User's Manual

Renesas Synergy™ Software Package（SSP）v1.1.0ユーザーズマニュアル（参考資料）

Renesas Synergy™ プラットフォーム

本資料は英語版を翻訳した参考資料です。内容に相違がある場合には英語版を優先します。資料によっては英語版のバージョンが更新され、内容が変わっている場合があります。日本語版は参考用としてご使用のうえ、最新および正式な内容については英語版のドキュメントをご参照ください。

資料番号 R01US0171EU0094、リビジョン Rev.0.94、発行日 2016 年 4 月 13 日の翻訳版です。

ルネサス エレクトロニクス
www.renesas.com

Rev.0.94　2016.10

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。

2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。

3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。

4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。

5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。

6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。

7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。

8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。

9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。

10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。

11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(2012.4)

参考資料

Renesas Synergy Software Package (SSP release v1.1) User's Manual

参考資料

第 1 章 Renesas Synergy™ Software Package（SSP）v1.1.0

このマニュアルでは、Synergy マイクロコントローラシリーズ対応のアプリケーションを、SSP を使用して作成する方法について説明します。下の図では、SSP ユーザーズマニュアルで説明する API リファレンスコンポーネントが青で示されています。このマニュアルではまた、SSP でのプログラミング手順を理解するため、e^2 studio ISDE の説明やチュートリアルと例など、その他のコンポーネントについても記載されています。

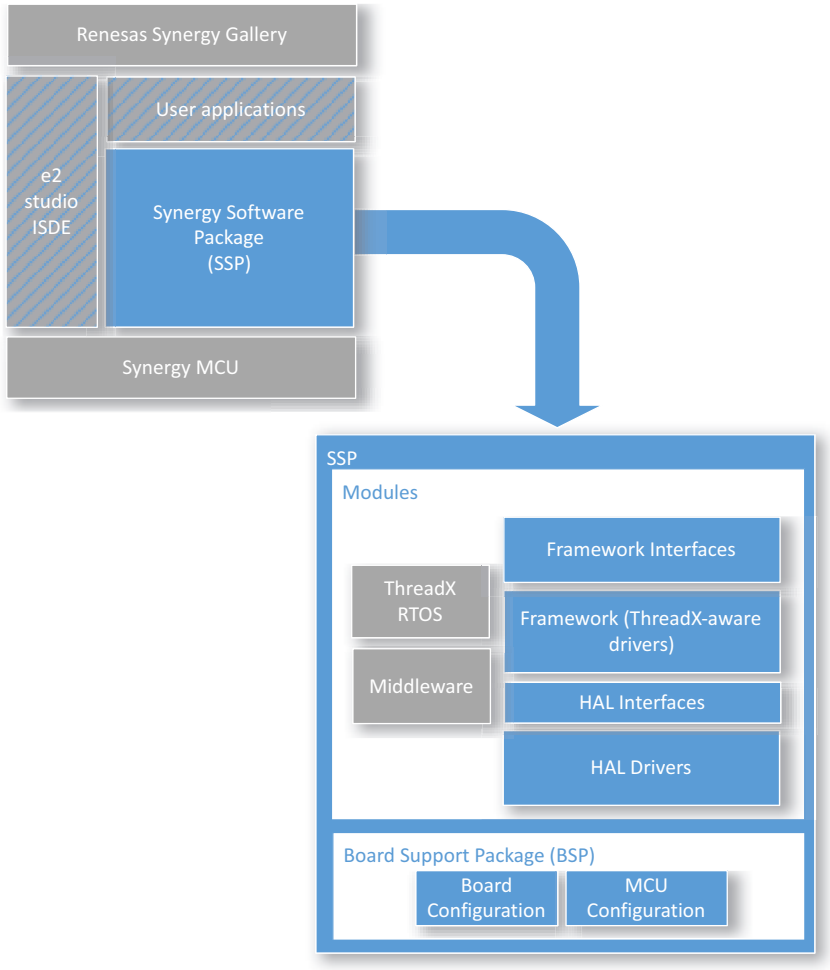

図 1: Synergy Software Package（SSP）のドキュメント

参考資料

1.1 このマニュアルの使用方法

Note：これらのページの内容は、Renesas Synergy Software Package v1.1.0 User's Manual, Rev.0.94 に基づいています。

このマニュアルの PDF バージョンについては、r01us0171eu0094_synergy_ssp.pdf を参照してください。

SSP アーキテクチャおよび SSP でのボードとチップレベルのサポートについては、以下を参照してください。

- SSP Architecture
- BSP Architecture

SSP を使用したプログラミングおよび e^2 studio ISDE の概要説明については、以下を参照してください：

- チュートリアル：e^2 studio ISDE ユーザーガイド

入門レベルのチュートリアルとアプリケーション例については、以下を参照してください。

- チュートリアル : Your First Synergy Project - Blinky
- チュートリアル : Using HAL Drivers - Programming the WDT

SSP モジュールについて説明したユーザーガイドについては、以下を参照してください。

- フレームワークレイヤー
- HAL レイヤー

このドキュメントには、以下の SSP コンポーネントの API リファレンスドキュメントが含まれています。

- API リファレンス フレームワークインタフェース : ThreadX 対応のフレームワークモジュールへのインタフェース
- API リファレンス フレームワークレイヤー : ThreadX 対応のフレームワークドライバモジュール
- API リファレンス HAL インタフェース : ハードウェア抽象化レイヤー（HAL）モジュールへのインタフェース
- API リファレンス HAL レイヤー : RTOS 独立の HAL ドライバモジュール
- Board Support Package：ボード固有、かつマイクロコントローラ固有の設定モジュールが含まれたボードサポートパッケージ（BSP）

1.2 ユーザーガイドおよび API リファレンスの使用方法

『ユーザーガイド』は、以下の説明をします。

- 各インタフェースの機能を理解する。
- アプリケーションから API 関数を呼び出す方法を確認する。
- インタフェースを使用するためのサンプルコードを参照する。
- インタフェースが適用される対象のハードウェアコンポーネントを確認する。
- インタフェースの制限を確認する。

『API リファレンス』は、以下の説明をします。

- すべての API 関数の定義を検索する。
- 設定構造体と制御構造体を参照する。
- 列挙値と定義を参照する。

参考資料

1.3 関連したドキュメント

以下の Express Logic 社の ThreadX コンポーネントに関連するドキュメントは、Renesas Synergy ギャラリーウェブサイトから入手できます。http://renesassynergy.com:

- NetX™
- FileX®
- USBX™
- GUIX™

参考資料

1.4 はじめに

1.4.1 目的

Renesas Synergy™ プラットフォームの一部である SSP は、使いやすく拡張性に優れた高品質の組み込みシステム設計ソフトウェアを提供するための完全な統合ソフトウェアパッケージです。Synergy ソフトウェアプラットフォームを使用することで、完全に統合された認証済みの組み込みソフトウェアプラットフォームが提供する、完全に最適化され、強化された業界屈指のリアルタイムオペレーティングシステム（RTOS）、ミドルウェア、通信スタック、機能ライブラリ、アプリケーションフレームワーク、ハードウェア抽象化されたローレベルデバイスドライバを利用でき、迅速な商品化が可能になります。

1.4.2 概要

SSP は、以下の 4 つの主要レイヤーで構成されています。

- API リファレンス フレームワークインタフェース：フレームワークライブラリは、ハードウェアを機能のユースケースとして抽象化する共通インタフェースを介して Synergy ハードウェア周辺機能に接続します。" インタフェースレイヤー " は、関数とパラメータの定義が入ったヘッダーファイルのグループであり、コード用のスペースは含まれていません。
- API リファレンス フレームワークレイヤー：" フレームワークレイヤー " は、RTOS 統合されたドライバおよび有益なアプリケーションコードで構成されています。
- API リファレンス HAL インタフェース："HAL レイヤー " インタフェースは、RTOS 独立の HAL レベルのドライバに接続します。
- API リファレンス HAL レイヤー：HAL レイヤーのドライバとハードウェアレジスタによって、インタフェースが実装されます。

1.4.3 使いやすさ

SSP は、統一的で直観的な API とそれに関する充実したドキュメントを提供しています。各モジュールには詳細なユーザードキュメントに加え、各関数のコードサイズや実行時間などを記載したソフトウェアデータシートが用意されています。

1.4.4 スケーラビリティ

ユーザーは、フレームワークレイヤー、インタフェースレイヤー、HAL レイヤーの中から、アプリケーションのニーズに最も適したレイヤーを使用してプラットフォーム機能と統合することができます。各モジュールをさらに拡張するには、パラメータチェッキングなどのビルド時オプションをコンパイルアウトすることで、小さく効率的なコードを作成できます。

参考資料

1.5 改訂履歴

1.5.1 Synergy Software Package v1.0.0-beta.3 User's Manual Rev.0.91

説明 : SSP v1.0.0-beta.3 の完全な文書化

日付 : 2015 年 12 月

ドキュメント ID: r01us0171eu0091_synergy_ssp

1.5.2 Synergy Software Package v1.0.0 User's Manual Rev.0.92

説明 : SSP v1.0.0 の完全な文書化

日付 : 2016 年 1 月

ドキュメント ID: r01us0171eu0092_synergy_ssp

1.5.3 Synergy Software Package v1.1.0-alpha User's Manual Rev.0.93

説明 : SSP v1.1.0-alpha の完全な文書化

日付 : 2016 年 2 月

変更点 :

- ユーザードキュメント : 新たに追加されたモジュールについては、SSP のリリースノートを参照してください。
- ユーザードキュメント : e^2 studio ISDE プロジェクトコンフィギュレータで使用されている名称と一致するようにすべてのユーザーモジュールガイドのタイトルを変更。
- ユーザードキュメント : メッセージングフレームワーク
  - 「サブスクライバーリストの設定」セクションを追加。
  - 「メッセージペイロード」セクションを更新。
  - 編集更新。
  - e^2 studio ISDE の [Messaging] タブの説明を追加。
- ユーザードキュメント : e^2 studio ISDE のタッチ パネル フレームワークにある [Messaging] タブの説明を追加。
- ユーザードキュメント : I2C フレームワークブロック図内の誤字を訂正。
- ユーザードキュメント : フラッシュ ドライバブロック図内の誤字を訂正。
- ユーザードキュメント : ディスプレイ ドライバ
  - パラメータ出力水平バックポーチサイクル数 : 説明を更新。

  - パラメータ出力垂直バックポーチサイクル数 : 説明を更新。
  - パラメータ出力データ有効性信号極性 : 設定を修正。
  - パラメータ出力同期エッジ : 設定を修正。
  - 「LCD タイミング」セクションを追加。
- ユーザードキュメント :「ADC ドライバ」セクションに ADC での温度センサーの使用を追加。
- ユーザードキュメント :「CTSU ドライバ」セクションの CTSU パラメータの設定を更新。
- ユーザードキュメント : BSP Architecture を Synergy S124 のパーツと DK-S124 キットが含まれるように更新。
- ユーザードキュメント :「SPI ドライバ」セクションに DTC （データトランスファコントローラ）を使用した SPI データ転送を追加。
- ユーザードキュメント :「セグメント LCD ドライバ」セクションの SLCDC ドライバのコード例を更新。
- API リファレンス : 新たに追加されたモジュールについては、SSP のリリースノートを参照してください。
- API リファレンス : インタフェースの詳細説明を若干、編集更新。
- API リファレンス : 構造体のドキュメント bsp_leds_t と bsp_lock_t を追加。

ドキュメント ID: r01us0171eu0093_synergy_ssp

1.5.4 Synergy Software Package v1.1.0 User's Manual Rev.0.94

説明 : SSP v1.1.0 の完全な文書化

日付 : 2016 年 4 月

変更点 :

- ユーザードキュメント : 新たに追加されたモジュールについては、SSP のリリースノートを参照してください。
- ユーザードキュメント : 次の SSP モジュールのユーザードキュメントを更新 :
  - 静電容量タッチフレームワーク
  - 静電容量タッチボタンフレームワーク
  - 通信フレームワーク
  - FileX 適応フレームワーク
  - GUIXTM Synergy ポートモジュール
  - JPEG デコード フレームワーク
  - メッセージングフレームワーク

参考資料

  - タッチ パネル フレームワーク
  - CTSU ドライバ
  - SD/MMC ドライバおよび SDIO ドライバ
  - タイマ ドライバ
  - JPEG デコード ドライバ
  - ローパワー モードドライバ
  - 低電圧検出ドライバ
  - ADC ドライバ
  - SPI ドライバ
  - CAN ドライバ
  - 入力キャプチャ ドライバ
  - I2C ドライバ
  - UART ドライバ
- ユーザードキュメント : e^2 studio ISDE 5.0 の e^2 studio ISDE ユーザーガイドを更新。
- ユーザードキュメント : BSP Architecture を更新。
- ユーザードキュメント : 静電容量タッチスライダフレームワークを追加。
- API リファレンス : 新たに追加されたモジュールについては、SSP のリリースノートを参照してください。

ドキュメント ID: r01us0171eu0094_synergy_ssp

第 2 章 SSP の概要

モジュールベースのアーキテクチャと機能的ソフトウェアレイヤーを使用して SSP でアプリケーションを開発する方法について確認します。複数のボードとボードサポートパッケージ（BSP）を使用した MCU で SSP アプリケーションを統合します。

以下のページでは、基礎的な SSP アーキテクチャについて説明します。

- SSP Architecture
- BSP Architecture

2.1 SSP Architecture

このセクションでは、SSP アーキテクチャと SSP アプリケーションプログラミングインタフェース（API）の使用方法について説明します。

2.1.1 SSP の概要

マイクロコントローラの複雑さが増すにつれ、幅広い知識がないと想定どおりに動作させることが難しくなっています。SSP は、ユーザーがより高いレベルで開発を開始できるようにする目的で提供されているソフトウェアパッケージです。ユーザーはハードウェアレジスタを参照する代わりに、充実したドキュメントが用意されている関数呼び出しを使用できます。

2.1.2 前提情報

ここからは、SSP を使用する場合の前提条件について説明します。

SSP ツールについては e^2 studio ISDE ユーザーガイドを参照してください。

2.1.3 C99 の使用

SSP では、ISO/IEC 9899:1999（C99）C プログラミング言語標準が使用されます。使用する C99 で導入された特定の機能には、標準の整数型（stdint.h）、ブール型（stdbool.h）、指定初期化子、および宣言とコードを組み合わせる機能が含まれます。

2.1.4 API パラメータ内での const の使用

const 修飾子は必要に応じて API パラメータと一緒に使用されます。実例を以下に示します。

```
ssp_err_t  (* open)(flash_ctrl_t  * const p_ctrl,

   flash_cfg_t  const * const p_cfg);
```

絶対確実というわけではありませんが、これは、SSP コード内で余分なチェックをいくらか実行することによって、変更すべきではない引数がそのように扱われることを保証します。

2.1.5 Doxygen

Doxygen は、SSP によって使用されるデフォルトのドキュメンテーションツールです。SSP ソースを探索すると、Doxygen コメントをドキュメントのさまざまな場所で見つけることができます。

参考資料

2.1.6 Weak シンボル

Weak シンボルは SSP 内で使用されています。このシンボルは、ユーザーがオプション関数を定義しなかった場合でもプロジェクトができるように使用されています。

2.1.7 SSP 用語

BSP 関連の用語	説明	参照先
モジュール	モジュールは、ペリフェラルドライバから単なるソフトウェアまでのすべてに該当します。各モジュールは、ソースコード、ドキュメンテーション、およびお客様がコードを効率的に使用するために必要な要素を含む 1 つのフォルダで構成されます。モジュールは独立した単位ですが、他のモジュールに依存する場合があります。SSP モジュールの例として以下が挙げられます。 ・UART ドライバー（UART インタフェース） ・Audio フレームワーク（Timer、DMA、および DAC ドライバーに依存する）（Audio フレームワークインタフェース） ・Messaging フレームワーク（Messaging フレームワークインタフェース） 複数のモジュールを組み合わせてユーザーに必要な機能を提供することによってアプリケーションを構築できます。	SSP モジュール
BSP	Board Support Package の略語。SSP では、BSP が他の SSP モジュールが問題なく連動できるようにするための必要最低限の機能を提供します。	BSP Architecture

参考資料

BSP 関連の用語	説明	参照先
コールバック関数	この用語はイベント発生時に呼び出される関数を意味します。たとえば、バスエラー割り込みハンドラーが r_bsp 内に実装されているとします。ユーザーは、バスエラーの発生を知りたい可能性があります。ユーザーに警告するため、コールバック関数を r_bsp に提供できます。バスエラーが発生すると、r_bsp が提供されたコールバック関数にジャンプして、ユーザーがそのエラーを処理できます。これは、この関数が呼び出されたときに、MCU はまだ割り込みの内部にあり、保留中の割り込みを後回しにします。	-
インタフェース	「SSP インタフェース」セクションを参照してください（SSP インタフェース）。SSP 内のすべてのインタフェースは、API リファレンス : フレームワーク インタフェースと API リファレンス : HAL インタフェース に記載されています。	SSP インタフェース
インスタンス	「SSP インスタンス」セクションを参照してください（SSP インスタンス）	SSP インスタンス
モジュールインスタンス	モジュールの単一で独立した構成。	-
アプリケーション	ユーザーが所有して維持管理するコード。アプリケーションコードは、Renesas から提供されるサンプルアプリケーションコードをベースに作成することができますが、責任はユーザー側にあります。	このマニュアルには、単純なアプリケーションの例がチュートリアルとして含まれています（チュートリアル : Using HAL Drivers - Programming the WDT）
ドライバ	ドライバは、MCU 上のレジスターを直接変更する特殊なモジュールです。	-
スタック	SSP アーキテクチャは、モジュールが連動してスタックを形成するように設計されています。最上位のモジュールから最下位のモジュールまでの依存関係が特定のスタックを形成します。	SSP スタック

BSP 関連の用語	説明	参照先
レイヤー / レベル	スタックは、複数のモジュールのレイヤーで構成されます。レイヤーは、1 つ上のレイヤーの要件に応じて 1 つまたは複数のモジュールで構成できます。レイヤーとレベルは区別なく使用されます。	SSP 定義レイヤー

2.1.8 SSP モジュール

モジュールは SSP の中核的構成要素です。モジュールを使用すれば、さまざまなことを実行できますが、すべてのモジュールが上位に機能を提供し、下位に機能を要求するという基本概念を共有しています。

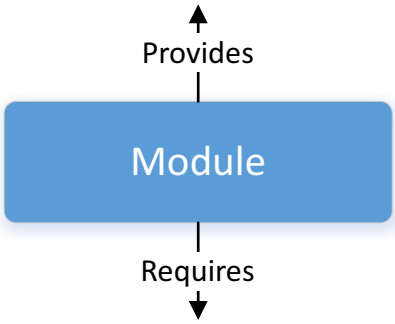

図 2:　　モジュール

モジュールによって提供される機能の量に制限はありませんが、通常は意味をなさなくなる限界点が存在します。提供される機能が多すぎると、将来、そのモジュールの再利用が困難になります。提供される機能が不十分な場合は、モジュールを想定どおりに機能させるための追加の処理が必要になります。

最も単純な SSP アプリケーションは、一番上にユーザーアプリケーションを含む 1 つのモジュールで構成されます。

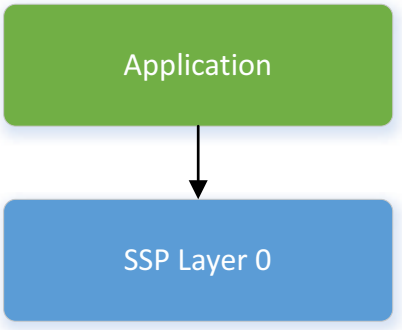

図 3:　　アプリケーションを含むモジュール

簡単にするために、この例ではボードサポートパッケージ（BSP）を無視しています。この図では、BSP は最下位レイヤー（SSP レイヤー 0）の下に配置されています。

2.1.9 SSP スタック

モジュールが上下に重なって、SSP スタックが形成されます。このスタッキングプロセスは、あるモジュールが提供しているものと別のモジュールが必要としているものが一致したときに実行されます。たとえば、オーディオ再生フレームワークモジュールには転送インタフェースが必要ですが、それはデータトランスファコントローラ（DTC）ドライバモジュールが使用できます。オーディオ再生モジュールに DTC コードが組み込まれるのではなく、それらが 2 つのモジュールに分割されています。これによりモジュールの再利用が可能になり、多くのメリットが得られます。

参考資料

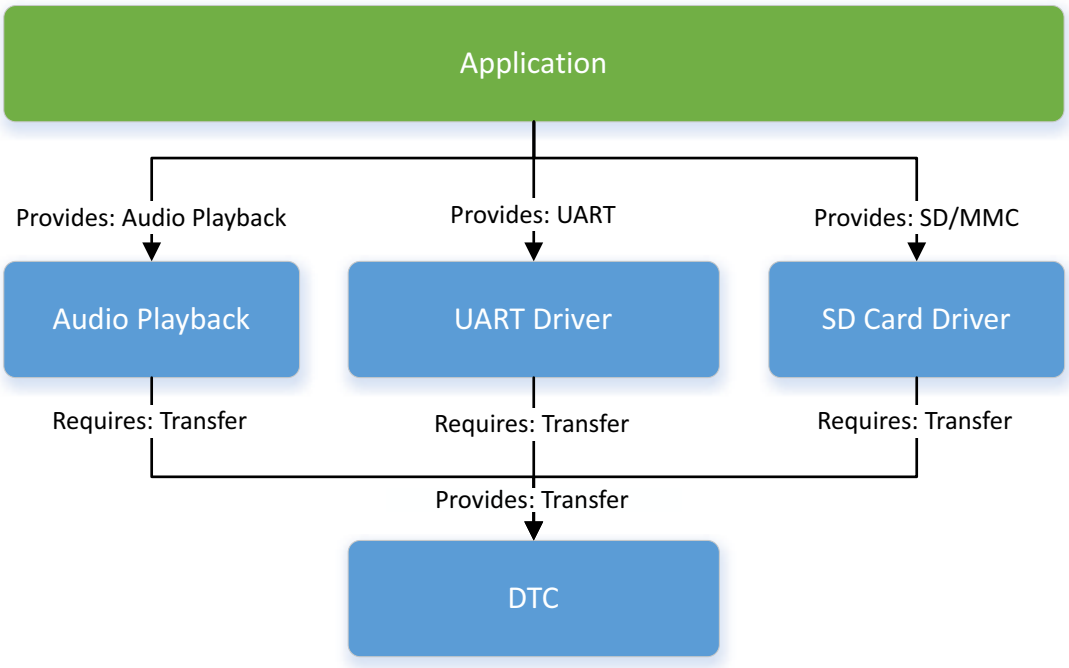

図 4:　　スタック – オーディオ再生

SSP モジュールを使ってレイヤーがスタックに加えられることにより、Synergy ハードウェアとハイレベルでインタフェースすることができます。

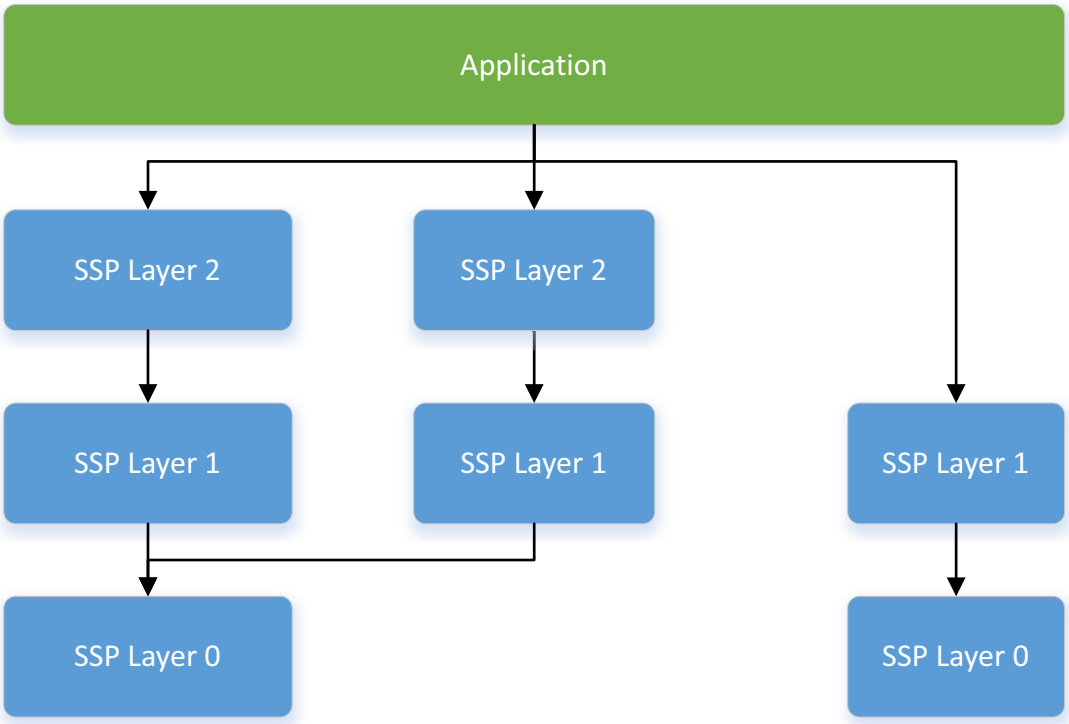

図 5:　　スタック

モジュールをスタックできると、アーキテクチャ全体の柔軟性が保証されるため、大きなメリットが得られます。モジュールが他のモジュールに直接依存している場合は、アプリケーション機能がさまざまなユーザー設計で機能できないときに問題が発生します。モジュールが再利用可能なことを保証するには、同じ機能を提供する他のモジュールに対してモジュールをスワップアウトできる必要があります。SSP アーキテクチャは、SSP インタフェースの使用を通してモジュールをスワップインまたはスワップアウトできる柔軟性を提供しています。

2.1.10 SSP インタフェース

アーキテクチャレベルでは、インタフェースが、モジュールが共通機能を提供する手段です。この共通性によって、同じインタフェースに接続されているモジュールを区別せずに使用できます。インタフェースは、2 つのモジュール間の契約と見なすことができます。モジュールは、契約内で同意された情報を使用して連動することに同意します。

Synergy ハードウェアでは、複数のペリフェラル間で機能が重複している場合があります。たとえば、I^2C 通信は IIC ペリフェラルまたは簡易 I^2C モードの SCI ペリフェラルの使用を通して実現できます。両方のペリフェラルから提供される機能のレベルに違いがあります。簡易 I^2C では、フル機能を備えた IIC の機能の一部しかサポートしません。

インタフェースは、ほとんどのユーザーが期待する共通機能のサポートを提供することを目的としています。これは、IIC などの周辺機器の一部の高度な機能がインタフェースで使用できない可能性があることを意味します。このような機能のほとんどはインタフェース拡張機能経由で使用することができます。

設計上は、インタフェースがヘッダーファイルとして実装されます。すべてのインタフェースヘッダーファイル名は "*_api.h" で終わります。ここからは、インタフェースの構成要素について説明します。

2.1.10.1 SSP インタフェース列挙

可能な場合、インタフェースは関数パラメータと構造体メンバーに対して型指定列挙を使用します。

```
typedef enum e_i2c_addr_mode

{

    I2C_ADDR_MODE_7BIT  = 1, //Use 7-bit addressing mode

    I2C_ADDR_MODE_10BIT         //Use 10-bit addressing mode

} i2c_addr_mode_t;
```

列挙を使用すると、パラメータで使用可能な値を決定する際の不確実性を取り除くことができます。また、列挙オプションは型の名前が使用可能なオプションに基づいてプレフィックスされる厳密な命名規則に従っていることにも注意してください。命名規則と e^2 studio ISDE で使用可能なオートコンプリート機能を組み合わせると、高度に読み取りやすいコードを維持しながら、コーディングが短時間で完成するメリットを得ることができます。

2.1.10.2 SSP インタフェースデータ構造体

少なくとも、すべての SSP インタフェースには、制御構造体、構成構造体、およびインスタンス構造体という 3 つのデータ構造体が含まれています。

制御構造体はモジュールを使用する場合の一意の識別子として使用されます。SSP モジュールが唯一のペリフェラルドライバの場合は、この制御構造体がチャネル番号に置き換えられます。その場合は、モジュールに対するすべての関数呼び出しでチャネル番号が使用されるため、コードで動作対象のペリフェラルチャネルを特定できます。SSP モジュールはデバイスドライバに制限されないため、制御構造体が使用されます。ユーザーは、制御構造体用のストレージを割り当ててから、その構造体へのポインタをモジュールの open() 呼び出しに渡します。この時点で、モジュールが必要に応じて構造体を初期化します。その後は、ユーザーがそれ以降のすべてのモジュール呼び出しに対して制御構造体へのポインタを送る必要があります。制御構造体の内容は、モジュールによって使用され、ユーザーによって変更されないようにする必要があります。

制御構造体の内容はモジュールによって異なります。デバイスドライバを実装するモジュールは、使用するチャネルを識別するだけの非常に小さな制御構造体を使用します。多くの場合、これはモジュールがメモリを内部で静的に割り当てることを意味します。この場合、モジュールは使用可能なチャネル数を正確に把握しています。単なるソフトウェアモジュールはそのように動作できません。これは、インスタンスの数が使用可能なメモリ容量によってのみ制限されることが多いためです。malloc() 関数と free() 関数を使用した動的メモリ割り当ては SSP モジュールでは使用されません。

構成構造体は、open() 呼び出し中のモジュールの初期設定に使用されます。この構造体は、チャネル、ビットレート、動作モードなどのメンバーで構成されます。この構造体は、モジュールへの入力のみに使用されます。この構造体は一意にする必要がなく、必要に応じてユーザーが初期化後に破棄することができます。

```
typedef struct st_i2c_cfg

{

/* Generic configuration */

    uint32_t channel; // Identifier recognizable by underlying instance

    i2c_rate_t  rate;  // Device's maximum clock rate in Hz, i.e. 400000

    uint16_t slave; // The address the device will respond to

    i2c_addr_mode_t  addr_mode; // Indicates how slave fields should be interpreted


    /* Parameters to control software behavior */

    void (*p_callback)(i2c_callback_args_t  * p_args); // Pointer to callback function

    void const * p_context; // Pointer to the user-provided context

    /* Instance-specific configuration */

    void const * p_extend; // Any configuration data needed by underlying hardware

} i2c_cfg_t;
```

上は I^2C インタフェースの設定構造体の例です。最後の 3 つの構造体メンバー（p_callback、p_context、および p_extend）は、ほぼすべてのモジュール設定に共通です。

p_callback および *p_context* メンバーについては、「SSP インタフェース：コールバック関数」のセクションで説明します。

p_extend メンバーは、特定のインスタンス用の現在のインタフェースを拡張するために使用されます。インタフェースは最も一般的な機能をサポートするように設計されています。インタフェースのインスタンスがそれ自体を正しく設定するために特別な情報が必要な場合があります。特別な情報が必要ない場合もありますが、ユーザーが特定のアプリケーションに合わせてモジュールを調整するために必要とする場合もあります。その場合、ユーザーは、*p_extend* メンバーを通じて渡すことにより、基礎となるインスタンスに追加の構成情報を提供できます。このメンバーを介して渡される情報は基礎となるインスタンスによって定義されるため、ユーザーはその構造体に忠実に従う必要があります。無効な情報が基礎となるドライバに渡された場合は、インスタンスでそのデータを正しく処理できないため、正常な動作が保証されません。詳細については、「インタフェース拡張機能」の項を参照してください。

構成構造体に現在のインタフェースに適用されるメンバーだけが入っていることも重要です。同じスタック内の複数のレイヤーで同じ設定パラメータが定義されていると、オプションをどこで変更するかを知るのが

難しくなります。たとえば、UART のボーレートはドライバレイヤーでのみ定義されます。UART インタフェースを使用するすべてのレイヤーは、ドライバレイヤーで指定されているボーレートに依存し、独自の構成構造体でそれを提供することはありません。

2.1.10.3 SSP インタフェースコールバック関数

コールバック関数を使用すれば、モジュールは、イベント発生時に非同期的にユーザーアプリケーションに警告することができます。イベントの例には、UART チャネル経由でのバイト受信があります。ユーザーアプリケーションコードで割り込みに対処するには、コールバックが必要です。SSP モジュールが MCU ペリフェラルの割り込みを定義して処理します。ユーザーが SSP モジュールと同時に割り込みサービスルーチンを定義しようとしても、そのコードはビルドされません。そのため、SSP モジュールを使用すれば、割り込み発生時に呼び出す関数を登録しておくことによって、ユーザーアプリケーションで割り込みに対処できます。

コールバック関数はユーザーアプリケーション内で定義する必要があります。この関数は常に *void* を返し、1 つのパラメータに対して 1 つの構造体を使用します。構造体 typedef はモジュールのインタフェースで指定され、*<interface>_callback_args_t* という名前です。構造体の内容はインタフェースによって異なる可能性がありますが、*event* と *p_context* の 2 つは共通です。

event メンバーは、アプリケーションが、コールバックが呼び出された理由を特定するために使用されます。前の UART の例では、バイトが受信された、すべてのバイトが送信された、またはフレーミングエラーが発生した場合にコールバックがトリガされた可能性があります。*event* メンバーはインタフェースで指定される列挙です。

p_context メンバーは、ユーザーが指定したデータをコールバック関数に渡すために使用されます。多くの場合、コールバック関数は、複数のチャネルまたはモジュールインスタンス間で共有されます。コールバックが発生すると、それを処理するコードがコールバック対象のモジュールインスタンスを特定するためにコンテキスト情報を必要とします。たとえば、コールバックで SSP API 呼び出しを発行する場合は、少なくともコールバックで制御構造体を使用する必要があります。これを簡単にするために、ユーザーは制御構造体へのポインタを *p_context* として指定できます。コールバックが発生するときは、コールバック構造体で渡される制御構造体を使用できます。

コールバック関数は割り込みサービスルーチン内から呼び出されます。そのため、ユーザーのシステムのリアルタイム性能に影響を与えないようにコールバック関数はできるだけ短くする必要があります。フラッシュインタフェースコールバックのスケルトン関数の例を以下に示します。

```
static void flash_callback (flash_callback_args_t * p_args)
{
    /* See what event caused this callback. */
    switch (p_args->event)
    {
        case FLASH_EVENT_ERASE_COMPLETE:
        /* Handle event. */
        break;
        case FLASH_EVENT_WRITE_COMPLETE:
        /* Handle event. */
        break;
        case FLASH_EVENT_BLANK:
        /* Handle event. */
        break;
        case FLASH_EVENT_NOT_BLANK:
        /* Handle event. */
        break;
        case FLASH_EVENT_ERR_DF_ACCESS:
```

```
            /* Handle error. */

            break;

            case FLASH_EVENT_ERR_CF_ACCESS:

            /* Handle error. */

            break;

            case FLASH_EVENT_ERR_CMD_LOCKED:

            /* Handle error. */

            break;

    }

}
```

モジュールがユーザーアプリケーション内で直接使用されていない（それがスタックの最上位レイヤーではない）場合は、そのコールバック関数が上記モジュールによって処理されます。UART インタフェースモジュールが必要なコンソールインタフェースモジュールが存在する場合は、コンソールモジュールが UART のコールバック関数を制御および使用します。この場合、ユーザーはアプリケーションコード内で UART モジュール用のコールバック関数を作成する必要がありません。

2.1.10.4 SSP インタフェース API 構造体

すべてのインタフェースには、サポートされているすべてのインタフェース関数の関数ポインタを含む API 構造体が含まれています。コメントが削除された、デジタル / アナログ変換（DAC）用の構造体の例を以下に示します。

```
typedef struct st_dac_api

{

    ssp_err_t  (*open)(dac_ctrl_t  *p_ctrl, dac_cfg_t  const *const p_cfg);

    ssp_err_t  (*close)(dac_ctrl_t  *p_ctrl);

    ssp_err_t  (*write)(dac_ctrl_t  *p_ctrl, dac_size_t  *p_value);

    ssp_err_t  (*start)(dac_ctrl_t  *p_ctrl);

    ssp_err_t  (*stop)(dac_ctrl_t  *p_ctrl);

    ssp_err_t  (*versionGet)(ssp_version_t  *p_version);

} dac_api_t;
```

API 構造体は、モジュールを同じインタフェースのインスタンスである他のモジュールに対して簡単にスワップインまたはスワップアウトできるようにするために使用されます。上記 DAC インタフェースを使用したアプリケーションの例を見てみましょう。

MCU は DAC を内蔵しています。DAC インタフェースで DAC API 構造体が使用されていない場合は、アプリケーションから直接モジュールを呼び出すことができます。下の例では、アプリケーションは r_dac モジュールで提供されている R_DAC_Write 関数を呼び出しています。

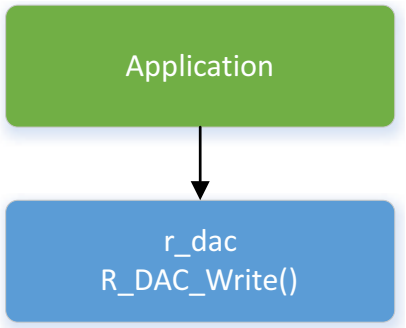

図 6: DAC 書き込みの例

ここで、MCU 上で使用可能な DAC チャネルだけでは足りず、ユーザーが r_dac_external という名前の新しい外部 DAC モジュールを追加する場合を考えます。外部 DAC は通信に I^2C を使用します。アプリケーションは 2 つのモジュールを区別する必要があるため、複雑さが増してアプリケーションに対する依存度が高まります。

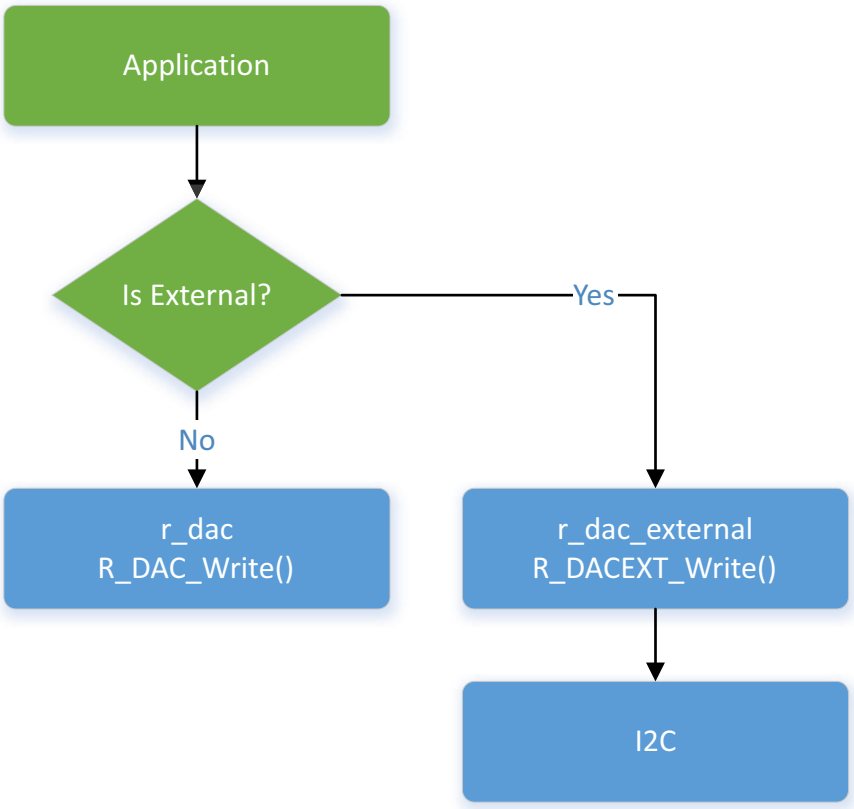

図 7: 2 つの書き込みモジュールを使用した DAC 書き込み

インタフェースと API 構造体の使用が理論上の DAC の使用を可能にします。これは、外部ロジックが不要で、アプリケーションが特定のハードコードされたモジュールに依存しないことを意味します。代わりに、アプリケーションは、任意の数のモジュールで実装可能な DAC インタフェース API に依存することになります。

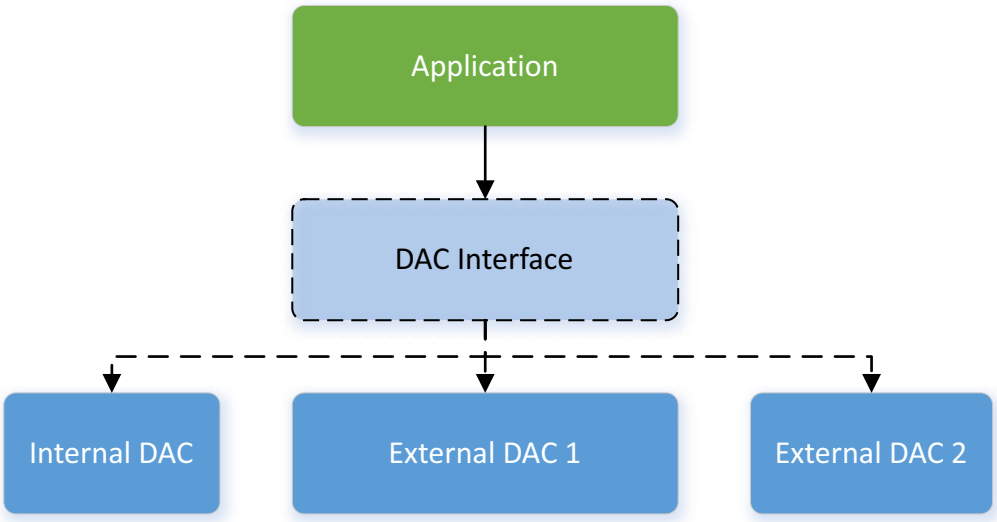

図 8: DAC インタフェース

API 構造体の内部関数は共通の名前に従っています。ほとんどのモジュールが open() 関数と close() 関数のペアを使用します。open() 関数は他の関数の前に呼び出す必要があります。唯一の例外は、ユーザーが指定した情報に依存しない versionGet() 関数です。

共通して使用される他の関数は、read()、write()、get()、および set() です。関数名は名詞の後に動詞が続くように設計されています。名前の例を以下に示します。

- read()、write()、writeRead()
- statusGet()
- calendarAlarmSet()、calendarAlarmGet()
- accessWindowSet()、accessWindowClear()

2.1.10.5 SSP インタフェースバージョン情報

すべてのインタフェースが versionGet() 関数を提供します。この関数は *ssp_version_t* という型の構造体を返します。この構造体は、インタフェース（API）用と現在使用されている基礎となるインスタンス用の 2 つのバージョンで構成されます。

```
typedef union st_ssp_version

{

uint32_t version_id;

struct

{

      uint8_t code_version_major; // Code major version

      uint8_t code_version_minor; // Code minor version

      uint8_t api_version_major;  // API major version

      uint8_t api_version_minor;  // API minor version

};

} ssp_version_t;
```

API バージョンは変更されないことが理想ですが、稀に変更される場合があります。API を変更するには、遡ってコードを変更する必要があります。コードバージョン、つまり、現在のインスタンスのバージョンは頻繁に更新される可能性があります。バグの修正、機能強化、および追加機能のすべては、コードバージョンの増加につながる可能性があります。ユーザーコードがインスタンスから提供される拡張機能を使用している場合、コードバージョンの変更ではユーザーコードだけを変更する必要があります。

2.1.10.6 SSP インスタンス

インタフェースは指定された機能を指示するのに対して、インスタンスは実際にそれらの機能を実装します。インスタンスごとに特定のインタフェースに関連付けられます。インスタンスは、インタフェースからの列挙、データ構造体、および API プロトタイプを使用します。これにより、インスタンスを使用するアプリケーションは必要に応じてインスタンスをスワップアウトすることができます。

MCU では、一部の周辺機器がインタフェースとインスタンスの 1 対 1 マッピングを使用しますが、それ以外の周辺機器は 1 対多マッピングを使用します。下の例では、IIC 周辺機器と SPI 周辺機器がそれぞれ 1 つのインタフェースにしかマッピングされていないのに対して、SCI 周辺機器は 3 つのインタフェースを実装しています。

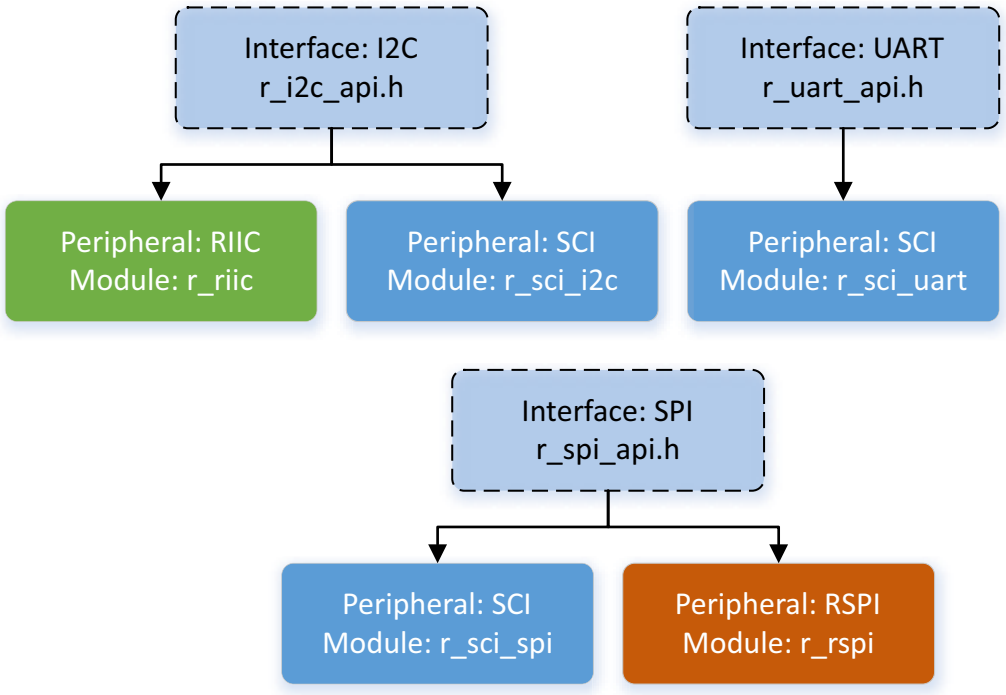

図 9: インスタンス

2.1.10.7 SSP インスタンスの API 構造体

各インスタンスには、インタフェースの API を実装するその関数を含む固定のグローバル構造体が含まれています。この構造体の名前は、*g_<interface>_on_<instance>* として標準化されています。例として、g_spi_on_spi、g_transfer_on_dtc、g_adc_on_adc などが挙げられます。この構造体は、インスタンスヘッダーファイル（それぞれ、r_spi.h、r_dtc.h、および r_adc.h）内の extern 経由で使用できます。

2.1.11 ビルド時間設定

すべてのモジュールに、ビルド時間設定ヘッダーファイルが割り当てられています。ほとんどの構成オプションは実行時に提供されます。稀にしか使用されない、または、モジュールのすべてのインスタンスに適用される一部のオプションはビルド時に移動される場合があります。ビルド時構成オプションを使用するメリットは、未使用の機能を削除することによってコードサイズを小さくできる可能性があることです。

すべてのモジュールに、それぞれのパラメータチェックを有効化または無効化するための 1 つ以上のビルド時オプションが付属しています。SSP モジュールは、可能な限り、関数引数の妥当性をチェックします。ユーザーは、テストの完了時にこの機能を無効化して、コードスペースを節約し、実行速度を上げることができます。

2.1.12 インタフェース拡張機能

一部の場合では、インスタンスがインタフェースから提供される情報以上の情報を必要とします。この状況は次の 2 つの場合に発生する可能性があります。

- あるインスタンスがインタフェースのほとんどのインスタンスに共通しない特別な機能を提供している。
- あるインタフェースを必要を超えて汎用的にしなければならない。インタフェースが汎用的になるほど、使用可能なインスタンスの数が増えます。この典型例がブロックメディアインタフェースです。

```
typedef struct st_sf_block_media_cfg

{

    uint32_t block_size; // Block size in bytes

    void * p_extend; // Instance dependent configuration

} sf_block_media_cfg_t;
```

ブロックメディアインタフェースの構成構造体は意図的に小さく作られています。これにより、ほぼ無限のインスタンスが可能になります。例として、SD カード、SPI フラッシュ、SDRAM、USB などが挙げられます。インスタンスごとに別々の設定情報が必要です。これは、*p_extend* パラメータ経由で情報を提供することによって実現されます。この *p_extend* 内で提供される構成データはインスタンス間で同じではありませんが、その後の API 呼び出しは同じです。これは、1 か所を変更するだけで済むことを意味します。

必ずしも、インタフェース拡張機能を使用する必要はありません。すべての機能がインタフェースで提供されるため、一部のインスタンスは拡張機能を提供しません。大抵の場合、*p_extend* メンバーを NULL に設定できます。NULL が指定され、インスタンスが拡張機能を提供する場合、インスタンスはこれを、デフォルトのオプションが使用されることを意味するものと見なします。各インスタンスのドキュメンテーションによって、インタフェース拡張機能が提供されているかどうかと、その使用が必須か任意かが指定されます。

2.1.13 SSP 定義レイヤー

SSP には、2 つの定義されたレイヤー（ドライバレイヤーとフレームワークレイヤー）があります。これらのレイヤーは、モジュールが別々のフォルダ内に存在し、プレフィックスが異なるため、簡単に識別できます。ドライバレイヤーモジュールは synergy\ssp\src\driver フォルダに配置され、フレームワークレベルモジュールは synergy\ssp\src\framework に配置されます。ドライバレイヤー内のモジュールは *r_* プレフィックスで始まりますが、フレームワークレベルモジュールは *sf_* プレフィックスで始まります。

両レイヤー間の機能的な主な違いとして、ドライバレイヤーモジュールには RTOS に対応した周辺機器ドライバでなければならないという制限がありますが、RTOS オブジェクトの使用や RTOS API 呼び出しを行うことはありません。これは、RTOS のある（またはない）アプリケーションでドライバレイヤーモジュールを使用できることを意味します。

フレームワークレイヤーモジュールはセマフォ、ミューテックス、イベントフラグなどの RTOS オブジェクトを自由に使用できます。フレームワークモジュールは、必要に応じて独自の RTOS オブジェクトを作

成することもできます。MCU にアクセスする必要があるフレームワークレイヤーモジュールは、通常、ドライバレイヤーインタフェースを通じてそれを行います。複数の周辺機器を同時に使用しなければならず、複数の個別インタフェースを使用することが実際的でない特殊な場合は、例外を認めることができます。

2.1.14 SSP フォルダの構造

SSP のハイレベルのフォルダ構造を以下に示します。

```
+-synergy
| +-ssp
|   +-inc
|   | +-bsp
|   | +-driver
|   | | +-api
|   | | +-instances
|   | +-framework
|   |   +-api
|   |   +-instances
|   | +-framework
|   |   +-api
|   |   +-instances
|   +-src
|     +-bsp
|     +-driver
|     | +-r_module
|     +-framework
|       +-sf_module
+-synergy_cfg
  +-ssp_cfg
    +-bsp
    +-driver
    +-framework
```

ベース *ssp* フォルダの直下で、フォルダは source フォルダと include フォルダに分岐しています。include フォルダは、include パスの参照とセットアップを容易にするために source から分離されています。*ssp/inc* フォルダと *ssp/src* フォルダの下に、同じフォルダのセット（*bsp*、*driver*、*framework*）があります。

BSP と違って、SSP の 2 つの事前定義レイヤー（ドライバとフレームワーク）が存在します。ドライバレイヤーモジュールは *ssp/src/driver* フォルダに位置していますが、フレームワークレイヤーモジュールは *ssp/src/framework* フォルダに位置しています。include ツリーの下では、ドライバレイヤーフォルダとフレームワークレイヤーフォルダにそれぞれ 2 つずつのフォルダ（*api* と *instances*）が含まれています。*api* フォルダには、そのレイヤー用のインタフェースヘッダーファイルが含まれています。*instances* フォルダには、そのレイヤー用のインスタンスヘッダーファイルが含まれています。両方のレイヤーが内部的に平坦であるため、プロジェクトに必要な include パスの数は制限されます。

ssp_cfg フォルダには、モジュールごとの構成ヘッダーファイルが保存されます。そのレイアウトは、*ssp* フォルダと同じで、BSP、Driver、および Framework レイヤーが別々の平坦なディレクトリで構成されます。これらのヘッダーファイルで提供される情報については、「ビルド時間設定」の項を参照してください。

2.1.15 SSP 接続レイヤー

SSP モジュールは再利用可能かつスタック可能に設計されています。モジュールは他のモジュールには依存しませんが、他のインタフェースには依存します。ユーザーは、ニーズに最適なインスタンスを使用したインタフェースを自由に構築できます。

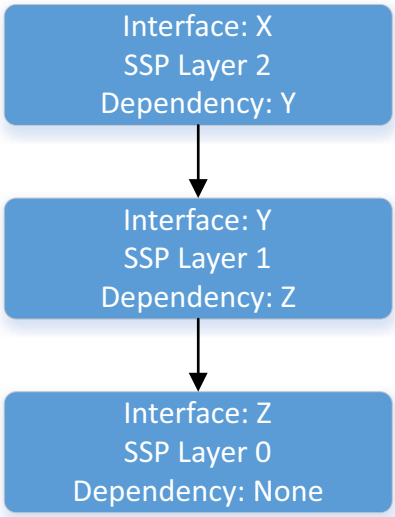

図 10: 接続レイヤー

上の図では、インタフェース Y がインタフェース X から依存されていると同時に、インタフェース Z に依存しています。インタフェース X はインタフェース Y にのみ依存しています。また、インタフェース X はインタフェース Z を認識していません。これがレイヤーを簡単にスワップアウトできることを保証するための要件です。これを次の図に示します。

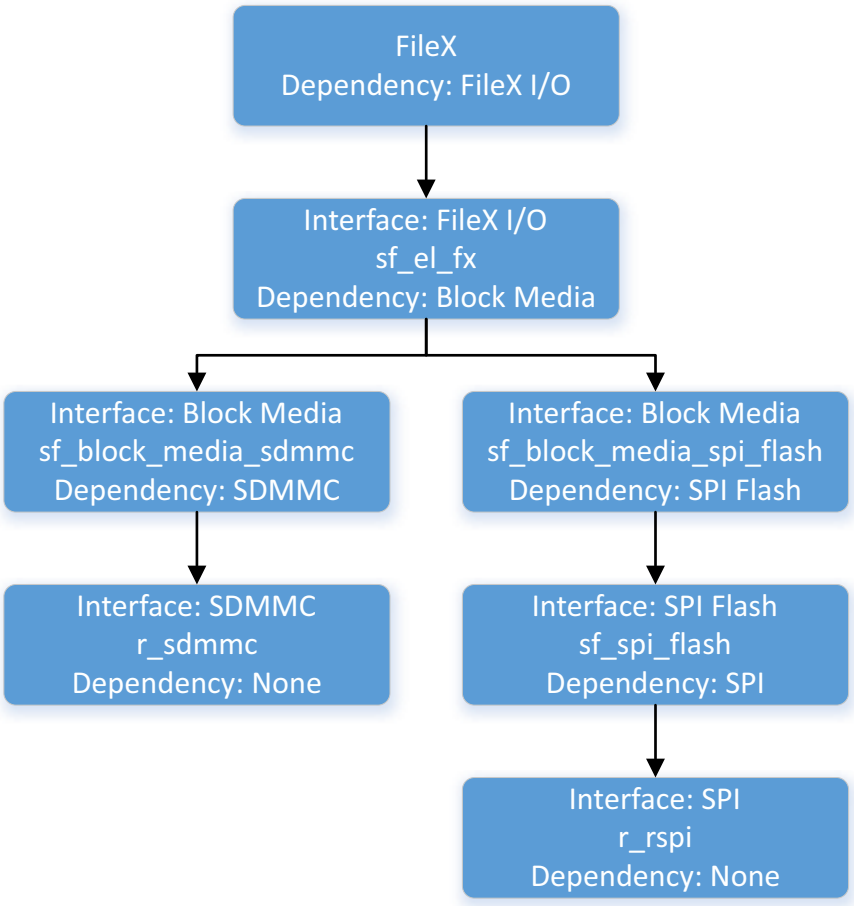

図 11: FileX インタフェースを使用した接続レイヤー

この例では、2 つのストレージメディア（SDMMC と SPI フラッシュ）上の Express Logic, Inc. FileX ファイルシステムを使用しています。SPI フラッシュインタフェースは SPI フラッシュプロトコルを処理しますが、実際の SPI バス通信には SPI インタフェースが必要です。SDMMC インタフェースはプロトコルとバス通信を処理するため、何にも依存しません。

SDMMC と FileX インタフェースの詳細については、FileX 適応フレームワークを参照してください。

2.1.16 実際の SSP アーキテクチャ

SSP スタック内の各レイヤーには、その依存対象の API 関数を呼び出す責任があります。このことは、ユーザーがインタフェースしているレイヤーで API 関数を呼び出す責任しかないことによっても説明することができます。上の FileX の例では、ユーザーにはアプリケーションコード内で FileX 関数を呼び出す責任しかありません。内部的に、この後 FileX は FileX I/O を呼び出し、次いで FileX I/O はブロックメディアインタフェースモジュールを呼び出します。ブロックメディアインタフェースは複数のドライバを呼び出すこ

とができます。少なくとも、上位レイヤーモジュールが、それが依存しているインタフェースの *open()* 関数を呼び出します。

モジュールを使用するアプリケーションを作成するには、次のことを決定する必要があります。

1) 呼び出す open() 関数を決定します。依存関係がインタフェースに基づくということは、モジュールが何らかの方法で呼び出すインスタンスを識別できなければならないことを意味します。
2) 設定パラメータを決定します。モジュールは、伝達する構成情報を認識する必要もあります。モジュールが特定の設定パラメータを設定しなければならない場合もあります。その場合は、モジュールがそれらの設定構造体メンバー自体を設定してから、残りの構造体を伝達します。残りの構成構造体メンバーはモジュール外部で指定する必要があります。
3) モジュールインスタンス固有であり、上位レイヤーモジュールで割り当て可能な制御構造体を提供します。

つまり、モジュールインスタンスとやり取りするには次のものが必要です。

- インスタンスの API 構造体へのポインタ
- モジュールインスタンスの構成構造体へのポインタ
- モジュールインスタンスの制御構造体へのポインタ

これは、任意のモジュールを使用するのに十分な情報です。API 構造体はインスタンスに固有である唯一の構造体であり、モジュールインスタンス固有ではありません。これは、API 構造体が同じインスタンスの複数の使用で変化しないためです。SPI が SCI チャネル 0 および 2 で使用されている場合、両方のモジュールインスタンスで同じ API 構造体が使用されますが、構成および制御構造体は異なります。

モジュールインスタンスをより使いやすくするため、これらのすべての部分は各インタフェースで見つかったインスタンス構造体内にカプセル化されます。これらの構造体は、*<interface>_instance_t* という標準化された名前を持っています。WDT インタフェースからの例を下に示します。

```
typedef struct st_wdt_instance

{

    wdt_ctrl_t  *p_ctrl; // Pointer to the control structure for this instance

    wdt_cfg_t  const *p_cfg; // Pointer to the configuration structure for this instance

    wdt_api_t  const *p_api; // Pointer to the API structure for this instance

}wdt_instance_t;
```

インタフェースに対する依存関係を持つ上部レイヤーモジュールは、インスタンス構造体を使用して、そのインタフェースのインスタンスとやり取りするのに必要なすべての情報を保持できます。上の WDT の例に続いて、スレッド監視フレームワークインタフェース構成構造体を下に示します。このスレッド監視インタフェースは WDT インタフェースに依存しています。

```
typedef struct st_sf_thread_monitor_watchdog_type

{

    wdt_instance_t  * p_lower_lvl_wdt; // Pointer to lower level watchdog instance

    bool profiling_mode_enabled;      // Enables or disables profiling mode

    UINT priority;                    // Priority of thread monitor thread

} sf_thread_monitor_cfg_t;
```

スレッド監視モジュールには、WDT インタフェースとやり取りするのに必要なすべての情報を *p_lower_lvl_wdt* 構造体メンバー内に持っています。

ある場合、モジュール依存関係はインタフェースではなくインスタンスで定義されています。たとえば、ブロックメディアインタフェースは、SDMCC、SPI フラッシュ、または他の多くのインスタンスで実装できます（も参照してください）。実装方法は広範囲にわたるため、特定のインタフェースのインスタンス構造体をブロックメディアインタフェースの構成構造体で直接使用することはできません。ブロックメディアインタフェースの構成構造体を再び下に示します。

```
typedef struct st_sf_block_media_cfg

{

    uint32_t block_size; // Block size in bytes

    void * p_extend; // Instance dependent configuration

} sf_block_media_cfg_t;
```

インスタンス構造体ポインタが指定されていることに注意してください。この理由は、前述のようにブロックメディアインタフェースが汎用的すぎて特定のインタフェースに対する依存関係を強制できないためです。

モジュールがブロックメディアなどの汎用インタフェースのインスタンスで、他のモジュールに依存している場合は、インタフェースの *p_extend* 設定メンバー経由で参照される拡張機能構造体内に下位レイヤーポインタを配置します。これは、インタフェースの拡張を強制せずにモジュールスタッキングを可能にして、複数のオプション構成メンバーを使用できるようにするために必要です。

```
typedef struct st_block_media_on_sdmmc_cfg

{

    sdmmc_instance_t  const * const p_lower_lvl_sdmmc; // Pointer to SDMMC instance structure

} sf_block_media_on_sdmmc_cfg_t;
```

2.1.17 SSP モジュールの使用

ここでは、SSP モジュールの使用方法に関する一般的な情報を提供します。

2.1.17.1 インタフェースの選択

必要な機能用のインタフェースを選択することから始めます。たとえば、UART 通信には UART インタフェースを使用します。

2.1.17.2 インタフェースに適切なインスタンスの検索

インタフェースを選択したら、それに対応するインスタンスを選択します。インタフェースの既知のインスタンスのリストは、インタフェースのドキュメンテーションコメント内に記載されています。選択したインスタンスのヘッダーファイルを、そのインスタンスを使用するアプリケーションのソースファイルにインクルードします。

2.1.17.3 制御構造体と構成構造体の割り当て

e^2 studio ISDE には、インタフェースおよびインスタンス構成構造体のパラメータを設定するためのグラフィカルユーザーインタフェースが備わっています。また、GUI で構成されたそれらの構造体は、アプリケーションコードにインクルード可能なアプリケーション固有のヘッダーファイルに自動的に含められます。

e^2 studio ISDE で構成が処理される方法については、『e^2 studio ISDE ユーザーズガイド』の e^2 studio ISDE ユーザーガイドを参照してください（プロジェクトの設定）。

構成および制御構造体の型は、それぞれ <interface>_ctrl_t* および *<interface>_cfg_t* という標準名に従います。e^2 studio ISDE では、e^2 studio ISDE が作成するアプリケーション固有のヘッダーファイル内に、これら両方の構造体用のストレージが割り当てられます。e^2 studio ISDE の [Properties] ビューを使用し、必要に応じて構成構造体のメンバーの値を設定します。多くのメンバーが、使用可能なオプションで列挙を参照可能な型指定列挙です。

インタフェースにコールバック関数オプションが付属している場合は、最初に、ソースコードでその関数を宣言して定義する必要があります。戻り値は常に型 *void* で、関数へのパラメータは *<interface>_callback_args_t* という名前の型指定構造体です。関数が定義されたら、その名前を構成構造体の *p_callback* メンバーに割り当てます。コールバックでコンテキスト情報が必要な場合、ユーザーは *p_context* メンバーへのポインタを指定できます。e^2 studio ISDE で選択したモジュールの [Properties] ビューを通じて、コールバック関数名を割り当てることができます。

インタフェース拡張機能が提供されているかどうかを確認するには、インスタンスドキュメンテーションを参照してください。提供されている場合は、*<interface>_on_<instance>_cfg_t* という名前のインスタンスヘッダーファイルに含められています。これは、インタフェースの設定構造体と同様に複数のメンバーで構成される場合があります。特定のインスタンスでドライバを選択する場合は、e^2 studio ISDE プロパティでそのインスタンスの構成構造体の任意のパラメータを選択できます。

2.1.17.4 インタフェースのインスタンス構造体を使用したやりとり

インスタンス構造体が定義された後は、必要に応じてインスタンスとやり取りできます。下に示すのは、SCI で実装されている UART インタフェースのインスタンス構造体を構築するコードです。Synergy で e^2 studio ISDE を使用する場合は、次のコードがユーザー用に自動的に生成されることに注意してください。

```
/* Include the header file of the Instance. */

\#include “r_sci_uart.h” //This will in turn include the r_uart_api.h Interface

/* Allocate control structure. */

uart_ctrl_t  my_uart_ctrl;

/* Setup extended UART configuration on SCI. */

uart_on_sci_cfg_t  my_uart_extended_cfg =

{

     /* Set extended configuration members... */

};

/* Configure standard UART Interface. */

uart_cfg_t  my_uart_cfg =

{
```

```
    .data_bits = UART_DATA_BITS_8,

   /* Continue configuring other members... */

    .p_extend = &my_uart_extended_cfg

};

/* Setup instance structure */

uart_instance_t  my_uart = {

    .p_ctrl  = &my_uart_ctrl,

    .p_cfg = &my_uart_cfg, //Extended configuration is brought through in p_extend

    .p_api = &g_uart_on_sci //Defined in r_sci_uart.h

};
```

これでインスタンス構造体は準備完了です。UART インタフェースとやり取りできます。e^2 studio ISDE では、インスタンス構造体の名前は、e^2 studio ISDE の [Properties] ビューでモジュールインスタンスを構成するときに指定した名前 * です。

```
ssp_err_t  err;

/* Initialize UART */

err = my_uart.p_api->open(my_uart.p_ctrl, my_uart.p_cfg);

/* Check return for errors. */

if (SSP_SUCCESS != err)

{

/* Handle error. */

}

/* Use other Interface functions. */

err = my_uart.p_api->write(my_uart.p_ctrl, ...);

err = my_uart.p_api->read(my_uart.p_ctrl, ...);
```

2.2 BSP Architecture

このセクションでは、BSP（ボードサポートパッケージ）について説明します。API リファレンスについては、Board Support Package を参照してください。BSP はボード固有であり、結果的に MCU 固有です。

2.2.1 BSP 関連用語

BSP 関連の用語	意味
Filename_xxxx.c	'xxxx' は MCU の種類を表します。たとえば、S7G2 を参照する場合は Filename_S7G2.c になります。
BSP	Board Support Package の略語。BSP には、一般に特定のボードに関連するソースファイルがあります。
コールバック関数	この用語はイベント発生時に呼び出される関数を意味します。ユーザーは、NMI システム例外の発生を知りたい可能性があります。ユーザーに通知するため、グループ割り込み（すべてが NMI に結び付けられている例外のグループ）に対してコールバック関数を設定できます。NMI が発生すると、BSP は指定されたコールバック関数にジャンプして、ユーザーがエラーを処理できます。割り込みコールバック関数は長くならないように慎重に扱う必要があります。これは、この関数が呼び出されたときに、MCU はまだ割り込みの内部にあり、保留中の割り込みが後回しにされるためです。

2.2.2 BSP の機能

BSP には、リセット以降の MCU をユーザーアプリケーション（main() 関数など）に到達できるようにする責任があります。ユーザーアプリケーションに到達する前に、BSP は、スタック、ヒープ、クロック、割り込み、および C ランタイム環境をセットアップします。また、BSP は、ポート I/O ピンを設定およびセットアップし、ボード固有の初期化も行います。

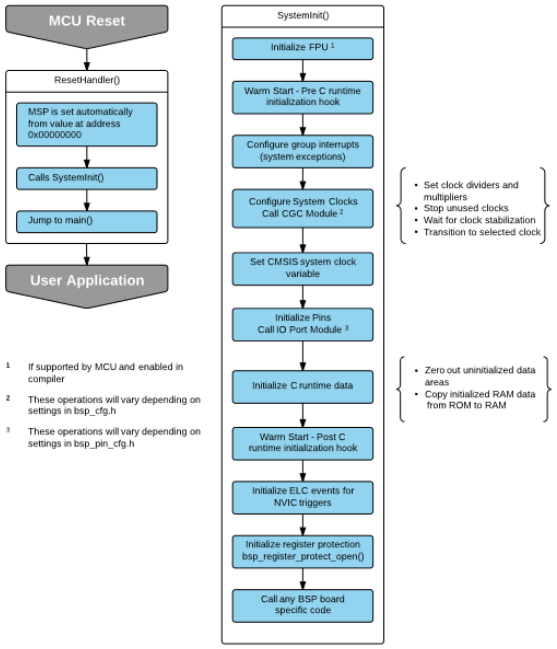

図 12: BSP のフロー

2.2.3 BSP の前提条件

2.2.3.1 サポートされるデバイスとボード

BSP は、S7G2、S124、S3A7 でテストされ、現在、DK-S7G2 および PE-HMI1、DK-S3A7、SK-S7G2、DK-S124 ボードをサポートしています。

2.2.4 BSP のディレクトリ構造

BSP は、MCU 情報、ボード固有情報、CMSIS 情報が格納されるフォルダに編成されます。

Synergy は CMSIS に準拠しており、CMSIS-Core に基づいています。そのためには、CMSIS の要件と命名規則に従う必要があります。

- プロセッサ周辺機器に対する標準化された定義
  - NVIC（ネスト型ベクター割り込みコントローラ）
  - Systick（システムティックタイマ）
  - MPU（メモリ保護ユニット）
- プロセッサ機能にアクセスするための標準化されたアクセス関数
  - NVIC_SetPriority()

  - NVIC_EnableIRQ
- システム例外ハンドラーのための標準化された関数名
  - Reset_Handler()
  - SysTick_Handler()
- システム初期化のための標準化された関数。
  - SystemInit()–system_S7G2.c で S7G2 用に定義
- クロック速度情報のための標準化されたソフトウェア変数
  - SystemCoreClock

BSP のディレクトリ構造を以下に示します。

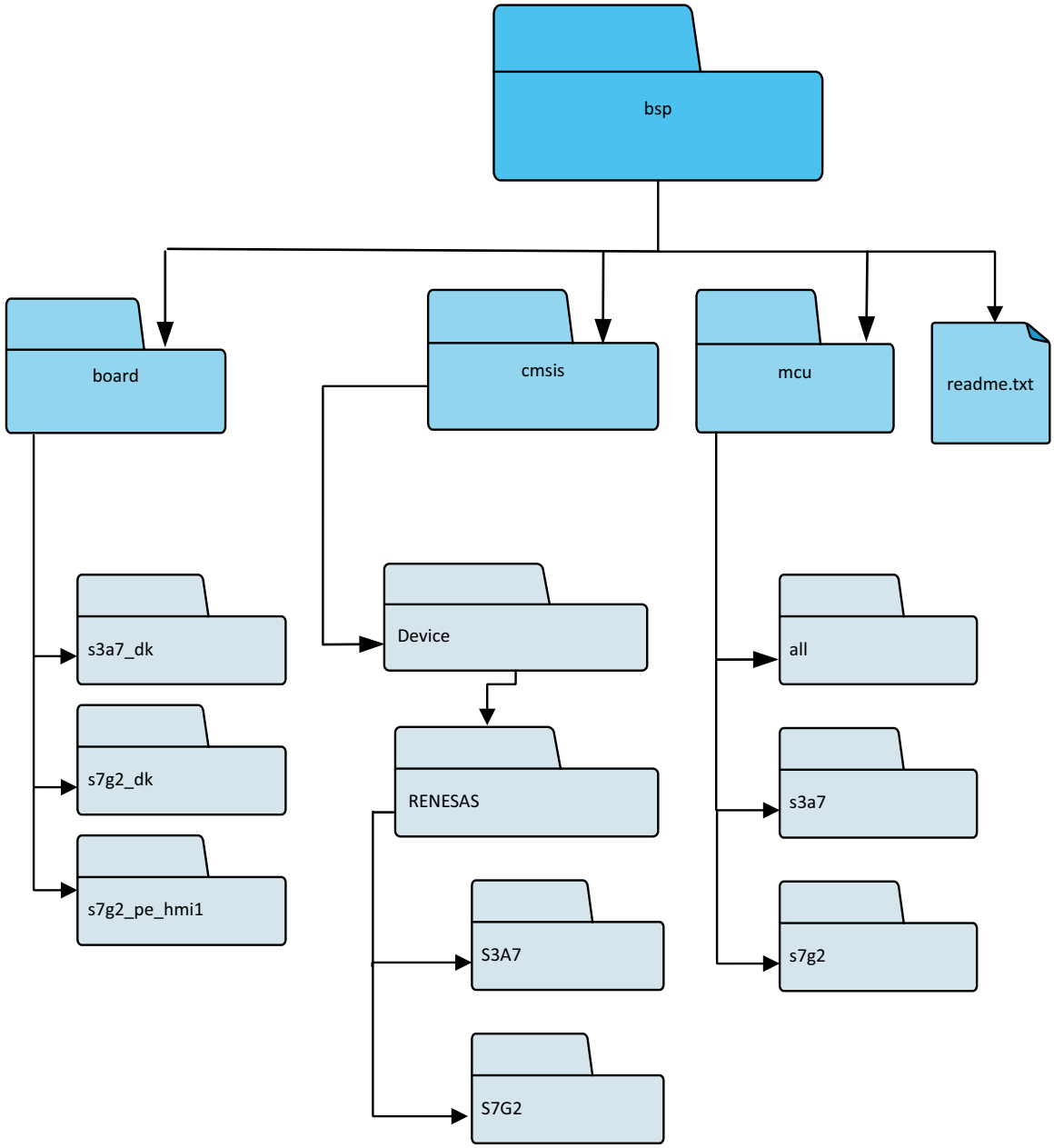

図 13: BSP のフォルダ構造

2.2.5 BSP の設定

BSP はデータ駆動型であり、ほとんどの機能は、設定ファイルの内容に基づいて設定されます。設定ファイルは、ユーザーによって指定された設定を表しており、[Generate Project Content] ボタンをクリックしたときに、e^2 studio ISDE によって生成されます。

2.2.6 BSP 設定の詳細

次の表では、BSP のそれぞれの変更可能な設定について説明します。これらの設定の多くは MCU 固有であり、サポートされるそれぞれの MCU で使用可能な設定に違いがあります。

表 : BSP 設定オプション

BSP プロパティ	説明
Part Package	MCU パッケージの種類とサイズ
Part memory size	使用可能なメモリ設定（ROM/RAM/ データフラッシュ）
Core and Frequency	ARM コアの種類と最大システム周波数
Part series	MCU パーツシリーズ
Main stack size (bytes)	メインスタックのサイズ。>0 よりも大きい必要があります。
Process stack size (bytes)	プロセススタックのサイズ。このスタックの使用はオプションです。0 の場合、PSP の使用が無効になります。
Heap size (bytes)	ピープのサイズ（バイト単位）。0 の場合、ヒープが無効になります。
OFS0 register settings IWDT Start Mode IWDT Timeout Period IWDT Dedicated Clock Frequency Divisor IWDT Window End Position IWDT Window Start Position IWDT Reset Interrupt Request Select IWDT Stop Control WDT Start Mode Select WDT Timeout Period WDT Clock Frequency Division ratio WDT Window End Position WDT Window Start Position WDT Reset Interrupt Request WDT Stop Control	オプション設定メモリは、リセット後の MCU の状態を決定します。これは、設定エリアとフラッシュメモリのプログラムフラッシュエリアに割り当てられます。詳細は MCU のユーザーマニュアルを参照してください。
OFS1 register settings Voltage Detection 0 Circuit Start Voltage Detection 0 Level HOCO OScillation Enable	詳細は MCU のユーザーマニュアルを参照してください。
MPU - PC 領域 0 を有効または無効にします	アクセスウィンドウ保護のための開始ブロックアドレス

BSP プロパティ	説明
MPU - PC0 の開始、MPU - PC0 の終了、MPU - PC 領域 1 の有効化または無効化、MPU - PC1 の開始、MPU - PC1 の終了、MPU - メモリ領域 0 の有効化または無効化、MPU - メモリ領域 0 の開始、MPU - メモリ領域 0 の終了、MPU - メモリ領域 1 の有効化または無効化、MPU - メモリ領域 1 の開始、MPU - メモリ領域 1 の終了、MPU - メモリ領域 2 の有効化または無効化、MPU - メモリ領域 2 の開始、MPU - メモリ領域 2 の終了、MPU - メモリ領域 3 の有効化または無効化、MPU - メモリ領域 3 の開始、MPU - メモリ領域 3 の終了	セキュア MPU ROM レジスタ設定。詳細はユーザーマニュアルを参照してください。
ID code 1 ID code 2 ID code 3 ID code 4	ブートモードとデバッガーアクセス保護のための ID コードを設定します。
MCU Vcc（mV）	一部のモジュール（LVD など）では、MCU に供給される電圧を知る必要があります。この情報はここから取得されます。
Parameter checking	パラメータチェックのグローバル設定を有効にするか無効にするかを定義します。ローカルモジュールはこの値をデフォルトで取得しますが、ローカルにオーバーライドすることができます。
RTOS being used	この BSP とともに RTOS を使用するかどうかを定義します。
Assert Failures	アサーション失敗が発生したときに行われる処理を定義します。
Error Log	エラーを ssp_error_log に記録するかどうかを定義します。

2.2.7 BSP の設定ファイル

設定ファイルは、BSP により、ROM レジスタ、クロック、割り込み、ELC イベント、および初期ピンを設定するために使用されます。これらの設定ファイルは、ssp_cfg\bsp にあります。

2.2.7.1 bsp_cfg.h

この設定ファイルは、BSP システム設定の値を表します。これは、e^2 studio ISDE の BSP プロパティタブで変更可能な設定です。これには、ROM レジスタ設定、スタックサイズ、パラメータチェック、およびエラーロギングの制御が含まれています。

[BSP] タブと [Properties] ビューを下に示します。

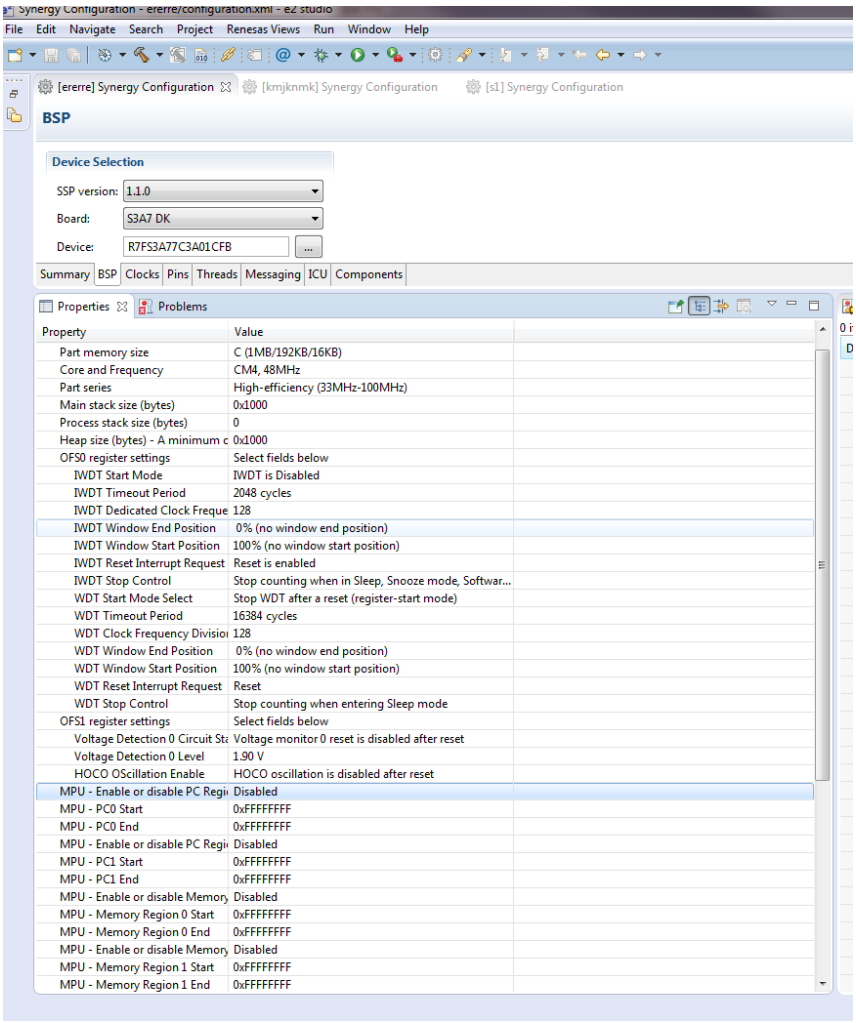

図 14: BSP プロパティビュー

一部のレジスタは ROM 内にあるため、コンパイル時に設定する必要があります。これには、いくつかのオプション設定メモリ（OFS）レジスタと、特定のメモリ保護レジスタが含まれます。

オプション設定メモリは、リセット後の MCU の状態を決定します。たとえば、IWDT の設定と有効化、電圧検出の有効化、HOCO 発振の有効化を行うことができます。これらのレジスタが設定されている場合、操作は、MCU のリセットベクトルがフェッチされ、例外が開始される前に完了します。

一部の MCU にはメモリ保護ユニット（MPU）が搭載されています。MPU は、プログラム可能なデバイスであり、各種メモリ領域について、メモリアクセス権限（たとえば、特権アクセス専用またはフルアクセス）と、メモリ属性（バッファリング可能、キャッシュ可能など）を定義するために使用できます。MPU は、最大で 8 個のプログラム可能メモリ領域をサポートでき、それぞれが独自のプログラム可能開始アドレス、サイズ、設定を持ちます。

e2 studio ISDE は、指定された MPU 設定の値を設定することで、これらのメモリ領域を設定します。これらのレジスタは慎重に設定する必要があります。設定が正しくないと、必要なメモリ領域にアクセスできなくなったり、MCU 全体にアクセスできなくなる可能性があります。

MCU には、MCU のメモリが読み取られるのを保護するための、以下の 2 つのメカニズムが備わっています。

1) ID コードの使用。ID コードは 16 バイトの値であり、MCU をデバッガーに接続したり、シリアルブートモードで接続するのを防ぐことができます。ID コードに設定できるさまざまな設定があります。使用可能なオプションについては、使用しているデバイスのユーザーズマニュアルを参照してください。
2) ROM コード保護と呼ばれる 4 バイト値の使用。この値は、MCU に対して同時に複数のプログラマーが実行できる読み取りおよび書き込みアクセスの内容を決定します。

2.2.8 BSP のピン設定

アプリケーションで使用するピンは、e2 studio ISDE のピンコンフィギュレータで設定できます。ピンの設定を参照してください。

2.2.8.1 bsp_pin_cfg.h

この設定ファイルには、ピン設定の配列が保存されます。起動時に main() が実行される前に、BSP はこの配列を反復処理し、配列内の設定に基づいて MCU のポートピンを初期化します。ユーザーがピン設定を行う前の初期状態では、e2 studio ISDE の [Pins] タブに、選択したボードの種類に対して定義されている初期基準設定が表示されています（ピンの設定を参照）。ユーザーがピン設定を変更し、[Generate Project Content] をクリックすると、新しい bsp_pin_cfg.h ファイルが生成され、新しいピン設定が格納されます。BSP は、常に ssp_cfg\bsp の bsp_pin_cfg.h ファイルをそのピン設定情報の元として使用しますが、[Generate Project Content] をクリックして生成されたピン情報は、隠しフォルダ ssp\cfg\bsp\.out 内のファイル bsp_pin_cfg.h に書き込まれます。

このようにして、ユーザーは、プロジェクト生成によってファイルが上書きされるのを心配することなく、ssp\bsp 内の bsp_pin_cfg.h を手動で編集できます。同時に、e2 studio ISDE によって生成されるピン設定情報も、参照用として、またはユーザーの設定ファイルとのマージ用として、引き続き使用できます。

2.2.9 BSP のクロック設定

すべてのシステムクロックは、BSP の初期化時に、bsp_clock_cfg.h の設定に基づいて設定されます。これらの設定は、e2 studio ISDE の [Clocks] タブ設定で指定したクロック設定情報が基になります。

- 比較的低速（たとえば 32 kHz）のクロック上で開始される可能性があることから、クロック設定は、起動処理を高速化するため、C ランタイム環境を初期化する前に実行されます。
- BSP は、選択したクロックが安定するために必要な遅延を実装します。

2.2.9.1 bsp_clock_cfg.h

この設定ファイルは、システムクロック設定の値を表します。これは、e^2 studio ISDE の [Clocks] タブで変更可能な設定です。クロックの設定を参照してください。

2.2.10 システム割り込み

MCU は、Cortex-M ARM アーキテクチャに基づいているため、ネスト型ベクトル割り込みコントローラ（NVIC）が例外と割り込み設定、優先順位付け、割り込みマスクを処理します。ARM アーキテクチャでは、NVIC が例外を処理します。一部の例外はシステム例外と呼ばれます。システム例外は、ベクタテーブルの先頭に静的に配置され、ベクトル番号 1 から 15 を占めます。ベクトル 0 は、メインスタックポインタ（MSP）用に予約されています。残りの 15 個のシステム例外を以下に示します。

- Reset
- NMI
- Cortex-M4 Hard Fault Handler
- Cortex-M4 MPU Fault Handler
- Cortex-M4 Bus Fault Handler
- Cortex-M4 Usage Fault Handler
- Reserved
- Reserved
- Reserved
- Reserved
- Cortex-M4 SVCall Handler
- Cortex-M4 Debug Monitor Handler
- Reserved
- Cortex-M4 PendSV Handler
- Cortex-M4 SysTick Handler

NMI とハード障害例外は、リセット以降に有効になっており、優先度は固定です。その他の例外の優先度は設定可能であり、一部は無効にできます。

2.2.11 グループ割り込み

グループ割り込みは、マスク不可能な割り込み（NMI）をトリガできる 12 のソースを表すために使用される用語です。NMI が発生した場合、NMI ハンドラーは NMISR（ステータスレジスタ）を調べ、割り込みのソースを判定します。NMI 割り込みはすべての割り込みより優先され、CPU 割り込みとしてのみ使用でき、データトランスファコントローラ（DTC）またはダイレクトメモリアクセスコントローラ（DMAC）のアクティブ化を行うことはできません。

グループ割り込みのソースとしては、以下のものが考えられます。

- IWDT アンダーフロー / 更新エラー
- WDT アンダーフロー / 更新エラー
- 電圧監視 1 割り込み
- 電圧監視 2 割り込み
- VBATT 監視割り込み
- 発振停止検出
- NMI ピン
- RAM パリティエラー
- RAM ECC エラー
- MPU バススレーブエラー
- MPU バスマスターエラー
- MPU スタックエラー

ユーザーは、BSP API 関数を使用してコールバックを登録することで、1 つ以上のグループ割り込みの通知を有効にすることができます（R_BSP_GroupIrqWrite）。NMI 割り込みが発生すると、NMI ハンドラーは、割り込み原因に対してコールバックが登録されているかどうかを確認し、登録されている場合はそのコールバック関数を呼び出します。

前述のように、ベクタテーブルの最初の 16 個のスロットはすでにシステム例外に割り当てられています。スロット 16 以降はユーザーが設定可能な割り込みです。外部割り込みか、周辺機器で生成される割り込みを設定できます。

NVIC 割り込みテーブルのサイズは、MCU の種類によって異なります（以下を参照）。

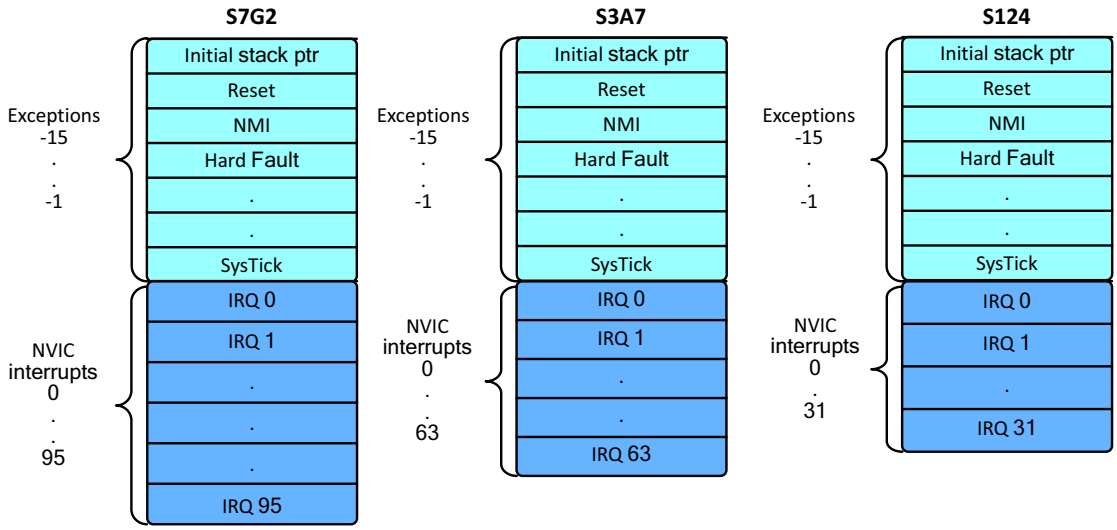

図 15: NVIC 割り込みベクタテーブル

NVIC 割り込みベクタテーブルの空きスロットの数は少なく見えるかもしれませんが、BSP では、割り込みを生成できるイベントが最大で 512 個定義されます。BSP は、イベントマッピングを使用することで、ユーザーが有効にしたイベントを NVIC 割り込みにマッピングします。S7G2 では、これらのイベントのうち 96 個だけを一度にアクティブにできますが、ユーザーは、アクティブなイベントを生成するイベントを柔軟に選択できます。

下図は S7G2 の割り込みベクタテーブルを示しています。

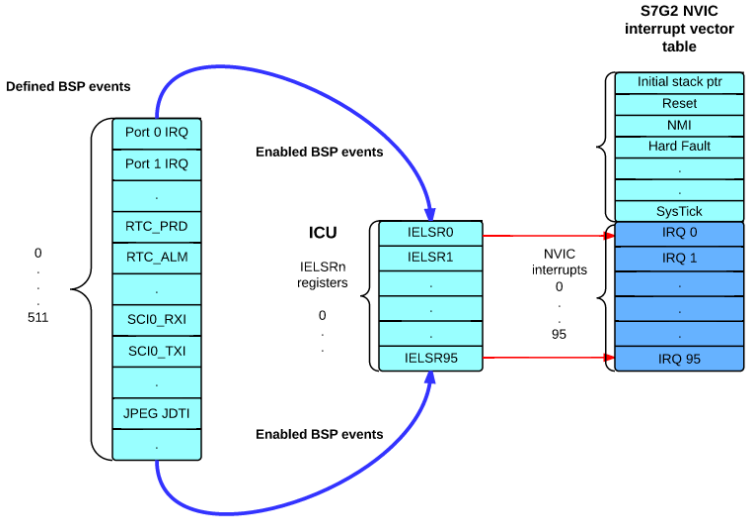

図 16: NVIC 割り込みベクタテーブル

割り込みソースとして関心があるイベントのみをユーザーが選択できるようにすることで、高速でイベント固有の割り込みサービスルーチンを提供できます。

たとえば、他のマイクロコントローラにおいて、標準の NVIC 割り込みベクタテーブルには、SCI0（シリアル通信インタフェース）周辺機器用の単一のベクトルエントリが格納される可能性があります。そのための割り込みサービスルーチンは、割り込みの「本当の」ソースのステータスレジスタを確認する必要があります。Synergy の実装では、関心がある SCI0 イベントごとに 1 つのベクトルエントリがあります。標準の NVIC テーブルと Synergy S7G2 NVIC テーブルの違いを下に示します。

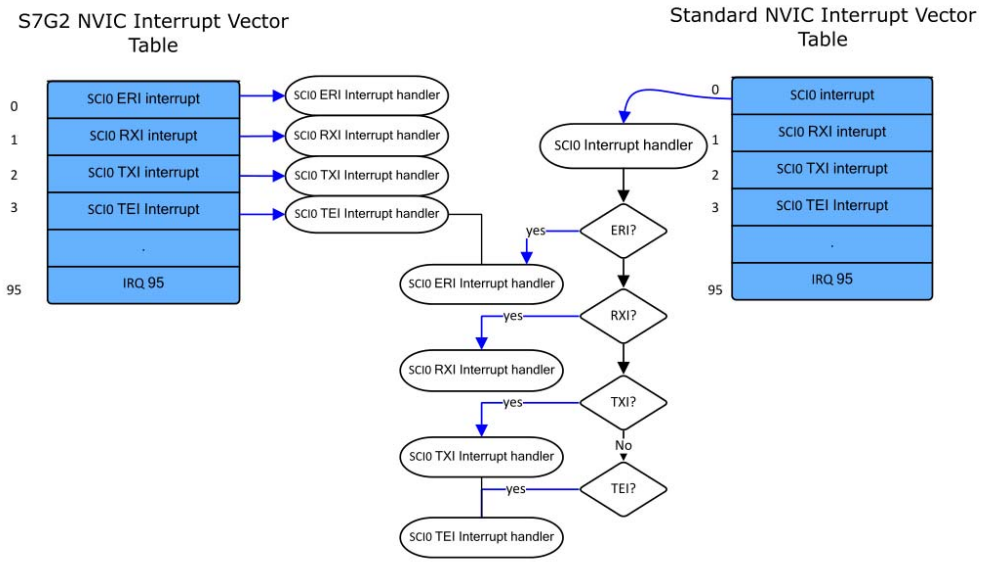

図 17: NVIC 割り込みベクタテーブルの例

割り込みの設定は、bsp_irq_cfg.h ファイルを通じて行います。e^2 studio ISDE の [ICU] タブで優先度「無効化不可」が設定されているすべてのイベントについて、BSP は対応する NVIC ベクタテーブルエントリを生成します。BSP は、可能性のあるイベントのリスト全体を反復処理し、NVIC スロットがその順序で割り当てられ、対応するイベント選択レジスタが同様に更新されます。

割り込みが発生した場合の最初の操作の 1 つは、BSP によって割り当てられた NVIC 割り込みスロットに対応する割り込み番号と共に R_BSP_IrqStatusClear を呼び出すことです。割り込み番号は、bsp_irq_cfg.h で有効にされた割り込みイベントに基づき、ビルド時に BSP によって作成されます。

R_BSP_IrqStatusClear は、特定の割り込みの割り込みステータスフラグ（IR）をクリアします。: 特定の割り込みの割り込みステータスフラグ（IR）をクリアします。割り込みがトリガされると、IR ビットがセットされます。

ここに示す例で、優先度が割り当てられたエントリ（たとえば BSP_IRQ_CFG_ICU_IRQ0）では、対応する weak ハンドラーのアドレスが、次に使用可能なベクトルスロットに格納されます。可能性のあるすべての割り込みソースがこのようにして反復処理されます。定義されている割り込みがベクタテーブルに設定されます。

2.2.11.1 bsp_irq_cfg.h

この設定ファイルには、割り込みを生成できるすべての周辺機器イベントのエントリが格納されています。e^2 studio ISDE の [ICU] タブで定義された周辺機器イベントを変更することで、周辺機器イベントに「BSP_IRQ_DISABLED」以外の値が指定されると、bsp_irg_cfg.h 中の対応するエントリが更新されます。BSP は、これらのエントリを繰り返し処理し、優先度が定義された（値が「BSP_IRQ_DISABLED」以外の）周辺機器イベントの割り込みサービスルーチンエントリが NVIC ベクタテーブルに追加されます。また、BSP は、その NVIC ベクトル番号について、各周辺機器イベントに対応するように、各 IELSRn レジスタを更新します。

2.2.11.2 Weak シンボル

BSP は、ユーザーが明示的に指定していない ISR アドレスを NVIC テーブルにどのようにして格納できるのか不思議に思うかもしれません。BSP に必要なのは、割り込みイベントに優先度を設定することだけです。

これは、「weak」属性を使用することで実現されます。weak 属性を使用すると、宣言がグローバルではなく weak シンボルとして定義されます。BSP は、実際には、512 個の可能性のある割り込みイベントそれぞれについて、ISR を CMSIS の startup_xxxx.c ファイル中で定義しています。weak シンボルは、同じ名前を持つ、付随する strong 参照によってオーバーライドされることが可能なシンボルです。BSP が関数を weak として宣言すると、ユーザーコードは同じ関数を定義できます。ユーザー定義の関数は、BSP 関数の代わりに使用されます。考えられるすべての割り込みソースを weak として定義することにより、ベクタテーブルをコンパイル時にビルドでき、ユーザー宣言（strong 参照）が実行時に使用されます。

weak シンボルは、ELF ターゲットでサポートされ、GNU アセンブラとリンカーを使用する場合は a.out ターゲットでもサポートされます。

CMSIS の system_xxxx.c では、ウォームスタートコールバック関数 R_BSP_WarmStart() の weak 定義（と関数本体）もあることに注意してください。この関数は、weak 宣言と同じファイルに定義されているため、「デフォルトの」実装として呼び出されます。本体をユーザーアプリケーションにコピーし、必要に応じて変更することにより、ユーザーが関数をオーバーライドできます。リンカーはこれを「strong」参照として認識して使用します。

2.2.11.3 ウォームスタートコールバック

BSP がボードをリセット状態から起動している最中に、ユーザーがコールバックを要求できる場所が 2 つあります。これらは、「Pre C」および「Post C」ウォームスタートコールバックとして定義されます。

前述のように、この関数は R_BSP_WarmStart() としてすでに weak 定義されているため、アプリケーションコードで関数を再定義する（または、CMSIS の system_xxxx.c から既存の本文をコピーする）だけで、コールバックを作成できます。R_BSP_Warmstart() は、実行中のウォームスタートコールバックの種類を示すイベントパラメータを受け取ります。

```
typedef enum e_bsp_warm_start_event

{

    BSP_WARM_START_PRE_C = 0, // Called almost immediately after reset.

    /* No C runtime environment, clocks, or IRQs. */

    BSP_WARM_START_POST_C // Called after clocks and C runtime environment have been set up.

} bsp_warm_start_event_t;
```

この関数は、有効 / 無効にされず、BSP のスタートアップの一部として、両方のイベントに対して必ず呼び出されます。そのため、ユーザーがオーバーライドするときには呼び出されない関数本体が必要です。関数の本体は system_xxxx にあります。この関数を使用するには、この関数をユーザーのコードにコピーし、要件を満たすように変更します。

2.2.11.4 Pre C ウォームスタートコールバック

このコールバックはリセット後ほぼすぐに呼び出されます。この時点では、C ランタイム環境、クロック、IRQ がセットアップされていません。

「Pre C」ウォームスタートコールバックにユーザーが関心を持つ理由

以下にいくつかの例を挙げます。

- スタートアップ処理の一部としてのセーフティコード（たとえば破壊的なメモリテスト）の実行。
- クラッシュダンプ調査の一環としてのグローバルメモリの検査。
- すでに実行している RTC の再初期化の防止。

2.2.11.5 Post C ウォームスタートコールバック

このコールバックは、クロックと C ランタイム環境がセットアップされた後で呼び出されます。

「Post C」ウォームスタートコールバックにユーザーが関心を持つ理由

以下にいくつかの例を挙げます。

- クロックがセットアップされている必要のあるテストの実行。
- ADC の診断。
- ROM/ 外部メモリシステムのチェック。

2.2.12 ボードサポート（BSP）

次の Synergy ボードがサポートされています。

- DK-S7G2
- PE-HMI1
- DK-S3A7
- DK-S124
- SK-S7G2

2.2.13 カスタムボードサポート（カスタム BSP）

いずれかの MCU に基づいて独自のボードを開発している場合はどうしますか。

e^2 studio ISDE 内から使用可能な外部コマンドラインユーティリティで、カスタム BSP の作成に使用できる Custom BSP Creator ツールがあります。このプロセスの詳細は、アプリケーションノート、R01AN3044EU0101 に説明されています。

2.2.14 BSP API 関数

BSP にはパブリック関数があり、BSP を使用するすべてのプロジェクトで使用できます。この関数を使用すると、BSP でサポートされる MCU とボード全体で共通する機能にアクセスできます。

- R_BSP_SoftwareLockInit: BSP は、アトミックロックを実装するための API 関数を提供しています。: BSP は、アトミックロックを実装するための API 関数を提供しています。これらのロックを使用すると、通常は RTOS のセマフォやミューテックスで行われるように、クリティカルなコード領域を保護することができます。
- R_BSP_SoftwareLock: 渡されたロックの取得を試みます。排他的ロード命令と排他的ストア命令は、排他的読み取り / 変更 / 書き込みを、入力ロックに対して実行するために使用されます。: 渡されたロックの取得を試みます。
  - 排他的ロード命令と排他的ストア命令は、排他的読み取り / 変更 / 書き込みを、入力ロックに対して実行するために使用されます。
  - この処理は以下のようになります。排他的ロード（LDREXB）を使用してロックの値を読み取ります。
  - ロックを使用できる場合は、ロック値を変更して確保します。
  - 返されたステータスビットをテストし、書き込みが実行されたかどうかを判断します。
- R_BSP_SoftwareUnlock: 既存のソフトウェアロックを解放します。
- R_BSP_HardwareLock: ハードウェアロックはソフトウェアロックに似ています。: 既存のソフトウェアロックを解放します。: ハードウェアロックはソフトウェアロックに似ています。実際に、BSP のソフトウェアロック関数は、ハードウェアロック関数によって呼び出されます。ハードウェアロックは特定の周辺機器に固有であり、使用可能なハードウェアロックのリストは bsp_hw_locks.h で定義されています。
- R_BSP_HardwareUnlock: 既存のハードウェアロックを解放します。たとえば、フラッシュ API の open() 関数が呼び出された場合、フラッシュハードウェアのロックを取得して、フラッシュ API の close() が呼び出されるまでロックを保持します。
- R_BSP_GroupIrqWrite: サポートされるいずれかのグループ割り込みに対し、コールバック関数を登録します。: サポートされるいずれかのグループ割り込みに対し、コールバック関数を登録します。前述のように、割り込みは 12 個あり、すべて NMI 例外にマッピングされています。NMI が発生した場合、NMI_Handler が NMISR（ステータスレジスタ）を参照し、割り込みのソースを決定します。コールバック引数に NULL が渡された場合は、過去に登録されたコールバックが登録解除されます。
- R_BSP_IrqStatusClear: 特定の割り込みの割り込みステータスフラグ（IR）をクリアします。: 特定の割り込みの割り込みステータスフラグ（IR）をクリアします。割り込みがトリガされると、IR ビットがセットされます。
- R_BSP_SoftwareDelay: ブロッキングソフトウェア遅延を実装します。: ブロッキングソフトウェア遅延を実装します。遅延は、システムクロックレートに基づいて実装されます。
- R_BSP_VersionGet: BSP のバージョンを返します。
- R_BSP_RegisterProtectEnable: レジスタ保護を有効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ（PRCR）と MPC の書き込み保護レジスタ（PWPR）を

使用することで可能になります。保護が可能なレジスタは、3 つのグループのいずれかにグループ化されます。

– BSP_REG_PROTECT_CGC- クロック発生回路に関連するレジスタ。
– BSP_REG_PROTECT_OM_LPC_BATT- 動作モード、低電力消費、バッテリーバックアップ機能に関連するレジスタ。
– BSP_REG_PROTECT_LVD–LVD（低電圧検出）に関連するレジスタ。

BSP レジスタ保護機能は、参照カウンタを利用して、特定のレジスタを指定した後で別の関数を呼び出すアプリケーションで、そのレジスタ保護設定が誤って変更されないようにします。

RegisterProtectDisable() が呼び出されるたびに、それぞれの参照カウンタがインクリメントされます。

RegisterProtectEnable() が呼び出されるたびに、それぞれの参照カウンタがデクリメントされます。

どちらの関数も、参照カウンタがゼロの場合は、保護状態のみを変更します。

以下の例に示すように、参照カウンタがないと、MODULE1 が保護解除したレジスターを MODULE2 が再度保護し、MODULE1 が書き込めなくなります。

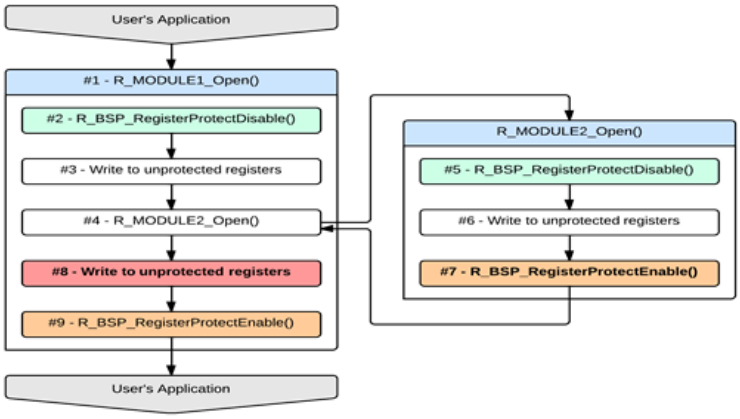

図 18: レジスタ保護

- R_BSP_RegisterProtectDisable: レジスタ保護を無効にします。保護されていないレジスタに書き込むことができます。レジスタ保護は、保護レジスタ（PRCR）と MPC の書き込み保護レジスタ（PWPR）を使用して無効にします。ここでも上記のレジスタのグループ化が適用されます。

第 3 章開発の開始

SSP を使用して開発を開始するには、e^2 studio ISDE をダウンロードして、対象となる Synergy 開発環境と評価ボードを入手し、この章にあるチュートリアルに従って実行します。e^2 studio ISDE ユーザーガイドとチュートリアルには、簡単なアプリケーションから始められる詳細なインストラクションが含まれています。SSP を始めるには、これらのページを参照してください :

- e^2 studio ISDE ユーザーガイド
- チュートリアル : Your First Synergy Project - Blinky
- チュートリアル : Using HAL Drivers - Programming the WDT

参考資料

3.1 e^2 studio ISDE ユーザーガイド

ここでは、e^2 studio ISDE（統合ソリューション開発環境）を使用して、SSP を使用したアプリケーションを作成する方法について説明します。SSP のアーキテクチャは、e^2 studio ISDE をどのように使用して Synergy アプリケーションを開発するかを直接決定します。このマニュアルに含まれる SSP アーキテクチャの詳細については、次のドキュメントを参照してください。

- SSP Architecture
- BSP Architecture

e^2 studio ISDE で生成および構築された簡単なサンプルプロジェクトについては、次を参照してください :

- チュートリアル : Using HAL Drivers - Programming the WDT
- チュートリアル : Your First Synergy Project - Blinky

このマニュアルのすべてのユーザーガイドでは、e^2 studio ISDE を使用してドライバを構成し、アプリケーションを開発する方法について説明しています。以下を参照してください。

- HAL レイヤー : HAL レイヤーユーザーガイド
- フレームワークレイヤー : フレームワークレイヤーユーザーガイド

3.1.1 e^2 studio ISDE の概要

e^2 studio ISDE（統合ソリューション開発環境）は、コードの開発、ビルド、デバッグを包含した、開発ツールです。e^2 studio ISDE はオープンソース Eclipse IDE および関連する C/C++ Development Tooling（CDT）に基づいています。特に MCU では、Synergy プロジェクト用に、SSP 用にコードを設定および自動生成するための、いくつかのグラフィカルユーザーインタフェース（GUI）ウィザードが e^2 studio ISDE に備わっています。e^2 studio ISDE にはスマートマニュアルも取り込まれており、ドライバとデバイスのドキュメントが、コード中でツールヒントの形で利用できます。

図 19: e² studio ISDE の開始画面

e² studio ISDE と Synergy プロジェクトコンフィギュレータは、特定のアプリケーションに必要な SSP モジュールをできるだけ簡単かつ迅速に選択し、プロジェクトに追加して設定できるように開発されています。e² studio ISDE は、MCU アプリケーション用に SSP のすべての要素を設定するグラフィカルユーザーインタフェースを提供しています。e² studio ISDE は、HAL モジュールとフレームワークモジュールに加えて、RTOS スレッド、セマフォ、ミューテックス、イベントフラグ、キューを追加および設定できます。これにより、RTOS のサポートをアプリケーションに非常に簡単に追加できます。プロジェクトが生成されたら、必要に応じて任意のモジュールと設定値を再設定できます。e² studio ISDE は構成ビューの選択から完全かつ正確な構成コードを生成するため、アプリケーションコードの記述に専念することができます。

SSP の要素は、e² studio ISDE の [Project Explorer] ビューに表示されます。SSP の構成構造とパラメータはすべて XML ファイルにマッピングされます。e² studio ISDE は、XML ファイルを使用することで、選択肢の視覚的リストをユーザーに提示できます。モジュールを設定するためのコードを生成するのに加えて、XML はモジュールの依存関係情報も提供します。

プロジェクトに SSP モジュールを追加すると、e² studio ISDE はそのモジュールの依存関係を確認し、すべての必要なドライバとフレームワークモジュールを追加して適切なスタックを作成します。ユーザーによる選択が必要な依存関係がある場合は、[Stack] ウィンドウでそのモジュールがハイライトされ、選択可能なオプションが e² studio ISDE により表示されます。

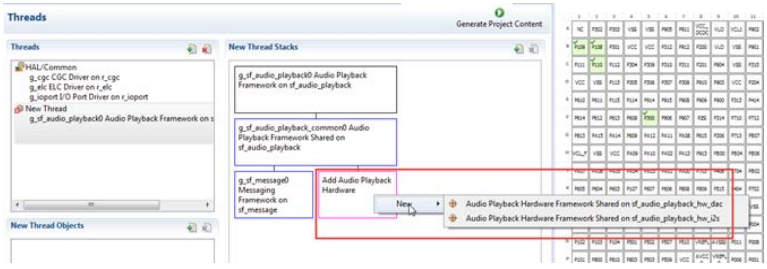

図 20: e² studio ISDE の依存関係チェック

エラーは、[HAL/Common Modules] または新しいスレッドモジュールペインのドライバ名の横にフラグされます。また、[Problems] ウィンドウでエラーを確認できます。

3.1.2 e² studio ISDE の前提条件

3.1.2.1 Synergy キットの準備

SSP でアプリケーションを開発するには、いずれかの Renesas Synergy キットの使用が便利です。Renesas Synergy キットは e² studio ISDE 上で動作するように設計されています。いくつかの種類のキットが提供されています。

- 開発キット（DK）
- スターターキット（SK）
- 製品事例（PE）

すべての Synergy キットは弊社のウェブサイトから入手できます。

3.1.2.2 PC 要件

e² studio ISDE を使用するには、ご使用のパソコンが次の最小要件を満たしていることを確認してください。

- Windows 7、Intel i5 または i7、あるいは AMD A10-7850K または FX
- メモリ : 8 GB DDR3 または DDR4 DRAM（16 GB DDR4/2400 MHz RAM を推奨）
- 最低 250 GB のハードディスク（500 GB 以上を推奨）
  （快適に使用する要件です。Windows7（32 ビット）， 4GB RAM， Intel i5 での動作実績があります）

3.1.2.3 e² studio ISDE および SSP のインストール

e² studio ISDE および SSP の詳細なインストール手順とインストーラーは、Renesas Synergy Gallery ウェブサイト（https://synergygallery.renesas.com）にあります。e² studio ISDE のリリースノートをレビューして、e² studio ISDE のバージョンが選択された SSP バージョンをサポートしていることを確認します。

関連項目 :SSP リリースの処理

3.1.2.4 ライセンス設定

デフォルトで、SSP ダウンロードには SSP の評価ライセンスが含まれます。Synergy Project の構成中にライセンスファイルを指定するようプロンプトされた場合は、インストール手順のライセンスに関する説明を参照してください。評価ライセンスファイルは以下のディレクトリにあります : <e2_studio_base_dir >/internal/projectgen/arm/Licenses/

評価ライセンスで、完全に機能する SSP を使用することができます。これは、SSP のすべてのモジュールをコンパイルして、アプリケーションにリンクできることを意味します。ただし、フレームワークレイヤー

のモジュールのソースコードは暗号化されており、変更できません（Express Logic X-Ware コンポーネントを含む）。この場合でも、選択したモジュールのソースコードを表示させることができます（Express Logic X-Ware コ ンポーネントを除く）。このタイプの暗号化コードは保護コードと呼ばれます。詳細は、ライセンスファイルを参照してください。

HAL レイヤーのモジュールのソースコードは保護されておらず、Synergy\ssp\src\driver\<module>\<module>.c ファイルをクリックして e^2 studio ISDE から表示できます。

他のライセンスタイプでは、デバッグフェーズのセキュアデバッグビューから e^2 studio ISDE で保護コードを表示できるか、選択したソースに対して完全な読み取り / 書き込みアクセスを提供します。

下の表では、利用可能なライセンスと機能を示しています。

ライセンス項目		評価ライセンス	開発・量産ライセンス	ソースライセンス
開発・デバッグ期間中のソースコードへのアクセス、閲覧	BSP/HAL ライバ	Yes	Yes	Yes
	アプリケーションフレームワーク		Yes	Yes
	ThreadX® RTOS、ライブラリ、X-Ware™ をはじめとしたスタック類		Yes	Yes
ソースコードのクリアテキストファイル（保存・印刷・改変可能。ただし、改変時は動作は保証されません）	BSP/HAL ライバ	Yes	Yes	Yes
	アプリケーションフレームワーク			Yes
	ThreadX® RTOS、ライブラリ、X-Ware™ をはじめとしたスタック類			Yes

3.1.3 プロジェクトとは

e^2 studio ISDE では、すべての SSP アプリケーションは Synergy プロジェクトに編成されます。Synergy プロジェクトの設定には以下の部分があります：

1) プロジェクトの作成
2) プロジェクトの設定

e^2 studio ISDE には、Synergy プロジェクト専用の多数のプロジェクトウィザードおよび構成ウィンドウがあります。Synergy Project Generator を使って新しい Synergy プロジェクトを作成することも、Synergy Project Editor を使って既存のプロジェクトの構成を編集することもできます。

e^2 studio ISDE を起動してワークスペースを選択すると、選択したワークスペースで以前に保存したすべてのプロジェクトが読み込まれ、[Project Explorer] ビューに表示されます。各プロジェクトには、configuration.xml という関連する構成ファイルがあり、プロジェクトのルートディレクトリに置かれています。

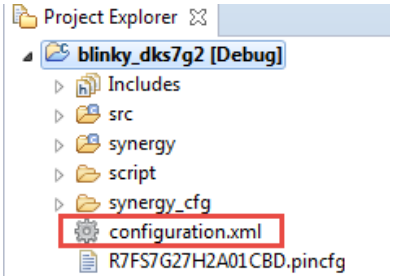

図 21: e^2 studio ISDE のプロジェクト構成ファイル

configuration.xml ファイルをダブルクリックすると Synergy Project Editor が開くので、このプロジェクトに関連するすべての構成設定を表示または変更できます。プロジェクト構成を編集するには、[Synergy Configuration] パースペクティブが e^2 studio ISDE ウィンドウの右上で選択されていることを確認してください。

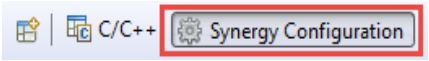

図 22: e^2 studio ISDE Synergy Configuration パースペクティブ

Synergy Project Editor にはいくつかのタブがあります。個別のタブの構成ステップとオプションについては、続くセクションで取り上げます。

Note: Synergy Project Editor で使用可能なタブは、e^2 studio ISDE のバージョンによって異なります。e^2 studio ISDE バージョン 5.0 では、[Messaging] 設定用のタブが新たにサポートされています。

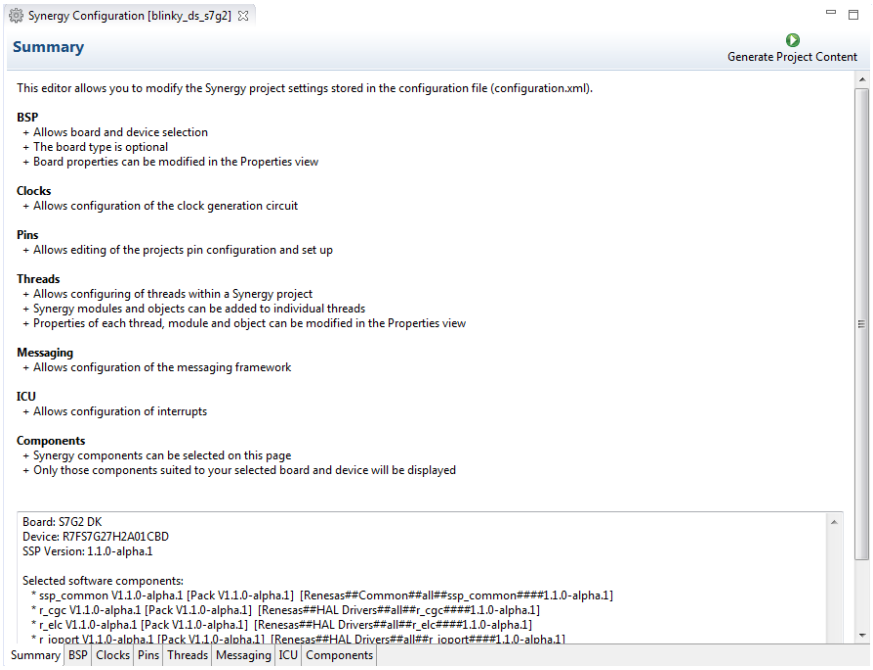

図 23: e² studio ISDE Project Editor

3.1.4 プロジェクトの作成

このセクションでは、Synergy プロジェクトを作成する詳細な手順について説明します。プロジェクトを作成すると、クロック、ピン、割り込みおよびアプリケーションの一部であるすべてのモジュールのパラメータを簡単に構成できます。

Synergy Project Generator で新しい Synergy プロジェクトを作成するには、プロジェクト名を選択し、アプリケーション用に MCU を選択し、ライセンスを確認または追加し、ツールチェーンを選択し、プロジェクトテンプレートを選択することによって事前設定されたクロック、ピン、および MCU 関連の設定を選べます。

3.1.4.1 新規プロジェクトの作成

Synergy アプリケーションでは、必ず次の方法で新しいプロジェクトを Synergy プロジェクトとして生成します。

1) [File] > [New] > [Synergy Project] の順にクリックします。

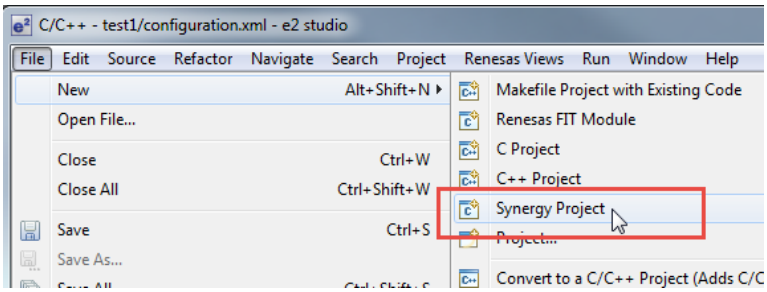

図 24: [New Synergy Project]

2) プロジェクト名と場所を選択します。

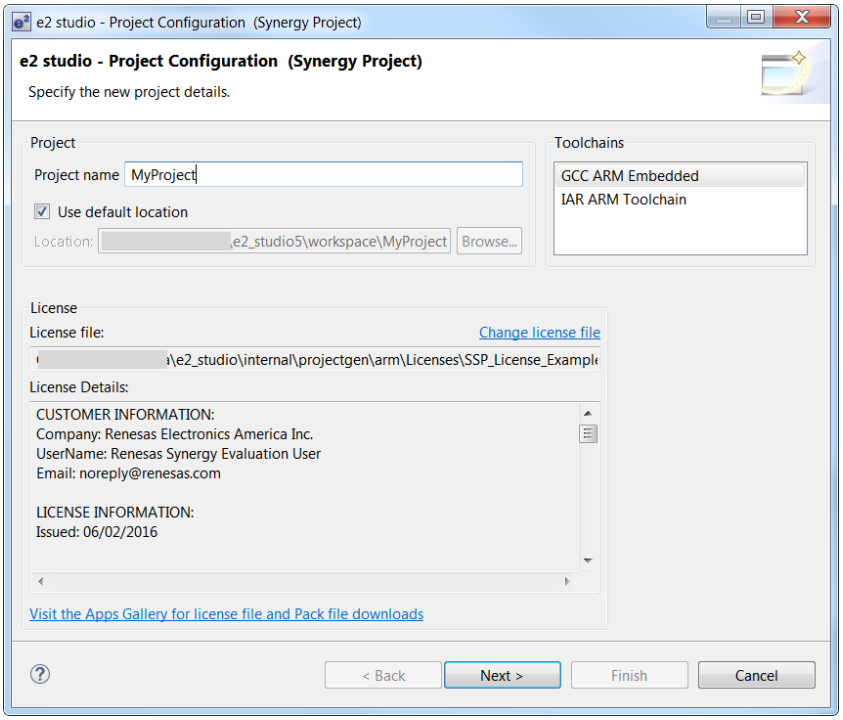

図 25: Synergy Project Generator（画面 1）

3) 最新の SSP ダウンロードのライセンスファイルがあることを確認します。SSP ダウンロードパックには評価ライセンスが含まれていますが、開発・量産ライセンスやソースライセンスなども利用可能です。ライセンスファイルの読み込み方法については、Synergy ウェブサイトにある

『**Installation_Guide**』を参照してください。e^2 studio ISDE および SSP を初めてインストールした場合、ライセンスペインは空です。それ以外の場合、ライセンスペインにはインストールした最新のライセンスが表示されます。複数のライセンスがある場合は、いずれかを選択できます。評価ライセンスファイルは以下のディレクトリにあります :<e2_studio_base_dir>/internal/projectgen/arm/Licenses/ また、ライセンス設定も参照してください。

4) [Next] をクリックします。

3.1.4.2 ボードとツールチェーンの選択

次の [Project Configuration] ウィンドウで、ハードウェアとソフトウェアの環境を選択します。

1) SSP のバージョンを選択します。
2) アプリケーションのボードを選択します。既存の Synergy キットを選択するか、独自の BSP 定義を持つ任意の MCU のカスタムユーザーボードを選択することができます。

   Note: 独自のアプリケーションを開発する場合、DK-S7G2 BSP などの基本テンプレートを選択します。プロジェクトのモジュールを構成する際にいつでも RTOS サポートを追加できます。
3) ツールチェーンのバージョンを選択します。
4) デバッガを選択します。J-Link ARM デバッガーがあらかじめ選択されています。
5) [Next] をクリックします。

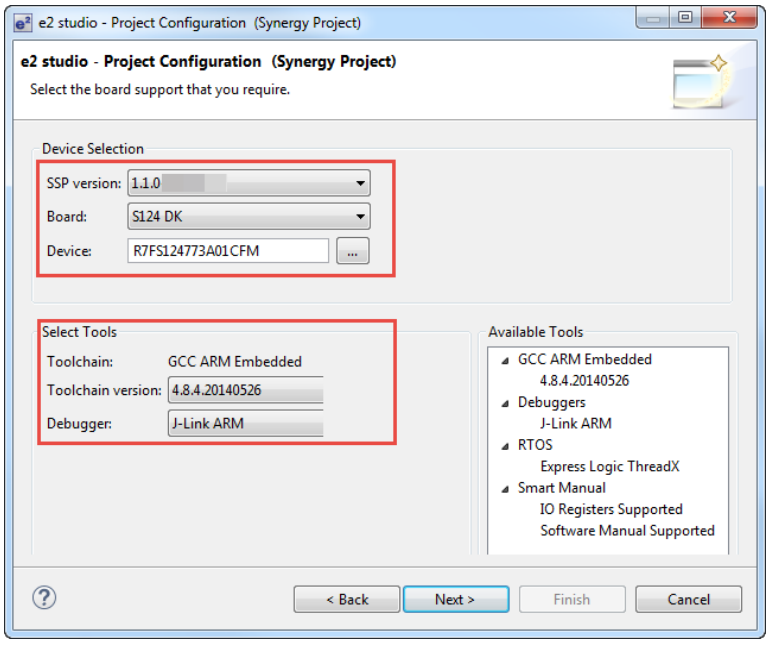

図 26: Synergy Project Generator（画面 2）

3.1.4.3 プロジェクトテンプレートの選択

続くウィンドウで、利用可能なテンプレートのリストからプロジェクトテンプレートを選択して、[Finish]をクリックします。

Note: 独自のアプリケーションを開発する場合、DK-S7G2 BSP などの基本テンプレートを選択します。プロジェクトのモジュールを構成する際にいつでも RTOS サポートを追加できます。

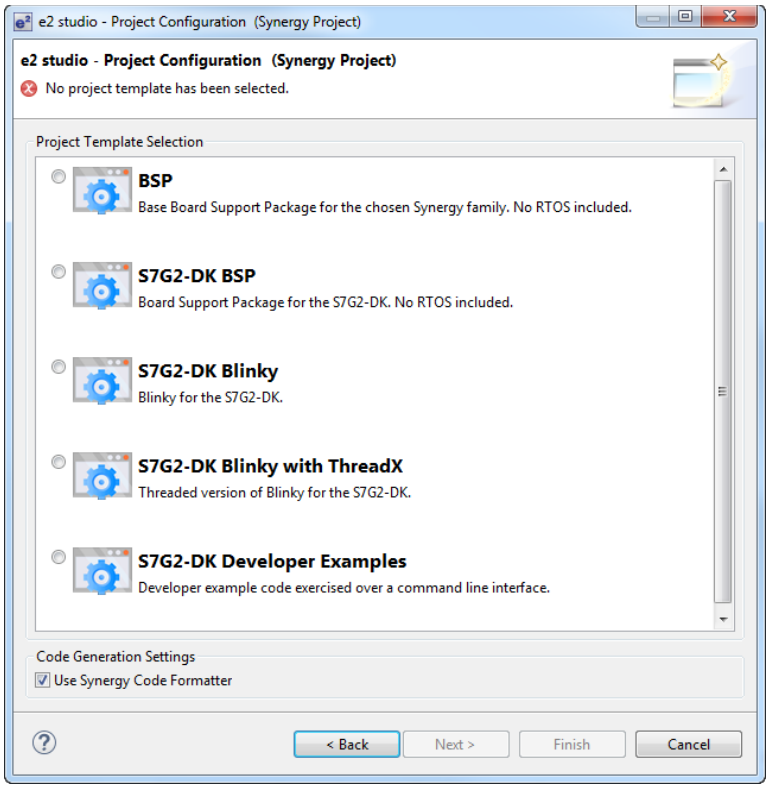

図 27: Synergy Project Generator（画面 3）

デフォルトでは、この画面には現在の SSP パックに含まれるテンプレートが表示されます。

プロジェクトが作成されると、e^2 studio ISDE は Synergy Project Editor で現在のプロジェクト構成の概要を表示します。

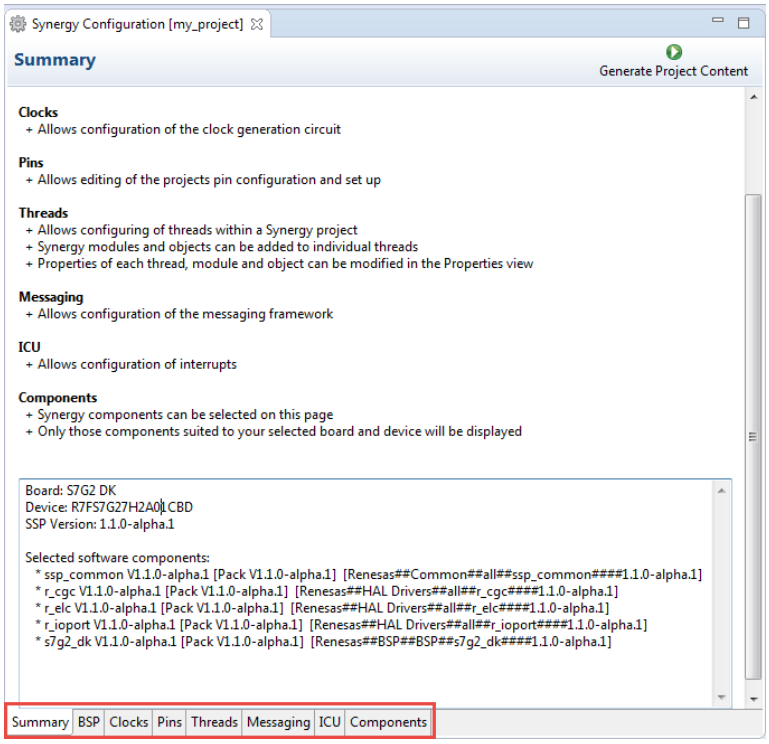

図 28: Synergy Project Editor とエディタータブ

Synergy Project Editor に、プロジェクトを構成できるタブがあります。

- [BSP] タブでは、初期プロジェクト選択からボード別のパラメータを変更できます。
- [Clocks] タブでは、プロジェクトの MCU クロック設定を構成できます。
- [Pins] タブでは、各ポートピンの電気特性と機能を構成できます。
- [Threads] タブでは、RTOS および RTOS 以外のアプリケーションの SSP モジュールおよびドライバを追加して、ドライバを構成できます。このタブで選択したモジュールまたはドライバごとに、プロパティウィンドウでは構成パラメータ、割り込み優先度、ピンの選択へのアクセスを提供します。
- [Messaging] タブでは、ThreadX ベースのプロジェクトのメッセージングフレームワークを設定できます。[Messaging] タブは、e^2 studio ISDE バージョン 5.0 以降に含まれています。
- [ICU] タブでは、割り込みを有効化し、割り込みの優先度を割り当てられます。
- [Components] タブでは、選択したモジュールの概要を提供しています。ここでは、特定の SSP リリースのドライバおよびアプリケーションサンプルコードを追加することもできます。

3.1.5 プロジェクトの設定

プロジェクトには 2 つのレベルの構成があります。

- [BSP]、[Clocks]、および [Pins] のタブは、リセット後にユーザーコードが実行される前の MCU の初期構成を決定します。プロジェクトの作成時にプロジェクトテンプレートを選択することにより、e^2 studio ISDE は選択したボードに適したデフォルト値を設定します。それらのデフォルト値は必要に応じて変更できます。
- スレッドでは、プロジェクトに SSP モジュールを追加でき、アプリケーションの必要に応じてモジュールの構成パラメータを設定できます。メッセージングフレームワークは ThreadX ベースのアプリケーションに不可欠な部分であるため、[Messaging] タブではメッセージングが必要な各スレッドのメッセージングフレームワークを設定できます（e^2 studio ISDE バージョン 5.0 以降の場合）。

3.1.5.1 e^2 studio ISDE での BSP の設定

[BSP] タブには、現在選択されているボード（選択されている場合）とデバイスが表示されます。[Properties] ビューは、下の [Project Configuration] ビューで左下に表示されています。

Note: [Properties] ビューが表示されていない場合、トップメニューバーで [Window] > [Show View] > [Properties] をクリックします。

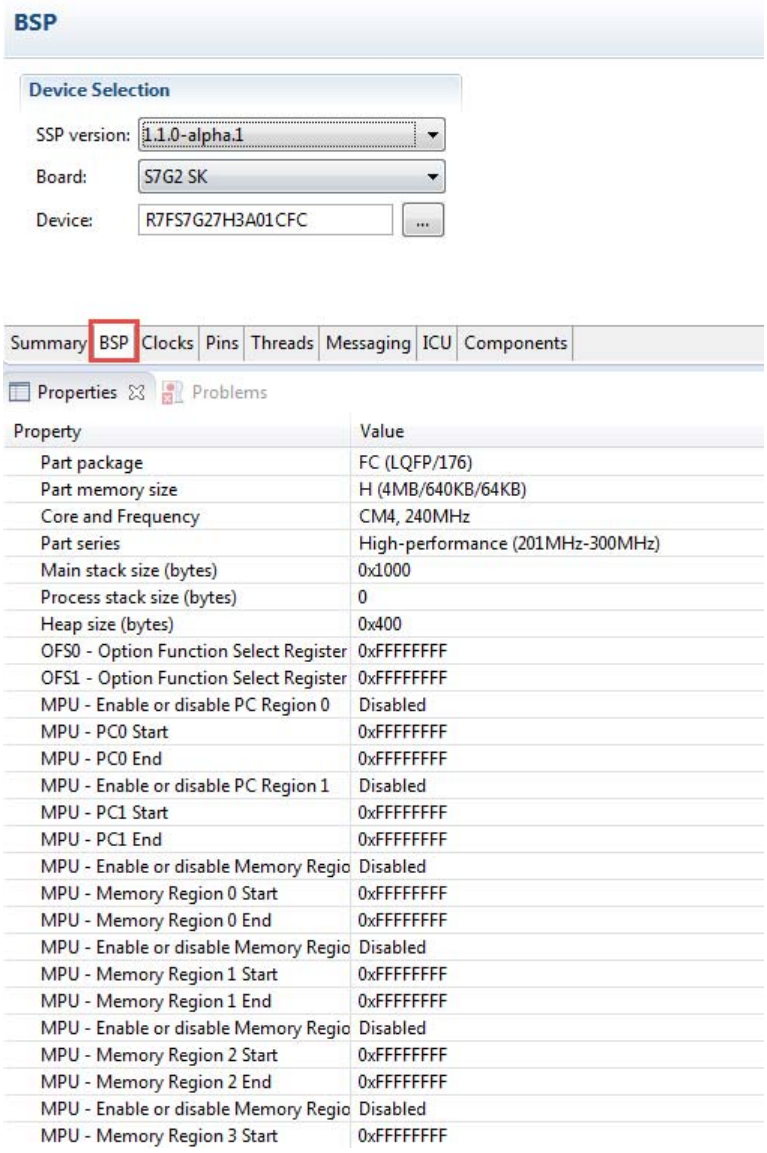

図 29: e² studio ISDE の [BSP] タブ

[Properties] ビューには、BSP で使用できる設定可能なオプションが表示されます。これらのオプションは、必要に応じて変更できます。BSP は、MCU 上の SSP レイヤーです。e² studio ISDE は、入力フィールドをチェックし、無効な入力を通知します。たとえば、スタックサイズには有効な数値のみを入力できます。

[Generate Project Content] ボタンを押すと、BSP の設定内容が次のファイルに書き込まれます。

synergy_cfg/ssp_cfg/bsp/bsp_cfg.h

このファイルが存在しない場合は作成されます。

Attention: このファイルは、[Generate Project Content] ボタンを押すたびに上書きされるため、編集しないでください。

3.1.5.2 クロックの設定

[Clocks] タブには、MCU のクロックツリーがグラフィカルに表示され、各種のクロック除算値とソースを変更できます。クロック設定が無効な場合、問題のあるクロック値は赤く強調表示されます。この設定でもコードを生成できますが、正しい動作は保証されません。次の図で、USB クロック UCLK の除算値が変更され、クロック周波数が必要な 48 MHz ではなく 60 MHz になっています。このパラメータは赤く表示されます。

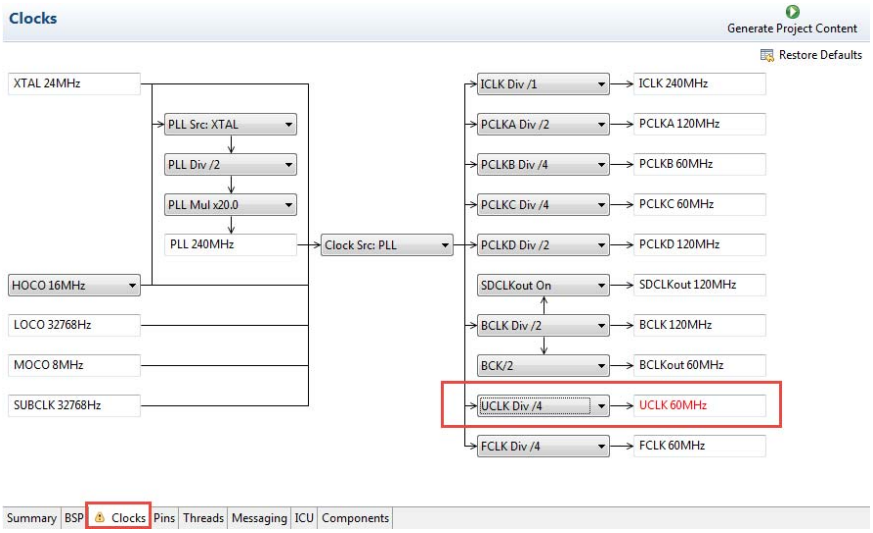

図 30: e^2 studio ISDE [Clocks] タブ

[Generate Project Content] ボタンを押すと、次のファイルにクロックの設定内容が書き込まれます :

synergy_cfg/ssp_cfg/bsp/bsp_clock_cfg.h

このファイルが存在しない場合は作成されます。

Attention: このファイルは、[Generate Project Content] ボタンを押すたびに上書きされるため、編集しないでください。

3.1.5.3 ピンの設定

[Pins] タブでは、MCU のピンを柔軟に設定できます。多数のピンが多数の機能を提供できるため、周辺機器ごとに設定できます。たとえば、SCI 周辺機器でシリアルチャネルを選択すると、そのモジュールとチャネルの送受信ピンの配置に関する複数のオプションが提供されます。設定されたピンは、[Package] ビューに緑色で表示されます。

Note: [Package] ビューウィンドウが e^2 studio ISDE で開いていない場合、トップメニューバーから [Window] > [Show View] > [その他] で [ビュー表示] ウィンドウを開き、[Pin Configurator] > [Package] を選択して開きます。

[Pins] タブでは、ピンまたは周辺機器ごとにエラーをハイライトし、オプションを表示することで、ピンが高度に多重化された大規模なパッケージを簡単に設定できます。DK-S7G2 などの特定のボードでプロジェクトテンプレートを選択した場合、ボードに接続された一部の周辺機器は事前に選択されています。

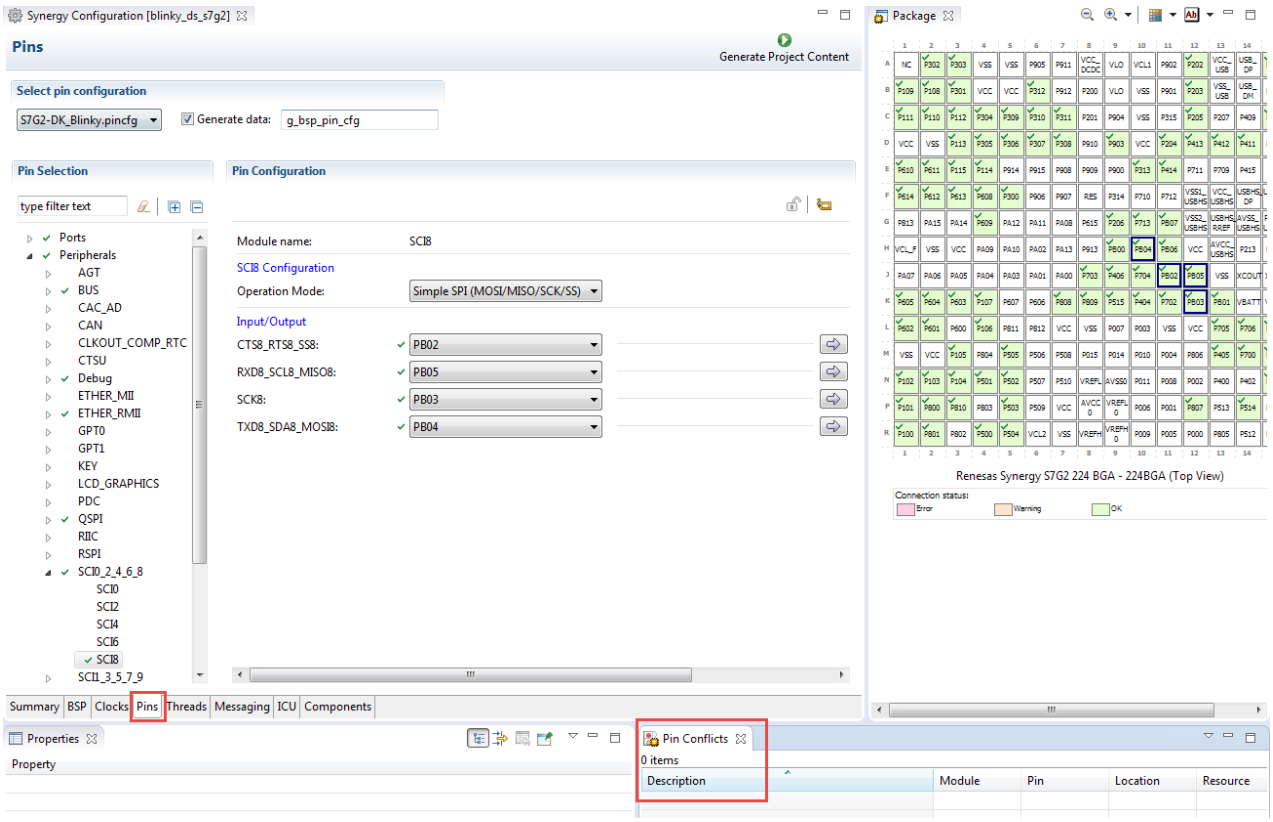

図 31: e^2 studio ISDE [Pins] タブ

ピンコンフィギュレータには、衝突チェッカーが組み込まれているため、同じピンが別の周辺機器や I/O 機能に割り当てられた場合、ピンが [Package] ビューで赤く表示され、メインの [Pins] タブの [Pin Selection] ペインと [Pin Configuration] ペインの赤い四角形の中に白い十字とともに表示されます。[Pin Conflicts] ビューにはすべての衝突が表示されるため、容易に識別および修正できます。

下の例では、ポート P103 は外部メモリ周辺機器によってすでに使用されており、このポートをシリアル通信インタフェース（SCI）につなげようとすると、ダングリング接続エラーが発生します。このエラーを修正するには、ピンのドロップダウンリストから別のポートを選択するか、タブの左側にある [Pin Selection] ペインで外部メモリ周辺機器を無効にします。

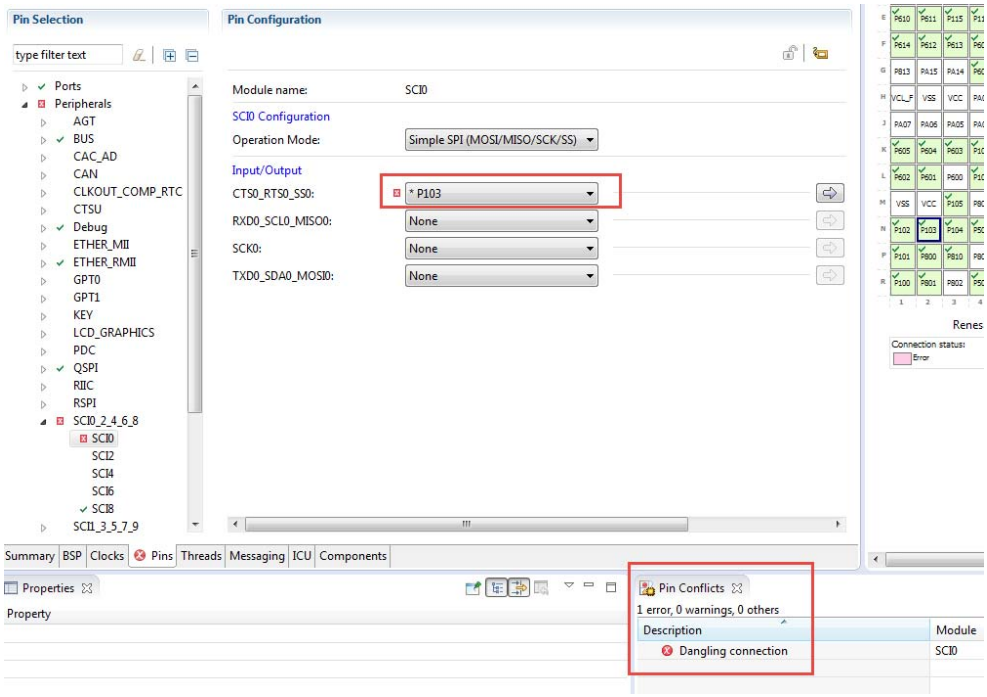

図 32: e^2 studio ISDE ピンコンフィギュレータ

ピンコンフィギュレータはまた、[Package] ビュー、および各ピンの電気特性または機能特性も表示します。

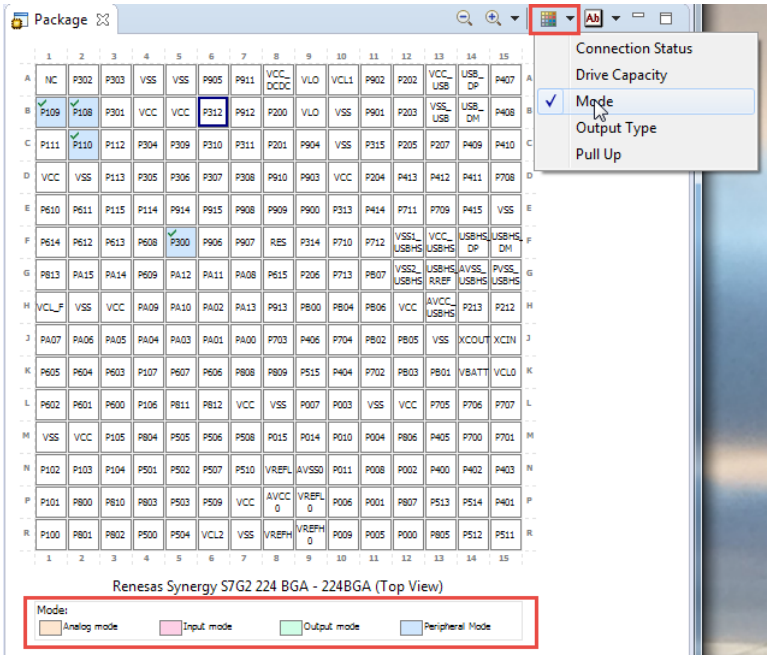

図 33: e² studio ISDE ピンコンフィギュレータ

[Generate Project Content] ボタンを押すと、次のファイルにピン構成の内容が書き込まれます :

synergy_cfg/ssp_cfg/bsp/bsp_pin_cfg.h

このファイルが存在しない場合は作成されます。

Attention: このファイルは、[Generate Project Content] ボタンを押すたびに上書きされるため、編集しないでください。

ピンに関する情報を簡単に共有できるよう、e² studio ISDE はピン設定を csv 形式でエクスポートして csv ファイルを synergy_cfg/ssp_cfg/bsp/pincfg_<MCU package>.csv にコピーします。

3.1.6 スレッドとドライバの追加

すべての ThreadX ベースの Synergy プロジェクトには、少なくとも 1 つの RTOS スレッドと、そのスレッドを実行する SSP モジュールのスタックが 1 つ含まれています。[Threads] タブは、スレッドに適切なモジュールを追加して、スレッドと、各スレッドに関連するモジュール双方のプロパティを設定できるグラフィカルユーザーインタフェースです。スレッドを設定すると、その構成に応じて e² studio ISDE が自動的にコードを生成します。

任意のドライバ、さらに一般的にはスレッドに追加する任意のモジュールに対し、e² studio ISDE は他のモジュールとのすべての依存関係を自動的に解決し、適切なスタックを作成します。このスタックは、選択されたモジュールとモジュールオプションに応じ、e² studio ISDE によって自動的に [Threads] ペインに入力表示されます。依存関係の要件を満たすモジュールが 2 つ以上ある場合、ドロップダウンメニューからモジュールを選択するよう表示されます。たとえば、スレッドにオーディオ再生フレームワークを追加する際には、再生のフレームワークとして DAC または I2S を選択する必要があります。オーディオ再生フレームワークではまた、タイマドライバが必要で、AGT、GPT ドライバ実装のいずれかと転送ドライバを選択し

なければなりません。転送ドライバの実装に選択肢があるかどうかは、この例では、選択された再生ハードウェアによって異なり、e2 studio ISDE では次のように表示されます。

- DAC を選択した場合、DMAC と DTC の実装のいずれかを転送ドライバとして選択するオプションがあります。
- I2S を選択した場合は、DTC のみがサポートされます。2 つのインスタンス（チャネルごとに 1 つ）の DTC が必要ですが、これは e2 studio ISDE によって追加されます。

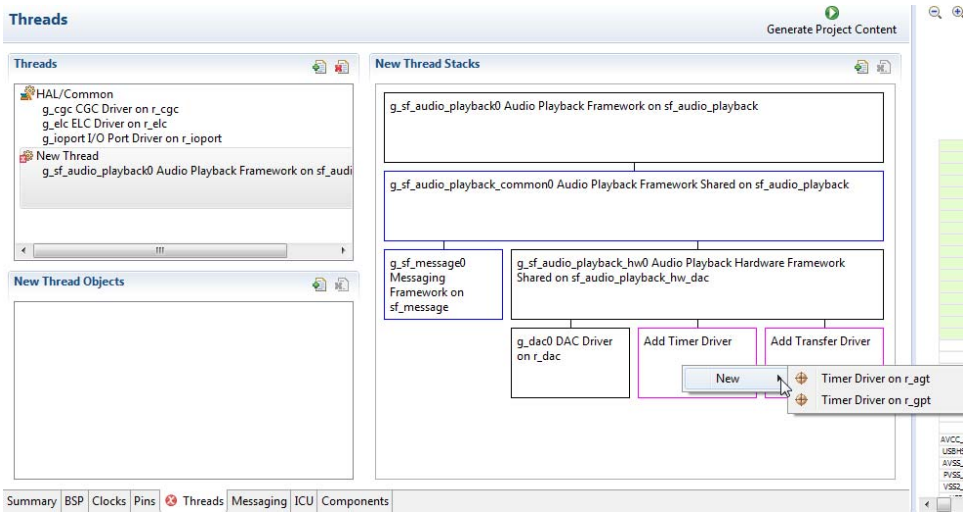

図 34: e2 studio ISDE プロジェクトコンフィギュレータ - 概要

[Threads] タブのデフォルトビューには、[HAL/Common] という共通スレッドが含まれています。このスレッドには、I/O 制御（IOPORT）、クロック発生回路（CGC）、イベントリンクコントローラ（ELC）のドライバが付属しています。デフォルトのスタックは [HAL/Common Stacks] ペインに表示されています。[HAL/Common] スレッドに追加されるデフォルトのモジュールは、SSP がそれぞれに単一のインスタンスのみを必要とするという点で特殊で、これは e2 studio ISDE によってすべてのユーザー定義スレッドにデフォルトで含まれます。

RTOS を使用しないアプリケーション、または RTOS 外で実行されるアプリケーションでは、[HAL/Common] スレッドがアプリケーションにドライバを追加できるデフォルトの場所となります。

モジュールとスレッドの追加および設定方法についての詳細は、以下のセクションを参照してください。

- HAL ドライバの追加と設定
- ドライバのスレッドへの追加とドライバの設定

モジュールを [HAL/Common] または新しいスレッドに追加したら、[Properties] ビューでドライバ設定オプションにアクセスできます。スレッドオブジェクトを追加したら、同様に [Properties] ビューでオブジェクト構成オプションにアクセスできます。

スレッドの設定方法については、こちらを参照してください : スレッドの設定

Note: ドライバやモジュールの選択肢や構成オプションは SSP パックで定義され、そのため SSP バージョン変更時に変更することができます。

3.1.6.1 HAL ドライバの追加と設定

RTOS の外または RTOS なしで実行されるアプリケーションの場合、[HAL/Common] スレッドを使用してアプリケーションに他の HAL ドライバを追加できます。ドライバを追加するには、次の手順を実行します。

1) [Threads] ペインで [HAL/Common] の領域をクリックします。[Stacks] ペインが [HAL/Common Stacks] に変わります。

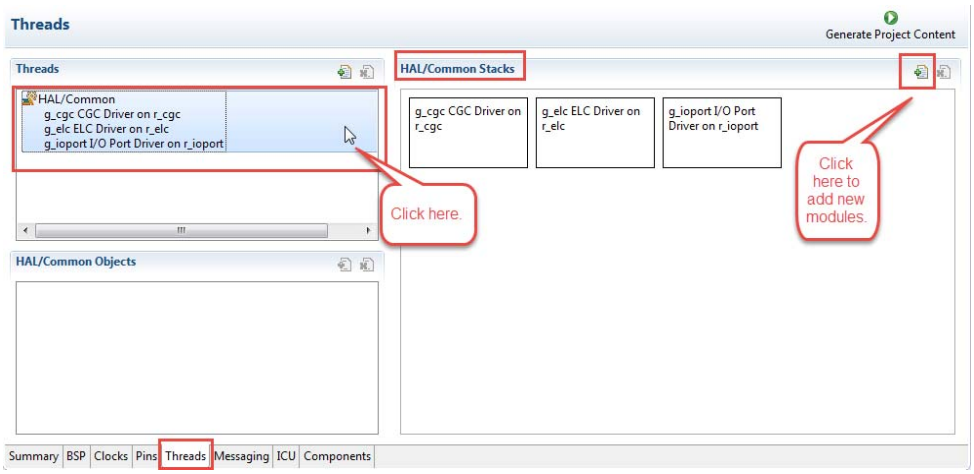

図 35: e² studio ISDE プロジェクトコンフィギュレータ - ドライバの追加

2) [New Stack] をクリックすると、SSP で使用可能な HAL レベルドライバのドロップダウンリストが表示されます。

3) [New Stack] > [Driver] からドライバを選択します。また、[New Stack] > [Framework] > [Service] > [Power Profiles Framework on sf_power_profiles] から RTOS 独立アプリケーションのパワープロファイルを選択することができます。他のすべてのモジュールは、ThreadX が存在する場合にのみ、スレッドに追加できます。

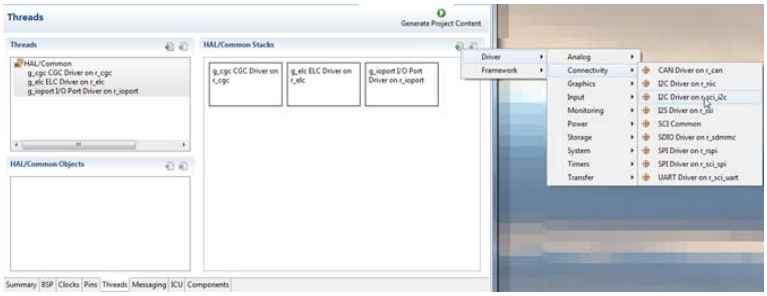

図 36: ドライバを選択します

e² studio ISDE は選択されたドライバのスタックを作成し、ドライバで有効化する必要のある追加のリソースがいつ必要かを知らせます。その場合、[Properties] ビューで割り込みを設定するか、[ICU] タブで後に不足している割り込みを追加できます。

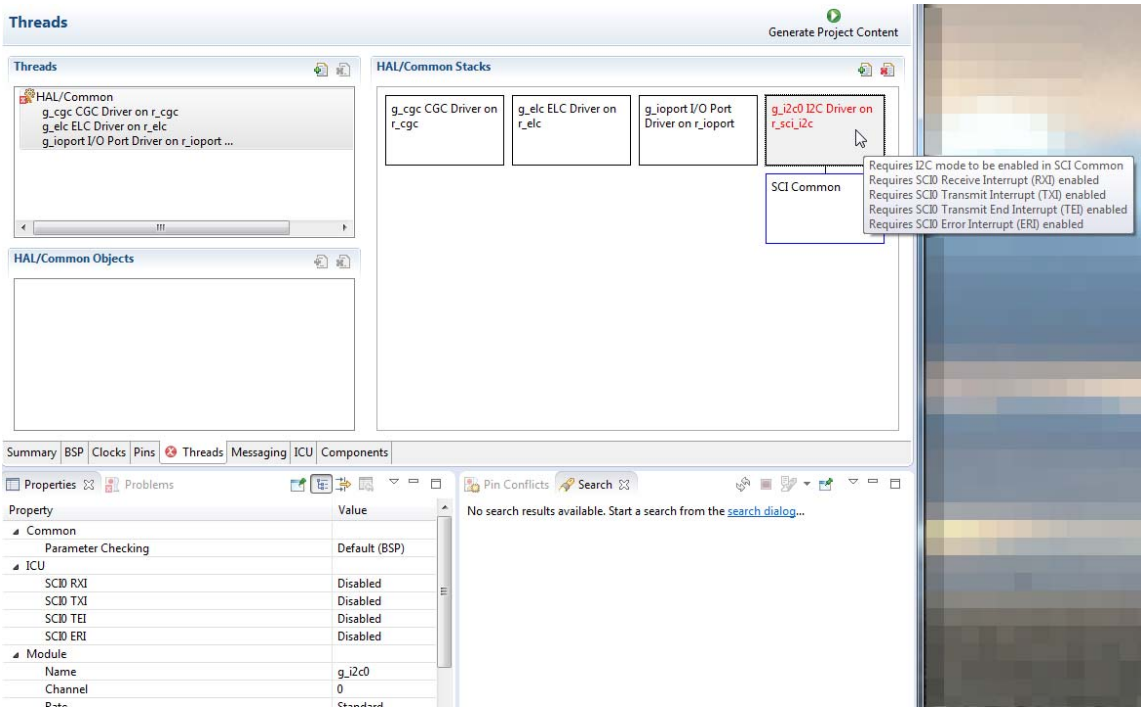

図 37: [Threads] タブでの依存関係の確認

4) ドライバモジュールを [HAL/Common Stacks] ペインで選択し、ドライバのプロパティを [Properties] ビューで構成します。

[Generate Project Content] ボタンをクリックすると、e^2 studio ISDE によって以下のファイルが追加されます。

- synergy/ssp ディレクトリ向けに選択したドライバモジュールとそのファイル。
- main() 関数と構成構造体、およびアプリケーション用のヘッダーファイルが下記の表に表示されています。

ファイル	内容	[Generate Project Content] で上書きされるか
src/synergy_gen/main.c	main() 呼び出しとユーザーコードが含まれます。呼び出されたときには、BSP により MCU が初期化されています。	はい
src/synergy_gen/hal_data.c	HAL ドライバのみのモジュールの設定構造体。	はい

ファイル	内容	[Generate Project Content] で上書きされるか
src/synergy_gen/hal_data.h	HAL ドライバのみのモジュールのヘッダーファイル。	はい
src/hal_entry.c	HAL ドライバのみのコード用のユーザーエントリポイント。ここにコードを追加します。	いいえ

追加されているすべてのモジュール用の構成ヘッダーファイルは、このフォルダに作成または上書きされます。

synergy_cfg/ssp_cfg/driver

3.1.6.2 ドライバのスレッドへの追加とドライバの設定

ThreadX RTOS で使用されるアプリケーションの場合、1 つ以上のスレッドを追加し、スレッドごとにそのスレッドで実行される少なくとも 1 つのモジュールを追加できます。ドライバまたはフレームワークのドロップダウンメニューからモジュールを選択できます。スレッドにモジュールを追加するには、次の手順を実行します。

1) [Threads] ペインで、[New Thread] アイコンをクリックし、スレッドを追加します。

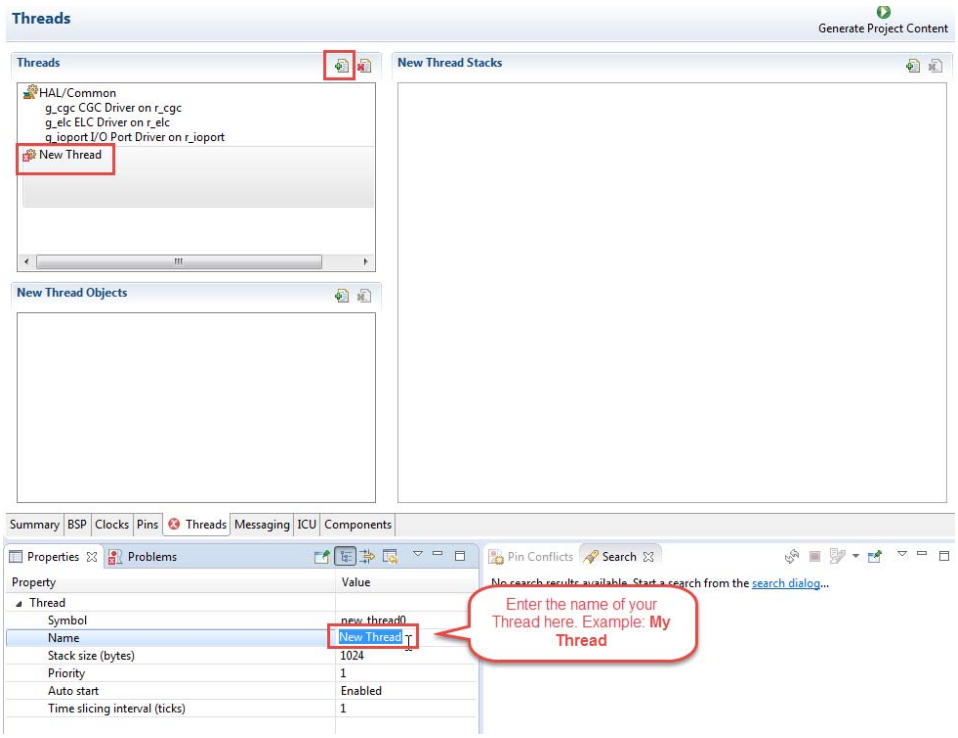

図 38: [Threads] タブでの新規 [RTOS スレッド] の追加

2) [Properties] ビューで、[Name] と [Symbol] のエントリーをクリックして、新しいスレッドに固有の名前と記号を入力します。

   Note: e^2 studio ISDE によって、スレッドスタックペインの名前が [My Thread Stacks] に更新されます。

3) [My Thread Stacks] ペインで、[New Stack] アイコンをクリックして、[Driver] と [Framework] のリストを表示します。フレームワークレベルのモジュールと HAL レベルのドライバの両方をここで追加できます。

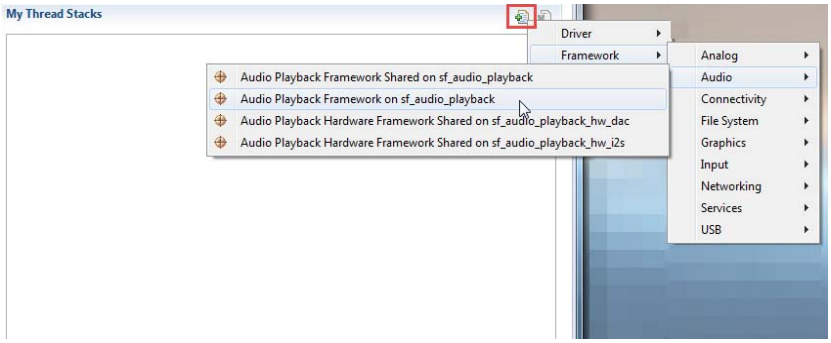

図 39:　　モジュールとドライバをスレッドに追加

4) リストからモジュールまたはドライバを選択します。

5) モジュールまたはドライバが依存関係を示している場合、不足しているリソースを選択します。

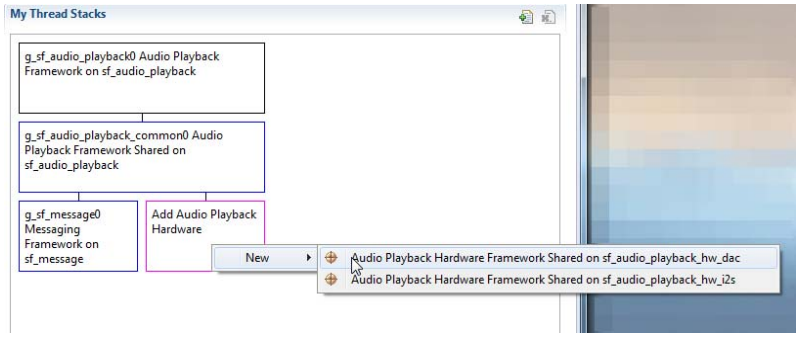

図 40:　　[Threads] タブでモジュールまたはドライバの依存関係の特定

6) 追加したドライバをクリックして、[Properties] ビューの構成パラメータを更新することにより、アプリケーションに必要なドライバを構成します。選択したモジュールまたはドライバを表示し、そのプロパティを編集するには、ドライバを含むスレッドが [Threads] ペインでハイライトされていることを確認します。

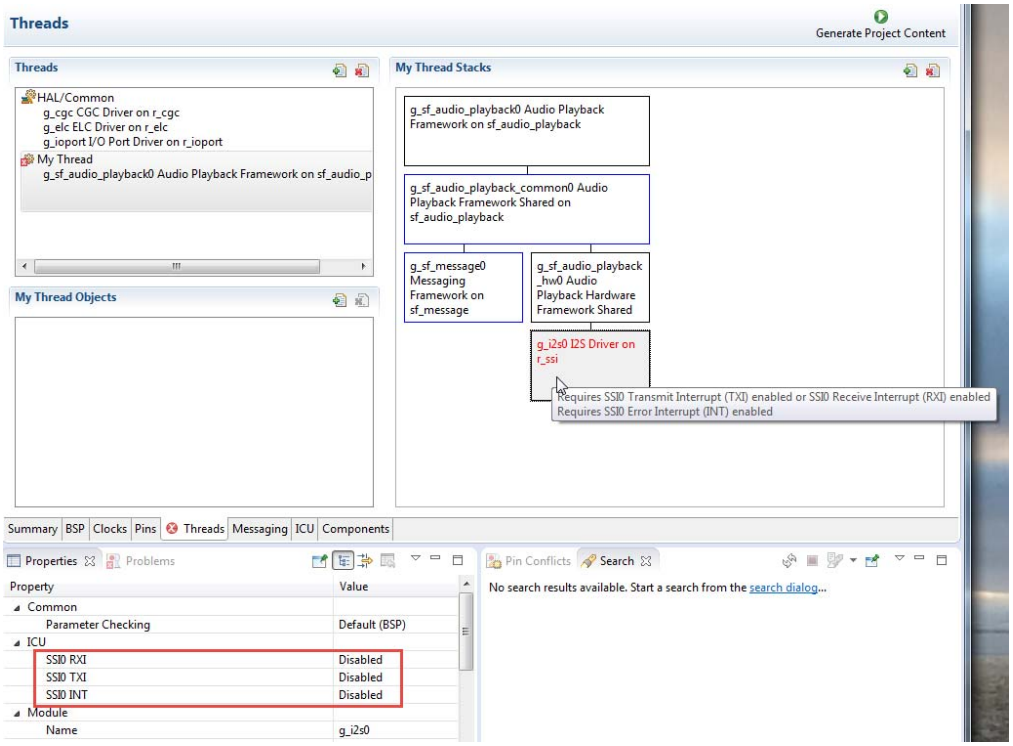

図 41:　モジュールまたはドライバのプロパティの設定

7) 必要に応じて、[Threads] ペインの [New Thread] をクリックして、別のスレッドを追加します。

上記の例で [Generate Project Content] ボタンを押すと、e^2 studio ISDE によって以下の表に示されているファイルが作成されます。

ファイル	内容	[Generate Project Content] で上書きされるか
src/synergy_gen/main.c	main() 呼び出しとユーザーコードが含まれます。呼び出されたときには、BSP により MCU が初期化されています。	はい
src/synergy_gen/my_thread.c	生成されたスレッド "my_thread" と、このスレッドに追加されたモジュールの構成構造体。	はい
src/synergy_gen/my_thread.h	スレッド "my_thread" 用のヘッダーファイル	はい

ファイル	内容	[Generate Project Content] で上書きされるか
src/synergy_gen/hal_data.c	HAL ドライバのみのモジュールの設定構造体。	はい
src/synergy_gen/hal_data.h	HAL ドライバのみのモジュールのヘッダーファイル。	はい
src/hal_entry.c	HAL ドライバのみのコード用のユーザーエントリポイント。ここにコードを追加します。	いいえ
src/my_thread_entry.c	スレッド "my_thread" 用のユーザーエントリポイント。ここにコードを追加します。	いいえ

追加されているすべてのモジュールとドライバ用の構成ヘッダーファイルは、次のフォルダに作成または上書きされます。

synergy_cfg/ssp_cfg/driver

synergy_cfg/ssp_cfg/framework

3.1.6.3 スレッドの設定

アプリケーションで ThreadX RTOS が使用されている場合、[Threads] タブを使用して、ThreadX スレッド、セマフォ、ミューテックス、イベントフラグを簡単に作成できます。

各スレッドのコンポーネントは、[Properties] ビューで構成できます。

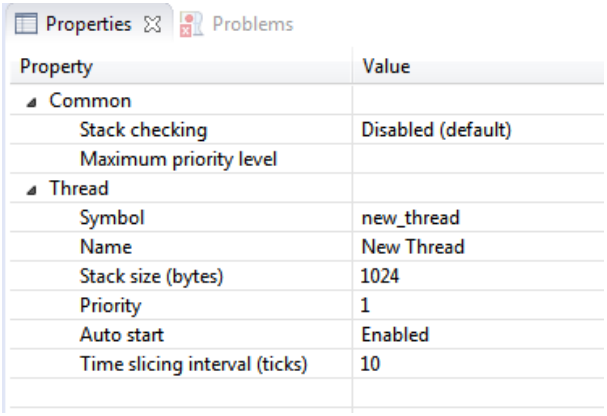

図 42: e^2 studio ISDE スレッドのプロパティ

[Properties] ビューには、すべてのスレッドに共通の設定（[Common]）と、この特定のスレッドの設定（[Threads]）が含まれています。

このスレッドインスタンスについて、スレッドの名前、プライオリティレベルやスタックサイズなどのプロパティを簡単に設定できます。e2 studio ISDE は、プロパティフィールドのエントリーが有効であることを確認します。たとえば e2 studio ISDE は、整数値を必要とする [Priority] フィールドなどに 0 ～ 9 の数値のみが含まれていることを確認します。

ThreadX リソースをスレッドに追加するには、[Thread Objects] ペインでスレッドを選択して [New Object] をクリックします。ペインは選択したスレッドの名前、この場合は [My Thread Objects] となります。

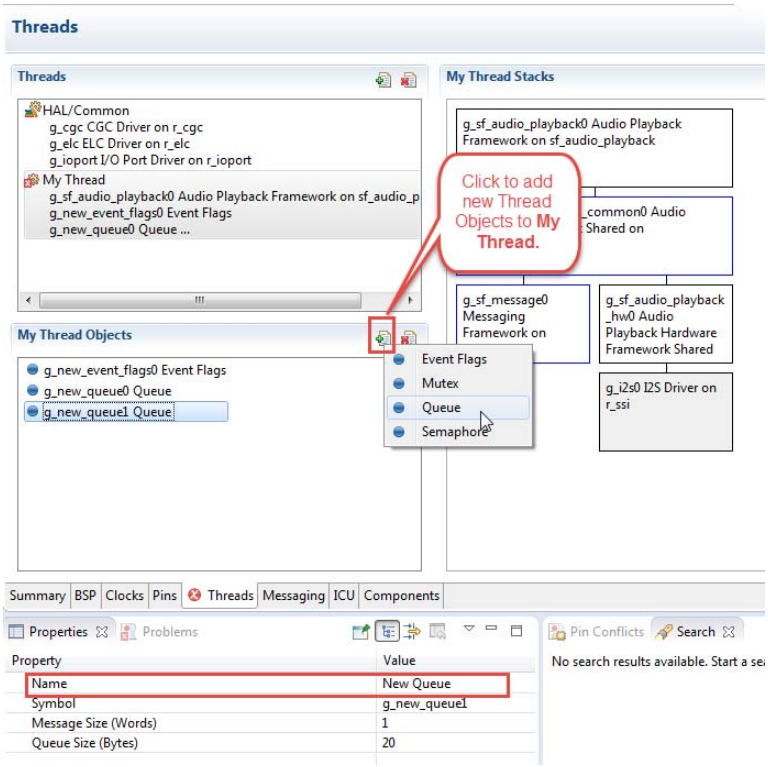

図 43:　スレッドプロジェクトのプロパティの設定

それぞれのスレッドオブジェクトに固有の名前と記号が付されるよう、[Name] と [Symbol] のエントリーを [Properties] ビューで更新します。

3.1.6.4 割り込みの設定

これには、[Threads] タブまたは [ICU]（割り込み制御ユニット）タブにある [Properties] ビューを使用して割り込みの優先度を設定し、割り込みを有効にします。[Threads] ペインで表示するスレッドを選択し、プロパティを編集します。

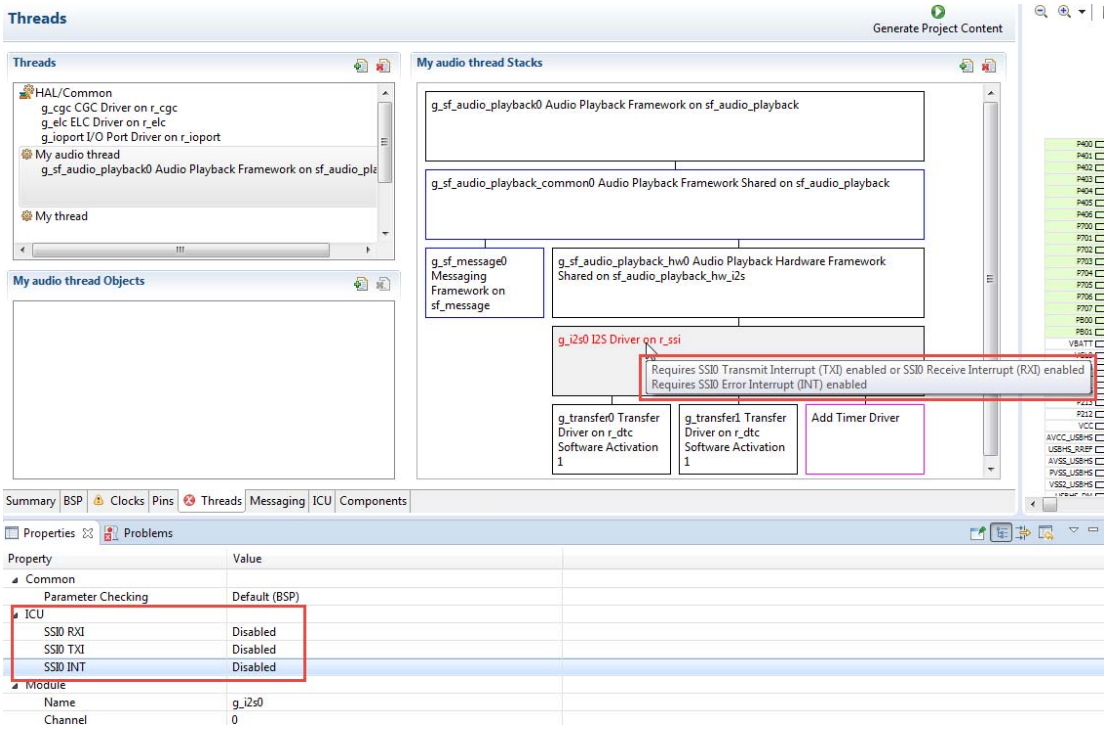

図 44: [Threads] タブで割り込みの設定

3.1.7 SSP メッセージングフレームワークの設定

メッセージングフレームワークは ThreadX メッセージングキュー機能に拡張される、最も重要な SSP モジュールの 1 つです。これは、スレッドが相互にメッセージを交換して通信するメカニズムを提供します。メッセージングフレームワークは、事前設定またはユーザー設定されたイベントが発生した際にスレッドがメッセージを送信（パブリッシュ）または受信待ち（リッスン）できるようにします。スレッドは、特定のイベントクラスをサブスクライブしているすべてのスレッドが受信待ちおよび対応できるイベントクラスを付随させたメッセージをパブリッシュできます。特定のイベントクラスを受信待ちできるスレッドのリストは、そのイベントクラスのサブスクライバーリストと呼ばれます。

メッセージングフレームワークを使用するには、まず [Threads] タブでメッセージングフレームワークインスタンスを 1 つ追加する必要があります。メッセージングフレームワークは、HAL/Common スレッド以外のすべてのスレッドに追加できます。プロジェクト内のすべてのスレッドがこのインスタンスを使用して相互に通信できます。タッチパネルフレームワークやオーディオ再生フレームワークなどのモジュールではメッセージングフレームワークが必要で、以下のオーディオ再生フレームワークの例に示されているとおり、自動的に追加されます。プロジェクトにモジュールなどのスレッドが含まれている場合は、アプリケーションにスレッドを追加しても、メッセージングフレームワークの他のインスタンスを追加する必要はありません。すべてのスレッドが 1 つのメッセージングフレームワークインスタンスを共有できます。

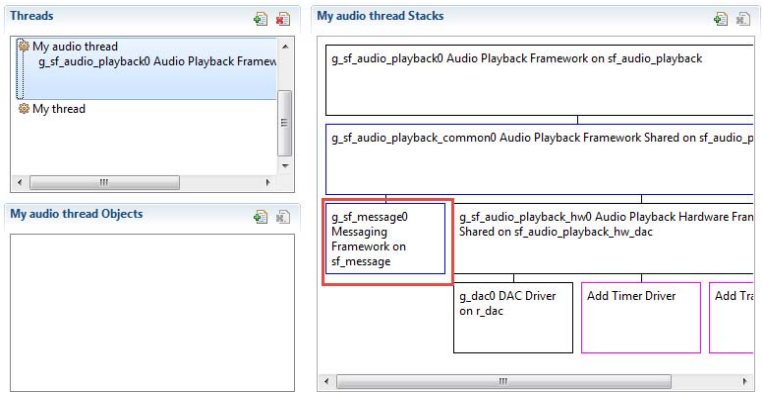

図 45:　スレッドへのメッセージングモジュールの追加

[Threads] タブにメッセージングフレームワークインスタンスを追加したら、[Messaging] タブを使用して独自のイベントクラスとイベントを定義し、どのスレッドがどのイベントクラスを受信待ちするかを決定できます。SSP には、タッチパネルおよびオーディオフレームワークモジュール用に事前定義されたイベントクラスとイベントが含まれています。これらのモジュールを追加した場合、事前定義されたイベントクラスとイベントが [Messaging] タブにも表示されます。オーディオ再生フレームワーク用に事前定義されたイベントクラスとイベントは、以下のとおりです。

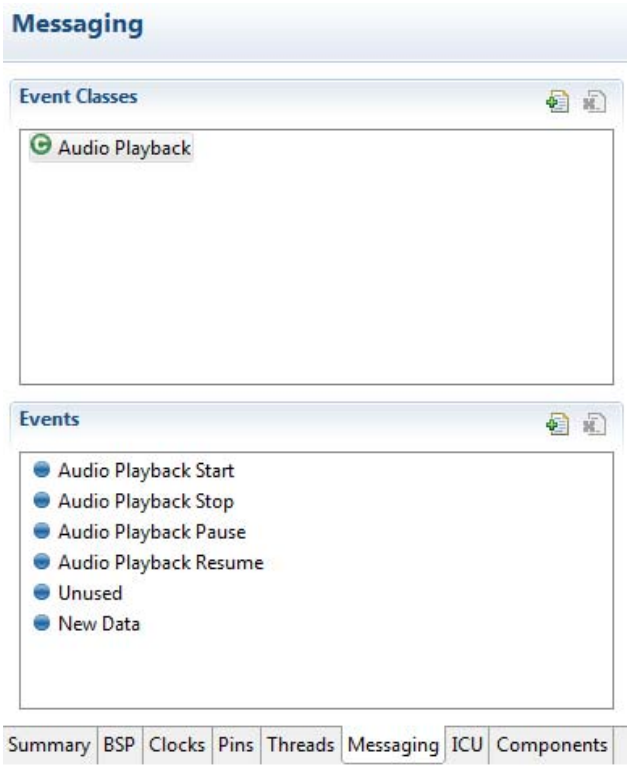

図 46:　オーディオ再生フレームワーク用に事前定義されたイベントクラス

3.1.7.1 イベントクラスの追加

独自のユーザー定義イベントクラスをメッセージングシステムに追加するには、次の手順を実行します。

1) [Messaging] タブで、[Event Classes] ペインを選択し、ボタンを追加します。
2) イベントクラスに一意の名前を入力します。
3) [OK] をクリックします。

   Note: ユーザー定義のイベントクラスは、イベントクラスアイコン右上の金色の四角形でマークされています。

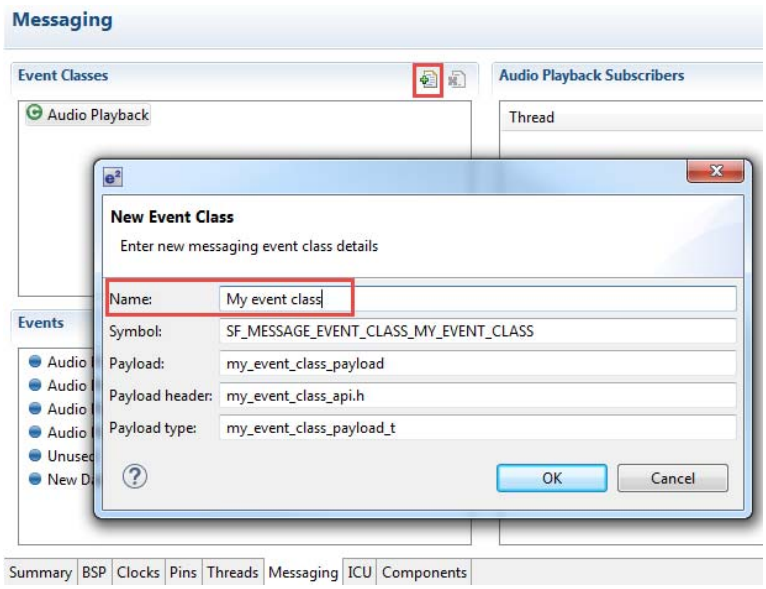

図 47: メッセージング – イベントクラスの追加

3.1.7.2 イベントの追加

独自のユーザー定義イベントをメッセージングシステムに追加するには、次の手順を実行します。

1) [Messaging] タブで、[Event] ペインを選択し、ボタンを追加します。
2) イベントクラスに一意の名前を入力します。
3) [OK] をクリックします。

   Note: ユーザー定義のイベントは、イベントアイコン右上の金色の四角形でマークされています。

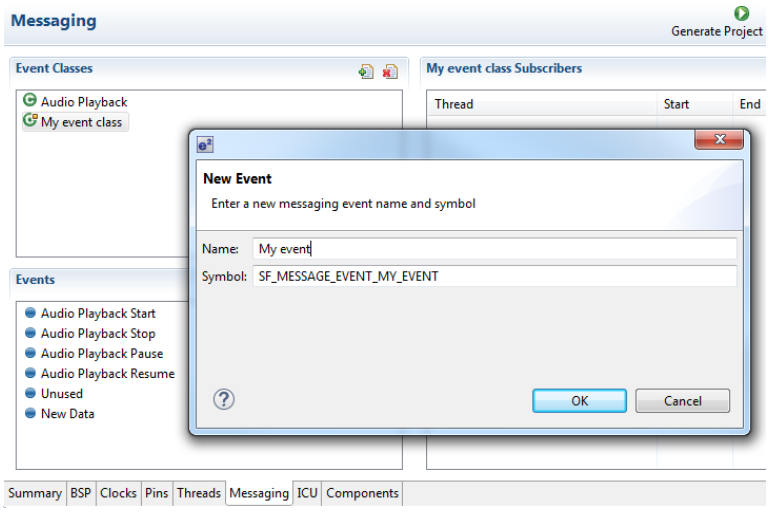

図 48: メッセージング – イベントの追加

3.1.7.3 メッセージングサブスクライバーリストの設定

サブスクライバーリストでは、メッセージパブリッシャーからのメッセージを受信待ちするスレッドを選択します。パブリッシュスレッドと受信待ちスレッド間の接続は、イベントクラスを通じて確立されます。そのため、プロジェクト内の各イベントクラスについてサブスクライバーリストを定義する必要があります。サブスクライバーリスト内のすべてのスレッドが、選択されたイベントクラスに属するメッセージを受信待ちして対応できます。

以下では、[Threads] タブで 2 つのスレッドが定義され、そのうちの 1 つがオーディオ再生フレームワークを使用することが想定されています。

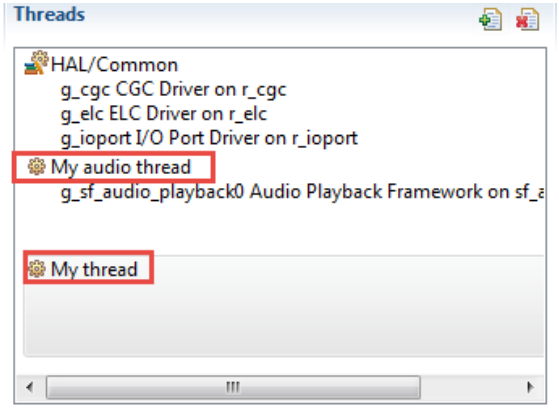

図 49: メッセージング – スレッドの例

イベントクラスのサブスクライバーリストを設定するには、次の手順を実行します。

1) [Messaging] タブの [Event Class] ペインでイベントクラスを選択します。[Subscriber List] ペインは選択したイベントクラスの名前になります。
2) 追加アイコンをクリックすると、[New Subscriber] ダイアログボックスが表示されます。
3) [Threads] ドロップダウンニューから、サブスクライバーリストの追加するスレッドを選択します。
4) 開始点と終了点を選択することによりインスタンス範囲を決定します。特定のイベントクラスのインスタンスが 1 つだけの場合は、開始値と終了値値をデフォルト値（0）に維持します。イベントクラスのインスタンスが 2 つ以上ある場合にインスタンス範囲を選択する方法については、メッセージングフレームワークユーザーガイドを参照してください。複数のチャネルでのオーディオストリーミングなど、同じイベントクラスを複数回使用するアプリケーションでは、イベントクラスのインスタンスが複数あると有用です。
5) 選択したイベントクラスのサブスクライバーリストに追加するそれぞれのスレッドについて、手順 3 と 4 を繰り返します。

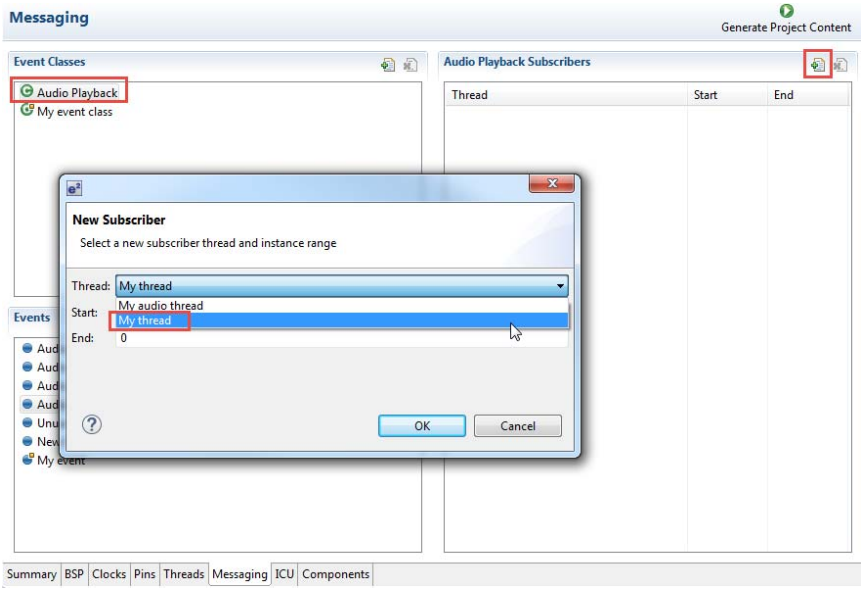

図 50: メッセージング – サブスクライバーリストの設定

3.1.7.4 メッセージングフレームワーク用ファイルの生成

[Generate Project Content] ボタンをクリックすると、e^2 studio ISDE によって以下の構成済みメッセージングフレームワーク用ファイルが生成されます。

ファイル	内容	[Generate Project Content] で上書きされるか
synergy_cfg/ssp_cfg/framework/sf_message_port.h	イベントクラスとイベントの列挙を含みます。	はい
synergy_cfg/ssp_cfg/framework/sf_message_payloads.h	イベントクラスペイロードへのポインタを含みます。	はい
synergy_cfg/ssp_cfg/framework/sf_message_payloads.h	メッセージングフレームワークのコンパイラオプション	はい

3.1.8 コンポーネントの確認と追加

[Components] タブでは、アプリケーションが必要とする個々のモジュールを追加または除外できます。すべての Synergy プロジェクトに共通するモジュールは、あらかじめ選択されています（たとえば、BSP>BSP> ボード固有 BSP および HAL ドライバ > すべて >r_cgc）。[Threads] タブで選択されたモジュールに必要なモジュールはすべて自動的に含まれます。モジュールを追加するには、必要なコンポーネントの横にあるボックスをオンにし、除外するにはボックスをオフにします。

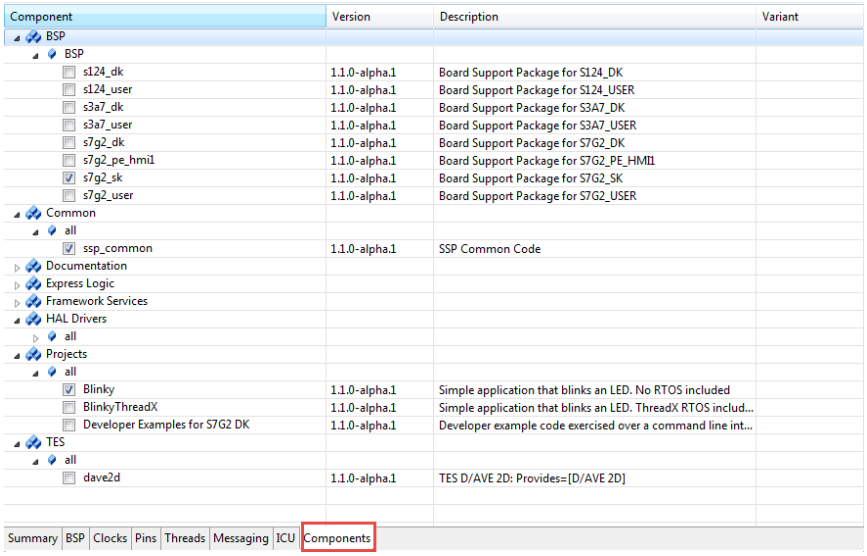

図 51: [Components] タブ

RTOS ThreadX、ファイルシステム FileX、TCP/IP ネットワーキング NetX™ など、HAL レイヤーおよびフレームワークレイヤーのコンポーネントは、Express Logic 製のコンポーネントとして使用できます。加えて、プロジェクトに追加するドキュメントを選択するか、プロジェクト全体を含めることができます。

[Components] タブでは、プロジェクトのモジュールを選択しますが、他のタブでモジュールそのものを構成する必要があります。[Generate Project Content] ボタンを押すと、パックファイル用の各コンポーネントの .c ファイルと .h ファイルが以下のフォルダにコピーされます。

synergy/ssp/inc/bsp

synergy/ssp/inc/driver

synergy/ssp/inc/framework

synergy/ssp/src/bsp

synergy/ssp/src/driver

synergy/ssp/src/framework

また e^2 studio ISDE は、残りの [Threads] タブと [ICU] タブからの構成オプションが含まれた構成ファイルを synergy_cfg/ssp_cfg フォルダに作成します。

3.1.9 アプリケーションの作成

モジュールとドライバを追加して、[Threads] タブで設定パラメータを設定した後、モジュールおよびドライバを呼び出すアプリケーションコードを追加できます。

Note: 構成を確認するには、プロジェクトをエラーなしで一度構築してから、独自のアプリケーションコードを追加します。

3.1.9.1 RTOS を使用しないアプリケーション

アプリケーションを作成するには、以下の手順を実行します。

1) [Threads] タブですべてのドライバとモジュールを追加し、不足している割り込みやドライバのような、e^2 studio ISDE によってフラグされたすべての依存関係を解決します。
2) [Properties] ビューでドライバを構成します。
3) [Project Configuration] ビューで [Generate Project Content] ボタンを押します。
4) [Project Explorer] ビューで、src/hal_entry.c ファイルをダブルクリックして、ソースファイルを編集します。

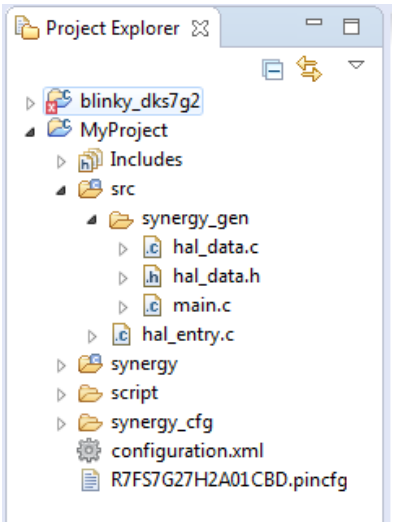

図 52: hal_entry.c

Note：アプリケーションで呼び出すドライバに必要なすべての構成構造体は、src/synergy_gen/hal_data.c で初期化されます。

Note：ディレクトリ src/synergy_gen のファイルは変更しないでください。これらのファイルは [Generate Project Content] ボタンを押すたびに上書きされます。

5) アプリケーションコードをここに追加します。

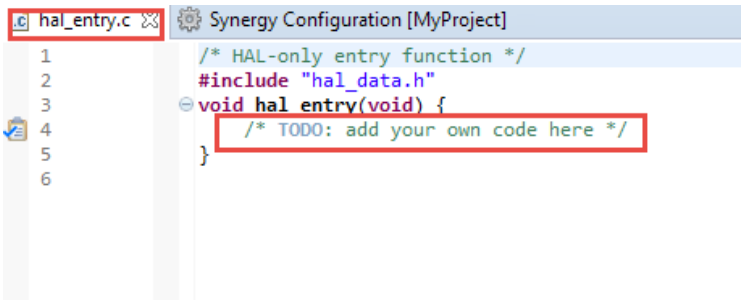

図 53: ユーザーコードを hal_entry.c に追加します。

6) [Project] > [Build Project] をクリックして、エラーなくプロジェクトをビルドします。

次のチュートリアルでは、上記の手順を実行してアプリケーションコードを追加する方法について説明しています：チュートリアル：Using HAL Drivers - Programming the WDT

WDT の例は HAL レベルアプリケーションで、RTOS を使用しません。各モジュールのユーザーガイドには、hal_entry.c に追加できる基本的なアプリケーションコードも含まれます。

3.1.9.2 ThreadX アプリケーション

ThreadX を使用した RTOS 対応アプリケーションコードを記述するには、次の手順を実行します。

1) Add a thread using the[Threads] タブを使用してスレッドを追加します。
2) このスレッドの一意の名前を [Properties] ビューで指定します。
3) このスレッドのすべてのドライバとリソースを構成し、不足している割り込みやドライバのような、e^2 studio ISDE によってフラグされたすべての依存関係を解決します。
4) スレッドオブジェクトを構成します。
5) 各スレッドオブジェクトの一意の名前を [Properties] ビューで指定します。
6) 必要に応じてさらにスレッドを追加し、手順 1 から 5 を繰り返します。
7) [Synergy Project Editor] で [Generate Project Content] ボタンを押します。
8) [Project Explorer] ビューで、src/my_thread_1_entry.c ファイルをダブルクリックして、ソースファイルを編集します。

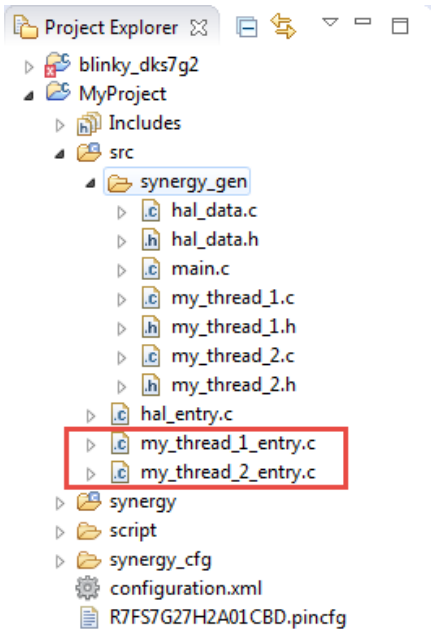

図 54: RTOS アプリケーション用 e^2 studio ISDE 生成ファイル

Note：アプリケーションで呼び出すドライバに必要なすべての構成構造体は、synergy_gen/my_thread_1.c と my_thread_2.c で初期化されます。

Note：ディレクトリ src/synergy_gen のファイルは変更しないでください。これらのファイルは [Generate Project Content] ボタンを押すたびに上書きされます。

9) アプリケーションコードをここに追加します。

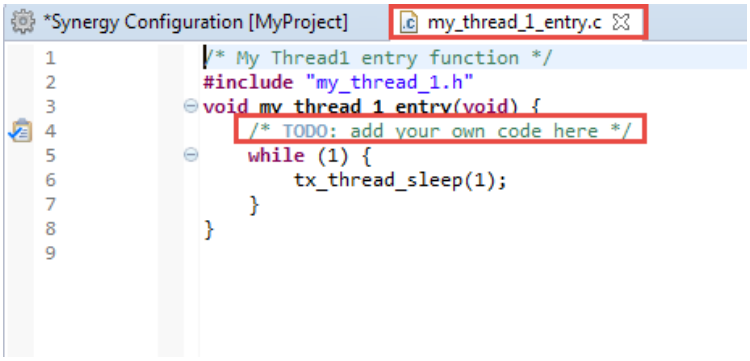

図 55: ユーザーコードを my_thread_1.entry に追加します。

10) 次のスレッドで手順 1 から 9 を繰り返します。

11) [Project] > [Build Project] をクリックして、エラーなくプロジェクトをビルドします。

3.1.10 プロジェクトのデバッグ

プロジェクトの構築がエラーなしで完了すると、デバッガーを使ってアプリケーションをボードにダウンロードして、実行できます。

アプリケーションをデバッグするには、次の手順を実行します。

1) デバッグアイコンの横にあるドロップダウンメニューで [Debug Configurations] を選択します。

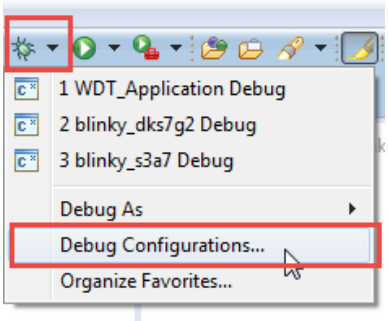

図 56: [Debug Configurations] ダイアログの呼び出し

2) [Debug Configurations] ビューで [MyProject Debug] としてリストされているプロジェクトをクリックします。

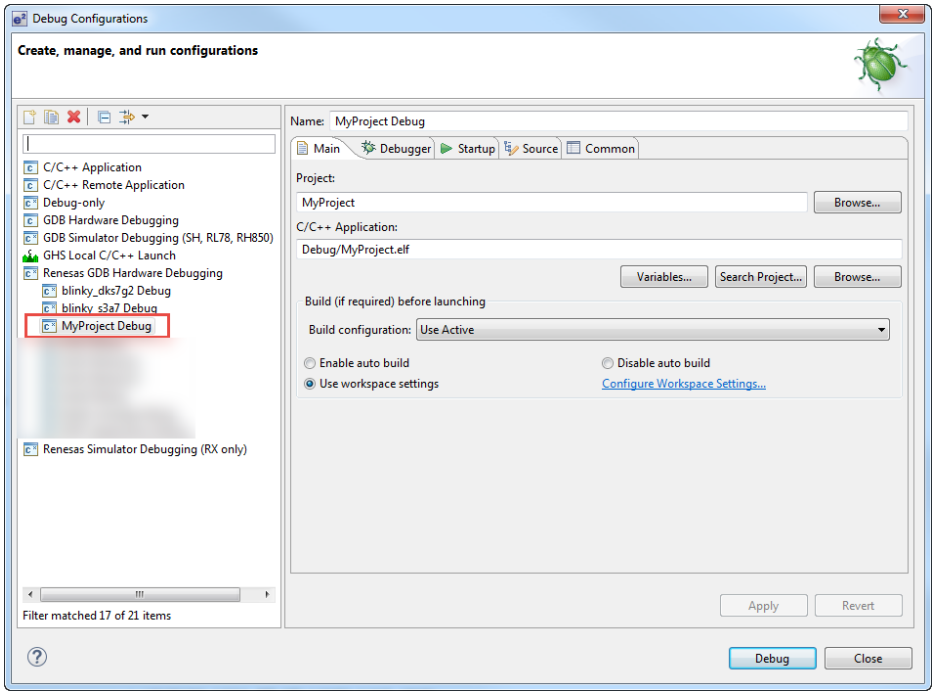

図 57: [Debug Configuration]

3) スタンドアロンの Segger J-Link デバッガまたは Segger J-Link On-Board（すべての Synergy DK および SK に含まれる）を使ってボードをパソコンに接続し、[Debug] をクリックします。

Note: J-Link の使用方法とボードをパソコンに接続する方法については、Synergy キットに含まれるクイックスタートガイドを参照してください。

3.1.11 Synergy プロジェクトを使用する TraceX の作成

Note: Synergy プロジェクトで TraceX を使用する前に、Synergy Gallery ウェブサイトから TraceX 実行ファイルをダウンロードする必要があります。

TraceX はホストベースの分析ツールで、リアルタイムのシステムイベントをグラフィカルに表示できます。TraceX はターゲットデバイスのデータを収集して、検査と分析のためにデータを表示します。MCU の TraceX バージョンが Synergy Gallery ウェブサイトからダウンロードでき、Synergy ライセンスと共に使用できます。

TraceX を使用するには、次の手順を実行します。

1) [Threads] タブを使用して、スレッドのプロパティウィンドウの TraceX を有効にします。TraceX バッファーのデフォルト名は g_tx_trace_buffer のままにします。

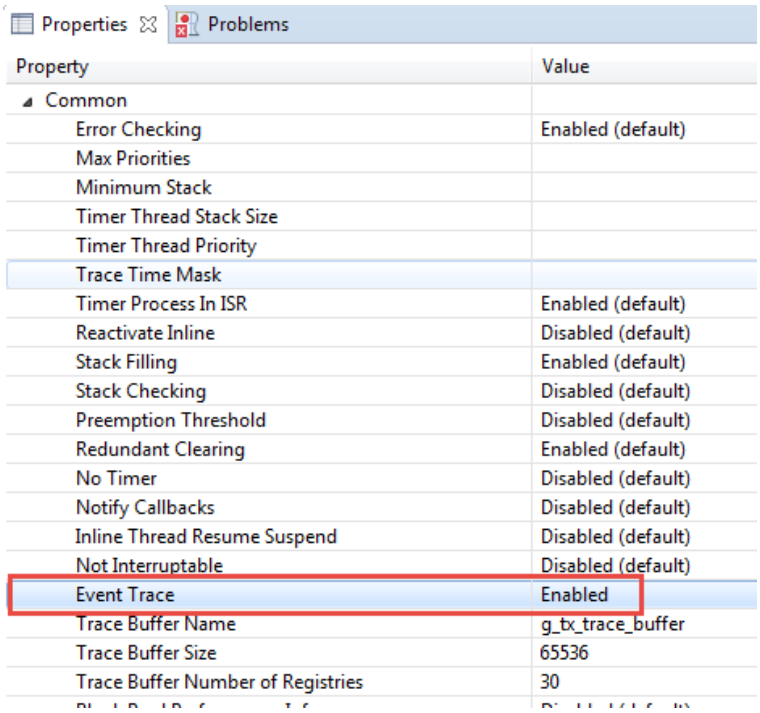

図 58: TraceX の設定

2) [Window] > [Preferences] から [C] > [Renesas] > [TraceX] で TraceX アプリケーションへのパスを設定します。

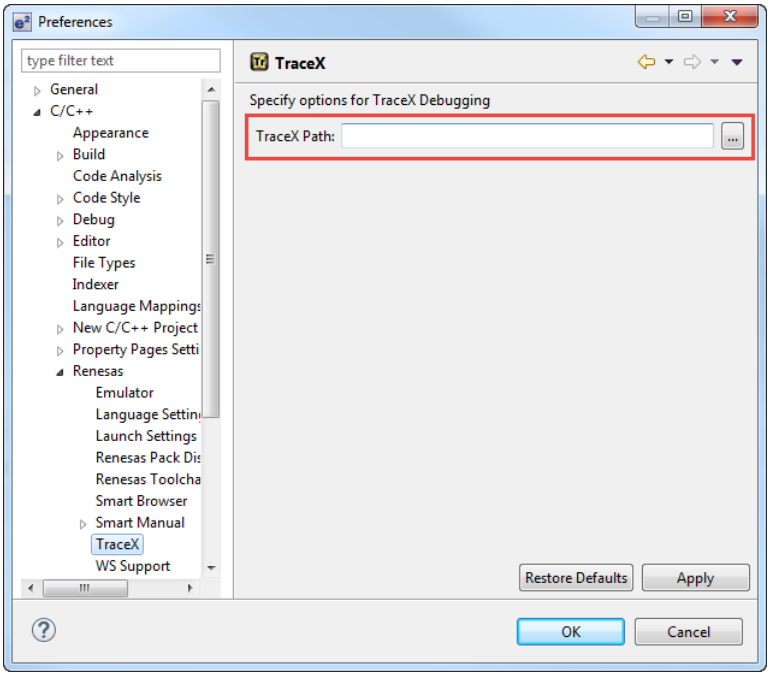

図 59: e2 studio ISDE TraceX パス

3) [Run] > [TraceX] で [Launch TraceX Debugging] を選択します。

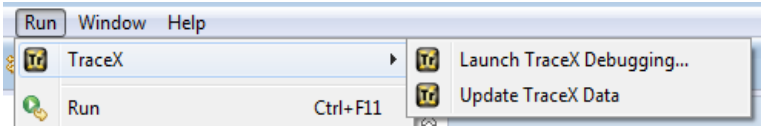

図 60: e2 studio ISDE TraceX 起動

4) [TraceX Debugging] ウィンドウで、[Buffer Start Address] を &g_tx_trace_buffer に設定します。

5) [TraceX Debugging] ウィンドウで、[Buffer Size (bytes)] を手順 1 のプロパティウィンドウで選択したバッファサイズに設定します。デフォルトは 65536 です。

図 61: e2 studio ISDE TraceX デバッグ

6) [OK] をクリックします。

TraceX の使用方法については、TraceX のユーザーズマニュアルを参照してください。

3.1.12 ツールチェーン設定の変更

使用するツールチェーンを変更することが必要な場合があります。(たとえば、コンパイラの最適化レベルを変更したり、リンカーにライブラリを追加する場合などです。) そのような変更を行うには、e2 studio ISDE で、プロジェクトが選択された状態でメニューから [Project] > [Renesas Tool Settings] を選択します。次のスクリーンショットは、GNU ARM ツールチェーンの設定ダイアログを示しています。このダイアログの外観は、使用するツールチェーンによって若干異なります。

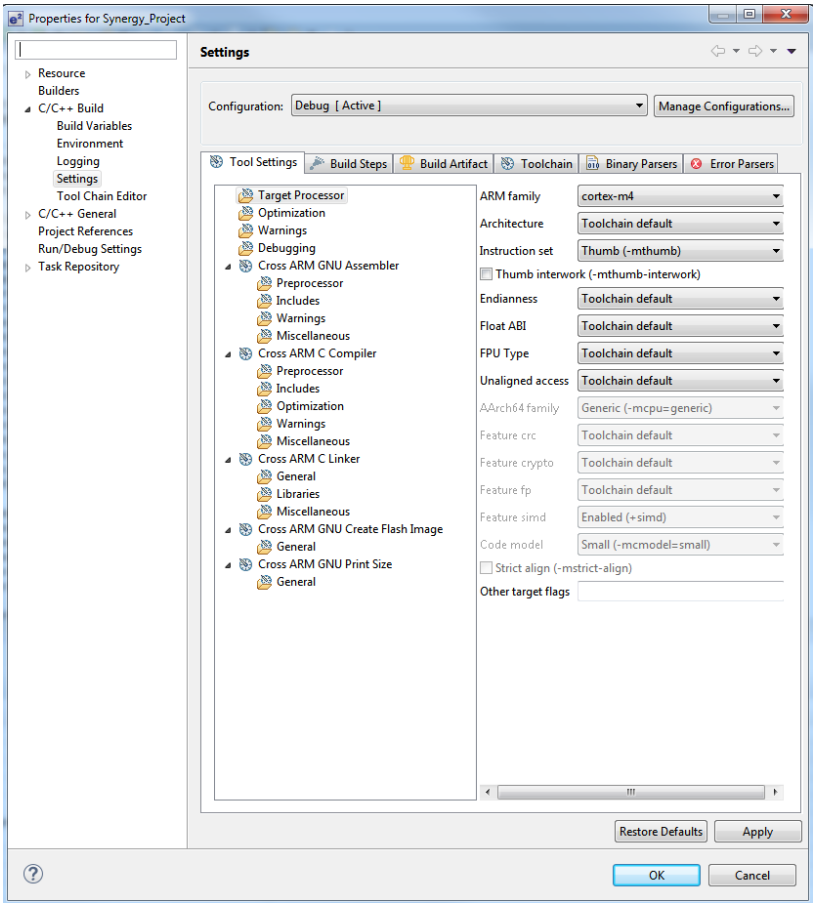

図 62: e2 studio ISDE プロジェクトツールチェーン設定

設定のスコープはプロジェクトのスコープで、変更対象のプロジェクトのみで有効です。

リンカーの、各種メモリセクションの場所を制御する設定は、使用するデバイスに固有のスクリプトファイルに格納されます。作成されたスクリプトファイルはプロジェクトに含まれ、スクリプトフォルダに格納されます（たとえば、/script/S7G2.ld）。

3.1.13 e2 studio ISDE 使用上の注意

3.1.13.1 SSP リリースの処理

SSP のいくつかのリリースパックが Synergy Gallery ウェブサイトで入手できます。

SSP パックには 2 種類あります。

1) X.Y.0 – 完全な SQA を含む生産リリース。X.Y.0 パックだけが含まれます。
2) X.Y.Z – リリースされた X.Y.0 バージョン（X= メジャー、Y= マイナー、Z= パッチ）に基づいたパッチリリース。X.Y.Z パックで追加または更新されたモジュールだけが含まれます（完全な X.Y.0 パックは含まれません）。

X.Y.0 生産リリースに加えて、複数の X.Y.Z パッチリリースをインストールすることができます。

たとえば、次の SSP バージョンをインストールすることができます :- SSP v1.0.0: 生産リリース - SSP v1.0.1: 最初のパッチリリース - SSP v1.0.2: 2 回目のパッチリリース - SSP v1.0.3: 3 回目のパッチリリース

パッチリリースにはボードサポートパッケージ（BSP）が含まれる場合も、含まれない場合もあります。Synergy Project Generator を使って新しい Synergy プロジェクトを作成する場合、[SSP Version] フィールドで BSP が付属する最新バージョンのパッチリリースを選択します。SSP のリリースに BSP が付属しているかどうかは、[Board] フィールドに選択できる BSP が表示されているかどうかで判断できます。選択したパッチリリースに BSP が含まれない場合（パッチリリース v0.91.3 など）、次のメッセージで示されます。

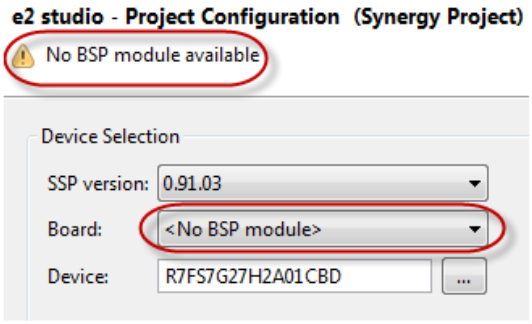

図 63: e² studio ISDE での複数の SSP バージョンのインストール

この場合、BSP を含む次に低い SSP バージョンを選択します。有効な BSP のある SSP バージョンを選択すると、e² studio ISDE はプロジェクトに含むことのできる最も高い SSP バージョンから SSP コンポーネントを自動で選択します。[Components] タブでモジュールをルックアップすることによって確認できます。

[Components] タブの SSP コンポーネントが選択した SSP バージョン（「パッチ」バージョンを含む）と正確に一致しない場合、[BSP] タブの [SSP Versions] フィールドの横に警告アイコンが表示されます。

次の例では、SSP バージョン 0.91.02 と v0.91.03 によってドライバモジュールが選択されています。この選択では、有効な BSP（0.91.02）が含まれますが、選択されたコンポーネントはより新しいパッチリリース（0.91.03）に属します。

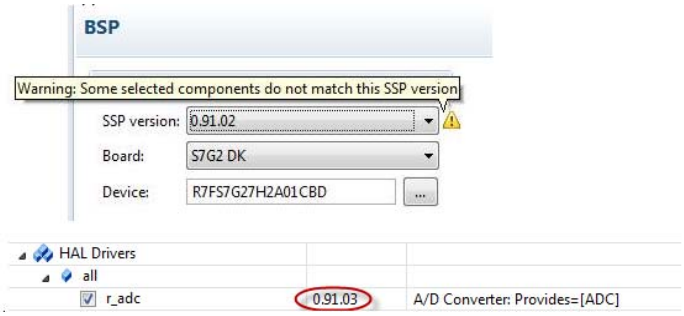

図 64: e² studio ISDE コンポーネントで複数の SSP バージョンをインストールすると不一致が発生する

Note: [Components] タブでは、同じコンポーネント（異なる SSP パックから）の複数のバージョンは選択しないでください。

3.1.13.2 ThreadX ソースを含む

以下のようにして、[Threads] タブを使用してプロジェクトに ThreadX ソースコードを含めることができます。

1) [Threads] ペインで HAL/Common アイコンをクリックします。[Stacks] ペインが [HAL/Common Stacks] に変わります。
2) [New] > [Framework] > [RTOS] > [ThreadX Source] の順に選択します。
3) TheadX RTOS プロパティを設定するには、[Properties] ウィンドウを確認してください。
4) [Generate Project Content] をクリックします。

e² studio ISDE により、ThreadX ソースコードが次のディレクトリに抽出されます。

Note: 適切なライセンスがある場合のみ、ThreadX ソースコードの表示と確認を行うことができます。ライセンス設定を参照してください。

Note: ThreadX ソースを抽出すると、プロジェクトのコンパイル時間は増加します。

3.2 チュートリアル : Your First Synergy Project - Blinky

このチュートリアルの目的は、e^2 studio ISDE を使用した単純なアプリケーションの作成と、そのアプリケーションの Synergy ボード上での実行手順を通じて、Synergy プラットフォームを学習することです。

3.2.1 Blinky の機能

このチュートリアルで使用するアプリケーションの Blinky は、新しい組み込み開発環境で伝統的に最初に実行するプログラムです。

Blinky は、マイクロコントローラの「Hello World」に相当します。LED が点滅することで、以下のことがわかります。

- ツールチェーンが正しく設定され、使用するチップ用の正常に動作する実行可能イメージがビルドされること。
- デバッガーがインストールされ、ドライバが動作し、ボードに適切に接続されていること。
- ボードの電源がオンになり、そのジャンパーとスイッチの設定がおそらく正しいこと。
- マイクロコントローラが作動し、クロックが動作し、メモリが初期化されていること。

このチュートリアルで使用するサンプルアプリケーション Blinky は、Synergy マイクロコントローラが搭載されている、Renesas 製のすべてのボードで同じように動作するよう設計されています。動作している特定のボード用の BSP（ボードサポートパッケージ）を呼び出して処理を実行します。これは以下の理由で正常に機能します。

- 1 つ以上の LED が GPIO ピンに接続されている。
- その 1 つの LED は、常にボード上で LED1 と明記されている。
- すべての BSP は、ボード上の LED、そのポート、ピン割り当てのリストを返す API をサポートしている。

3.2.2 前提条件

このチュートリアルを実行するには、以下のものが必要です。

- Windows ベースの PC
- e^2 studio ISDE
- SSP
- Synergy ボードキット

3.2.3 Blinky 用の新規プロジェクトの作成

Synergy プロジェクトの作成と設定は、アプリケーションを作成するための最初のステップです。ベース SSP パックには、シンプルですべての Renesas Synergy ボードで動作する、作成済みの Blinky サンプルアプリケーションが含まれています。

Synergy プロジェクトを作成するには以下の手順に従います。

1) e^2 studio ISDE で、[File] > [New] > [Synergy Project] の順にクリックします。
2) この新しいプロジェクトに名前を付けます。このチュートリアルでは Blinky という名前を使用します。
3) ライセンスウィンドウが空白の場合はライセンスファイルを特定します。このバージョンのプラットフォームのライセンスファイルは、<e^2 studio ISDE ベースディレクトリ > 内の \internal\projectgen\arm\Licenses\SSP_License_Example_EvalLicense_<rev>.xml にあります。
4) [Next] をクリックします。[Project Configuration] ウィンドウに選択内容が表示されます。

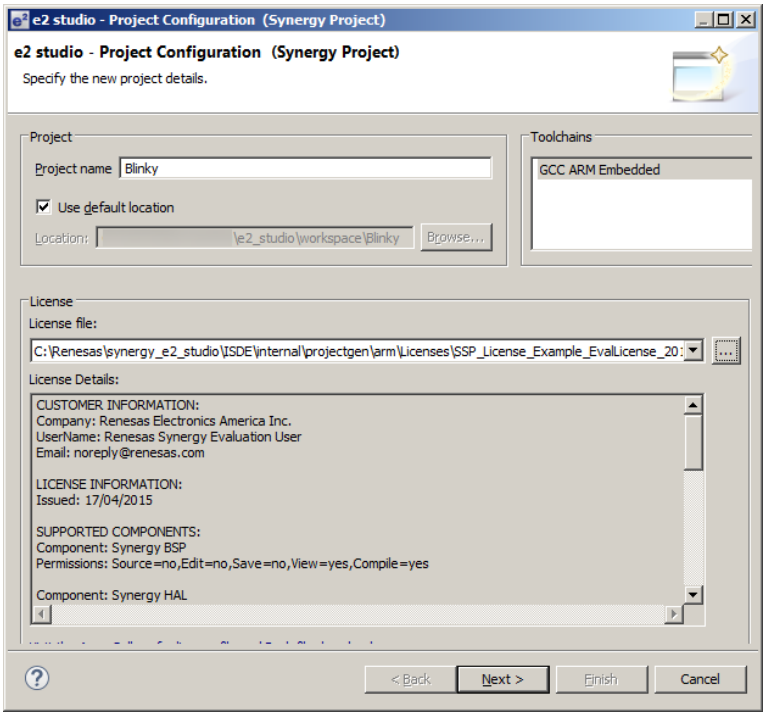

図 65: e^2 studio ISDE の [Project Configuration] ウィンドウ（パート 1）

5) ボードサポートパッケージを選択するため、[Board] ドロップダウンリストから使用するボード名を選択し、[Next] をクリックします。

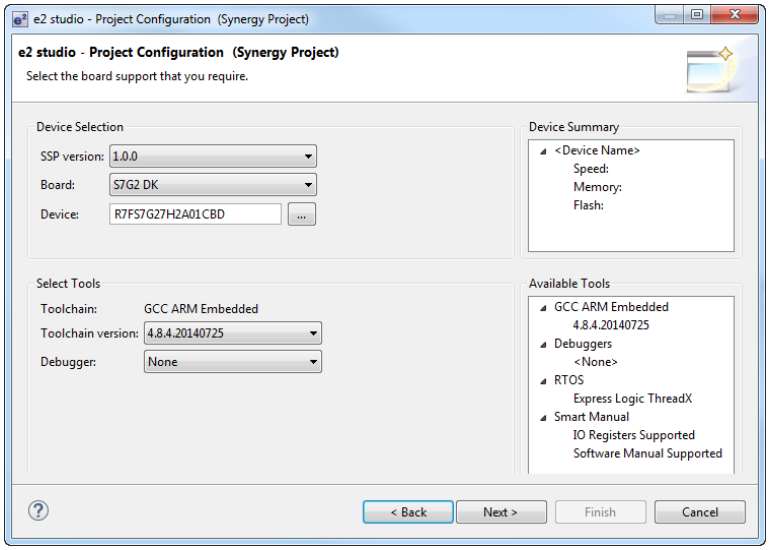

図 66: e^2 studio ISDE の [Project Configuration] ウィンドウ（パート 2）

6) 使用するボード用の Blinky テンプレートを選択し、[Finish] をクリックします。

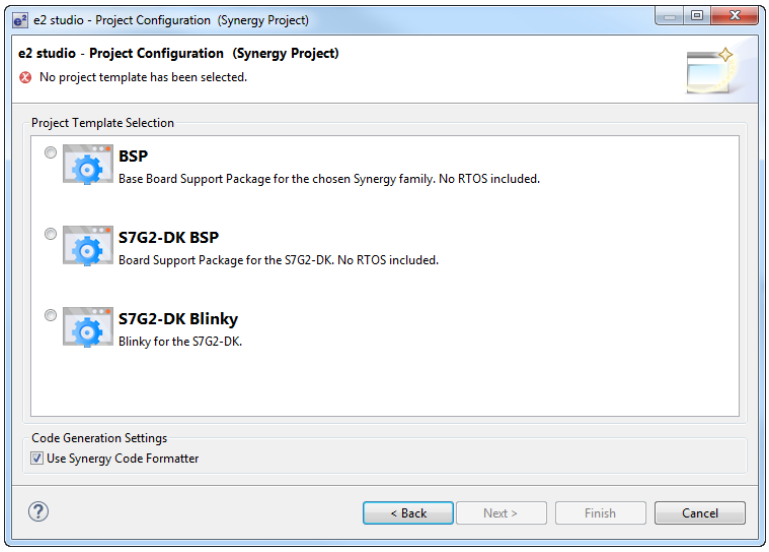

図 67: e^2 studio ISDE の [Project Configuration] ウィンドウ（パート 3）

プロジェクトが作成されると、プロジェクトの名前が e^2 studio ISDE の [Project Explorer] ウィンドウに表示されます。[Project Configuration] ウィンドウの右上隅にある [Generate Project Content] ボタンを押し、ボード固有のファイルを生成します。

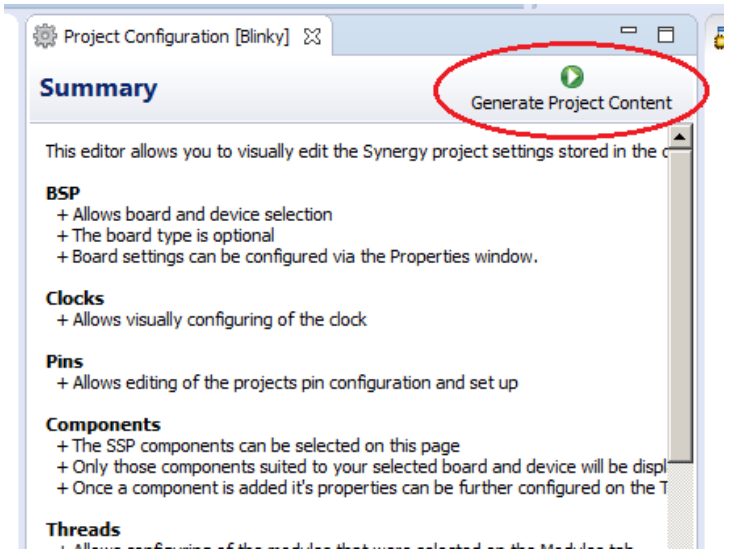

図 68: e² studio ISDE の [Project Configuration] タブ

これで新しいプロジェクトが作成、設定され、ビルドの準備ができました。

3.2.3.1 Blinky の設定に関する詳細

[Generate Project Content] ボタンは、構成ヘッダーファイルを作成し、テンプレートからソースファイルをコピーして、通常は [Project Configuration] 画面の状態に基づいてプロジェクトを構成します。

たとえば、[Components] タブのモジュールの横にあるチェックボックスをオンにし、[Generate Project Content] ボタンを押すと、そのモジュールをプロジェクトに追加するために必要なすべてのファイルがコピーまたは作成されます。その同じチェックボックスをオフにすると、それらのファイルが削除されます。

3.2.3.2 Blinky のクロックの設定

Blinky テンプレートを選択することで、e² studio ISDE により Blinky アプリケーション用にクロックが設定されます。Blinky のクロック構成は、e² studio ISDE の [Project Configuration] 画面の [Clocks] タブに表示されます（クロックの設定を参照）。Blinky のクロック構成は、BSP クロック構成ファイルに保存されます（BSP のクロック設定を参照）。

3.2.3.3 Blinky のピン設定

Blinky テンプレートを選択することで、LED1 を切り替えるために使用する GPIO ピンが、e² studio ISDE により Blinky アプリケーション用に設定されます。e² studio ISDE の [Project Configuration] 画面の [Pins] タブには、Blinky アプリケーション用のピン構成が表示されます（ピンの設定を参照）。Blinky のピン構成は、BSP 構成ファイルに保存されます（BSP のピン設定を参照）。

3.2.3.4 Blinky コンポーネント用のパラメータの設定

Blinky プロジェクトは、e^2 studio ISDE コンポーネント内の次の HAL コンポーネントを自動的に選択します。

- r_cgc
- r_elc
- r_ioport

いずれかのコンポーネントの構成パラメータを表示するには、それぞれのドライバの [HAL] ウィンドウの [Properties] タブを確認します（HAL ドライバの追加と設定を参照）。

3.2.3.5 main() の場所

main 関数は < プロジェクト >/src/synergy_gen/main.c にあります。これは、プロジェクト作成段階で生成されるファイルの 1 つであり、hal_entry() の呼び出しのみを含んでいます。生成されるファイルの詳細については、HAL ドライバの追加と設定を参照してください。

3.2.3.6 Blinky のサンプルコード

Blinky アプリケーションは、hal_entry.c ファイルに格納されます。このファイルは、Blinky プロジェクトテンプレートを選択したときに e^2 studio ISDE によって生成され、プロジェクトの src/ フォルダ内に配置されます。

アプリケーションは次の手順を実行します。

1) BSP HAL 関数を呼び出すことにより、選択されたボードの LED 情報を取得します（R_BSP_LedsGet）。
2) 選択されたボードの LED を制御する GPIO ピンの出力レベル HIGH を定義します。
3) 選択されたシステムクロック速度を取得し、LED のトグルを観察できるようにクロックをスケールダウンします。
4) 次のものを使用して GPIO ピンに書き込むことにより、LED をトグルします。

```
g_ioport.p_api->pinWrite()
```

3.2.4 Blinky プロジェクトのビルド

新しいプロジェクトを [Project Explorer] ウィンドウでハイライトし、ビルドします。

プロジェクトをビルドするには、以下の 3 つの方法があります。

a. メニューバーの [Project] をクリックし、[Build Project] を選択します。

b. ハンマーアイコンをクリックします。

c. プロジェクトを右クリックし、[Build Project] を選択します。

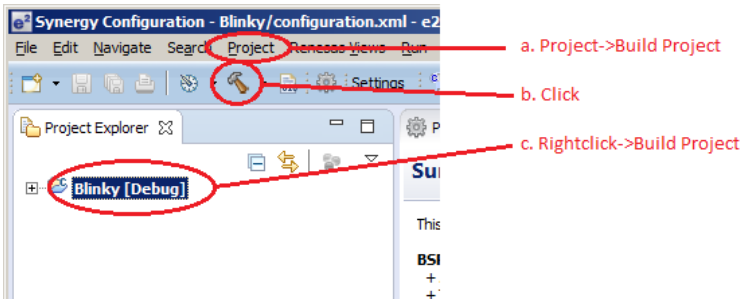

図 69: e² studio ISDE の [Project Explorer] ウィンドウ

ビルドが完了すると、メッセージがビルドの [Console] ウィンドウに表示され、最終的なイメージファイル名と、そのイメージ内のセクションサイズが表示されます。

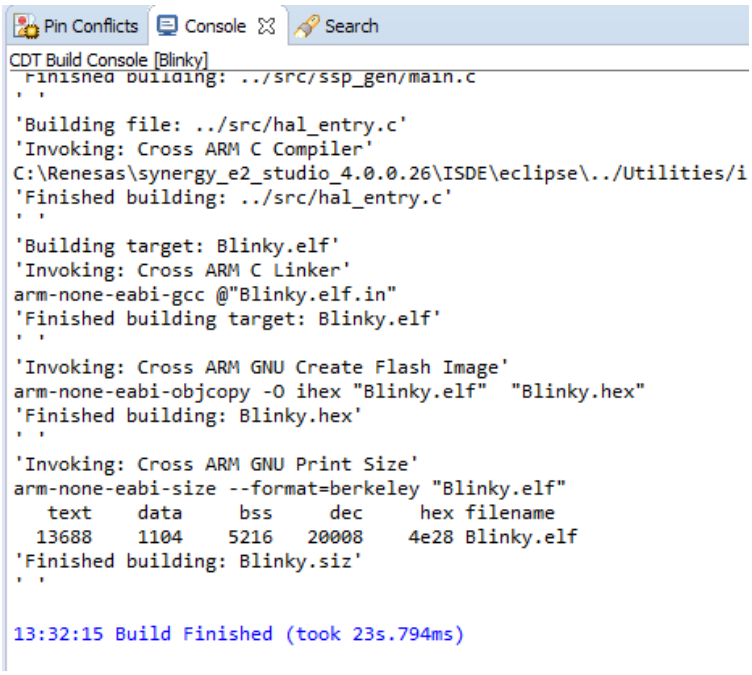

図 70: e² studio ISDE のプロジェクトビルドコンソール

3.2.5 Blinky プロジェクトのデバッグ

3.2.5.1 デバッグの前提条件

ボード上でプロジェクトをデバッグするには、以下のものが必要です。

- e² studio ISDE に接続するボード
- ボードと通信するように設定されるデバッガー

- マイクロコントローラにプログラムされるアプリケーション

アプリケーションは、マイクロコントローラの内蔵フラッシュから実行します。アプリケーションを実行またはデバッグするには、まずアプリケーションがマイクロコントローラのフラッシュにプログラムされている必要があります。そのための方法としては、以下の 2 つの方法があります。

- JTAG デバッガー
- UART または USB を通じた組み込みブートローダー

一部のボードにはオンボード JTAG デバッガーが搭載されていますが、他のボードでは、ボード上のヘッダーに接続された外部 JTAG デバッガーが必要です。

ボードのユーザーマニュアルで、JTAG デバッガーを e^2 studio ISDE に接続する方法を確認してください。

3.2.5.2 デバッグ手順

Blinky アプリケーションをデバッグするには、以下の手順を実行します。

1) プロジェクト用にデバッガを設定するため、[Run] > [Debugger Configuration] をクリックします。

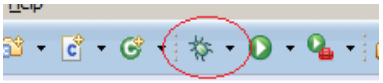

図 71: e^2 studio ISDE のデバッグアイコン

または、バグアイコンの横にあるドロップダウンメニューで [Debugger Configuration] を選択します。

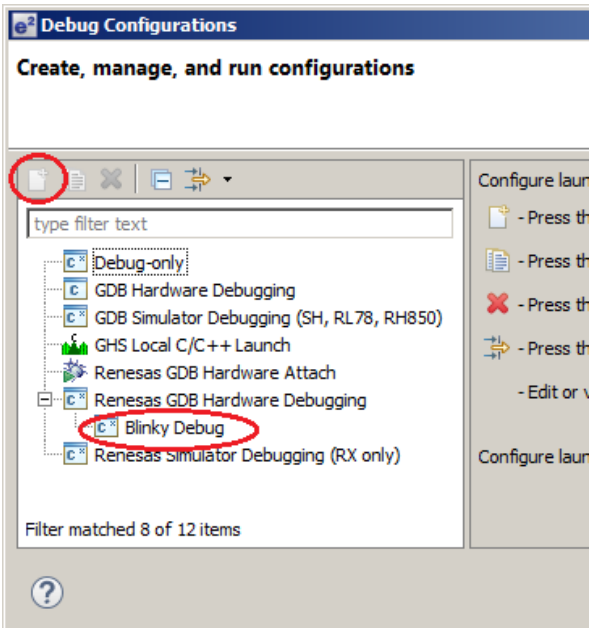

図 72: e^2 studio ISDE の [Debugger Configuration] ウィンドウ

2) ウィンドウでデバッガー設定を選択します。デバッガー設定が表示されていない場合は、ウィンドウの左上隅にある [New] アイコンをクリックして作成する必要があります。選択すると、[Debug Configuration] ウィンドウに、Blinky プロジェクトのデバッグ設定が表示されます。

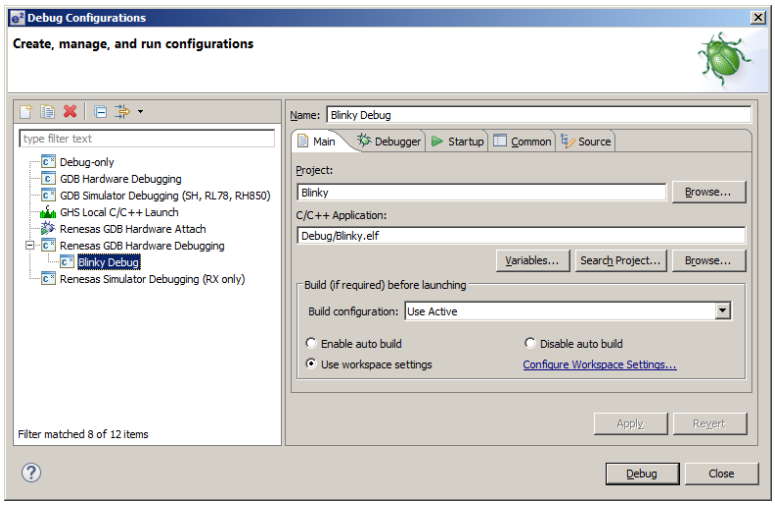

図 73: e^2 studio ISDE の [Debugger Configuration] ウィンドウと Blinky プロジェクト

3) [Debug] を押してアプリケーションのデバッグを開始します。

3.2.5.3 デバッグ手順の詳細

デバッグモードでは、e^2 studio ISDE は次のタスクを実行します。

1) アプリケーションイメージをマイクロコントローラにダウンロードし、イメージを内蔵フラッシュメモリにプログラミングします。
2) main() にブレークポイントを設定します。
3) スタックにスタックポインタレジスタを設定します。
4) プログラムカウンタレジスタに、リセットベクトルのアドレスをロードします。
5) プログラムカウンタが指しているスタートアップコードを表示します。

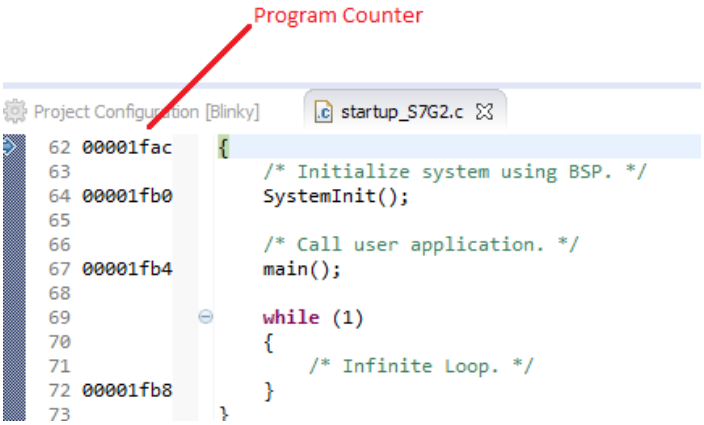

図 74: **e^2 studio ISDE のデバッガメモリウィンドウ**

3.2.6 Blinky プロジェクトの実行

デバッグモードで、[Run] > [Resume] をクリックするか、[Play] アイコンを 2 回クリックします。

図 75: e^2 studio ISDE デバッガの [Play] アイコン

ボード上の LED1 とマークされた LED が点滅します。

3.3 チュートリアル : Using HAL Drivers - Programming the WDT

このアプリケーションは、WDT HAL ドライバの WDT によって実装されている WDT インタフェースを使用します。このドキュメントでは、e^2 studio ISDE と SSP を使用して、MCU のウォッチドッグタイマ（WDT）周辺デバイス用のアプリケーションを作成する方法について説明します。このアプリケーションは、以下の SSP モジュールを利用します。

- Board Support Package（ボードサポートパッケージ）
- CGC（クロック発生回路）
- WDT（ウォッチドッグタイマ）
- IOPORT（GPIO）

3.3.1 SSP と e^2 studio ISDE を使用した WDT アプリケーションの作成

3.3.1.1 SSP および e^2 studio ISDE の使用方法

Renesas 製の SSP は、Synergy アプリケーションを開発するためのドライバライブラリ一式を提供します。SSP は、ハードウェア抽象化レイヤー（HAL）ドライバ、ボードサポートパッケージ（BSP）ドライバ、および開発者がアプリケーションを作成するために使用する上位のフレームワークアプリケーションを提供します。SSP は、eclipse ベースの e^2 studio 統合ソリューション開発環境（e^2 studio ISDE）に統合されており、ビルド（エディター、コンパイラ、リンカー）およびデバッグフェーズと、拡張された GNU Debug（GDB）インタフェースを提供します。

3.3.1.2 WDT アプリケーション

WDT アプリケーションのフローチャートを以下に示します。

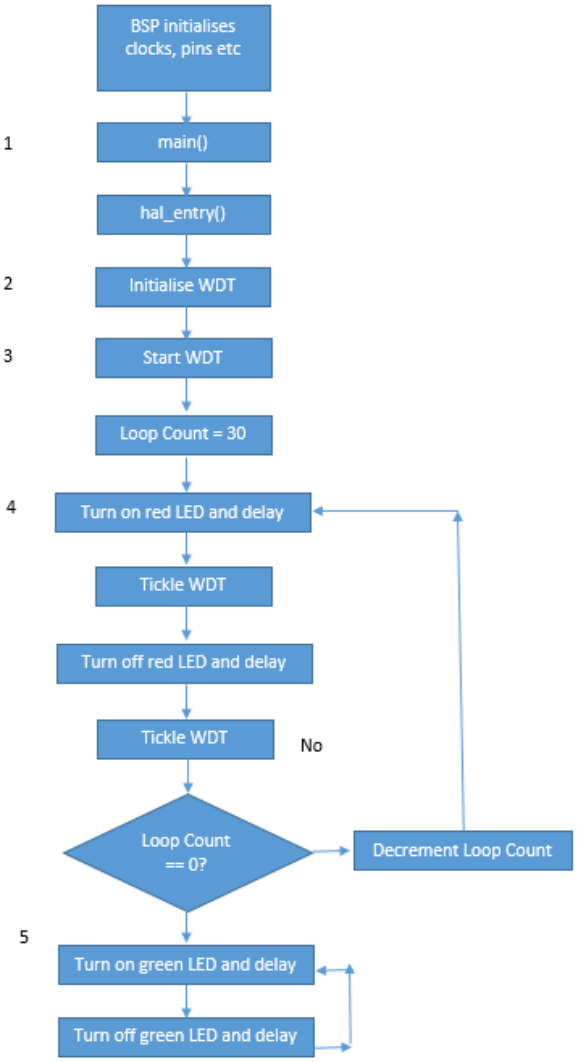

図 76: WDT アプリケーションのフロー図

3.3.1.3 WDT アプリケーションのフロー

WDT アプリケーションの主な部分は以下のとおりです。

1) main() が hal_entry() を呼び出します。関数 hal_entry() は SSP によって作成され、ユーザーコードのプレースホルダが含まれます。WDT 用のコードをこの関数に追加します。
2) WDT を初期化します。ただし、まだ開始しません。
3) WDT をリフレッシュすることによって、WDT を開始します。
4) 赤い LED が 30 回点滅し、LED の状態が変化するたびにウォッチドッグがリフレッシュされます。

5) 緑色の LED が点滅しますが、ウォッチドッグはリフレッシュされません。ウォッチドッグのタイムアウト期間の後、デバイスがリセットされます。その様子は、シーケンスが繰り返されるときに赤い LED が再度点滅することで確認できます。

3.3.2 e² studio ISDE でのプロジェクトの作成

e² studio ISDE を起動し、以下のようにして新しい Synergy プロジェクトを設定します。

1) [File] > [New] > [Synergy Project] の順に選択します。

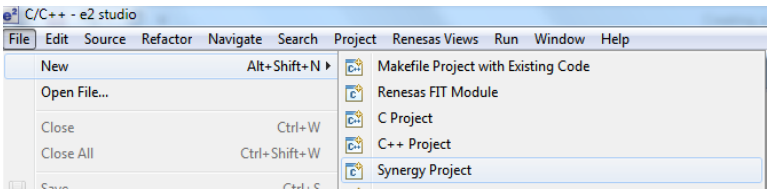

図 77:　新規プロジェクトの作成

2) e² studio ISDE プロジェクトの [Project Configuration] ウィンドウで、プロジェクト名（例 :WDT_Application）を入力します。また、ツールチェーンとライセンスファイルを選択します。[Next] をクリックします。

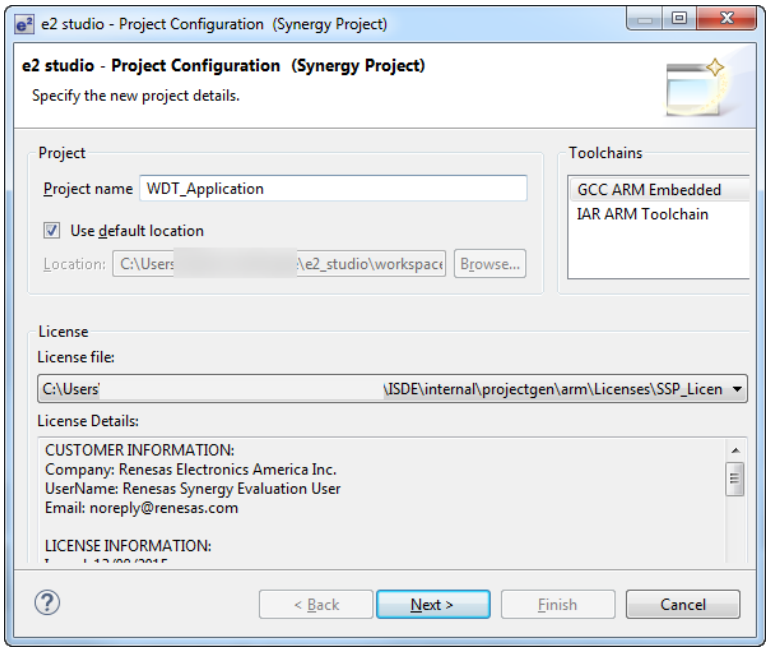

図 78:　Project Configuration（パート 1）

3) このアプリケーションは、Synergy S7G2 ベースの DK-S7G2 ボードで動作します。そのため、[Board] には [S7G2 DK] を選択します。これにより、[Device] ドロップダウンに、このボードで使用

される正しいデイバイスが自動的に設定されます。ツールチェーンのバージョンを選択します。J-Link ARM をデバッガとして選択します。このアプリケーションでは RTOS は使用されていませんが、デフォルトの Express Logic ThreadX のままで構いません。[Next] をクリックしてプロジェクトを構成します。

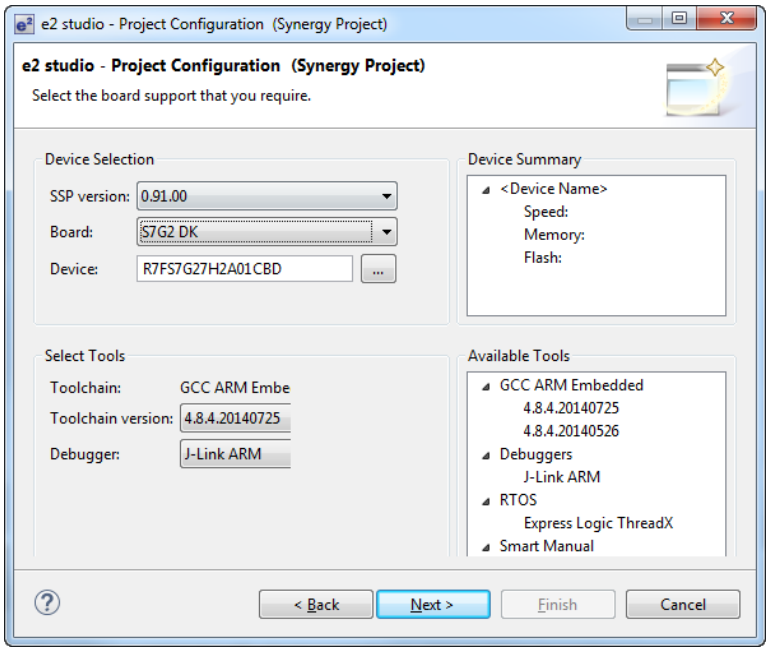

図 79: Project Configuration（パート 2）

次にプロジェクトテンプレートを選択します。RTOS は必要ないため、[S7G2-DK BSP] を選択します。

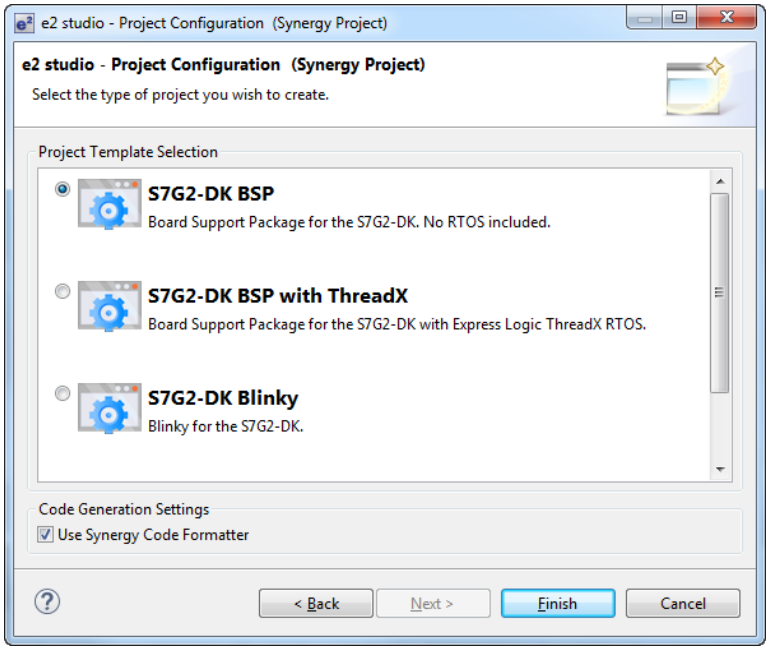

図 80: Project Configuration（パート 3）

4) [Finish] をクリックします。

プロジェクトが作成されて、プロジェクト構成の概要が示された [Project Explorer] ビューと [Summary] タブの表示が開きます。

3.3.3 e² studio ISDE でのプロジェクトの設定

e² studio ISDE は、プロジェクトを設定するためのオプションを選択する GUI インタフェースを備えているため、プロジェクト構成手順が簡略化され、時間も短縮されます。

e² studio ISDE には、進行中の操作に応じて各種のウィンドウを表示する、いくつかのパースペクティブが備わっています。デフォルトのパースペクティブは、[C/C++]、[Synergy Configuration]、[Debug] です。パースペクティブを変更するには、e² studio ISDE の右上にあるボタンから新しいパースペクティブを選択します。

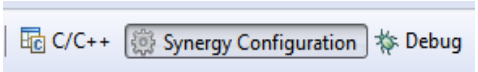

図 81: パースペクティブの選択

C/C++ パースペクティブは、コード編集用に選択されたレイアウトを提供します。[Synergy Configuration] パースペクティブには Synergy プロジェクトを設定するための要素があり、[Debug] パースペクティブはデバッグに適したビューを備えています。

1) プロジェクトを設定するには、[Synergy Configuration] パースペクティブが選択されていることを確認します。
2) プロジェクト構成 [WDT Application] が開かれていることを確認します。要約情報が表示されている場合はすでに開かれています。ここで、または任意のタイミングで [Project Configuration] を開くには、[Synergy Configuration] パースペクティブが選択されていることを確認し、e^2 studio ISDE の右側にある [Project Explorer] ペインで configuration.xml ファイルをダブルクリックします。

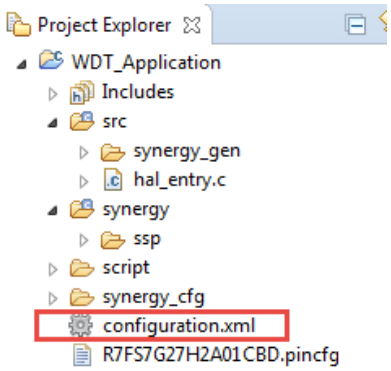

図 82: Synergy プロジェクト設定

[Project Configuration] ビューの下部には、プロジェクトを設定するためのタブがいくつかあります。プロジェクトでは、これらのタブの一部または全部に対する変更が必要になる可能性があります。これらのタブについて以下に説明します。

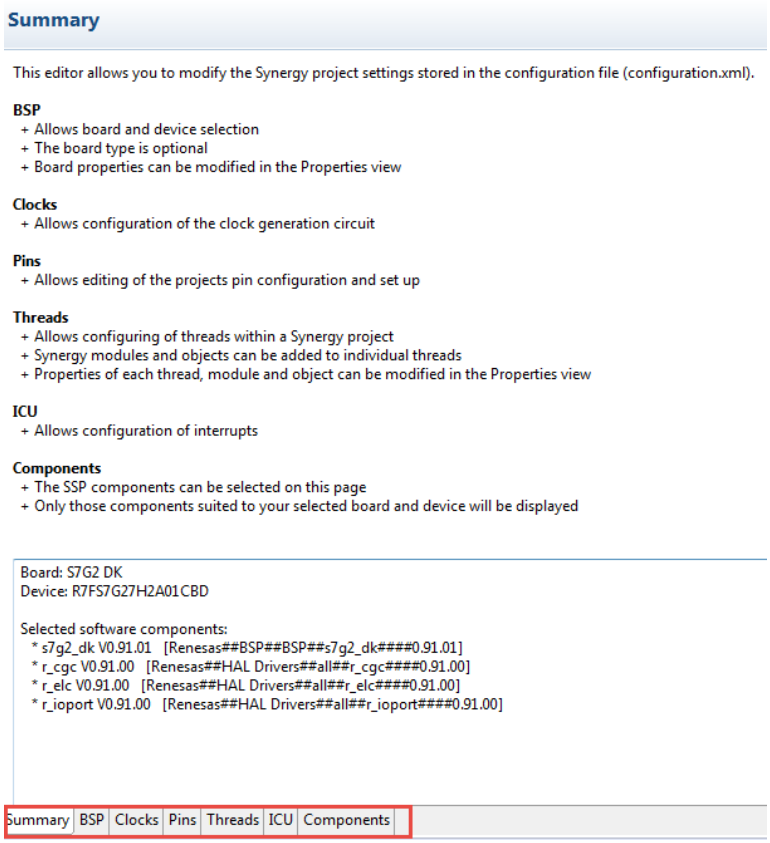

図 83:　　プロジェクト設定タブ

3.3.3.1 BSP タブ

[BSP] タブでは、ボードサポートパッケージ（BSP）オプションをデフォルト値から変更できます。この特定の WDT プロジェクトについては、変更は不要です。ただし、WDT をオートスタートモードで使用する場合は、[BSP] タブで OFS0（オプション機能選択レジスタ 0）レジスタを設定できます。WDT のオートスタートモードの詳細については、MCU ユーザーズマニュアルを参照してください。

3.3.3.2 [Clocks] タブ

[Clocks] タブには、デバイスのクロックツリーがグラフィカルに表示されます。GUI のドロップダウンボックスを使用して、各種クロックを設定できます。WDT は PCLCKB を使用します。このクロックのデフォルト出力周波数は 60 MHz です。クロックがこの値を出力していることを確認してください。

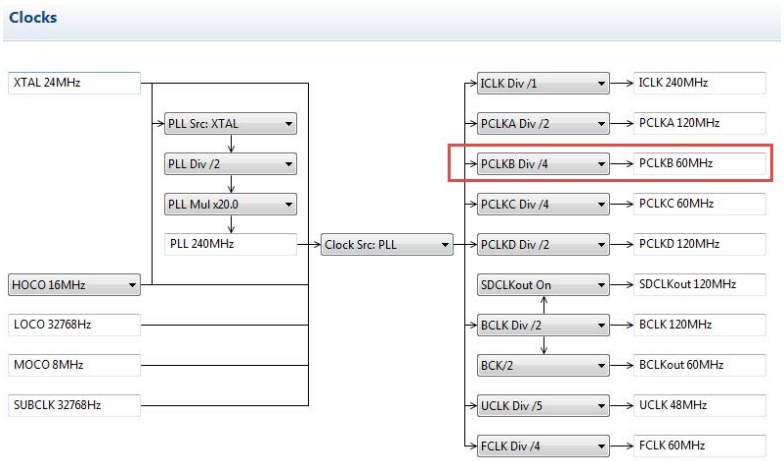

図 84:　　クロック設定

3.3.3.3 [Pins] タブ

[Pins] タブでは、デバイスのピンの機能を設定するためのグラフィカルツールを利用できます。WDT プロジェクトでは、ピンの設定は不要です。プロジェクトではデバイス上のピンに接続された 2 個の LED を使用しますが、これらのピンは出力 GPIO ピンとして BSP によりあらかじめ設定されています。

3.3.3.4 [Threads] タブ

[Threads] タブでは、任意のドライバをプロジェクトに追加できます。クロック発生回路、イベントリンクコントローラ、および IO ポートピン用の HAL ドライバは、プロジェクトの設定時に e^2 studio ISDE によって自動的に追加されます。WDT アプリケーションは ThreadX リソースを使用しないため、追加するのは HAL WDT ドライバだけにしてください。

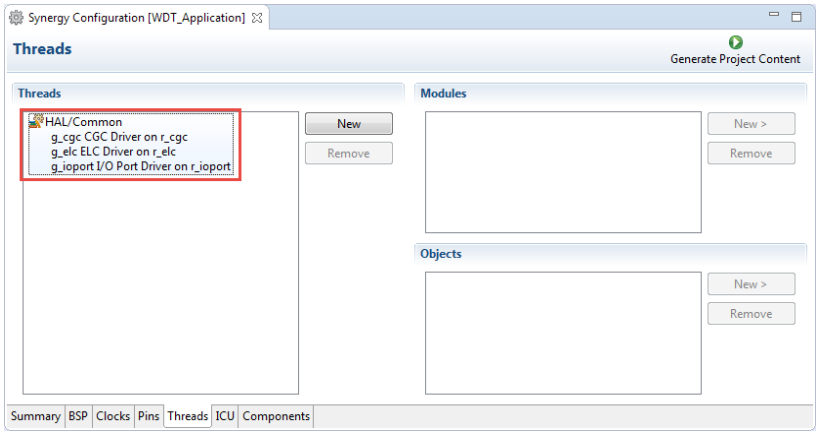

図 85: [Threads] タブ

1) 上の図に示すように、[Threads] ウィンドウで [HAL/Common] パネルをクリックします。[Stacks] パネルが [HAL/Common Stacks] パネルになり、e^2 studio ISDE によってあらかじめ選択されたモジュールがパネルに表示されます。
2) [New Stack]> をクリックして、使用可能な HAL レベルドライバを含むポップアップウィンドウを表示します。
3) WDT の WATCGDOG ドライバを選択します。

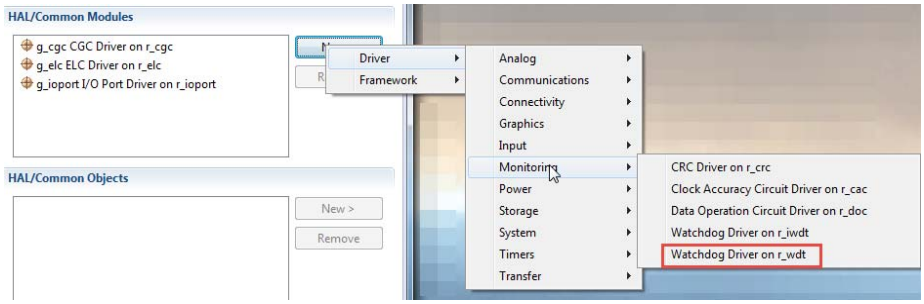

図 86: モジュールの選択

選択した HAL WDT ドライバが [HAL/Common Stacks] パネルに追加され、選択されたモジュールのすべての構成オプションが [Properties] ウィンドウに表示されます。WDT の [Properties] タブが画面の左下に表示されます。表示されない場合は、[Synergy Configuration] パースペクティブが選択されていることを確認します。

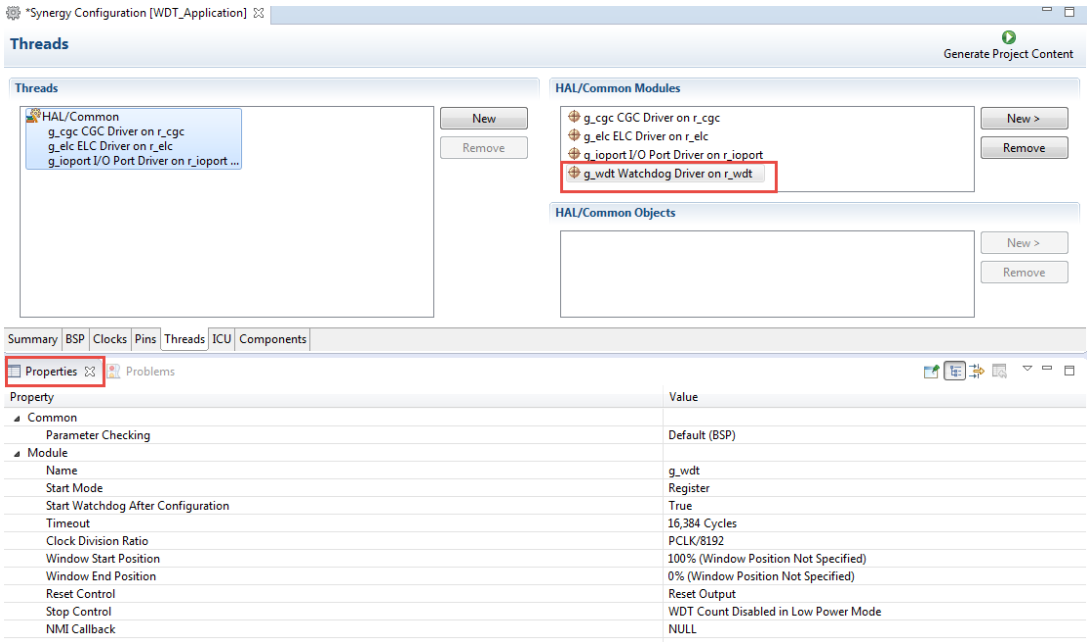

図 87:　　モジュールのプロパティ

パラメータ [Start Watchdog After Configuration] を [True] から [False] に変更します。他のパラメータはデフォルト値のままで構いません。[Start Watchdog After Configuration] を [False] に設定すると、WDT ドライバに対し（その open API 呼び出しを通じて）、WDT を設定するものの、開始しないように指示します。後で更新することで開始されます。

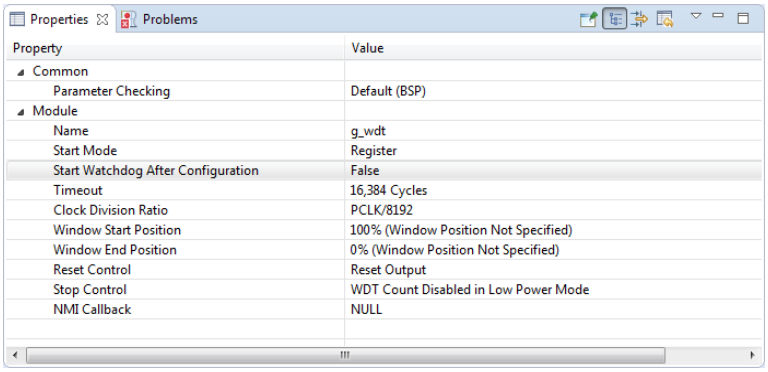

図 88:　　WDT プロパティ上の g_wdt ウォッチドッグドライバ

PCLKB が 60 MHz で動作している状態で、WDT は最後の更新から 2.23 秒後にデバイスをリセットします。

WDT クロック = 60 MHz/8192 = 7.32 kHz

サイクル時間 = 1/7.324 kHz = 136.53 マイクロ秒

タイムアウト = 136.53 マイクロ秒 x 16384 = 2.23 秒

[Project Configuration] ファイルを保存して、[Project Configuration] ペインの右上隅にある [Generate Project Content] ボタンをクリックします。

図 89: [Generate Project Content] ボタン

e^2 studio ISDE はプロジェクトファイルを生成します。

3.3.3.5 [Components] タブ

[Components] タブは参照用に存在し、ここでプロジェクトに含まれているモジュールを確認できます。[Threads] タブでモジュールを追加すると、それらのモジュールが [Components] ビューで自動的に選択されます。

WDT プロジェクトでは、以下のモジュールが選択されていることを確認してください。

1) HAL_Drivers -> r_cgc
2) HAL_Drivers -> r_elc
3) HAL_Drivers -> r_ioport
4) HAL_Drivers -> r_wdt

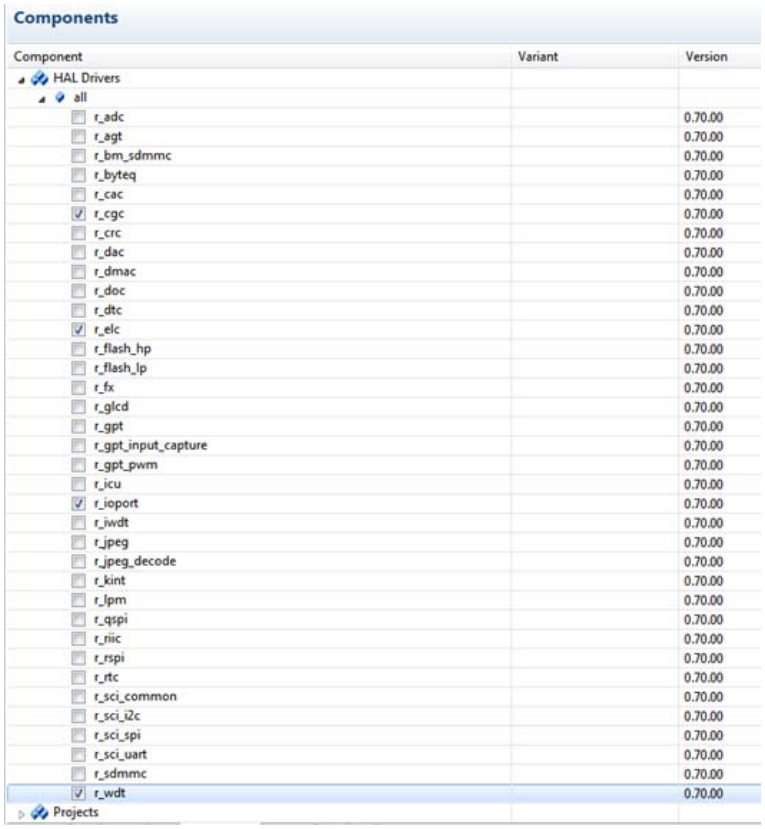

図 90: コンポーネント選択

Note: [Components] タブに表示されるモジュールの一覧は、インストールされている SSP のバージョンによって異なります。

3.3.4 WDT の生成されるプロジェクトファイル

[Generate Project Content] ボタンを押すと以下のタスクが実行されます。

- r_wdt フォルダと WDT ドライバの内容が次の場所に作成されます。
  - synergy/ssp/src/driver/
- r_wdt_api.h が次の場所に作成されます。
  - synergy/ssp/inc/api
- r_wdt.h が次の場所に作成されます。
  - synergy/ssp/inc/driver/instances

上記のファイルは、WDT HAL モジュールの標準的なファイルです。これらのファイルには、プロジェクト固有の内容は含まれていません。これは WDT のドライバファイルです。これらのファイルの内容の詳細については、WDT HAL モジュールのドキュメントを参照してください。

WDT プロジェクトでの WDT HAL モジュールの設定情報は次の場所にあります。

- synergy_cfg/ssp_cfg/driver/r_wdt_cfg.h

上記のファイルの内容は、[g_wdt Watchdog Driver on WDT Properties] ペインの [Common] 設定に基づいています。

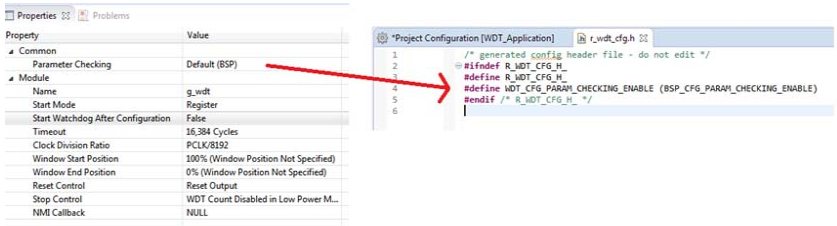

図 91: r_wdt_cfg.h の内容

Attention: これらのファイルは、[Generate Project Content] ボタンを押すたびに再生成され、変更内容が上書きされるため、編集しないでください。

r_ioport フォルダは、synergy/ssp/src/driver には作成されません。このモジュールは BSP に必要なため、すでに存在するからです。これは、正しいヘッダーファイルを src/synergy_gen/hal_data.h でインクルードするために、WDT プロジェクトに含まれています。詳細については、このドキュメントで後述します。同じ理由により、他の IOPORT ヘッダーファイル（synergy/ssp/inc/driver/api/r_ioport_api.h と synergy/ssp/inc/instances/r_ioport.h）も、すでに存在するため、作成されません。

e^2 studio ISDE は、WDT 用の HAL ドライバファイルと IOPORT ファイルを生成するのに加えて、WDT 用の設定データが格納されたファイルと、ユーザーコードを安全に追加できるファイルも生成します。これらのファイルを以下に示します。

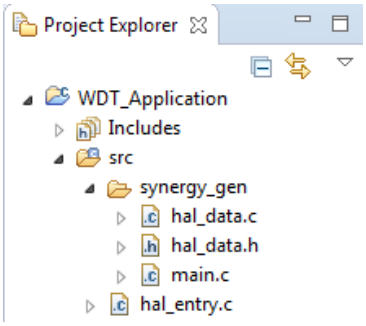

図 92: WDT プロジェクトファイル

3.3.4.1 これは、正しいヘッダーファイルを src/synergy_gen/hal_data.h でインクルードするために、WDT プロジェクトに含まれています。

hal_data.h の内容は以下のとおりです。

```
/* generated HAL header file - do not edit */

#ifndef HAL_DATA_H_

#define HAL_DATA_H_

#include <stdint.h>

#include "bsp_api.h"

#include "r_wdt.h"

#include "r_wdt_api.h"

#include "r_ioport.h"

#include "r_ioport_api.h"

#include "r_elc.h"

#include "r_elc_api.h"

#include "r_cgc.h"

#include "r_cgc_api.h"

extern const wdt_instance_t g_wdt0.

#ifdef NULL
#define WATCHDOG_ON_WDT_CALLBACK_USED (0)

#else

#define WATCHDOG_ON_WDT_CALLBACK_USED (1)

#endif

#if WATCHDOG_ON_WDT_CALLBACK_USED

void NULL(wdt_user_cb_data_t * p_args);

#endif

extern const ioport_instance_t g_ioport;

extern const elc_instance_t g_elc;

#endif /* HAL_DATA_H_ */
```

hal_data.h には、e2 studio ISDE で生成されたプロジェクトに必要なヘッダーファイルが格納されています。また、WDT HAL ドライバに使用される構成、制御、API の各構造体へのポインタを含む g_wdt インスタンス構造体への外部参照も含まれます。

Attention: このファイルは、[Generate Project Content] ボタンを押すたびに再生成されるため、編集しないでください。

3.3.4.2 WDT プロジェクトファイル

hal_data.c の内容は以下のとおりです。

```
/* generated HAL source file - do not edit */

#include "hal_data.h"

#if WATCHDOG_ON_WDT_CALLBACK_USED

#if defined(__ICCARM__)

#define NULL_WEAK_ATTRIBUTE

#pragma weak NULL = NULL_internal

#elif defined(__GNUC__)

#define NULL_WEAK_ATTRIBUTE __attribute__ ((weak,
alias("NULL_internal")))

#endif

void NULL(wdt_user_cb_data_t * p_args) NULL_WEAK_ATTRIBUTE;

#endif

static wdt_ctrl_t  g_wdt0.ctrl;

static const wdt_cfg_t  g_wdt0.cfg =

{

    .start_mode = WDT_START_MODE_REGISTER,

   .autostart = false,

    .timeout = WDT_TIMEOUT_16384,

    .clock_division = WDT_CLOCK_DIVISION_8192,
```

```
    .window_start = WDT_WINDOW_START_100,

    .window_end = WDT_WINDOW_END_0,

    .reset_control = WDT_RESET_CONTROL_RESET,

    .stop_control = WDT_STOP_CONTROL_DISABLE,

    .p_callback = NULL,

};

/* Instance structure to use this module. */

const wdt_instance_t  g_wdt0 =

{

    .p_ctrl  = &g_wdt0_ctrl,

    .p_cfg = &g_wdt0_cfg,

    .p_api = &g_wdt0_on_wdt

};

#if WATCHDOG_ON_WDT_CALLBACK_USED

/* This is a weak example callback function. It should be */
/* overridden by defining a user callback function  */
/* with the prototype below.  */

/* - void NULL(wdt_user_cb_data_t * p_args) */

/* parameter p_args Callback arguments used to identify what caused the callback. */

void NULL_internal(wdt_user_cb_data_t * p_args)

{

}

#endif

const ioport_instance_t  g_ioport =

{
```

```
    .p_api  = &g_ioport_on_ioport,

    .p_cfg = NULL

};

const elc_instance_t  g_elc =

{

    .p_api  = &g_elc_on_elc,

    .p_cfg = NULL

};

void g_hal_init(void) {

}
```

hal_data.c には、WDT HAL ドライバのこのインスタンス用の制御構造体である g_wdt_ctrl が含まれています。この構造体は、ドライバがオープンされたときに初期化されるため、初期化しないでください。

g_wdt_cfg の内容は、[Synergy Configuration] の [Threads] タブの [g_wdt0 Watchdog Driver on r_wdt] ペインを使用してこのファイルに設定されます。この構造体の内容に、e^2 studio ISDE で行った設定が反映されていない場合は、[Project Configuration] の設定が e^2 studio ISDE で保存されているのを確認してから、[Generate Project Content] ボタンを押してください。

Attention: このファイルは、[Generate Project Content] ボタンを押すたびに再生成されるため、編集しないでください。

3.3.4.3 この構造体は、ドライバがオープンされたときに初期化されるため、初期化しないでください。

BSP のスタートアップコードで呼び出される main() が格納されています。main() は、ユーザーが開発したコードが含まれている hal_entry() を呼び出します（次のファイルを参照）。main.c の内容は以下のとおりです。

```
/* generated main source file - do not edit */

extern void hal_entry(void);

void main(void) {

hal_entry();

}
```

Attention: このファイルは、[Generate Project Content] ボタンを押すたびに再生成されるため、編集しないでください。

3.3.4.4 WDT main.c

このファイルには、main() から呼び出される関数 hal_entry() が格納されています。ユーザーが開発したコードは、このファイルと関数に格納します。

WDT プロジェクトでは、このファイルの内容を編集して、以下のコードを追加します。このコードは、このドキュメントの概要セクションにあるフローチャートを実装します。

```
/* HAL-only entry function */

#include "hal_data.h"

#define RED_LED_NO_OF_FLASHES 30

#define RED_LED_PIN IOPORT_PORT_08_PIN_08

#define GREEN_LED_PIN IOPORT_PORT_08_PIN_07

#define RED_LED_DELAY_COUNT 1500000

#define GRN_LED_DELAY_COUNT 1200000

volatile uint32_t delay_counter;

volatile uint16_t loop_counter;

void hal_entry(void) {

    /* TODO: add your own code here */

    /* Open the WDT */

    g_wdt0.p_api->open(g_wdt0.p_ctrl, (wdt_cfg_t  *const)g_wdt0.p_cfg);

  /* Start the WDT by refreshing it */
```

```
    g_wdt0.p_api->refresh(g_wdt0.p_ctrl);

/* Flash the red LED and tickle the WDT for a few seconds */

    for(loop_counter=0; loop_counter<RED_LED_NO_OF_FLASHES; loop_counter++)

    {

        /* Turn red LED on */

        g_ioport.p_api->pinWrite(RED_LED_PIN, IOPORT_LEVEL_HIGH);

     /* Delay */

        for(delay_counter=0; delay_counter<RED_LED_DELAY_COUNT; delay_counter++);

        /* Refresh WDT */

        g_wdt0.p_api->refresh(g_wdt0.p_ctrl);

     /* Turn red off */

        g_ioport.p_api->pinWrite(RED_LED_PIN, IOPORT_LEVEL_LOW);

     /* Delay */

        for(delay_counter=0; delay_counter<RED_LED_DELAY_COUNT; delay_counter++);

        /* Refresh WDT */

        g_wdt0.p_api->refresh(g_wdt0.p_ctrl);

}

    /* Flash green LED but STOP tickling the WDT. WDT should reset the
    device */

    while(1)
```

```
    {   /* Turn green LED on */

    g_ioport.p_api->pinWrite(GREEN_LED_PIN, IOPORT_LEVEL_HIGH);

  /* Delay */

    for(delay_counter=0; delay_counter<GRN_LED_DELAY_COUNT; delay_counter++);

    /* Turn green off */

    g_ioport.p_api->pinWrite(GREEN_LED_PIN, IOPORT_LEVEL_LOW);

  /* Delay */

    for(delay_counter=0; delay_counter<GRN_LED_DELAY_COUNT; delay_counter++);

    }

}
```

WDT HAL ドライバは、r_wdt.h で定義されているインタフェース g_wdt_on_wdt を通じて呼び出されます。WDT HAL ドライバは、r_wdt_api.h で定義されているインスタンスを使用して、open API 呼び出しを通じてオープンされます。

```
g_wdt0.p_api->open(g_wdt0.p_ctrl, (wdt_cfg_t  *const)g_wdt0.p_cfg);
```

最初に渡されるパラメータは、hal_data.c でインスタンス化される制御構造体 g_wdt_ctrl へのポインタです。2 番目のパラメータは、同じ hal_data.c ファイルでインスタンス化される構成データ g_wdt_cfg へのポインタです。

WDT は、次の API 呼び出しによって開始および更新されます。

```
g_wdt0.p_api->refresh(g_wdt0.p_ctrl);
```

ここでも、この API に渡される最初の（この場合は唯一の）パラメータは、ドライバのこのインスタンスの制御構造体へのポインタです。

3.3.5 プロジェクトのビルドとテスト

プロジェクトは、[Build] > [Build Project] でビルドします。プロジェクトはエラーなくビルドされる必要があります。

プロジェクトをデバッグするには

1) ターゲットボードとホスト PC の間に JLink デバッガーを接続します。ボードの電源を投入します。
2) e^2 studio ISDE の左側にある [Project Explorer] ペインで、WDT プロジェクト WDT_Application を右クリックし、[Debug As] > [Debug Configuration] を選択します。

3) [Renesas GDB Hardware Debugging] で、以下に示すように [WDT_Application Debug] を選択します。

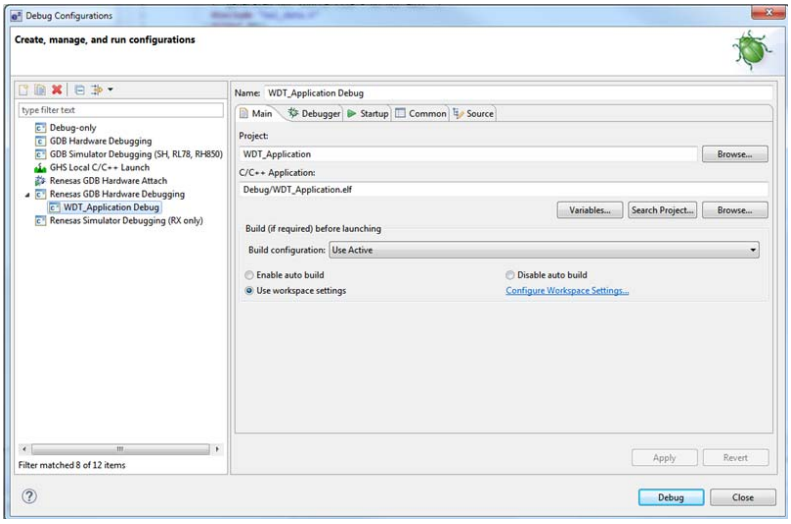

図 93: デバッグ設定

4) [Debug] ボタンを押します。質問されたらデバッグパースペクティブに切り替えます（[Yes] を選択）。

5) コードは Reset_Handler() 関数まで実行されます。

6) [Run] > [Resume] で実行します。main() の hal_entry() の呼び出しで実行が停止します。

7) 再度、[Run] > [Resume] で実行を再開します。

赤い LED が点滅を開始します。30 回点滅した後、緑の LED が点滅を開始し、赤い LED の点滅が止まります。

緑の LED が点滅している間、WDT がアンダーフローしてデバイスをリセットし、その結果シーケンスが繰り返されて再び赤い LED が点滅します。ただし、デバッガに接続されているときには WDT は動作しないため、このシーケンスはデバッガを使用しているときには発生しません。

1) e^2 studio ISDE で、[Run] > [Terminate] を選択してデバッガを停止します。

2) ターゲットボード上のリセットボタンを押します。LED が点滅を開始します。

第 4 章ユーザーガイド

ユーザーガイドでは、各 SSP モジュールの機能や e^2 studio ISDE を使用してプロジェクトをモジュールに含む方法や MCU のモジュールの設定方法、モジュールの API の使い方、モジュールがその他のモジュールとどのように機能するのかについて説明しています。ThreadX RTOS を必要とするモジュールや基礎となる HAL ドライバがないモジュールについてはフレームワークレイヤーに記載されています。RTOS に依存せず HAL ドライバを通してアクセスハードウェア周辺機器アクセスするモジュールについては HAL レイヤーに記載されています。

ユーザーガイドのリストは以下のページで見つけることができます。

- フレームワークレイヤー
- HAL レイヤー

4.1 フレームワークレイヤー

4.1.1 ADC 周期フレームワーク

ADC 周期フレームワークは、ThreadX 対応 ADC アプリケーション用の汎用 API で、SF_ADC_Periodic に実装されます。このフレームワークを利用すれば、使用可能な任意のチャネルを指定したレートでサンプリングし、指定したサンプリング繰り返し回数分のデータをバッファに格納してからアプリケーションに通知できます。ADC 周期フレームワークは、MCU 上の ADC、GPT、および DTC 周辺デバイスを使用します。このセクションでは、e^2 studio ISDE を使用して ADC 周期フレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Analog] > [SF_ADC_Periodic on SF_ADC_Periodic] を選択することで、ADC 周期フレームワークモジュールを追加および構成できます。詳細については、以下を参照してください。e^{2}studio ISDE による ADC 周期フレームワークを使用するアプリケーションの作成

ADC 周期フレームワークの API リファレンスは、次の ADC 周期フレームワークインタフェースの説明内に記載されています :ADC 周期フレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

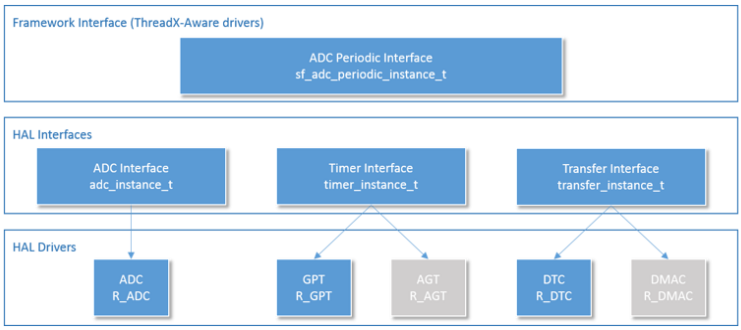

図 94: ADC 周期 - ブロック図

4.1.1.1 ADC 周期フレームワークの機能

ADC 周期フレームワークは、ADC データをサンプリングしてバッファに格納します。設定されたサンプル数のバッファリングが完了すると、アプリケーションに通知されます。

ADC 周期フレームワークは次のように機能します。初期設定の後、スキャンプロセスが開始されると、ハードウェアタイマを使用して ADC スキャンがワンショットモードでトリガされます。各スキャンは 1 つ以上のチャネルで構成できます。各スキャンが完了すると、DTC によって ADC 割り込みが捕捉され、スキャン結果がユーザーバッファに移動します。各スキャンは「繰り返されるサンプリング手順の 1 回の実行」と定義されており、各スキャンで生成されるサンプルの数は、チャネルが連続している場合（たとえば、チャネル 1、2、3、4、5 の場合）にはチャネルの数と一致します。チャネルが連続していない場合（たとえば、チャネル 1、3、4、5 の場合）、各スキャンによって生成されるサンプルには、中間にある未使用のチャネルからのデータも含まれます。したがって、2 つ目の例では、毎回 5 個のサンプルがユーザーバッファに格納されます。ユーザーは、その回数のサンプリングが完了した後に通知を受ける、サンプリング繰り返し回数の総数を指定します。指定されたサンプリング繰り返し回数が実行され、各回のデータがユーザーバッファに格納されたら、バッファ内の有効なデータのインデックスを含むコールバックと、指定された繰り返し回数のサンプリングが完了したことを示すイベントが発生して、ユーザーに通知されます。ユーザーがスキャンプロセスを停止しない限り、引き続きタイマによってスキャンがトリガされ、データがユーザーバッファに書き込まれます。ユーザーバッファはフレームワークによって循環バッファとして扱われます。バッファの名前と長さは ISDE コンフィギュレータによって指定します。詳細については、ADC 周期フレームワークの使用上の注意を参照してください。

タイマインタフェースの GPT 実装と AGT 実装はどちらも、このフレームワークモジュールによって使用できます。

同様に、転送インタフェースの DTC 実装と DMAC 実装も、このフレームワークモジュールによって使用できます。

Note: このフレームワークモジュールで使用されるタイマおよび転送ドライバに関する制限事項については、ADC 周期フレームワークの制限事項を参照してください。

4.1.1.2 e²studio ISDE による ADC 周期フレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

参考資料

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

ADC 周期フレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Framework ADC Periodic	Threads	[Framework] > [Analog] > [SF ADC Periodic on SF_ADC-Periodic]

さらに、MCU で実行できる ADC 周期アプリケーションを作成するには、ADC、DTC、および GPT ドライバが必要です。

リソース	ISDE タブ	選択
ADC Driver	Threads	[Driver] > [Analog] > [ADC Driver on r_adc]
GPT Driver	Threads	[Driver] > [Timers] > [Timer Driver on r_gpt]
DTC Driver	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dtc]

ADC 周期フレームワークのクロックの設定

e2 studio ISDE で、[Clocks] タブを使用して ADC クロックを設定します（クロックの設定を参照）。

ADC 周期フレームワークには、固有のクロック設定は不要です。ADC HAL ドライバのクロック構成は、ADC HAL の使用上の注意（ADC クロックの設定）に記載されています。

ADC 周期フレームワークのピンの設定

e2 studio ISDE を使用して、[Pins] タブから ADC ピンを設定します（ピンの設定を参照）。

ADC 周期フレームワークでは、アナログピンとしてサンプリングする ADC チャネル用のピンを設定します。

ADC 周期フレームワークの割り込みの設定

ADC 周期フレームワークは割り込みを直接使用しませんが、内部で ADC、DTC、および GPT タイマイベントが使用されます。これらについては以下で説明します。

ADC 周期フレームワークパラメータの設定

e^2 studio ISDE を使用して、SF_ADC_Periodic 上の SF ADC 周期ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

[SF ADC Periodic Framework on SF_ADC_Periodic] を追加すると、ssp_cfg_framework ディレクトリに sf_adc_periodic_cfg.h ファイルが作成され、各構成パラメータに対して選択した設定がこのファイルに格納されます。

表 : ADC 周期フレームワークのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define SF_ADC_PERIODIC_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータチェック用のコードを含めるかどうかを制御します。

Note: どのような構成エラーも捕捉できるように、パラメータチェックを有効にした状態でアプリケーションを開発することをお勧めします。実稼働コードではパラメータチェックを無効化して、コードスペースと実行時間を節約できます。

表 : ADC 周期フレームワークの構成

ISDE プロパティ	設定	設定値	説明
Name of the data-buffer to store samples	p_data_buffer	Valid C name	サンプルを格納する 16 ビットデータバッファの名前。
Length of the data-buffer	data_buffer_length	Integer value > 0	データが格納されるバッファの長さ。
Number of sampling iterations	sample_count	Integer value > 0	ADC 周期フレームワークの内部スレッドのプライオリティ。
GPT Timer channel used to trigger the scan	channel	Any pull-down value	ELC イベントエントリ scan_trigger の生成に使用されます。
Callback	p_callback	Valid C name	“sample_counts” 数のデータがバッファに格納された後に呼び出されるユーザー関数。

ISDE プロパティ	設定	設定値	説明
Lower Level ADC Name	p_lower_lvl_adc	adc_instance_t	ADC 周辺デバイスのインスタンス構造体へのポインタ。
Lower Level Timer Name	p_lower_lvl_timer	timer_instance_t	GPT タイマ周辺デバイスのインスタンス構造体へのポインタ。
Lower Level Transfer Name	p_lower_lvl_transfer	transfer_instance_t	DTC 周辺デバイスのインスタンス構造体へのポインタ。

HAL ADC ドライバパラメータの設定

e^2 studio ISDE を使用して、g_adc ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

ADC 周期フレームワークと連携させるため、このフレームワークとともに使用する ADC ドライバの構成パラメータを適切に設定する必要があります。特定の値に設定しなければならない構成プロパティの中には、ロックされていてユーザーが変更できないものがあります。そのような構成パラメータは、sf_adc_periodic_cfg_t::p_lower_lvl_adc::p_cfg が指す構造体に含まれています。次の表に、ADC ドライバを構成するために設定しなければならないオプションの一覧を示します。

Note: 次の表には、このフレームワークと連携させるために設定可能なオプションのみを示します。残りのオプションはこの表から除外されているか、設定不可になっています。

表 : ADC モジュールの設定

ISDE プロパティ	設定	設定値	説明
Unit	unit	0 or 1	使用する ADC ユニットを指定します。S7G2 には、0 と 1 の 2 つのユニットがあります。
Mode	mode	adc_mode_t	このフィールドは ADC フレームワークによって事前設定されており、ロックされています。
Resolution	resolution	adc_resolution_t	このユニットの変換解像度を指定します。
Alignment	alignment	adc_alignment_t	変換結果のアラインメントを指定します。

ISDE プロパティ	設定	設定値	説明
Add Average Count	add_average_count	adc_add_t	このユニット内のいずれかのチャネルに対して、加算または平均化を行う必要があるかどうかを指定します。実際のチャネルはチャネルマスク add_mask を使用して指定します。
Clearing	clearing	adc_clear_t	変換結果を読み取った後で、結果レジスタを自動的にクリアするかどうかを指定します。
Trigger	trigger	adc_trigger_t	このフィールドは ADC フレームワークによって事前設定されており、ロックされています。
Callback	p_callback	Locked for use with the ADC Framework	ADC フレームワークはこのコールバックを内部で使用します。

表 : ADC チャネルの設定

ISDE プロパティ	設定	設定値	説明
Scan Mask	scan_mask	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels.	通常の動作モードでは、このビットマスクフィールドを使用して、その ADC ユニットで有効なチャネルを指定します。たとえば、0x101 に設定されている場合、チャネル 0 と 2 が有効になります。グループモードでは、このフィールドはグループ A に属するチャネルを指定するために使用されます。
Scan Mask Group B	scan_mask_group_b	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels.	このモードはシングルスキャンモードに固定されているため、ADC フレームワークでは使用しないでください。

ISDE プロパティ	設定	設定値	説明
Priority Group A	priority_group_a	adc_group_a_t	このモードはシングルスキャンモードに固定されているため、ADC フレームワークでは使用しないでください。
Add Mask	add_mask	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels.	このフィールドは、add_average_count が有効になっている場合にのみ有効です。このフィールドは、平均化または合計するチャネル結果を決定するために使用します。
Sample Hold Mask	sample_hold_mask	Use #define ADC_MASK_CHANNEL_0 , ADC_MASK_CHANNEL_1 , ADC_MASK_CHANNEL_2 which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels. Only channels 0, 1 and 2 are supported.	チャネル 0、1、2 のどれが、sample_hold_states で指定されたサンプルアンドホールド状態の更新値を使用するかを決定します。このフィールドは、チャネル 0、1、および 2 のデフォルトのサンプルアンドホールドカウント値を変更する場合にのみ設定する必要があります。
Sample Hold States	sample_hold_states	4-255	チャネル専用のサンプルアンドホールド回路用の、更新されたサンプルアンドホールドカウントを指定します。このフィールドは、sample_hold_mask が 0 でない場合にのみ有効です。チャネル 0、1、2 のみに専用のサンプルアンドホールド回路があります。Note：値をサンプリングするデフォルトの状態数（24）を変更するにはこのフィールドを使用します。各状態は、1/ADCLK 時間に等しくなります。

GPT タイマパラメータの設定

e^2 studio ISDE を使用して、g_timer ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

ADC 周期フレームワークと連携させるため、このフレームワークとともに使用する GPT タイマドライバの構成パラメータを適切に設定する必要があります。特定の値に設定しなければならない構成プロパティの中には、ロックされていてユーザーが変更できないものがあります。そのような構成パラメータは、sf_adc_periodic_cfg_t::p_lower_lvl_timer::p_cfg が指す構造体に含まれています。次の表に、GPT HAL ドライバを構成するために設定しなければならないオプションの一覧を示します。

Note: 次の表には、このフレームワークと連携させるために設定可能なオプションのみを示します。残りのオプションはこの表から除外されているか、設定不可になっています。

表 : ADC 周期フレームワークのタイマ設定

設定	設定値	説明
mode	One-shot	このフィールドは ADC フレームワークによって事前設定されており、ロックされています。
period	-	ADC スキャンをトリガするタイマ周期を設定します。
unit	timer_unit_t	上で設定したタイマ周期の単位を設定します。
channel	0-13 for S7G2, 0-9 for S3A7, 0-7 for S124	このフィールドは、ADC フレームワークで選択されたチャネルに基づいて事前設定されており、ロックされています。
autostart	[False]	このフィールドは ADC フレームワークによって事前設定されており、ロックされています。
p_callback	NULL	このフィールドは ADC フレームワークによって事前設定されており、ロックされています。

転送パラメータの設定

DTC 転送モジュールは、ADC 周期フレームワークによって内部で設定されます。このため、ユーザーはモジュール名の指定を除いて、設定を行う必要はありません。

4.1.1.3 ADC 周期フレームワークアプリケーションの作成

ISDE によって生成されたソースファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_adc_periodic_instance_t  g_sf_adc_periodic =

{

    .p_ctrl  = &g_sf_adc_periodic_ctrl,

    .p_cfg = &g_sf_adc_periodic_cfg,

    .p_api = &g_sf_adc_periodic_on_sf_adc

};
```

ISDE コンフィギュレータを使用して ADC フレームワークモジュールと必要な低位 HAL モジュール（GPT タイマ、ADC、DTC 転送モジュールなど）を追加し、それらを設定します。これらのモジュールを設定してファイルが生成されたら、以下の手順に従って ADC 周期フレームワークをスレッドで使用します。以下の手順は、ADC 周期フレームワークのユーザー定義名が g_sf_adc_periodic であることを前提とします。

1) ADC 周期フレームワークフィールド p_callback に設定したコールバック関数の本体を定義します。データが使用可能になると、このコールバックによってユーザーに通知されます。ユーザーは、サンプリングデータが使用可能になったことをアプリケーションの残りの部分に知らせるコードをここに追加する必要があります。
2) アプリケーションスレッドで、open() 関数呼び出しを使用してフレームワークを初期化します。

```
g_sf_adc_periodic.p_api->open(g_sf_adc_periodic.p_ctrl, g_sf_adc_periodic.p_cfg);
```

3) start() 関数を呼び出してスキャンを開始します。

```
g_sf_adc_periodic.p_api->start(g_sf_adc_periodic.p_ctrl);
```

4) stop() 関数を呼び出してスキャンを停止します。

```
g_sf_adc_periodic.p_api->stop(g_sf_adc_periodic.p_ctrl);
```

5) start() 関数をもう一度呼び出してスキャンを再開します。

```
g_sf_adc_periodic.p_api->start(g_sf_adc_periodic.p_ctrl);
```

6) close() 関数を呼び出してフレームワークを閉じます。

```
g_sf_adc_periodic.p_api->close(g_sf_adc_periodic.p_ctrl);
```

4.1.1.4 ADC 周期フレームワークの使用上の注意

このフレームワークで使用する ADC HAL ドライバを設定するとき、チャネルを 1 つ以上選択してください。そうしなければ、API によってエラーが返されます。

ADC フレームワークのスキャンレート（GPT タイマ周期）を設定するとき、選択したすべてのチャネルをスキャンできる十分な長さの周期を設定してください（Synergy S7G2 デバイスでは、各チャネル変換に約 2 マイクロ秒かかります)。

周期 ADC フレームワークは、各スキャンで収集したすべてのチャネルのデータをユーザー指定のバッファに格納します。指定されたサンプリング繰り返し回数が完了すると、ユーザーに通知されます。したがって、5 つのチャネルを選択していて（チャネル 1、2、3、4、5)、サンプルカウントを 3 に設定している場合は、5 x 3 = 15 のサンプルが使用可能になるとユーザーに通知されます。サンプルの順序は次のようになります。

図 95: ADC 周期 – サンプル順序

周期 ADC フレームワーク構成でデータバッファ長を選択するとき、バッファの長さが、生成されるサンプル数の長さの 2 倍以上になるようにしてください（この例では 15 x 2 = 30)。そのようにする理由は、ユーザーがデータ使用可能の通知を受けた後も、引き続き新しいデータがサンプルレートでバッファリングされるためです。このバッファは循環バッファとして扱われるため、データが意図せずに上書きされる可能性があります。つまり、バッファサイズが生成されるサンプル数以下の場合、アプリケーションでデータが使用可能になる前にデータが上書きされてしまいます。

アプリケーションコールバックによって、有効なデータが存在するバッファ内の適切な位置を指すインデックスが通知されます。

4.1.1.5 ADC 周期フレームワークでサポートされるハードウェア実装

このドライバは S7G2 MCU でテストされています。

4.1.1.6 ADC 周期フレームワークでサポートされるオペレーティングシステム

このフレームワークはミューテックスのような ThreadX に固有のオブジェクトを使用し、ThreadX RTOS を必要とします。

4.1.1.7 ADC 周期フレームワークの制限事項

ADC 周期フレームワークでは現在、GPT 以外のタイマはサポートされていません。また、グループスキャンモードもサポートされていません。

ADC 周期フレームワークの現在のバージョンでは、DTC 実装のみがサポートされています。

このフレームワークで使用する ADC チャネルを設定するとき、他の使用可能なチャネルも合わせて選択する場合は、温度センサーと電圧センサーを選択しないでください。温度センサーだけ、電圧センサーだけ、または任意の数の通常 ADC チャネルを使用できます。

4.1.2 オーディオ再生フレームワーク

オーディオ再生フレームワークは、同期を処理してモノラル 16 ビットパルス符号変調（PCM）サンプルを再生します。ハードウェアにアクセスするためにハードウェアポートの DAC オーディオ再生フレームワークまたは I2S オーディオ再生フレームワークを使用します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Audio] > [Audio Playback Framework on sf_audio_playback] を選択することで、オーディオ再生フレームワークモジュールを追加および構成できます。詳細については、e^2 studio ISDE によるオーディオ再生フレームワークを使用するアプリケーションの作成を参照してください。

フレームワークオーディオ再生フレームワークの API リファレンスはオーディオフレームワークインタフェースにあります。

SSP レイヤー	オーディオ再生フレームワークコンポーネント
API リファレンス : フレームワーク レイヤー	オーディオフレームワークインタフェース , DAC オーディオ再生フレームワーク , I2S オーディオ再生フレームワーク
API リファレンス : HAL インタフェース	DAC インタフェース , タイマインタフェース , 転送インタフェース , I2S インタフェース
API リファレンス : HAL レイヤー	-
Board Support Package	-

4.1.2.1 オーディオ再生フレームワークの機能

オーディオ再生フレームワークは、以下の機能を実装します。

- データを扱いやすいチャンクに分割することによって長いバッファを再生します。
- ThreadX タイムアウトが発生するまで再生を繰り返します（正弦波音やループするバックグラウンドミュージックのように音声が繰り返される場合）。
- 最後のバッファの再生が開始された後、コールバックを使用して次のデータを要求します。
- ソフトウェア音量制御。
- 一時停止と再開機能。
- 符号付き 16 ビット PCM データを符号なし 12 ビット DAC の範囲にシフトするためのスケーリング
- 複数のストリームについての基本的なミキシング。

オーディオ再生フレームワークは、単一のハードウェアポート上の複数のストリームをサポートします。2 つのストリームを使用する場合に必要なモジュールのブロック図を下図に示します。

SSP オーディオ再生スタック

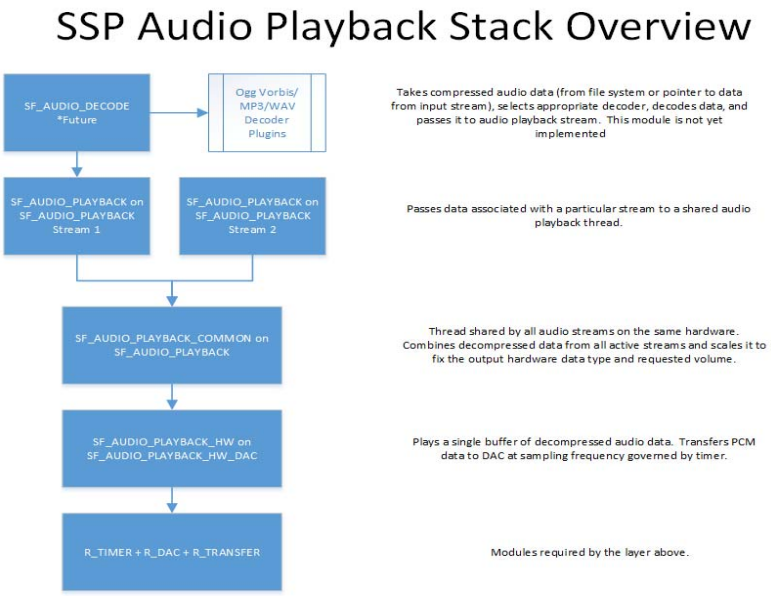

図 96: SSP オーディオ再生スタック

オーディオ再生フレームワークは、オーディオ再生をサポートするためのスレッドを内部で作成します。下図に、オーディオ再生フレームワークのスレッドとパブリックオーディオ再生フレームワーク API とのやり取りを表すフローチャートを示します。

オーディオ再生フレームワークフローチャート

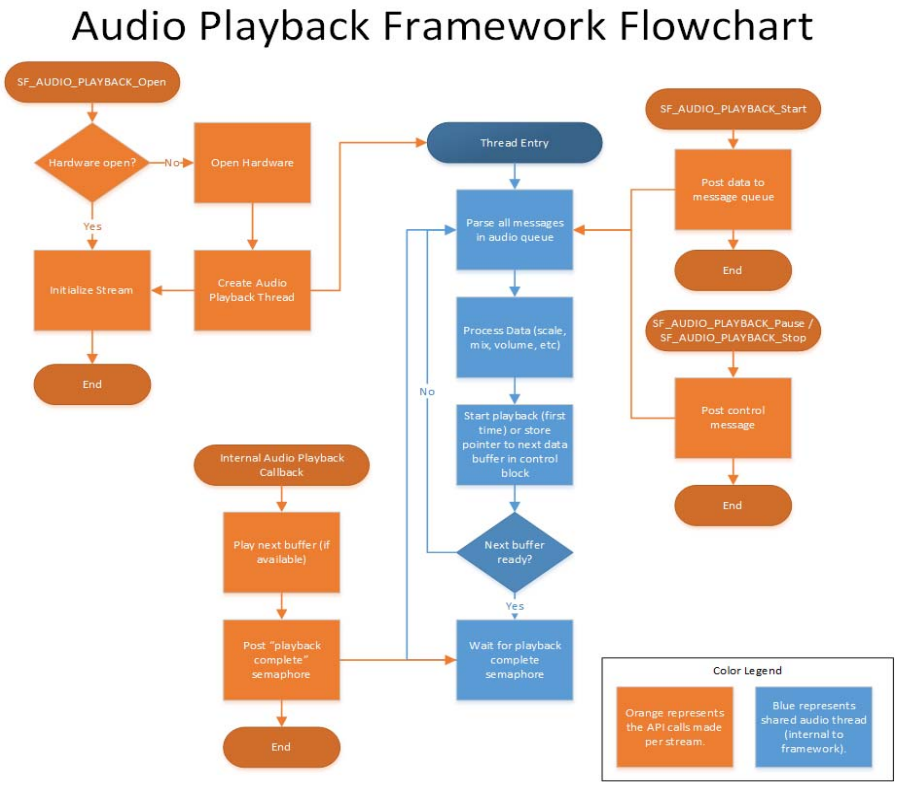

図 97:　　オーディオ再生フレームワークフローチャート

4.1.2.2 e² studio ISDE によるオーディオ再生フレームワークを使用するアプリケーションの作成

フレームワークは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

オーディオ再生フレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Audio Playback Framework	Threads	[Framework] > [Audio] > [Audio Playback Framework on sf_audio_playback]

リソース	ISDE タブ	選択
Audio Playback Framework Shared	Threads（HAL/Common）	[Framework] > [Audio] > [Audio Playback Framework Shared on sf_audio_playback]
Queue	Threads（Objects）	Queue-[Audio Playback Framework Shared on sf_audio_playback] の [Properties] で指定された名前と一致する必要があります（デフォルトは g_sf_audio_playback_queue）。
Messaging Framework	Threads（HAL/Common）	[Framework] > [Services] > [Messaging Framework on sf_message]

[Messaging] タブのメッセージングフレームワークコンフィギュレータを使用して、メッセージングフレームワークを設定します。

1) オーディオ再生イベントクラスをハイライトします。
2) 新しいサブスクライバーを追加します。以下の設定を選択して、sf_audio_playback モジュール上のオーディオ再生フレームワークの [Properties] タブでメッセージクラスインスタンスが開始インスタンスと終了インスタンスの間に適切に設定されていることを確認します。
   - スレッド : アプリケーション内の任意のスレッド。
   - 開始 : アプリケーションで使用されている最初のオーディオインスタンス。
   - 終了 : アプリケーションで使用されている最後のオーディオインスタンス。
3) オーディオ再生サブスクライバーで新しいサブスクライバーをハイライトします。シンボル名を記録します。
4) [Threads] タブに戻ります。
5) HAL/Common のオーディオ再生フレームワーク共有モジュールをハイライトして、オーディオメッセージキュー名をオーディオ再生サブスクライバーのシンボル名に設定します。

また、オーディオ再生フレームワークでは、単一のハードウェアポートが必要です。SSP では以下のハードウェアポートが提供されています。

- AUDIO_PLAYBACK_DAC_CFG オーディオフレームワークの DAC ハードウェアポートの使用
- I2S オーディオフレームワークの I2S ハードウェアポートの使用

オーディオフレームワークの DAC ハードウェアポートの使用

オーディオフレームワークの DAC ハードウェアポートは、転送 API を使用して、内蔵タイマで定義されたサンプリング周波数で再生バッファから DAC にオーディオデータを転送します。オーディオ再生 DAC ハードウェアポートを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Audio Playback Hardware Framework Shared	Threads（HAL/Common）	[Framework] > [Audio] > [Audio Playback Hardware Framework Shared on sf_audio_playback_hw_dac]

オーディオフレームワーク DAC ハードウェアポートには、タイマ、DAC、および転送 API モジュールに対する依存関係があります。

- タイマモジュールを追加します。
  - 周波数（Hz 単位）をオーディオデータのサンプリング周波数に設定します。
  - DTC を転送モジュールとして使用する場合は（推奨）、割り込みを有効にします。
- DAC モジュールを追加します。
- r_dtc 上の転送モジュールを追加します。
  - DAC チャネル 0 を使用する場合は宛先ポインタを &R_DAC->DADRn[0] に設定し、DAC チャネル 1 を使用する場合は &R_DAC->DADRn[1] に設定します。
  - アクティベーションソースを上で選択したタイマ割り込みに設定します。

オーディオフレームワークの I2S ハードウェアポートの使用

オーディオフレームワーク I2S ハードウェアポートは、I2S 通信インタフェースを使用して、オーディオデータを外部バスに書き込みます。オーディオ再生 DAC ハードウェアポートを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Audio Playback Hardware Framework Shared	Threads（HAL/Common）	[Framework] > [Audio] > [Audio Playback Hardware Framework Shared on sf_audio_playback_hw_i2s]

オーディオフレームワーク I2S ハードウェアポートには、I2S モジュールに対する依存関係があります。I2S ドライバモジュールは DTC を使用して高速化できます（推奨）。

- I2S ドライバモジュールを追加します。
  - オーディオクロック周波数（ヘルツ単位）を、使用されている入力オーディオクロックの周波数に設定します。
  - サンプリング周波数（ヘルツ単位）をオーディオデータのサンプリング周波数に設定します。

– データビットとワード長を 16 ビットに設定します（オーディオフレームワークでは 16 ビットのみが受け入れられます）。

– SSIn TXI および SSIn INT 割り込みを有効にします。

- （推奨）r_dtc 上の転送モジュールを追加します。
- アクティベーションソースを SSIn TXI 割り込みに設定します。

4.1.2.3 オーディオ再生フレームワークアプリケーションの作成

ISDE によって生成されたソースファイルでプロジェクトを作成すると、オーディオおよびメッセージフレームワークモジュール用に以下のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_audio_playback_instance_t  g_sf_audio_playback =

{

    .p_ctrl  = &g_sf_audio_playback_ctrl,

    .p_cfg = &g_sf_audio_playback_cfg,

    .p_api = &g_sf_audio_playback_on_audio_playback

};
```

```
// Instance structure to use this module.

const sf_message_instance_t  g_sf_message = {

    .p_ctrl  = &g_sf_message_ctrl,

    .p_cfg = &g_sf_message_cfg,

    .p_api = &g_sf_message_on_sf_message

};
```

4.1.2.4 オーディオ再生フレームワーク制限

このモジュールを使用するうえで既知の制限事項はありません。最新のソフトウェアバージョンの詳細については、SSP のリリースノートを参照してください。

オーディオ再生フレームワークの推奨される使用方法を以下に示します。

1) セマフォを作成します（たとえば、g_sf_audio_playback_semaphore）。これは [Threads] タブで行うことができます。初期値を 2 に設定します（オーディオ再生フレームワークでは、ストリームあたり最大 2 つのデータメッセージを保持できます）。

2) コールバック関数を作成します（たとえば、sf_audio_playback_callback）。オーディオ再生フレームワークインスタンスにコールバック関数の名前を入力します。このコールバック関数は、オーディオ再生フレームワークがデータの処理を終了したときに呼び出されます。コールバックで、上記の手順 1 で作成したセマフォを配置します。例 :

```
void sf_audio_playback_callback(sf_message_callback_args_t * p_args)

{

  tx_semaphore_put(&g_sf_audio_playback_semaphore);

}
```

3) メインループ内で、データを再生する前にセマフォを取得します。データを再生するには、まずメッセージングフレームワークからバッファを取得し、次にオーディオ再生データ構造をバッファ内に作成します。例 : 以下の例では、4096 のサンプルを含む符号付き 16 ビットモノラル PCM データの PCM バッファ（g_audio_buffer）を再生しています。

```
tx_semaphore_get(&g_sf_audio_playback_semaphore, TX_WAIT_FOREVER);

ssp_err_t  err;

sf_audio_playback_data_t  * p_audio_playback_data;

sf_message_acquire_cfg_t  acquire_cfg =

{

  .buffer_keep  = false

};

err = g_sf_message.p_api->bufferAcquire(g_sf_message.p_ctrl, (sf_message_header_t  **) &p_audio_playbac
k_data, &acquire_cfg, 10);

if (SSP_SUCCESS != err)

{

  while(1);

}
```

```
p_audio_playback_data->header.event_b.class = SF_MESSAGE_EVENT_CLASS_AUDIO;

p_audio_playback_data->header.event_b.class_instance = 0;

p_audio_playback_data->header.event_b.code = SF_MESSAGE_EVENT_AUDIO_START;

p_audio_playback_data->type.is_signed  = 1;

p_audio_playback_data->type.scale_bits_max  = 16;

p_audio_playback_data->size_bytes  = (uint32_t) 4096 *sizeof(int16_t);

p_audio_playback_data->p_data  = &g_audio_buffer[0];

p_audio_playback_data->loop_timeout  = TX_NO_WAIT; /* Play buffer once */

p_audio_playback_data->stream_end  = *false*; /* Lock stream to this thread. */

index += p_audio_playback_data->size_bytes;

err = g_sf_audio_playback->p_api->start(g_sf_audio_playback->p_ctrl, p_audio_playback_data, TX_WAIT_FOREVER);
```

4.1.2.5 オーディオ再生フレームワークでサポートされるデバイス

このフレームワークは、以下のファミリでテストされています。

- S7G2

オーディオ再生フレームワークは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.1.3 通信フレームワーク

通信フレームワークは、ThreadX RTOS を使用する通信アプリケーション用の汎用 API です。このフレームワークは現在、UART（sf_uart_comms）、USBX™ CDC ACM（sf_el_ux_comms）、および Telnet（sf_el_nx_comms）に実装されています。このセクションでは、e² studio ISDE を使用して通信フレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e² studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで以下のモジュールのいずれかを選択することで、通信フレームワークモジュールを追加および構成できます。

- [New] > [Framework] > [Connectivity] > [Communications Framework on sf_uart_comms]
- [New] > [Framework] > [Connectivity] > [Communications Framework on sf_el_ux_comms]

- [New] > [Framework] > [Connectivity] > [Communications Framework on sf_el_nx_comms]

詳細については、以下を参照してください。e² studio ISDE による通信フレームワークを使用するアプリケーションの作成

通信フレームワークの API リファレンスは、次の通信フレームワークインタフェースの説明内に記載されています : 通信フレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

4.1.3.1 通信フレームワークの機能

このモジュールは、ThreadX 対応の通信フレームワークです。このモジュールは、ミューテックスなどの ThreadX オブジェクトを使用してブロックを行い、イベントフラグなどの同期手法を使用してトランザクションを完了します。

4.1.3.2 e² studio ISDE による通信フレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

どのタイプの通信インタフェースを使用するかに応じて（UART、NetX™ Telnet サーバー、USBX™ CDC ACM）、以下のいずれかのセクションを選択してください。

UART 通信フレームワークの追加

UART フレームワークインタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
UART Communications Framework	Threads	[Framework] > [Connectivity] > [Communications Framework on sf_uart_comms]

さらに、MCU で実行できる UART アプリケーションを作成するには、UART HAL ドライバが少なくとも 1 つ必要です。

UART フレームワークと SCI モジュールを使用する場合は、以下のリソースを追加します。

リソース	ISDE タブ	選択
Driver for UART on SCI	Threads	r_sci_uart 上の UART ドライバ
SCI Common driver	Threads	SCI 共通 ; このドライバの [Properties] ウィンドウで調歩同期式モード（r_sci_uart）を有効にします。
Interrupts	ICU	SCI>SCIn>SCIn RXI、SCIn TXI、SCIn TEI、SCIn ERIn 選択した SCI チャネル n の割り込み。

USBX™ 通信フレームワークの追加

USBX™ フレームワークインタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Communications Framework	Threads	[Framework] > [Connectivity] > [Communications Framework on sf_el_ux_comms]

USBX™ CDC ACM 通信フレームワークを使用する場合は、以下の依存リソースを追加します。

リソース	ISDE タブ	選択
USBX Device Class CDC ACM	Threads	[Framework] > [USB] > [USBX Device Class CDC ACM on ux_device_class_cdc_acm]
USBX Stack	Threads	[Framework] > [USB] > [USBX on ux]
USBX Port	Threads	[Framework] > [USB] > [USBX Port on sf_el_ux]

sf_el_ux_comms 上の新しい通信フレームワークをハイライトします。このモジュールのプロパティは以下の表に説明されています。この表において、「プロパティ」は ISDE 内での [Properties] タブ名を表します。

USBX™ 通信フレームワークの構成

ISDE プロパティ	設定値	説明
Memory Size (Bytes)	Default: 65536	ux_system_initialize() に渡されるメモリ。ホストモードが使用される場合は、65536 から開始することが推奨されます。デバイスのみのマクロが定義されている場合は、3072 から開始することが推奨されます。メモリサイズは最終的にはアプリケーションによって決定される必要があります。
Read Input Buffer Size (Bytes)	Default: 128	これは、read() API で一度に受信できる最大バイト数です。

Telnet 通信フレームワークの追加

USBX™ フレームワークインタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Communications Framework	Threads	[Framework] > [Connectivity] > [Communications Framework on sf_el_nx_comms]

USBX™ CDC ACM 通信フレームワークを使用する場合は、以下の依存リソースを追加します。

リソース	ISDE タブ	選択
NetX Telnet Server	Threads	[Framework] > [Networking] > [NetX Telnet Server on nx]
NetX Stack	Threads	[Framework] > [Networking] > [NetX on nx]
NetX Port	Threads	[Framework] > [Networking] > [NetX Port on sf_el_nx]

sf_el_nx_comms 上の新しい通信フレームワークをハイライトします。このモジュールのプロパティは以下の表に説明されています。この表において、「プロパティ」は ISDE 内での [Properties] タブ名を表します。

ISDE プロパティ	設定値	説明
Channel	デフォルト : 0	イーサネットドライバによって使用される基底チャネル
IP Address Byte < n >	192.168.0.0	IP アドレスバイト
Subnet Mask Byte < n >	255.255.255.0	サブネットマスクバイト

4.1.3.3 通信フレームワークアプリケーションの作成

open 呼び出しは、ISDE によって、モジュールが追加された src/synergy_gen/< スレッド名 >.c ファイル内に生成されます。必要なハードウェア接続が確立されていれば、実行が src/< スレッド名 >_entry.c に達するまでにモジュールが使用可能になります。通信フレームワークを呼び出すには、API を直接呼び出します。

ISDE によって生成されたソースファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_comms_instance_t  g_sf_comms =

{

    .p_ctrl  = &g_sf_comms_ctrl,

    .p_cfg = &g_sf_comms_cfg,

    .p_api = &g_sf_comms_on_sf_comms

};
```

以下は書き込みの呼び出し例です。

```
char p_src[] = "Hello World!\\r\\n";

g_sf_comms.p_api->write(g_sf_comms.p_ctrl, p_src, sizeof(p_src) - 1, TX_NO_WAIT);
```

以下はブロッキング読み取りの呼び出し例です。

```
char p_dest[10];

g_sf_comms.p_api->read(g_sf_comms.p_ctrl, p_dest, 10, TX_WAIT_FOREVER);
```

4.1.3.4 通信フレームワークの制限事項

このモジュールを使用するうえで既知の制限事項はありません。最新のソフトウェアバージョンの詳細については、SSP のリリースノートを参照してください。

4.1.3.5 通信フレームワークでサポートされるデバイス

この通信フレームワークモジュールは、S7G2 でテストされています。

UART 通信フレームワークは、API への変更なしに、SCI UART および HAL ドライバを使用するすべての MCU をサポートするように設計されています。

USBX™ CDC 通信フレームワークは、USB コントローラを使用するすべての MCU をサポートするように設計されています。

NetX™ Telnet 通信フレームワークは、イーサネットコントローラを使用するすべての MCU をサポートするように設計されています。

4.1.4 コンソールフレームワーク

コンソールフレームワークは、ThreadX RTOS を使用するコンソールコマンドラインインタフェース（CLI）アプリケーション用の汎用 API で、sf_console に実装されています。この実装は、SSP の UART（sf_uart_comms）、USBX™ CDC ACM（sf_el_ux_comms）、および Telnet（sf_el_nx_comms）に実装されている通信インタフェースを使用します。このセクションでは、e² studio ISDE を使用してフレームワークコンソールインタフェースを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e² studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Services] > [Console Framework on sf_console] を選択することで、コンソールフレームワークモジュールを追加および構成できます。詳細については、e² studio ISDE によるコンソールフレームワークを使用するアプリケーションの作成を参照してください。

コンソールフレームワークは、以下の機能をサポートしています。

- メニューに基づくコマンドラインインタフェースの作成
- サブメニューと、単一呼び出しによる複数メニュー内の移動
- 親メニューへの移動、またはメニューのルートに戻る
- メニューごとのヘルプメニュー
- NULL 終端文字列の書き込みと、改行文字までの読み取り
- 引数をコマンドラインに解析する API
- 大文字と小文字を区別しない入力

API リファレンスは、次のコンソールフレームワークインタフェースの説明内に記載されています : コンソールフレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

4.1.4.1 コンソールフレームワークの機能

このモジュールは、ThreadX 対応のコンソールコマンドラインインタフェース（CLI）フレームワークです。このモジュールは、ミューテックスなどの ThreadX オブジェクトを使用してブロックを行い、イベントフラグなどの同期手法を使用してトランザクションを完了します。

4.1.4.2 e² studio ISDE によるコンソールフレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

コンソールフレームワークの追加

コンソールフレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
UART Console Framework	Threads	[Framework] > [Services] > [Console Framework on sf_console]

さらに、MCU で実行できるコンソールアプリケーションを作成するには、通信フレームワークが少なくとも 1 つ必要です。コンソールフレームワークの詳細については、通信フレームワークを参照してください。

4.1.4.3 コンソールフレームワークアプリケーションの作成

open 呼び出しは、ISDE によって、モジュールが追加された src/synergy_gen/< スレッド名 >.c ファイル内に生成されます。この open 呼び出しを実行するには、コンフィギュレータで設定した名前（デフォルトは g_sf_console_root_menu）と一致する変数名を持つ sf_console_menu_t 型のルートメニューをアプリケーションで定義する必要があります。

必要なハードウェア接続が確立されていれば、実行が src/< スレッド名 >_entry.c に達するまでにモジュールが使用可能になります。コンソールフレームワークを使用するには、まずメニュー、コマンド構造体、およびコールバックを設定します。次に、スレッド内の while ループからプロンプトを実行します。プロンプト API は、現在のメニューをプロンプトとして出力してから、入力を読み取り、それをコンソールにエコーバックします（プロパティでエコーが無効にされている場合は除く）。

コンソールに入力するとき、BackSpace キーはカーソルの前の文字を消去し、Delete キーはカーソルの後の文字を消去します。左右の矢印キーを押すとカーソルが移動します。上矢印キーを押すと、他に何も入力していない場合にのみ、最後のコマンドが入力されます。最後のコマンドより前の履歴は保持されません。つまり、上矢印キーを 2 回押しても、最後のコマンドの前に入力されたコマンドの情報は残っていないため、最後のコマンドが引き続き表示されます。

読み取り入力時に改行文字 ('/n') が現れると、入力文字列が解析され、関連するコールバックが呼び出されるか、次のメニューに切り替わります（コマンドのコールバックの代わりに SF_CONSOLE_CALLBACK_NEXT_FUNCTION が使用されている場合）。解析はコールバック関数が呼び出されるまで続行されます。プロンプトが再び呼び出された場合は、コールバック関数を含むメニューを表示して入力待ちになります。親メニューに移動するには、'^' を押します。サブメニューからルートメニューに移動するには、'~' を押します。

以下に、comms_thread というシンボル名のスレッドで単一メニューを持つコンソールを作成する例を示します。

ISDE によって生成されたソースファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_console_instance_t  g_sf_console =

{

    .p_ctrl  = &g_sf_ console_ctrl,

    .p_cfg = &g_sf_ console_cfg,

    .p_api = &g_sf_ console_on_sf_console

};
```

この例では、BSP によって提供された LED リスト内の最初の LED をトグルする単一のコマンドを使用します。このコードを使用するには、コンソールを開いて次のコマンドを入力します。

Root>led toggle

また、次のコマンドを入力してヘルプメニューを見ることもできます。

Root>?

```
void led_toggle_callback(sf_console_callback_args_t * p_args);
const sf_console_command_t g_sf_console_commands[] =
{
    { .command = "LED TOGGLE", .help = "Toggle an LED", .callback = led_toggle_callback, .context = NULL},
};
const sf_console_menu_t g_sf_console_root_menu =
{
    .menu_prev = NULL,
    .menu_name = "Root",
    .num_commands = (sizeof(g_sf_console_commands)) / (sizeof(g_sf_console_commands[0])),
    .command_list = &g_sf_console_commands[0]
};
void led_toggle_callback(sf_console_callback_args_t * p_args)
{
  bsp_leds_t leds;
  ioport_level_t level;
  R_BSP_LedsGet(&leds); // Get LED list from BSP
```

```
  g_ioport.p_api->pinRead(leds.p_leds[0], &level); // Read current level

 g_ioport.p_api->pinWrite(leds.p_leds[0], !level); // Write opposite level

}

void comms_thread_entry(void)

{

    while (1)

    {

        g_sf_console.p_api->prompt(g_sf_console.p_ctrl, NULL, TX_WAIT_FOREVER);

  }

}
```

以下に、"comms_thread" というシンボル名のスレッドで "Root" メニューと "LED" サブメニューを持つコンソールを作成する例を示します。これらのメニューを結び付けるため、コマンドコールバックを SF_CONSOLE_CALLBACK_NEXT_FUNCTION に設定し、コマンドコンテキストをメニューへのポインタ（この例では g_sf_console_led_menu）に設定します。このコードを使用するには、コンソールを開いて次のコマンドを入力します。

Root>led<LED メニューに切り替えます。>toggle<led_toggle_callback コールバックを呼び出して LED をトグルします。

また、次のコマンドを入力してヘルプメニューを見ることもできます。

Root>?< ヘルプメニューを表示します。

```
const sf_console_command_t g_led_commands[] =

{

    { .command = "Toggle", .help = "Toggle LED1", .callback = led_toggle_callback, .context = NULL},

};

extern const sf_console_menu_t g_sf_console_root_menu;

const sf_console_menu_t g_sf_console_led_menu =

{

    .menu_prev = &g_sf_console_root_menu,

    .menu_name = "LED",

    .num_commands = (sizeof(g_led_commands)) / (sizeof(g_led_commands[0])),

    .command_list = &g_led_commands[0]

};

const sf_console_command_t g_sf_console_commands[] =

{

    { .command = "LED", .help = "Enter LED menu", .callback = #define
 SF_CONSOLE_CALLBACK_NEXT_FUNCTION, .context = &g_sf_console_led_menu},

};

const sf_console_menu_t g_sf_console_root_menu =

{

    .menu_prev = NULL,

    .menu_name = "Root",

    .num_commands = (sizeof(g_sf_console_commands)) / (sizeof(g_sf_console_commands[0])),

    .command_list = &g_sf_console_commands[0]

};

void led_toggle_callback(sf_console_callback_args_t * p_args)
```

```
{

    bsp_leds_t  leds;

    ioport_level_t  level;

    R_BSP_LedsGet(&leds);

    g_ioport.p_api->pinRead(leds.p_leds[0], &level);

    g_ioport.p_api->pinWrite(leds.p_leds[0], !level);

}

void comms_thread_entry(void)

{

    while (1)

    {

        g_sf_console.p_api->prompt(g_sf_console.p_ctrl, NULL, TX_WAIT_FOREVER);

    }

}
```

コマンドラインから引数を解析するには、argumentFind API を使用します。以下の例では、argumentFind を使用して “LED” という引数を探し、その文字列の後の数字を “led_num” という変数に格納しています。これを上記のどちらかの例の led_toggle_callback に追加し、コンソールに “led toggle led 2” と入力すると、この呼び出しの後、“led_num” が 2 に設定されます。見つかった引数の後のスペースは無視されるため、以下はどちらも機能します。

- led toggle led2
- led toggle led 2

```
int32_t led_num;

g_sf_console_on_sf_console.argumentFind("LED", p_args->p_remaining_string, NULL, &led_num);
```

4.1.4.4 コンソールフレームワークの制限事項

このモジュールを使用するうえで既知の制限事項はありません。最新のソフトウェアバージョンの詳細については、SSP のリリースノートを参照してください。

4.1.4.5 コンソールフレームワークでサポートされるデバイス

このコンソールフレームワークモジュールは、S7G2 でテストされています。このモジュールはソフトウェアのみであり、基礎となる通信フレームワークドライバをサポートするすべての MCU で機能します。

4.1.5 静電容量タッチフレームワーク

静電容量タッチフレームワークは、ThreadX RTOS を使用する静電容量タッチアプリケーション用の汎用 API です。静電容量タッチフレームワークは、MCU で利用可能な CTSU 周辺機器をサポートし、sf_touch_ctsu に実装されます。このフレームワークは、Workbench 6 ツールによって生成された構成データと組み合わせて使用するために設計されています。このセクションでは、e2 studio ISDE を使用してフレームワーク CTSU インタフェースを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Input] > [Cap Touch Framework on sf_touch_ctsu] を選択することで、静電容量タッチフレームワークモジュールを追加および構成できます。詳細については、以下を参照してください。e^2 studio ISDE による静電容量タッチフレームワークを使用するアプリケーションの作成

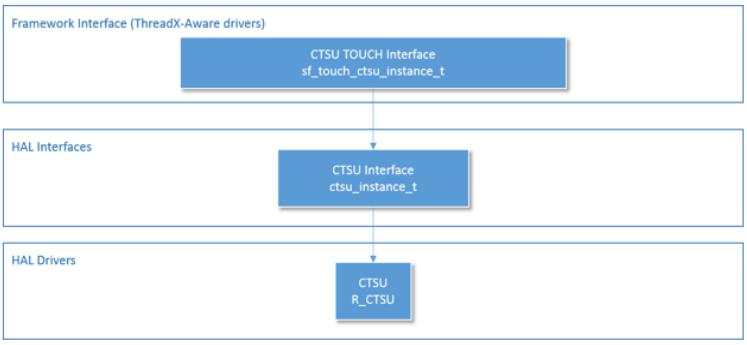

図 98: 静電容量タッチフレームワーク - ブロック図

フレームワーク CTSU インタフェースの API リファレンスは次の場所にあります : CTSU フレームワークインタフェース。

CTSU ドライバも参照してください。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

4.1.5.1 静電容量タッチフレームワークの機能

CTSU フレームワークインタフェースは、静電容量タッチパネルのハードウェアスキャンを実行してパネルを周期的に更新するプライベートスレッドを作成します。このフレームワークモジュールは、HAL レイヤー CTSU ドライバを使用してスキャン結果を読み取ります。

スキャンが完了すると、アプリケーションレイヤーによって登録されたコールバックが呼び出されます。複数の上位レイヤーがこのフレームワークを使用している場合（ボタンやスライダ、ホイールなど）、このレイヤーは初期化を行った順にこれら各レイヤーのコールバックを呼び出します（NULL でない場合）。内部

スレッドの周波数（update_hz）は、このフレームワークに実行される後続の各 open() 呼び出しで設定 / 変更されます。

4.1.5.2 e² studio ISDE による静電容量タッチフレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

CTSU フレームワークインタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Framework CTSU Driver	Threads	[Framework] > [Input] > [Cap Touch Framework on sf_touch_ctsu]
CTSU Interrupts	ICU	アプリケーション例については、静電容量タッチフレームワークの割り込みの設定

静電容量タッチフレームワークのクロックの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。CTSU クロック速度を、静電容量タッチ調整ツール Workbench 6 で選択したクロック速度と同じ値に設定します。

CTSU ドライバを参照してください。

静電容量タッチフレームワークのピンの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。ピンをタッチセンサーピン TSxx として設定し、TSCAP 機能を有効にします。

CTSU ドライバを参照してください。

静電容量タッチフレームワークの割り込みの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。[ICU] タブで 3 つの CTSU 割り込みをすべて同じプライオリティに設定して、これらの割り込みを有効にします。どのような値を設定しても構いません。

CTSU ドライバを参照してください。

静電容量タッチフレームワークのパラメータの設定

e² studio ISDE を使用して、静電容量タッチフレームワークのパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

静電容量タッチフレームワークは、SF_CTSU_Open() API で sf_ctsu_cfg_t 型のポインタを渡すことによって設定されます。

静電容量タッチフレームワークの設定

ISDE プロパティ	設定	設定値	説明
Name	-	Arbitrary symbol	ドライバ名
Lower Level Touch Driver Name	p_ctsu_instance	Name of lower level driver	-
Thread Priority	priority	ThreadX Priority:	システムの他のスレッドのプライオリティに基づいてユーザーが決定します。高いプライオリティを使用することを推奨します。
Update Hz	update_hz	>10 Hz	CTSU ハードウェアスキャンを実行するレート
Callback	p_callback	>10Hz	各スキャンが完了したときに呼び出されるユーザーコールバックの名前。新しい処理済みデータが使用可能になったときにも呼び出されます。使用しない場合は NULL に設定できます。

CTSU HAL ドライバパラメータの設定

e^2 studio ISDE を使用して、HAL レイヤー CTSU ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

静電容量タッチフレームワークアプリケーションの作成

1) Workbench 6 によって生成されたデータを含むフォルダが ${PROJECT_ROOT}\src フォルダに存在し、ビルドに含まれていることを確認します。
2) [Threads] タブで新しいスレッドを作成するか、既存のスレッドに [Cap Touch Framework on sf_touch_ctsu] を追加します。スレッドの [Name] を確認します。
3) 同じスレッドで、[Touch Driver on r_ctsu] を追加します。
4) CTSU 上のタッチドライバの [Properties] タブを使用して、モジュールの名前 ** と [CTSU Configuration Used] を確認します。(CTSU HAL モジュールの設定の詳細については、CTSU HAL モジュールの使用上の注意を参照してください)。
5) [Cap Touch Framework on sf_touch_ctsu] の [Properties] タブを使用して、このドライバの [Name] を設定します。

6) 提供されたローレベルタッチドライバ名が CTSU モジュール上のタッチドライバのエントリと同じであることを確認します。
7) [Thread Priority] フィールドを高いプライオリティ（5 など）に設定します。
8) [Update Hz] フィールドを 100 に設定します。
9) [Generate Project Content] をクリックして、必要な構造体を生成します。
10) ssp_gen の下にある Thread_name.h ファイルを開いて、生成された構造体を確認します。
11) プロジェクトをビルドします。
12) SF_TOUCH_CTSU Name.api->read を使用して、スキャンされたデータを読み取ります。
13) ハードウェアを接続し、実行可能ファイルをダウンロードして実行します。
14) Name.api->read から読み取ったデータをウォッチウィンドウに追加します。

これで、静電容量タッチセンサーにタッチすると、構成に従ってバイナリビットマスクが変更されます。

4.1.5.3 CTSU フレームワークの制限事項

このモジュールを使用するうえで既知の制限事項はありません。SSP のリリースノートも参照してください。

4.1.5.4 CTSU フレームワークインタフェースファイル

プロジェクト設定中、e^2 studio ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
CTSU Framework Interface	synergy/ssp/inc/framework/api/sf_touch_ctsu_api.h
CTSU Framework Instance	synergy/ssp/inc/framework/instances/sf_touch_ctsu.h
CTSU Framework Driver	synergy/ssp/src/framework/sf_touch_ctsu
CTSU HAL Interface	synergy/ssp/inc/driver/spi/r_ctsu_api.h
CTSU HAL Instance	synergy/ssp/inc/driver/instances/r_ctsu.h
CTSU HAL Driver	synergy/ssp/src/driver/r_ctsu

4.1.5.5 CTSU フレームワークでサポートされるデバイス

このドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S7G2

- S3A7
- S124

CTSU フレームワークは、API への変更なしに、CTSU 周辺機器と CTSU ドライバを使用するすべての MCU をサポートするように設計されています。

4.1.6 静電容量タッチボタンフレームワーク

静電容量タッチボタンフレームワークは、ThreadX RTOS を使用する静電容量タッチボタンアプリケーション用の汎用 API で、sf_touch_ctsu ボタンに実装されています。静電容量タッチボタンフレームワークでは、MCU で静電容量タッチフレームワークと CTSU ドライバモジュールが必要です。

このフレームワークは、Workbench 6 ツールによって生成された構成データと組み合わせて使用するために設計されています。ボタンやイベント、およびその他の項目は Workbench ツールで設定します。

このセクションでは、e^2 studio ISDE を使用して静電容量タッチボタンフレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Input] > [Cap Touch Button Framework on sf_touch_ctsu_button] を選択することで、静電容量タッチボタンフレームワークモジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による静電容量タッチボタンフレームワークを使用するアプリケーションの作成。

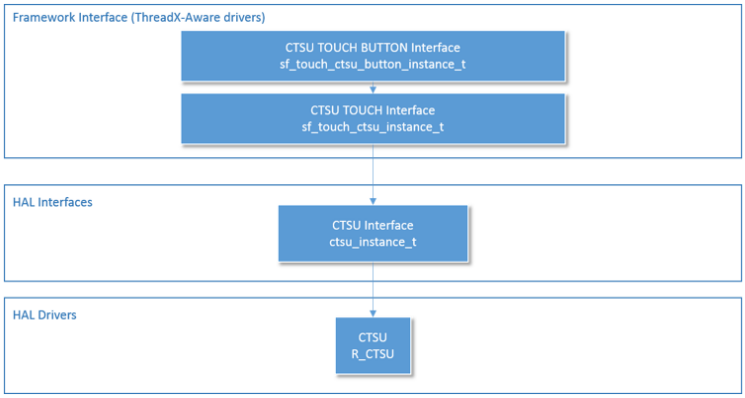

図 99: CTSU タッチボタンインタフェースブロック図

API リファレンスは、次の CTSU ボタンフレームワークインタフェースの説明内に記載されています :CTSU ボタンフレームワークインタフェース

静電容量タッチフレームワーク、CTSU ドライバも参照してください。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

4.1.6.1 静電容量タッチボタンフレームワークの機能

静電容量タッチボタンフレームワークインタフェースは、システムに存在するすべてのボタンの CTSU データを解釈するために使用されます。また、CTSU フレームワークレイヤーの初期化も行います。静電容量タッチボタンフレームワークには、処理されたデータが使用可能になるたびに CTSU フレームワークレイヤーを伴って呼び出されるコールバックがあります。静電容量タッチボタンフレームワークはこの処理済みのデータ（バイナリ）を使用してデバウンシングを実行して、設定されたイベント（Press、Release、LongTouch など）のいずれが各ボタンに対して有効かを判断し、各ボタンのコールバックをボタン構成テーブルに列挙された順に呼び出します。

4.1.6.2 e^2 studio ISDE による静電容量タッチボタンフレームワークを使用するアプリケーションの作成

ドライバは、e^2 studio ISDE の SSP に組み込まれています（e^2 studio ISDE ユーザーガイドを参照）。

e^2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

静電容量タッチボタンフレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Framework CTSU Button Driver	Threads	[Framework] > [Input] > [Cap Touch Button Framework on sf_touch_button_ctsu]
Framework CTSU Driver	Threads	[Framework] > [Input] > [Cap Touch Framework on sf_touch_ctsu]
CTSU HAL Driver	Threads	-
CTSU Interrupts	Threads	アプリケーション例については、静電容量タッチボタンフレームワークの割り込みの設定

静電容量タッチボタンフレームワークのクロックの設定

CTSU ボタンフレームワークは CTSU HAL ドライバと同じ構成を使用します。CTSU クロック速度を、静電容量タッチ調整ツール Workbench 6 で選択したクロック速度と同じ値に設定します。

CTSU ドライバを参照してください。

静電容量タッチボタンフレームワークのピンの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。ピンをタッチセンサーピン TSxx として設定し、TSCAP 機能を有効にします。

CTSU ドライバを参照してください。

静電容量タッチボタンフレームワークの割り込みの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。[ICU] タブで 3 つの CTSU 割り込みをすべて同じプライオリティに設定して、これらの割り込みを有効にします。どのような値を設定しても構いません。

CTSU ドライバを参照してください。

静電容量タッチボタンフレームワークパラメータの設定

e^2 studio ISDE を使用して、SF タッチ CTSU ボタン上の SF タッチ CTSU ボタンドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

CTSU ボタンフレームワークを追加すると、ssp_cfg_framework ディレクトリに、各構成パラメータに対して選択した設定を含む sf_touch_ctsu_button_cfg.h ファイルが作成されます。

表 : CTSU ボタンフレームワークのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define SF_TOUCH_CTSU_BUTTON_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータチェック用のコードを含めるかどうかを制御します。
Number of buttons	#define SF_TOUCH_CTSU_BUTTON_CFG_USER_SUPPORTED_BUTTONS	1	Workbench 6 で設定されたボタンの数
Multi touch enable	#define SF_TOUCH_CTSU_BUTTON_CFG_MULTI_TOUCH_ENABLE	Enabled/Disabled	マルチタッチ検出を有効化 / 無効化します
Short hold debounce multiplier	#define SF_TOUCH_CTSU_BUTTON_CFG_SHORT_HOLD_DEBOUNCE_MULTIPLIER	1	ボタンの短押しイベントの乗数を指定します
Long hold debounce multiplier	#define SF_TOUCH_CTSU_BUTTON_CFG_LONG_HOLD_DEBOUNCE_MULTIPLIER	5	ボタンの長押しイベントの乗数を指定します

ISDE プロパティ	設定	設定値	説明
Enable stuck at condition detection	#define SF_TOUCH_CTSU_BUTTON_CFG_STUCK_IN_DEBOUNCE_MULTIPLIER	7	ボタンのスタックイベントの乗数を指定します

Note：どのような構成エラーも捕捉できるように、パラメータチェックを有効にした状態でアプリケーションを開発することをお勧めします。実稼働コードではパラメータチェックを無効化して、コードスペースと実行時間を節約できます。

SPI フレームワークは、SF_TOUCH_CTSU_Button_OpenAPI で sf_touch_ctsu_button_cfg_t 型のポインタを受け渡しすることにより設定されます。

CTSU ボタンフレームワークの構成

ISDE プロパティ	設定	設定値	説明
Name	-	Arbitrary symbol	ドライバ名
Lower Level CTSU Framework Name	p_lower_lvl_touch_framework	Name of lower level CTSU Framework	
Button Configuration	pp_button_cfgs		Workbench 6 によって生成されたボタン構成構造体の名前
Callback		Name of callback function created by user application	

静電容量タッチボタンフレームワークアプリケーションの作成

1) Workbench 6 によって生成されたデータを含むフォルダが ${PROJECT_ROOT}\src フォルダに存在し、ビルドに含まれていることを確認します。
2) [Threads] タブで新しいスレッドを作成するか、既存のスレッドに [Cap Touch Button Framework on sf_touch_button_ctsu] を追加します。スレッドの [Name] を確認します。
3) 同じスレッドで、[Cap Touch Framework on sf_touch_ctsu] を追加します。
4) SF タッチ CTSU モジュールの [Properties] タブを使用して、モジュールの名前 ** を確認します (CTSU フレームワークモジュールの設定の詳細については、CTSU フレームワークモジュールの使用上の注意を参照してください)。
5) 同じスレッドで、[CTSU Driver on r_ctsu] を追加します。

6) CTSU 上のタッチドライバの [Properties] タブを使用して、モジュールの名前 ** と [CTSU Configuration Used] を確認します。(CTSU HAL モジュールの設定の詳細については、CTSU HAL モジュールの使用上の注意を参照してください)。
7) [Cap Touch Button Framework on sf_ touch_button_ctsu] の [Properties] タブを使用して、このドライバの [Name] を設定します。
8) 提供されたローレベルフレームワーク名が SF CTSU タッチモジュールのエントリと同じであることを確認します。
9) [Number of Buttons] フィールドを、Workbench 6 で設定したボタンの数に設定します。
10) 短押しデバウンスの乗数を設定します。これは、より高い耐ノイズ性が必要でない限り、1 に設定する必要があります。
11) 長押しデバウンスの乗数を設定します。これは、短押しデバウンスの乗数より大きくする必要があります。長押しデバウンスの乗数により、アプリケーションにおける短押しと長押しの検出間の時間の比率を変更できます。
12) マルチタッチ機能を有効化 / 無効化します。
13) スタック検出機能を有効化 / 無効化します。この機能では、タッチされた状態とタッチされていない状態間の過度のジッターを検出できます。これは、環境内に過度の偶発的な接触またはノイズがある場合に発生する可能性があります。
14) フレームワークによって使用されるボタン構成の配列の名前（ボタン構成構造体名）を指定します (Workbench 6 によって生成)。これは、sf_touch_ctsu_button_individual_t オブジェクトへのポインタとして src/captouch_configs/button_configs/self_button_config.c にあります (sf_touch_ctsu_button_individual_t* touch_buttons[])
15) スライダフレームワークによって使用されるコールバックを指定します。関数のシグネチャーは、以下のとおりです。

```
void callback_name(sf_touch_ctsu_slider_callback_args_t * p_args);
```

このコールバックは、ボタンフレームワークからの通知を処理します。通知のリストは、sf_touch_button_state_t にあります。このコールバックは、通知が関連するボタンを判断するために、ボタン ID を切り替えます。このコールバックは、sf_touch_button_state_t にリストされているイベントの一部または全部を処理または無視できます。

16) [Generate Project Content] をクリックして、必要な構造体を生成します。
17) ssp_gen の下にある Thread_name.h ファイルを開いて、生成された構造体を確認します。
18) プロジェクトをビルドします。
19) ハードウェアを接続し、実行可能ファイルをダウンロードして実行します。
20) ボタンにタッチすると、そのボタンのコールバックが発生し、コールバックを発生させた正確なイベントを指定する引数が渡されます。
21) アプリケーションスレッドで SF_TOUCH_CTSU_BUTTON の Name.api->disable を使用して、すべてのボタンのコールバックを無効にします。

22) アプリケーションスレッドで SF_TOUCH_CTSU_BUTTON の Name.api->enable を使用して、すべてのボタンのコールバックを有効にします。

4.1.6.3 静電容量タッチボタンフレームワークの制限事項

このモジュールを使用するうえで既知の制限事項はありません。SSP のリリースノートも参照してください。

4.1.6.4 静電容量タッチボタンフレームワークファイル

プロジェクト設定中、e^2 studio ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
CTSU Button Framework Interface	synergy/ssp/inc/framework/api/sf_touch_ctsu_button_api.h
CTSU Button Framework Instance	synergy/ssp/inc/framework/instances/sf_touch_ctsu_button.h
CTSU Button Framework Driver	synergy/ssp/src/framework/sf_touch_ctsu_button
CTSU Framework Interface	synergy/ssp/inc/framework/api/sf_touch_ctsu_api.h
CTSU Framework Instance	synergy/ssp/inc/framework/instances/sf_touch_ctsu.h
CTSU Framework Driver	synergy/ssp/src/framework/sf_touch_ctsu
CTSU HAL Interface	synergy/ssp/inc/driver/spi/r_ctsu_api.h
CTSU HAL Instance	synergy/ssp/inc/driver/instances/r_ctsu.h
CTSU HAL Driver	synergy/ssp/src/driver/r_ctsu

4.1.6.5 静電容量タッチボタンフレームワークでサポートされるデバイス

このドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S7G2
- S3A7
- S124

CTSU ボタンフレームワークは、API への変更なしに、CTSU フレームワークを使用するすべての MCU をサポートするように設計されています。

4.1.7 静電容量タッチスライダフレームワーク

静電容量タッチスライダフレームワークは、ThreadX RTOS を使用する静電容量タッチスライダおよびホイールアプリケーション用の汎用 API で、sf_touch_ctsu_slider ボタンに実装されています。静電容量タッチスライダフレームワークでは、MCU で静電容量タッチフレームワークと CTSU ドライバモジュールが必要です。

このフレームワークは、Workbench 6 ツールによって生成された構成データと組み合わせて使用するために設計されています。スライダやホイール、チャネル、およびその他の項目は Workbench 6 ツールで設定します。

このセクションでは、e^2 studio ISDE を使用して静電容量タッチスライダフレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Input] > [Cap Touch Slider Framework on sf_touch_ctsu_slider] を選択することで、静電容量タッチスライダフレームワークモジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による静電容量タッチスライダフレームワークを使用するアプリケーションの作成。

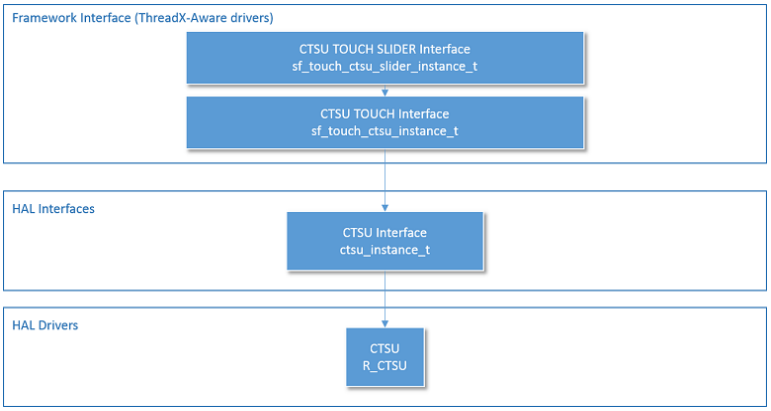

図 100: CTSU タッチスライダインタフェース - ブロック図

API リファレンスは、次の CTSU スライダフレームワークインタフェースの説明内に記載されています : CTSU スライダフレームワークインタフェース

静電容量タッチフレームワーク、CTSU ドライバも参照してください。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

4.1.7.1 静電容量タッチスライダフレームワークの機能

静電容量タッチスライダフレームワークインタフェースは、システムによって初期化されるすべてのスライダ構成の CTSU データを解釈するために使用されます。また、CTSU フレームワークレイヤーの初期化も行います。静電容量タッチスライダフレームワークには、処理されたデータが使用可能になるたびに CTSU フレームワークレイヤーを伴って呼び出されるコールバックがあります。スライダフレームワークは、このデータ（未加工値）を使用して、タッチまたは解放が発生したかどうか、また発生した場合はどこで発生したかを判断します。状態に変化があった場合、フレームワークはイベントと位置とともにスライダ構成テー

ブルに列挙された順に各スライダのコールバックを呼び出します。スライダフレームワークは、タッチイベントと解放イベント間の更新レート（sf_touch_ctsu configuration update_hz）でコールバックを実行します。アプリケーションコードは、これらのコールバックを使用してスライダ上の位置を追跡する必要があります。

スライダはまた、マルチタッチ検出もサポートしており、ユーザーアプリケーションはスライダと同時にスライダ以外のウィジェットがタッチされた場合にマルチタッチイベントのコールバックを取得します。この機能は、ビルド時にオプションで無効にできます。

4.1.7.2 e² studio ISDE による静電容量タッチスライダフレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

静電容量スライダボタンフレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Framework CTSU Slider Driver	Threads	[Framework] > [Input] > [Cap Touch Slider Framework on sf_touch_slider_ctsu]
Framework CTSU Driver	Threads	[Framework] > [Input] > [Cap Touch Framework on sf_touch_ctsu]
CTSU HAL Driver	Threads	-
CTSU Interrupts	Threads	アプリケーション例については、静電容量タッチスライダフレームワークの割り込みの設定

静電容量タッチスライダフレームワークのクロックの設定

CTSU スライダフレームワークは CTSU HAL ドライバと同じ構成を使用します。CTSU クロック速度を、静電容量タッチ調整ツール Workbench 6 で選択したクロック速度と同じ値に設定します。

CTSU ドライバを参照してください。

静電容量タッチスライダフレームワークのピンの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。ピンをタッチセンサーピン TSxx として設定し、TSCAP 機能を有効にします。

CTSU ドライバを参照してください。

静電容量タッチスライダフレームワークの割り込みの設定

CTSU フレームワークは CTSU HAL ドライバと同じ構成を使用します。[ICU] タブで 3 つの CTSU 割り込みをすべて同じプライオリティに設定して、これらの割り込みを有効にします。どのような値を設定しても構いません。

CTSU ドライバを参照してください。

静電容量タッチスライダフレームワークパラメータの設定

e^2 studio ISDE を使用して、SF タッチ CTSU ボタン上の SF タッチ CTSU スライダドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

CTSU スライダフレームワークを追加すると、ssp_cfg_framework ディレクトリに、各構成パラメータに対して選択した設定を含む sf_touch_ctsu_slider_cfg.h ファイルが作成されます。

表 : CTSU スライダフレームワークのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define SF_TOUCH_CTSU_SLIDER_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータチェック用のコードを含めるかどうかを制御します。
Number of sliders	#define SF_TOUCH_CTSU_SLIDER_CFG_USER_SUPPORTED_SLIDERS	1	Workbench 6 で設定されたスライダの数
Multi-Touch enable	#define SF_TOUCH_CTSU_SLIDER_CFG_MULTI_TOUCH_ENABLE	Enabled/Disabled	マルチタッチ検出を有効化 / 無効化します

Note: どのような構成エラーも捕捉できるように、パラメータチェックを有効にした状態でアプリケーションを開発することをお勧めします。実稼働コードではパラメータチェックを無効化して、コードスペースと実行時間を節約できます。

フレームワークは、SF_TOUCH_CTSU_Slider_OpenAPI で sf_touch_ctsu_slider_cfg_t 型のポインタを受け渡しすることにより設定されます。

CTSU スライダフレームワークの設定

ISDE プロパティ	設定	設定値	説明
Lower Level CTSU Framework Name	p_lower_lvl_touch_framework	Name of lower level CTSU Framework	

ISDE プロパティ	設定	設定値	説明
Slider/Wheel Configuration Structure Name	p_extend		Workbench 6 によって生成されたスライダ構成構造体の名前
Callback		Name of callback function created by user application	

静電容量スライダボタンフレームワークアプリケーションの作成

1) Workbench 6 によって生成されたデータを含むフォルダが ${PROJECT_ROOT}\src フォルダに存在し、ビルドに含まれていることを確認します。
2) [Threads] タブで新しいスレッドを作成するか、既存のスレッドに [Cap Touch Slider/Wheel Framework on sf_touch_slider_ctsu] を追加します。スレッドの [Name] を確認します。
3) sf_touch_ctsu 上の静電容量タッチフレームワークと r_ctsu 上の CTSU ドライバを含む、スライダフレームワークに必須のすべてのローレベルモジュールは、この時点で自動的に追加されます。
4) モジュールがすでにプロジェクトに追加されている場合（ボタンフレームワークなど）は、これら追加済みのモジュールはピンク色の境界線で囲まれた形で表示されます。これらのモジュールを再利用するには、ピンク色のブロックを右クリックし、[Use Existing] オプションを選択します。
5) CTSU フレームワークモジュールと CTSU HAL モジュールの設定の詳細については、CTSU フレームワークモジュールの使用上の注意を参照してください。
6) [Cap Touch Slider Framework on sf_touch_slider_ctsu] の [Properties] タブを使用して、このドライバの [Name] を設定します。
7) 提供されたローレベルフレームワーク名が SF CTSU タッチモジュールのエントリと同じであることを確認します。
8) [Number of Sliders] フィールドを、Workbench 6 で設定したスライダの数に設定します。
9) マルチタッチ機能を有効化 / 無効化します。
10) フレームワークによって使用されるスライダ / ホイール構成の配列の名前（スライダ / ホイール構成構造体名）を指定します（Workbench 6 によって生成）。これは、sf_slider_on_ctsu_cfg_t オブジェクトへのポインタとして src/captouch_configs/slider_configs/self_slider_config.c にあります（sf_slider_on_ctsu_cfg_t* all_sliders[])
11) スライダフレームワークによって使用されるコールバックを指定します。関数のシグネチャーは、以下のとおりです。

```
void callback_name(sf_touch_ctsu_slider_callback_args_t * p_args);
```

このコールバックは、スライダフレームワークからの通知を処理します。通知のリストは、sf_touch_ctsu_slider_state_t にあります。このコールバックは、通知が関連するスライダを判断するために、スライダ ID を切り替えます。このコールバックは、sf_touch_ctsu_slider_state_t にリストされているイベントの一部または全部を処理または無視できます。一部のイベントは同時に発生する場合があります。特に、マルチタッチイベントの

SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH は SF_TOUCH_CTSU_SLIDER_STATE_TOUCHED または SF_TOUCH_CTSU_SLIDER_STATE_HELD と同時に発生する可能性があります。コールバックコードは、マルチタッチイベントを無視するか、イベント引数をビットフィールドとして処理することで組み合わせイベントをチェックすることができます。コード例：

```
    switch(p_args->event)
    {
         case SF_TOUCH_CTSU_SLIDER_STATE_TOUCHED:
        case SF_TOUCH_CTSU_SLIDER_STATE_TOUCHED | SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH:
        if(0 != (p_args->event & (uint32_t)SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH))
                {
                    printf("\n Wheel initial multi-touch detected at %d \n", (int)p_args->current_position);
                }
                else
                {
                    printf("\n Wheel initial touch detected at %d \n", (int)p_args->current_position);
                }
                break;
         case SF_TOUCH_CTSU_SLIDER_STATE_HELD:
         case SF_TOUCH_CTSU_SLIDER_STATE_HELD | SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH:
                if(0 != (p_args->event & (uint32_t)SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH))
                {
                    printf("\n Wheel multi-touch sustained at %d \n", (int)p_args->current_position);
                }
                else
                {
                    printf("\n Wheel touch sustained at %d \n", (int)p_args->current_position);
                }
                break;
         case SF_TOUCH_CTSU_SLIDER_STATE_RELEASED:
                printf("\n Wheel released \n");
                break;
         default:
                break;
}
```

12) [Generate Project Content] をクリックして、必要な構造体を生成します。

13) ssp_gen の下にある **Thread_name.h** ファイルを開いて、生成された構造体を確認します。

14) プロジェクトをビルドします。

15) ハードウェアを接続し、実行可能ファイルをダウンロードして実行します。

16) スライダにタッチすると、そのスライダのコールバックが発生し、コールバックを発生させた正確なイベントを指定する引数が渡されます。

17) アプリケーションスレッドで SF_TOUCH_CTSU_BUTTON の **Name.api->disable** を使用して、すべてのスライダのコールバックを無効にします。

18) アプリケーションスレッドで SF_TOUCH_CTSU_BUTTON の Name.api->enable を使用して、すべてのスライダのコールバックを有効にします。

4.1.7.3 静電容量タッチスライダフレームワークの制限事項

このフレームワークは、セルフキャパシタンス動作モードで配置された静電容量タッチスライダのみをサポートします。それ以外には、このモジュールを使用するうえで既知の制限事項はありません。SSP のリリースノートも参照してください。

4.1.7.4 静電容量タッチスライダフレームワークファイル

プロジェクト設定中、e^2 studio ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
CTSU Slider Framework Interface	synergy/ssp/inc/framework/api/sf_touch_ctsu_slider_api.h
CTSU Button Framework Instance	synergy/ssp/inc/framework/instances/sf_touch_ctsu_slider.h
CTSU Slider Framework Driver	synergy/ssp/src/framework/sf_touch_ctsu_slider
CTSU Framework Interface	synergy/ssp/inc/framework/api/sf_touch_ctsu_api.h
CTSU Framework Instance	synergy/ssp/inc/framework/instances/sf_touch_ctsu.h
CTSU Framework Driver	synergy/ssp/src/framework/sf_touch_ctsu
CTSU HAL Interface	synergy/ssp/inc/driver/spi/r_ctsu_api.h
CTSU HAL Instance	synergy/ssp/inc/driver/instances/r_ctsu.h
CTSU HAL Driver	synergy/ssp/src/driver/r_ctsu

4.1.7.5 静電容量タッチスライダフレームワークでサポートされるデバイス

このドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S7G2
- S3A7
- S124

CTSU スライダフレームワークは、API への変更なしに、CTSU フレームワークを使用するすべての MCU をサポートするように設計されています。

4.1.8 外部 IRQ フレームワーク

外部 IRQ フレームワークは、ThreadX RTOS による外部ピン割り込みを使用するアプリケーション用の汎用 API です。このフレームワークは、MCU で利用可能な外部 IRQ ピンをサポートし、sf_external_irq に実装されます。このセクションでは、e^2 studio ISDE を使用して外部 IRQ フレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Input] > [External IRQ Framework on sf_external_irq] を選択することで、外部 IRQ フレームワークモジュールを追加および構成できます。詳細については、以下を参照してください。e^2 studio ISDE による外部 IRQ フレームワークを使用するアプリケーションの作成

API リファレンスは、次のフレームワーク外部 IRQ インタフェースの説明内に記載されています : 外部 IRQ フレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

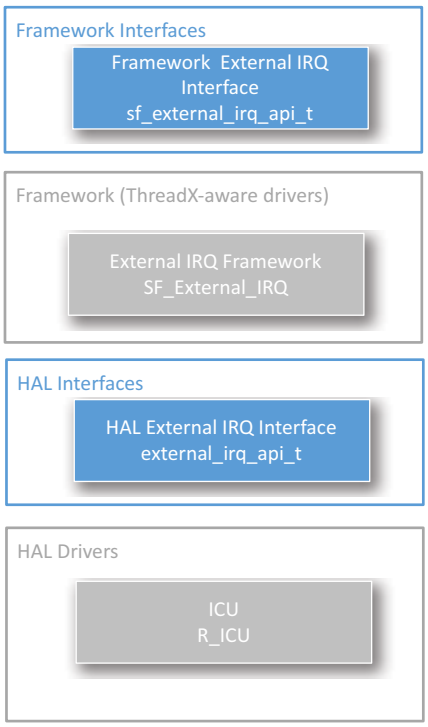

図 101: 外部 IRQ フレームワーク - ブロック図

関連項目：

- 外部 IRQ ドライバ

4.1.8.1 外部 IRQ フレームワークの機能

外部 IRQ フレームワークは、ThreadX 対応の一連のフレームワーク API です。外部 IRQ フレームワークを使用すると、スイッチなどの外部入力は、内部セマフォを通じてスレッドに信号を送信したり、イベントリンクコントローラ（ELC）を通じて転送をトリガすることができます。

4.1.8.2 e² studio ISDE による外部 IRQ フレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

外部 IRQ フレームワークを使用するアプリケーションでは、新しいスレッドを作成して、以下のリソースを追加します。

リソース	ISDE タブ	選択
Framework External IRQ Driver	Threads	[Framework] > [Input] > [External IRQ Framework on sf_external_irq]
HAL External IRQ Driver	Threads	[Driver] > [Input] > [External IRQ Driver on r_icu]。[Properties] ウィンドウで [Channel] プロパティを変更することにより、外部割り込みピンを選択できます。
Interrupts	Threads	[PORT] > [PORTn] > [PORTn IRQ]（n = 選択したチャネル）

外部 IRQ フレームワークのクロックの設定

外部 IRQ フレームワークには、固有のクロック構成は不要です。

外部 IRQ フレームワークのピンの設定

e² studio ISDE で、[Pins] タブを使用して外部 IRQ ピンの IRQn を設定します（ピンの設定を参照）。

e² studio ISDE のピンコンフィギュレータで、外部割り込みとして使用するピンを IRQn ピンとして設定します。[Pin Selection] ウィンドウに [IRQ] というドロップダウンボックスがあり、そこに外部割り込み機能をサポートするピンが一覧表示されます。このボックスで、このピンの外部割り込み機能を有効にできます。構成データは ssp_cfg/bsp/bsp_pin_cfg.h に書き込まれます。

Note: 外部割り込み機能をサポートするポートピンを調べるときや、特定のポートピンの外部 IRQ 番号が知りたいときには、プログラムする MCU のデータシートを参照してください。

Note: 外部 IRQ 番号は、e2 studio ISDE で外部 IRQ ドライバの [Properties] ウィンドウに示されるチャネルの設定に対応します。

外部 IRQ フレームワークの割り込みの設定

予想されるハードウェアイベントが発生したことをフレームワークドライバに通知するには、PORTn（n は IRQ 番号）の割り込みを BSP で有効にする必要があります。

Note: external_irq ICU モジュールの割り込みを有効にするには、e2 studio ISDE の [Project Configurator] の [ICU] タブで、[PORT]>[PORTn IRQ] 割り込み（n は IRQ 番号）の優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

外部 IRQ フレームワークモジュール

外部 IRQ ICU 周辺機器用の外部 IRQ フレームワークを設定するには、以下のモジュールを設定する必要があります。

1) 外部 IRQ ICU HAL モジュールを設定します。
2) 外部 IRQ フレームワークモジュールを設定します。

以降のセクションでは、これらの各モジュールの設定方法について説明します。

外部 IRQ ICU ドライバパラメータの設定

e2 studio ISDE を使用して、g_external_irq ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

ICU 上の External_IRQ ドライバを追加すると、ファイル sf_external_irq_cfg.h が ssp_cfg_framework ディレクトリに作成され、各構成パラメータの選択した設定が格納されます。

外部 IRQ フレームワークの設定 – sf_external_irq_cfg.h（共通パラメータ）

ISDE プロパティ	設定	設定値	説明
Parameter checking	#define SF_EXTERNAL_IRQ_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータチェック用のコードを含めるかどうかを制御します。

外部 IRQ ICU ドライバモジュールの設定

ISDE プロパティ	設定	設定値	説明
Channel	channel	0-15 for S7G2, 0-5 for S3A7, 0-7 for S124	使用するハードウェア IRQ チャネルを指定します。

ISDE プロパティ	設定	設定値	説明
Trigger condition	trigger	external_irq_trigger_t	エッジトリガまたはレベルトリガを設定します。
Digital Filtering	filter_enable	true/false	デジタルフィルタ有効 / 無効。
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	pclk_div	external_irq_pclk_div_t	ノイズフィルタサンプリング期間を設定します。
Interrupt enabled after initialization	autostart	true/false	-
Callback	p_callback	User-defined	ユーザーコールバック関数を open で登録できます。このコールバック関数が指定されている場合、IRQn がトリガされるたびに割り込みサービスルーチン（ISR）から呼び出されます。 Attention: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
-	p_context	User-defined	コールバックがユーザー定義データを受け取ることができるようにします。
-	p_extend	User-defined	ハードウェアに依存する設定。

Note：外部 IRQ フレームワークモジュールは、コールバックとコンテキストを内部で設定します。そのため、ユーザーが指定した値はすべて無視されます。

4.1.8.3 外部 IRQ フレームワークパラメータの設定

e^2 studio ISDE を使用して、外部 IRQ ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

外部 IRQ フレームワークモジュールの設定パラメータを設定する必要があります。モジュール「ICU 上の SF 外部 IRQ モジュール」を追加し、その中の各パラメータを設定します。

外部 IRQ フレームワークは、g_sf_external_irq_on_sf_external_irq.open()API で sf_external_irq_cfg_t 型のポインタを渡すことによって設定されます。

外部 IRQ フレームワークモジュールの設定

ISDE プロパティ	設定	設定値	説明
Lower Level API	p_lower_lvl_irq	external_irq_instance_t	外部 IRQ ICU モジュールへのローレベルインタフェースを指すポインタ。
Event	event	sf_external_irq_event_t	IRQ がトリガされたときに発生するイベント。例 : 内部セマフォをポスト。

4.1.8.4 外部 IRQ フレームワークアプリケーションの作成

以下で説明するアプリケーションは、ICU 周辺機器を使用するように外部 IRQ フレームワークのインスタンスを設定します。

ICU を使用する外部 IRQ フレームワークアプリケーションは、以下のようにして作成します。

1) RTOS を使用する Synergy プロジェクトを作成します。
2) [Threads] タブを選択し、前述のように以下の構造体を設定することで、モジュールを追加および設定します。
   - external_irq_cfg_t
   - sf_external_irq_cfg_t
3) 前述のように ICU を設定します。
4) e^2 studio ISDE でプロジェクトコンテンツを生成します。これにより、フレームワーク関連のヘッダーファイルと設定ファイルが自動的に作成されます。設定構造体の例を以下に示します。

参考資料


```
/* Configuration structure for low level external IRQ module */
external_irq_cfg_t g_external_irq_cfg =
{
    .channel = 0,
    .trigger = EXTERNAL_IRQ_TRIG_FALLING,
    .filter_enable = false,
    .pclk_div = EXTERNAL_IRQ_PCLK_DIV_BY_64,
    .autostart = false,
    .p_callback = NULL,
    .p_context = &g_external_irq,
    .p_extend = NULL,
};
/* Control structure for low level external IRQ module */
static external_irq_ctrl_t g_external_irq_ctrl;
/* Instance structure to use this module. */
const external_irq_instance_t g_external_irq =
{
    .p_ctrl = &g_external_irq_ctrl,
    .p_cfg = &g_external_irq_cfg,
    .p_api = &g_external_irq_on_icu
};
sf_external_irq_cfg_t g_sf_external_irq_cfg =
{
```

```
    .event = SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT,
    .p_lower_lvl_irq = &g_external_irq,
};
/* Instance structure to use this module. */
const sf_external_irq_instance_t g_sf_external_irq =
{
    .p_ctrl = &g_sf_external_irq_ctrl,
    .p_cfg = &g_sf_external_irq_cfg,
    .p_api = &g_sf_external_irq_on_sf_external_irq
};
```

5) プロジェクトをエラーなくビルドします。

6) ICU として実装された外部 IRQ を使用して、外部 IRQ フレームワークインスタンスをオープンします。外部 IRQ フレームワークは、次のインタフェースを使用してオープンします。

```
g_sf_external_irq.p_api->open(g_sf_external_irq.p_ctrl, g_sf_external_irq.p_cfg)
```

ここで、g_sf_external_irq.p_ctrl と g_sf_external_irq.p_cfg は、e^2 studio ISDE による外部 IRQ フレームワーク設定ステップの後に自動生成されます。外部 IRQ フレームワークをオープンすると、ICU デバイスのハンドルが返されます。上の例では g_sf_external_irq です。このハンドルを使用して、各種の External_IRQ 操作を実行します。以下に、open 操作を行った後に、このハンドルを使用して wait 操作を行うコード例を示します。

```
    ssp_err_t  err;

    err = g_sf_external_irq.p_api->open(g_sf_external_irq.p_ctrl, g_sf_external_irq.p_cfg);

  while (1)

    {

    /* Do something… */

    /* Wait (forever) for IRQ event to occur. */

    err =
    g_sf_external_irq.p_api->wait(g_sf_external_irq.p_ctrl, TX_WAIT_FOREVER);

}
```

4.1.8.5 外部 IRQ フレームワークの制限事項

このモジュールには既知の制限事項はありません。最新のソフトウェアバージョンの詳細については、SSP のリリースノートを参照してください。

4.1.8.6 外部 IRQ フレームワークファイル

プロジェクト設定中、e^2 studio ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
External IRQ Framework Instance	synergy/ssp/inc/framework/instances/sf_external_irq.h
External IRQ Framework Driver	synergy/ssp/src/framework/sf_external_irq
External IRQ HAL Interface	synergy/ssp/inc/driver/api/r_external_irq_api.h
External IRQ HAL Instance	synergy/ssp/inc/driver/instances/r_external_irq.h
ICU Driver	synergy/ssp/src/driver/r_icu

4.1.8.7 外部 IRQ フレームワークでサポートされるデバイス

外部 IRQ フレームワークは、API への変更なしに、外部 IRQ 入力および HAL ドライバを使用して、すべての MCU をサポートするように設計されています。

このドライバは、S7G2 でテストされています。

外部 IRQ フレームワークは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.1.9 FileX 適応フレームワーク

SSP ファイルシステムモジュールは以下の 4 つのコンポーネントで構成されています。

1) FileX
2) FileX ポートブロックメディアフレームワーク
3) ブロックメディアインタフェース / 適応ドライバ
4) メディアドライバ

このセクションでは、SSP FileX モジュールを使用してファイルシステムアプリケーションを作成する方法について説明します。

e^2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [Framework] > [File system] > [FileX Port Block Media Framework on sf_el_fx] を選択することで、ファイルシステムを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による FileX 適応フレームワークを使用するアプリケーションの作成。

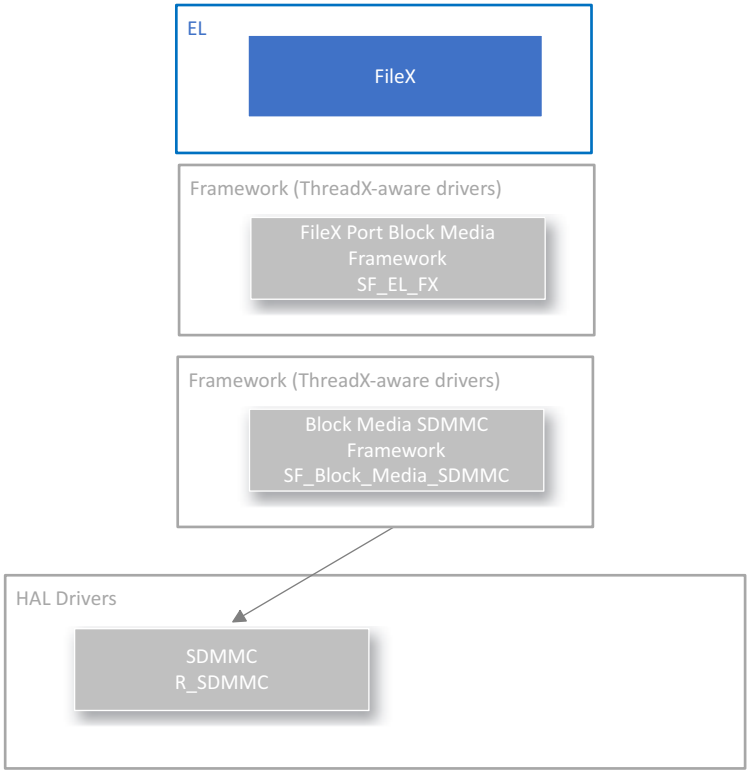

図 102: FileX 適応フレームワーク – ブロック図

4.1.9.1 FileX

FileX は、ディープエンベデッドアプリケーション用の完全なファイルアロケーションテーブル（FAT）フォーマットメディアであり、ファイル管理システムでもあります。

FileX は以下のメディアデバイスをサポートします。同時にサポートできるデバイスの数に制限はありません。

- RAM ディスク
- FLASH マネージャー
- その他複数の物理デバイス

FileX は、12 ビット、16 ビット、および 32 ビットの FAT フォーマットと連続的なファイル割り当てをサポートします。FileX はサイズと性能の両面で高度に最適化されています。

FileX モジュールの API リファレンスは次の場所にあります :Express Logic FileX API リファレンス

4.1.9.2 FileX ポートブロックメディアフレームワーク（FileX I/O ドライバ）

FileX ポートブロックメディアモジュール（sf_el_fx）は、FileX がブロックメディアインタフェースと適応レイヤーを介して Synergy メディアドライバにアクセスするための I/O 呼び出しを提供します。

FileX ポートブロックメディアフレームワークモジュールの API リファレンスは次の場所にあります :FX_IO フレームワーク

4.1.9.3 ブロックメディアフレームワーク

ブロックメディアフレームワークインタフェースは単なる抽象インタフェースであり、直接関数呼び出しの代わりに関数ポインタを使用します。関数は FileX と SSP ブロックメディアドライバ（SDMMC や SPI フラッシュなど）の間で呼び出されます。このインタフェースはどのメディアドライバに対しても変わらないため、すべてのメディアドライバがファイル I/O レイヤーで機能的に同一に見え、コードを変更せずにメディアドライバを互いに交換できます。アプリケーションはブロックメディアフレームワークインタフェースを介してデバイス適応ドライバ（sf_block_media_sdmmc など）にアクセスし、これらのドライバがメディア I/O 操作の実行に必要なデバイス固有のコードを提供します。ブロックメディア関数呼び出しによって渡される構成構造体と制御構造体も、通常はデバイス固有です。

ブロックメディアインタフェースの API リファレンスは次の場所にあります : ブロックメディアフレームワークインタフェース

フレームワークブロックメディアドライバは、SD/MMC メディアドライバを介して SD カード、eMMC 組み込みデバイス、SDHI（SD ホストインタフェース）周辺デバイスなどの読み取り、書き込み、および制御を実行するために SD/MMC バスプロトコルを実装できます。このドライバは、ブロックメディアインタフェースを介してファイルシステムとやり取りするために必要なすべての機能を備えています。

SD/MMC ドライバには以下の機能があります。

- S7G2 上の 2 チャネル SD/MMC ホストインタフェース SDHI をサポート
- SDSC（SD 標準容量）、SDHC（SD 大容量）、および eMMC（組み込み）をサポート
- 1 ビット、4 ビット、または 8 ビット（eMMC のみ）のデータバスをサポート

フレームワークブロックメディアドライバは、Synergy マイクロコントローラハードウェア上の SDMMC 周辺デバイスをサポートしています。

4.1.9.4 e² studio ISDE による FileX 適応フレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

FileX ドライバ（fx）は ThreadX RTOS に依存しています。FileX フレームワークモジュールを使用するアプリケーションでは、新しいスレッドを作成して、以下のリソースを追加します。

リソース	ISDE タブ	選択
FileX	Threads	[Framework] > [File system] > [fx 上の FileX]
FileX Port Block media Framework	Threads	[Framework] > [File system] > [sf_el_fx 上の FileX ポートブロックメディアフレームワーク]
Block Media Framework	Threads	[Framework] > [File system] > [sf_block_media_sdmmc 上のブロックメディアフレームワーク]
SDMMC Driver	Threads	[Framework] > [Driver] > [Storage] > [r_sdmmc 上の SD/MMC ドライバ]

また、[Framework] > [File system] > [FileX Source] の順に選択して Express Logic FileX ソースファイルを追加することも可能です。FileX ソースファイルの表示が許可されているかどうかはライセンスファイルを確認してください。

FileX 適応フレームワークの SD/MMC モジュールの設定

FileX アプリケーションは、SDMMC HAL ドライバを使用して Synergy マイクロコントローラ上の SDMMC 周辺デバイスにアクセスします。SDMMC API 関数にアクセスするには、SDMMC インタフェース API を使用します。SDMMC の API リファレンスは次の場所にあります : SDMMC インタフェース。

FileX 適応フレームワークの SD/MMC クロックの設定

SDHI は PCLKA をそのクロックソースとして使用します。データレートを最適化する必要がない限り、SDMMC 周辺デバイスに対して固有のクロックを設定する必要はありません。SDMMC ドライバは、PCLKA 周波数と、SD または eMMC デバイスで許可される最大クロックレート（これはメディアデバイスの初期化時に取得されます）に基づいて、適切な内蔵分周器を選択します。

FileX 適応フレームワークの SD/MMC ピンの設定

e^2 studio ISDE ピンコンフィギュレータを使用して、SDMMC 周辺デバイスの出力ピンを設定します。

FileX 適応フレームワークの SD/MMC パラメータの設定

e^2 studio ISDE を使用して、**SDMMC** ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

e^2 studio ISDE によって /synergy_cfg/ssp_cfg/driver ディレクトリに r_sdmmc_cfg.h ファイルが作成され、各共通構成パラメータに対して選択した設定がこのファイルに格納されます。

SDMMC には、以下のコンポーネントの構成が必要です。

- SDMMC 共通構成
- SDMMC_on_SDMMC:sdmmc_hw_t

SD/MMC の共通構成パラメータ

ISDE プロパティ	設定	設定値	説明
Parameter Checking Enable	#define SDMMC_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータエラーチェックを有効または無効にします。

SD/MMC ICU 構成パラメータ

ISDE プロパティ	設定	設定値	説明
SDHIMMC ACCS	-	Disabled (default) priority level 0-15	SD および eMMC、SDIO への読み取り、書き込み、エラーに必要となる割り込み優先順位。有効にすることで、割り込みコールバックを呼び出します。
SDHIMMC CARD	-	Disabled (default) priority level 0-15	（オプション）カード取り外しの割り込み。カードが取り外された際、自動でデバイスを閉じます。有効にすることで、割り込みコールバックを呼び出します。
SDHIMMC SDIO	-	Disabled (default) priority level 0-15	SDIO の読み取り、書き込み、エラーに必要となる割り込みプライオリティレベル。有効にすることで、割り込みコールバックを呼び出します。メモリカードに使用されません。

SD/MMC の設定 sdmmc_hw_t

ISDE プロパティ	設定	設定値	説明
Name	-	Arbitrary symbol (Default: "g_sdmmc")	SDMMC モジュール制御ブロックインスタンスに使用する名前。この名前は、他の可変インスタンスのプレフィックスにも使用されます。

ISDE プロパティ	設定	設定値	説明
Channel	channel	0,1	SDMMC 周辺機器のチャネル（チャネル 0 または 1）
Media Type	media_type	Card, Embedded	メディアは、カードと組み込みデバイスのどちらかです。この設定により、カードの挿入 / 取り出しがあるのか、あるいは書き込み保護ピンが存在するのかをファームウェアに知らせます。
Bus Width	bus_width	1 Bit, 4 Bits, 8 Bits	ハードウェアインタフェースによって定義されたバス幅。（8 ビットは eMMC のみ）
Lower level Transfer name	transfer_instance_t	NULL (default) Name of transfer module	（必須）r_dmac または r_dtc モジュールで転送ドライバ名に設定します。r_dmac を使用する場合は r_dmac の割り込みを有効化します。他の転送パラメータ（モード、転送サイズ、転送先など）を設定する必要はありません。
Callback	-	NULL (default) Name of user callback function.	（FileX を使用する場合は不要）ユーザーコールバック関数の名前に設定します。Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR

必要な割り込みは次のとおりです。

DTC で SD/MMC を使用 :

- SDHIMMCx ACCS

- SDHIMMCx DMA REQ

DMAC で SD/MMC を使用 :

- SDHIMMCx ACCS
- DMACx（DMAC 転送割り込み）

DTC で SDIO を使用 :

- SDHIMMCx ACCS
- SDHIMMCx SDIO
- SDHIMMCx DMA REQ

DMAC で SDIO を使用 :

- SDHIMMCx ACCS
- SDHIMMCx SDIO
- DMACx（DMAC 転送割り込み）

SDHIMMCx CARD はオプションです。

x= 使用するチャネル

FileX 適応フレームワークのブロックメディアパラメータの設定

e2 studio ISDE を使用して、sf_block_media_sdmmc ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

e2 studio ISDE によって /synergy_cfg/ssp_cfg/framework ディレクトリに sf_block_media_sdmmc_cfg.h ファイルが作成され、各共通構成パラメータに対して選択した設定がこのファイルに格納されます。

sf_block_media_sdmmc には、以下のコンポーネントの構成が必要です。

- sf_block_media_sdmmc 共通構成
- sf_block_media_cfg_t

ブロックメディアの共通構成パラメータ

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define BLOCK_MEDIA_SDMMC_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータエラーチェックを有効または無効にします。

ブロックメディア設定 sf_block_media_cfg_t

ISDE プロパティ	設定	設定値	説明
Name	-	Arbitrary symbol (Default: "g_sf_block_media_sdmmc")	sf_block_media_sdmmc モジュール制御ブロックインスタンスに使用する名前。
Block size of media in bytes	block_size	512	メディアブロックサイズ
Lower Level SDMMC Name	p_extend	Arbitrary symbol (Default: " g_sdmmc ")	ローレベルインスタンス名。これはローレベルメディアドライバの名前になります（この場合は g_sdmmc）。

FileX ポートブロックメディアフレームワークのパラメータの設定

e2 studio ISDE を使用して、sf_el_fx 上の FileX I/O ドライバ FileX ポートブロックメディアフレームワークパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

e2 studio ISDE によって /synergy_cfg/ssp_cfg/framework ディレクトリに sf_el_fx_cfg.h ファイルが作成され、各共通構成パラメータに対して選択した設定がこのファイルに格納されます。

sf_el_fx には、以下のコンポーネントの構成が必要です。

sf_el_fx 共通構成

sf_el_fx_t sf_el_fx

sf_el_fx の共通構成パラメータ

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define SF_EL_FX_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータエラーチェックを有効または無効にします。

sf_el_fx 構成 sf_el_fx_t

ISDE プロパティ	設定	設定値	説明
Name	-	Arbitrary symbol (Default: " g_sf_el_fx ")	sf_el_fx モジュール制御ブロックインスタンスに使用する名前。

ISDE プロパティ	設定	設定値	説明
Block media driver	-	On SDMMC	これは sf_block_media_sdmmcapi を選択します。現時点では、これが可能な唯一の設定です。
Lower level block media name	p_lower_lvl_block_media	Arbitrary symbol (Default: " g_sf_block_media_sdmmc ")	ローレベルインスタンス名。これはローレベルブロックメディアドライバの名前になります（この場合は g_sf_block_media_sdmmc）。

FileX 適応フレームワークの fx パラメータの設定

e2 studio ISDE を使用して、fx 上の FileX ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

各設定のスレッド初期化ソースファイルのコードは e2 studio ISDE によって生成されます。

FileX の設定

設定	設定値	説明
Name	Arbitrary symbol (Default: " g_fx_media ")	fx モジュール制御ブロックインスタンスに使用する名前。
sf_el_fx_t	Arbitrary symbol (Default: " g_sf_el_fx")	ローレベル インスタンス名。これはローレベル sf_el_fx ドライバの名前になります （この場合は g_sf_el_fx.）。
Format media during initialization	Enabled, Disabled	有効にした場合、初期化時にメディアをフォーマットするコードをコンパイルします。
File System is on SDMMC	True, False	これを true に設定すると、フォーマットする前にメディアデバイスからセクターサイズとセクター数を読み取るコードが使用されます。false に設定すると、下記の構成で指定したセクターサイズとセクター数がフォーマット関数で使用されます。

設定	設定値	説明
Volume Name	Volume 1	これはメディアデバイスのフォーマット時に使用するボリューム名です。11 文字以下にする必要があります。
Number of FATs	1	ファイルアロケーションテーブルの数。
Directory Entries	256	許可されるディレクトリエントリの数。
Hidden Sectors	0	隠しセクターの数。
Total Sectors	3751936	フォーマット関数の使用時にフォーマットするセクターの数。この値を使用するには、[ファイルシステムが SDMMC 上に存在] を false に設定する必要があります。
Bytes per sector	512	フォーマット関数の使用時にフォーマットするセクターサイズ。この値を使用するには、[ファイルシステムが SDMMC 上に存在] を false に設定する必要があります。
Sectors per Cluster	1	クラスタあたりのセクター数。この値は SD または eMMC メディアでは使用されません。
Working Media Size	512	これは FileX がメディアの読み書きに使用するバイト数です。1 セクターのサイズ以上にする必要があります。

4.1.9.5 FileX 適応フレームワークを使用した SD/MMC アプリケーションの作成

4 つのコンポーネントを設定してコードが生成されると、完全なファイルシステムが使用可能になります。

ファイルの作成、書き込み、および読み取りを行うためには、メディアを少なくとも 1 回フォーマットする必要があります。

FileX は fx_media_open 関数を使用してメディアを開きます。ユーザーコードでこの関数を呼び出さなければならないのは、リセット後の一度だけです。メディアを閉じる必要はありません。このコードは、スレッド初期化ソースファイル内に生成されます。

FileX 関数の詳細については、FileX ユーザーガイドを参照してください。

4.1.9.6 FileX 適応フレームワークでサポートされるデバイス

このドライバは、S7G2 と S3A7 でテストされています。

4.1.10 GUIX™ Synergy ポートモジュール

SSP フレームワークレイヤー内の GUIX™ Synergy ポートモジュール（SF_EL_GX）は、GUIX™ ディスプレイドライバインタフェースを介して SSP グラフィックスデバイスドライバを Express Logic GUIX™ に結び付けます（GUIX™ ユーザーガイドの第 5 章「GUIX™ ディスプレイドライバ」を参照）。このモジュールは、デバイスへのアクセスを相互排除するため、およびグラフィックス用画像データのレンダリング動作と表示動作とのタイミングを同期させるために、ThreadX サービス呼び出しを使用します。

このモジュールは、2DG および JPEG モジュール用の RTOS 対応デバイスドライバと、ディスプレイ HAL デバイスドライバ（通常は GLCDC モジュール）を使用します。下図に、Synergy グラフィックスソリューションのコンポーネントと、このソリューションでのグラフィックスデータの流れを示します。

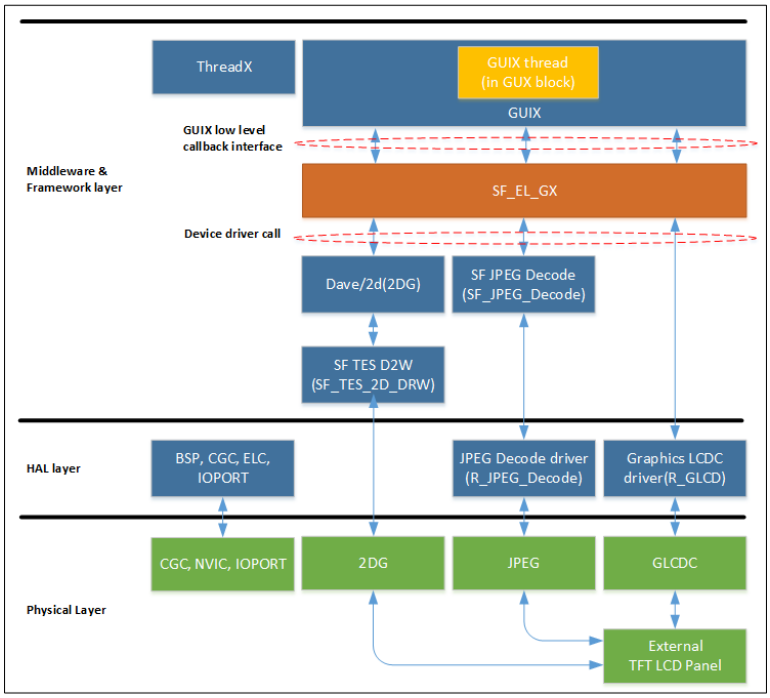

図 103: GUIX™ Synergy SF_EL_GX を使用した SSP グラフィックスソリューションのブロック図

e² studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで > [New] > [Framework] > [Graphics] > [GUIX Port on sf_el_gx] を選択することで、GUIX Synergy ポートモジュールを追加および構成できます。詳細については、以下を参照してください：e² studio ISDE による GUIXTM Synergy ポートモジュールを使用するアプリケーションの作成。

API リファレンスは、次の GUIX™ Synergy ポートモジュールインタフェースの説明内に記載されています（GUIXTM インタフェースを参照）。

関連インタフェースの API リファレンス：

- ディスプレイインタフェース
- JPEG デコードフレームワークインタフェース

Note：このドキュメントでは、GUIXTM の GUIXTM Synergy ポートモジュールについて説明します。

4.1.10.1 GUIX™ Synergy ポートモジュールの機能

フレームワークは以下の機能をサポートしています。

- GUIX™ を SSP の一番上に適応する
- SSP ディスプレイインタフェースドライバを GUIX™ ディスプレイドライバインタフェースに設置する
- D2W（2DG）エンジンにより、GUIX™ が高速でウィジェットを描画できるようにする
- JPEG エンジンにより、GUIX™ が高速でウィジェットを描画できるようにする
- 切れ目が発生することなく画面の移行を可能とする、ダブルバッファ切り替え制御をサポートする
- ユーザーコールバック関数をサポートする

4.1.10.2 GUIX™ Synergy ポートモジュールのアーキテクチャ

SSP に準拠した API 設計

GUIX™ Synergy ポートモジュールの SF_EL_GX モジュールは、SSP フレームワークレイヤーで SSP に準拠した API セットを定義します。このモジュールは、GUIX™ 定義に一致した引数を持つ特別な API で構成されています。SSP 準拠の API と GUIX™ 準拠の API を組み合わせることで、GUIX™ を SSP ベースのグラフィックスシステムに適応させることができます。他の SSP フレームワークモジュールと同様に、SF_EL_GX モジュールも、モジュールインスタンスにアクセスするために下記のインタフェースを備えています。このインタフェースには、制御ブロック、構成ブロック、および API 関数が含まれます。詳細については、sf_el_gx_api.h の sf_el_gx_instance_t* 構造体を参照してください。p_api を介して SF_EL_GX モジュールのインスタンスにアクセスできます。詳細については、このドキュメントのコード例を参照してください。

```
typedef struct sf_st_el_gx_driver_instance
{
    sf_el_gx_ctrl_t  * p_ctrl

    sf_el_gx_cfg_t  const * p_cfg;

    sf_el_gx_api_t  const * p_api;

} sf_el_gx_instance_t;
```

SF_EL_GX モジュールは、以下の SSP 準拠 API で構成されています。

- open – SF_EL_GX モジュールを開きます。この API は、1 つのスレッドからのみ呼び出すことができます。この API は、sf_el_gx_api_t::p_cfg を渡してローレベルグラフィックスデバイスドライバとフ

レームバッファを設定し、ユーザーコールバック関数を登録します。この関数は実際にはローレベルドライバを初期化しません。ローレベルドライバの初期化は API setup によって行われます。その理由については、下記の setup の説明を参照してください。

- close – SF_EL_GX モジュールを閉じます。この API はローレベルドライバを閉じます。通常、GUIX™ は初期化後に閉じられないため、この API は呼び出されません。
- versionGet – モジュールのバージョンを返します。

SF_EL_GUIX には、GUIX™ ポートに以下の特別な API が含まれます。

- setup – これはローレベルグラフィックスデバイスドライバを初期化するためのインタフェースであり、GUIX™（Studio）サービス呼び出し gx_studio_display_configure() を介してこれを関数ポインタとして GUIX™ に渡す必要があります。続いて GUIX™ はこの API をコールバックし、そのときに open によって渡された構成に基づいて SSP デバイスドライバが設定されます。このような手順でローレベルドライバを初期化する理由は、この API は GUIX™ 準拠の引数（GX_DISPLAY*）型を持っており、この API では e² studio ISDE から生成された SSP グラフィックスデバイスドライバの詳細な構成を適用できないためです。
- canvasInit – これは GUIX™ キャンバスのメモリアドレスを決定する GUIX™ ヘルパー API です。この API には（GX_WINDOW_ROOT*）型の引数があり、キャンバスメモリの開始アドレスを GUIX™ に提供します。このアドレスは、ローレベルグラフィックスデバイスドライバがイメージを描画 / 表示するために必要となります。

ピンポンフレームバッファ管理

GUIX™ Synergy ポートモジュールは、ピンポンフレームバッファを備えたグラフィックスシステムでバッファ切り替え動作を管理します。下図に、SF_EL_GX モジュールによって実装されたピンポンバッファグラフィックスシステムを示します。このモジュールは、GUIX™ とローレベル周辺デバイスドライバを使用してイメージの描画（DRW（2DG）または JPEG）とイメージの表示（ディスプレイモジュール、たとえば GLCDC など）を行います。SF_EL_GX 構成では、単一のフレームバッファを持つ設計も可能です。しかし、2 つのフレームバッファを使用すると、単一フレームバッファシステムで起こり得る切れ目の問題がなくなります。

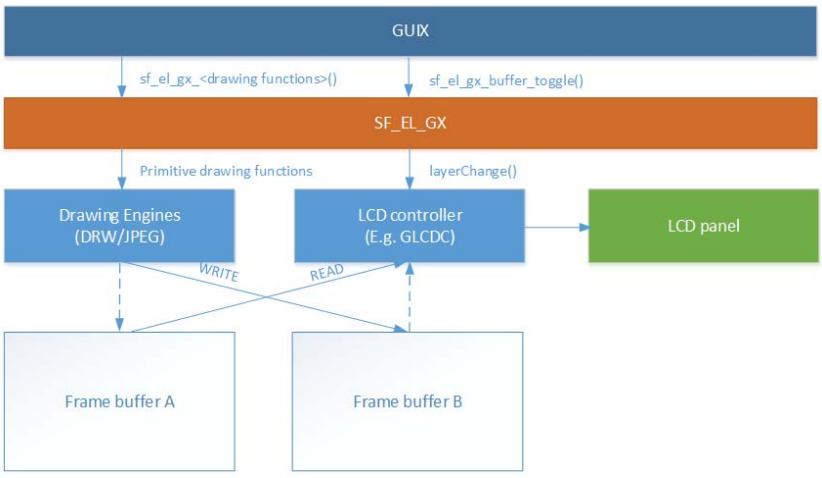

図 104:　ピンポンバッファシステム

上図は、GUIX™ キャンバスバッファを使用しない場合を示しています。画面ローテーション機能が有効な場合、GUIX はまず SF_EL_GX モジュールを介して GUIX キャンバスバッファに画面更新を描画します。その後、この画面はフレームバッファの背面にコピーされます。詳細については、画面のローテーションを参照してください。

4.1.10.3 e² studio ISDE による GUIX™ Synergy ポートモジュールを使用するアプリケーションの作成

フレームワークモジュールは、e² studio ISDE 内の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します （ドライバのスレッドへの追加とドライバの設定を参照）。

プロジェクトの設定の際に、e² studio ISDE で以下のモジュールを追加します （ドライバのスレッドへの追加とドライバの設定を参照）。

リソース	e² studio ISDE のタブ	選択
GUIX on gx (mandatory)	Threads	[Framework] > [Graphics] > [GUIX on gx]
GUIX Port on sf_el_gx (mandatory)	Threads	[Framework] > [Graphics] > [GUIX Port on sf_el_gx]
Display Driver (mandatory)	Threads	[Driver] > [Graphics] > [Display Driver on r_glcd]
D/AVE 2D Driver on dave2d (optional)	Threads	[Framework] > [Graphics] > [D/AVE 2D Driver on dave2d]
D/AVE 2D Port on sf_tes_2d_drw (optional)	Threads	[Framework] > [Graphics] > [D/AVE 2D Port on sf_tes_2d_drw]
JPEG_Decode Framework Driver (optional)	Threads	[Framework] > [Graphics] > [JPEG Decode Framework on sf_jpeg_ decode]
JPEG_Decode Driver (optional)	Threads	[Driver] > [Graphics] > [JPEG Driver on r_jpeg]

また、[Framework] > [Graphics] > [GUIX Source] の順に選択して GUIX™ ソース ファイルを追加することも可能です。この手順はオプションです。

GUIX™ ライブラリのデフォルト オプションを変更したい倍は、GUIX ソースをプロジェクトに追加します。以下の GUIX™ ソース オプションは、GUIX™ ソース モジュールの [Properties] ウィンドウで設定できます。

- マルチスレッド サポートの無効化 （Disable Multithread Support）：このオプションは、GUIX™ システム リソース保護のための GUIX™ システム ミューテックスを取得する GUIX™ 内部スレッドを無効化します。システムが単一のスレッドからのみ GUIX™ にアクセスする場合は、このオプションを有効にして、GUIX™ にリソース ロック手順を省略させることで、システム スループットを向上できます。このオプションは、システムのパフォーマンスを調整する必要がない限り、無効のままにしておくことが推奨されます。
- UTF8 サポートの無効化（Disable UTF8 Support）：このオプションは、GUIX™ で UTF8 形式の文字列エンコードを無効化し、8 ビット ASCII 文字と Latin-1 コードのページ文字エンコードのみを許可します。
- システムタイマ（ミリ秒）：このオプションは GUIX™ タイマレートを定義します。使用可能なレートは 10（ミリ秒）以上の整数です。

GUIX™ ソースファイルの表示が許可されているかどうかは SSP ライセンスを確認してください。特定のライセンスについては、こちらを参照してください : https://synergygallery.renesas.com/about

GUIX™ Synergy ポートモジュールのクロックの設定

GUIX™ Synergy ポートモジュールは、論理モジュールであるため、ARM Cortex-M コア SysTick タイマ設定を除く、クロックなどのハードウェア設定は不要です。

GUIX™ Synergy ポートモジュールのピンの設定

GUIX™ Synergy ポートモジュールは、論理モジュールであるため、ピンのハードウェア設定は不要です。

GUIX™ Synergy ポートモジュールのパラメータの設定

e² studio ISDE を使用して、ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

表 : GUIX™ の設定

ISDE プロパティ	設定	設定値	説明
GX_USE_SYNERGY_DRW	Synergy 2D Drawing Engine Support	Enabled(Default), Disabled	DRW によるハードウェア レンダリング（2DG）を有効、または無効にします。有効にすると、DRW ハードウェアにより GUIX™ ウィジェットの描画速度が上昇します。このオプションは、S7G2 デバイスファミリの場合のみ有効です。Synergy 2D 描画エンジンのサポートを参照してください。

ISDE プロパティ	設定	設定値	説明
GX_USE_SYNERGY_JPEG	Synergy JPEG Support	Enabled, Disabled (Default)	JPEG によるハードウェアレンダリングを有効、または無効にします。有効にすると、JPEG ハードウェアにより GUIX™ ウィジェットの描画速度が上昇します。このオプションは、S7G2 デバイスファミリの場合のみ有効です。Synergy JPEG のサポートを参照してください。

表 : SF_EL_GX の設定

ISDE プロパティ	設定	設定値	説明
SF_EL_GX_CFG_PARAM_CHECKING_ENABLE	Parameter Checking	Enabled (Default), Disabled	エラーパラメータチェックを有効または無効にします。
Name of SF_EL_GX instance	Name	Arbitrary symbol name(Default: "g_sf_el_gx")	ISDE が生成する SF_EL_GX インスタンスの名前。このモジュールのインスタンス名を指定します。名前は有効な C シンボルである必要があります。
Name of Display Driver Run-time Configuration	p_display_runtime_cfg	Arbitrary symbol name(Default: "g_display0_runtime_cfg_bg")	Synergy 構成で指定したディスプレイモジュールのランタイム設定名を指定します。名前には必ず有効な C シンボルを使用してください。NULL を設定することはできません。

ISDE プロパティ	設定	設定値	説明
Name of Frame Buffer A	p_framebuffer_a	Arbitrary symbol name(Default: g_display0_fb_background [0])	フレームバッファの名前を指定します。Synergy 設定のディスプレイモジュール設定には、作成するフレームバッファの名前が含まれています。フレームバッファの名前をここに設定します。名前には必ず有効な C シンボルを使用してください。NULL を設定することはできません。
Name of Frame Buffer B	p_framebuffer_b	Arbitrary symbol name(Default: g_display0_fb_background [1])	もう 1 つのフレームバッファの名前を指定します。シングルフレームバッファを使用するグラフィックシステムを設定する場合は、このパラメータに NULL を設定するか、またはパラメータ「フレームバッファ A の名前」を使用して同じフレームバッファ名を設定します。単一のバッファ設計における切れ目を参照してください。
Name of User Callback function	p_callback	Arbitrary symbol name (Default: “NULL”)	イベント発生時にモジュールによって呼び出されたユーザーコールバック関数の名前。必ず有効な C シンボルを使用してください。NULL も使用できます。
Screen Rotation Angle (Clockwise)	rotation_angle	0 (Default), 90, 180, 270	画面のローテーション角度（度）。0 以外の値が選択された場合、画面のローテーションが有効になり、GUIX™ は画面イメージを反時計回りに指定された角度回転させてフレームバッファに描画します。

ISDE プロパティ	設定	設定値	説明
GUIX Canvas Buffer	p_canvas	Not used (Default), Used	画面のローテーションを有効にする場合、キャンバスバッファを使用する必要があります。キャンバスバッファのサイズは、ディスプレイモジュールのフレームバッファとまったく同じでなければなりません。
Size of JPEG Work Buffer	jpegbuffer_size	Arbitrary integer value (Default: "768000")	JPEG 作業バッファサイズ (バイト単位)。値には必ず有効な整数値を使用してください。JPEG 高速化を使用しない場合はゼロも設定できます。バッファサイズが大きいほど描画時間が短縮されます。JPEG 作業バッファのサイズを参照してください。
Section name for GUIX Canvas Buffer	-	Arbitrary symbol name (Default: "sdram")	GUIX™ キャンバスバッファを割り当てるメモリセクション名。リンカースクリプトファイルで定義されている有効なセクション名を入力します。名前は有効な C シンボルである必要があります。
Section name for JPEG Work Buffer	-	Arbitrary symbol name (Default: "sdram")	JPEG 作業バッファを割り当てるメモリセクション名。リンカースクリプトファイルで定義されている有効なセクション名を入力します。名前は有効な C シンボルである必要があります。

4.1.10.4 GUIX™ Synergy ポートモジュールの設定上の注意

Synergy 2D 描画エンジンのサポート

Express Logic GUIX™ Studio（v5.2.8 以降）の [Configure Project] ウィンドウから、[Target CPU] 設定で [Renesas Synergy] が選択され、[Synergy Advanced Settings] ウィンドウの [Enable Graphics Accelerator] がチェックされていることを確認してください。

[Edit Pixel map] ウィンドウの Pixelmap 出力フォーマットについては、[Compress Output] を選択してください。[Raw Format] を選択しないでください。このように設定することで、GUIX™ Studio により Targa

RLE フォーマットでエンコードされた画像リソースデータが生成されます。D2W ハードウェアはこのフォーマットをリードできるうえに、フレームバッファに画像のデコードと描画を実行できます。

Synergy JPEG のサポート

GUIX™ Studio（v5.2.8 以降）の [Configure Project] ウィンドウから、[Target CPU] 設定で [Renesas Synergy] が選択され、[Decoder Types JPEG:] ドロップダウンメニューの [Hardware JPEG Decoder] がチェックされていることを確認してください。

[Exit Pixel map] ウィンドウの Pixelmap 出力フォーマットについては、[Raw Format] を選択してください。このように設定することで、GUIX™ Studio により、圧縮されていない JPEG エンコードされた画像リソースデータが生成されます。JPEG ハードウェアはこのフォーマットをリードできるうえに、フレームバッファに画像のデコードと描画を実行できます。

GUIX™ キャンバスバッファ

GUIX™ キャンバスバッファは、画面イメージのローテーションを実行するために使用されます。GUIX™ キャンバスバッファのサイズは、ディスプレイモジュールのフレームバッファとまったく同じでなければなりません。

画面のローテーション

GUIX™ Synergy ポートモジュールでは、GUIX™ によって描画された画面イメージを回転させることができます。この機能は、GUI 画面の設計などで、横長の形状にする必要があるが、ディスプレイパネルのハードウェアが縦長の画面になっている場合に有用です。このような場合、GUIX™ Studio で横長の GUI を設計し、このモジュールによって表示デバイスの画面を回転させて表示することができます。サポートされているローテーション角度は反時計回りに 90、180、270 度で、これは open で設定できます。動的な画面ローテーションはサポートされていません。画面のローテーション機能を有効にするには、GUIX™ キャンバスバッファを使用する必要があります。GUIX™ はまずキャンバスに画面更新を描画し、その後 GUIX™ ポートがフレームバッファへの画面コピーを反時計回りにイメージを回転させて処理します。Synergy 2D 描画エンジン（DRW）サポートが有効になっている場合、回転は Synergy DRW によってテクスチャマッピングで処理されます。有効になっていない場合、回転はソフトウェアコピーによって処理されます。

設定 : 画面のローテーション角度 = [0]。GUIX™ キャンバスバッファ = [Used] は許可されますが、画面イメージのコピーに余分なバス帯域を消費します。そのため、この場合 GUIX™ キャンバスバッファは [Not used] にしてください。その後、GUIX™ はメモリとバス帯域を保存できるよう、フレームバッファに直接、画面更新を描画します。

JPEG 作業バッファのサイズ

JPEG 作業バッファと JPEG のデコード速度は、バッファサイズに対してトレードオフの関係にあります。画面のウェジェットが JPEG でフォーマットされている場合は、JPEG 作業バッファは一時ストレージメモリとして使用され、デコードされた画像が作成されます。画像全体をデコードするにあたってバッファサイズの大きさが不足している場合は、JPEG デコーディングは出力バッファストリーミングモードで実行されます。DRW の BitBLT では、バッファの JPEG ラスターイメージのピースがデコードされ、その後フレームバッファに転送されます。

一時作業バッファを使用するのは、メモリアラインメントとピクセル数に JPEG ハードウェアの制限があり、またウィジェットにアルファ値が存在する場合にはアルファブレンディングが必要となるからです。

ウィジェット全体の描画は、上記の説明のように反復処理により行われるため、JPEG 作業バッファのサイズが小さい場合ではデータ転送オーバーヘッドによりデコード処理速度が低下します。バッファサイズが最小要件を満たしていないと、JPEG デコーディングは実行されません。

最小バイト数は、水平ラインのピクセル数 x ディスプレイフォーマットの bpp（1 ピクセルあたりのバイト数）x8（ライン）となる必要があります。たとえば、デコードした画像が水平ライン 800 ピクセルで RGB565 フォーマットの場合では、この数は 800 x 2 x 8= 12800（バイト）となります。スループットを向上させるには、[JPEG 作業バッファのサイズ] のパラメータを最小値よりも大きい値にしてください。

単一のバッファ設計における切れ目

一般的に、シングルフレームバッファを使用するシステムは、汎用 RGB インタフェースを採用した LCD パネルにおける切れ目問題の原因となります。ご使用のシステムがシングルフレームバッファを扱うように設計されている場合、このモジュールでは切れ目問題を回避できません。

4.1.10.5 e² studio ISDE で生成した GUIX™ インスタンスのコード例

SF_EL_GX モジュールのインスタンスは、e² studio Synergy コンフィギュレータによって生成されます。コード例を以下に示します。

```
/* Frame buffer (for GNU GCC build) */

uint8_t g_display_fb_background[2][768000] __attribute__((aligned(64), section(".sdram")));

/* GUIX Canvas buffer (for GNU GCC build) */
uint8_t g_sf_el_gx_canvas[sizeof(g_display_fb_background[0])] __attribute__ ((aligned(4), section(".sdram")));

/* JPEG Work buffer for decode processing (for GNU GCC build) */

uint8_t g_sf_el_gx_jpegbuffer[192000] __attribute__ ((aligned(64), section(".sdram")));

/* DISPLAY module control block */

static display_ctrl_t  g_display_ctrl;

/* DISPLAY module configuration */

static const display_cfg_t  g_display_cfg = {

… (omit the code sample) …

}

/* DISPLAY module run-time configuration */

display_runtime_cfg_t  g_display_runtime_cfg_bg = {

… (omit the code sample) …

}

/* DISPLAY module instance. */
```

```
const display_instance_t g_display = {
    .p_ctrl = &g_display_ctrl,
    .p_cfg = (display_cfg_t *) &g_display_cfg,
    .p_api = (display_api_t *) &g_display_on_glcd
};
/* SF_EL_GX module control block */
static sf_el_gx_ctrl_t g_sf_el_gx_ctrl;
/* SF_EL_GX module configuration */
static const sf_el_gx_cfg_t g_sf_el_gx_cfg = {
    .p_display_instance = (display_instance_t *) &g_display,
   .p_display_runtime_cfg = &g_display_runtime_cfg_bg,
    .p_canvas = g_sf_el_gx_canvas,
    .p_framebuffer_a = g_display_fb_background[0],
    .p_framebuffer_b = g_display_fb_background[1],
    .p_callback = user_callback_guix_driver,
    .p_jpegbuffer = g_sf_el_gx_jpegbuffer,
    .jpegbuffer_size = sizeof(g_jpeg_buffer)
    .screen_rotation = 90
};
/* SF_EL_GX module instance. */
sf_el_gx_instance_t g_sf_el_gx = {
    .p_api = &sf_el_gx_on_guix,
    .p_ctrl = &g_sf_el_gx_ctrl,
    .p_cfg = &g_sf_el_gx_cfg
};
```

ユーザー定義のコールバック関数例

GUIX™ Synergy ポートモジュールは、open を介してユーザー定義のコールバック関数を呼び出すことができます。このコールバック関数を利用して、ディスプレイデバイスのブランキング期間の開始タイミングを知って追加処理を実行したり、ローレベルデバイスドライバで発生したエラーを検出したりできます。コード例を以下に示します。

```
void user_callback_guix_driver(sf_el_gx_callback_args_t *p_arg)

{

    switch (p_arg->event)

    {

        case SF_EL_GX_EVENT_DISPLAY_VSYNC:

            /* Do processing */

            break;

        case SF_EL_GX_EVENT_UNDERFLOW:

            /* Do processing */

            break;

        case SF_EL_GX_EVENT_ERROR:

            if (SF_EL_GX_DEVICE_DISPLAY == p_arg->device)

      {
```

```
                    if(SSP_ERR_INVALID_LAYER_FORMAT == p_arg->error)
                    {
                        /* Do processing */
                    }
                }
                break;
            default:
            break;
        }
}
```

初期化コード例

GUIX™ Synergy ポートモジュールは、e² studio ISDE から生成されたインスタンスと構成を使用して初期化できます。コード例を以下に示します。

```
void <your thread>_thread_entry(void)
{
    /* Initializes GUIX. */
    gx_system_initialize();
    /* Initializes GUIX drivers. Call this API before calling gx_studio_display_configure() */
    g_sf_el_gx.p_api->open  (g_sf_el_gx.p_ctrl, g_sf_el_gx.p_cfg);
  /* Configures GUIX system. */
    gx_studio_display_configure (MAIN_DISPLAY,
                                 g_sf_el_gx.p_api->setup,
                      LANGUAGE_ENGLISH,
                                 MAIN_DISPLAY_THEME_1,
                                 &p_window_root);
```

```
    /* Initializes memory address of the canvas. Call this API after
    calling gx_studio_display_configure() and before calling
    gx_studio_named_widget_create() */

    g_sf_el_gx.p_api->canvasInit(g_sf_el_gx.p_ctrl, p_window_root);

  /* Creates the primary screen. */

    gx_studio_named_widget_create("splash_screen", (GX_WIDGET *)p_window_root, GX_NULL);

    /* Shows the root window to make it and patients screen visible. */

    gx_widget_show(p_window_root);

    /* Lets GUIX run. */

    gx_system_start();

    …

}
```

4.1.10.6 GUIX™ Synergy ポートモジュールの制限事項

- SF_EL_GX は、3 つ以上のフレームバッファを持つシステムをサポートしていません。
- SF_EL_GX は、GUIX™ キャンバスシステムを 1 つだけサポートします。
- SF_EL_GX は、ディスプレイモジュールのレイヤーを 1 つだけ利用します。
- GUIX™ が TES D/AVE 2D モジュールおよび TES D/AVE 2D ポートモジュールを使用する場合、これらのモジュールに直接アクセスしないでください。
- GUIX™ が JPEG デコードフレームモジュールおよび JPEG デコード HAL モジュールを使用する場合、これらのモジュールに直接アクセスしないでください。

4.1.10.7 サポートされているデバイス

このモジュールは、次のファミリをサポートするように設計されており、またこれらのファミリでテストも実施されています。

- S7G2

モジュールインタフェースは、API への変更なしに、以下のファミリをサポートしています。

- なし

Note：S3A7 ファミリデバイスは、DRW（2DG）、JPEG、および GLCDC モジュールをサポートしていません。この場合では、SF_EL_GX モジュールの使用は大きく制限されるため、SSP ディスプレイインタフェースをサポートするドライバモジュールを開発する必要があります。次にこのドライバにより、LCD

画面を更新して、SF_EL_GX モジュールハンドルにフレームバッファの切り替えを処理させるコールバック関数を呼び出す必要があります。

4.1.11 I²C フレームワーク

I²C フレームワークは、ThreadX RTOS を使用する I²C 通信アプリケーション用の汎用 API です。このフレームワークは、MCU で利用可能な SCI および IIC 周辺機器をサポートし、sf_i2c として実装されます。このセクションでは、e² studio ISDE を使用して I²C フレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e² studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Connectivity] > [I²C Framework Device on sf_i2c] を選択することで、外部 I²C フレームワークモジュールを追加および構成できます。詳細については、以下を参照してください。e² studio ISDE による I2C フレームワークを使用するアプリケーションの作成

API リファレンスは、次のフレームワーク I²C インタフェースの説明内に記載されています : I2C フレームワーク。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

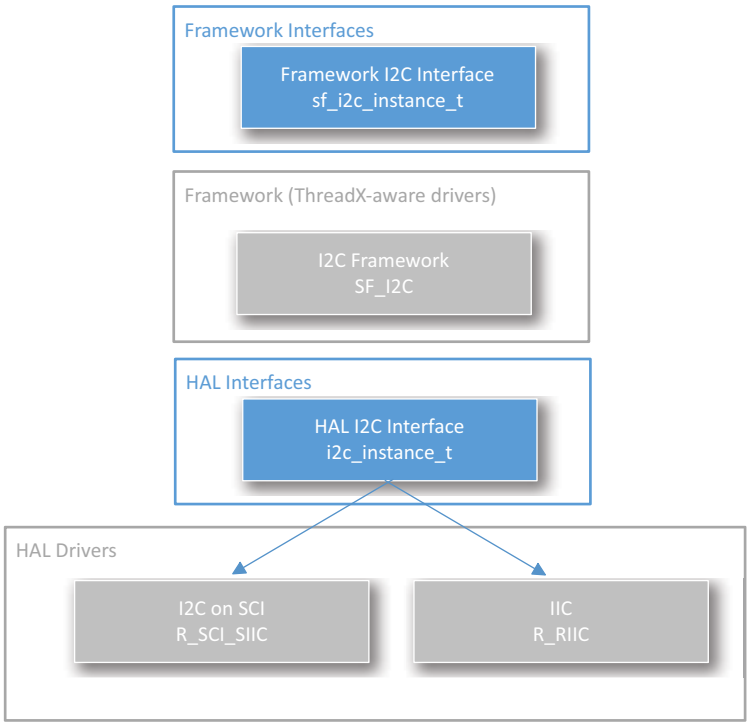

図 105: I²C フレームワーク - ブロック図

4.1.11.1 I²C フレームワークの機能

I²C フレームワークは、ThreadX 対応の一連のフレームワーク API です。I²C フレームワークは、I²C バス上の複数の I²C 周辺機器の統合と同期を処理します。I²C フレームワークを使用すると、1 つ以上の I²C バスを作成し、複数の I²C 周辺機器をそれぞれの I²C バスに接続できます。I²C フレームワークは、単一のインタフェースを使用して、SCI I²C ドライバと RIIC ドライバの両方にアクセスします。

I²C フレームワークを使用すると、1 つ以上の I²C バスを作成し、複数の I²C 周辺機器をそれぞれの I²C バスに接続できます。I²C フレームワークは、単一のインタフェースを使用して、SCI I²C ドライバと RIIC ドライバの両方にアクセスします。

フレームワークドライバアーキテクチャは、「バス」と「バス上のデバイス」によるアーキテクチャを使用します。フレームワークドライバアーキテクチャは、「バス」と「バス上のデバイス」によるアーキテクチャを使用します。下位では一度に 1 つのデバイスのみが設定され、残りのデバイスは、読み取りまたは書き込み操作の際に再設定されます。ユーザーは、バス、フレームワーク、バスに接続されている各デバイスのローレベルドライバレイヤーを設定する必要があります。共通の開始および停止手順が、すべての I²C データ転送操作に使用されます。ユーザーは、バス、フレームワーク、バスに接続されている各デバイスのローレベルドライバレイヤーを設定する必要があります。

共通の開始および停止手順が、すべての I²C データ転送操作に使用されます。ロックすると、一定期間（ロックとロック解除の間）、デバイス用にバスを予約できるようになります。これにより、一部のニーズに対応して、デバイスが複数の読み書き操作を中断することなく完了できるようになります。

4.1.11.2 I²C フレームワークの I²C モジュールの選択

このフレームワークモジュールは、SCI I²C モジュールと RIIC HAL モジュールの両方をサポートしています。どのモジュールを選択するかを決定するには、HAL I²C インタフェースユーザーガイド（I2C モジュールの選択）を参照してください。

4.1.11.3 e² studio ISDE による I²C フレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

I²C フレームワークを使用するアプリケーションの場合は、新しいスレッドを作成して、次のリソースを追加します。

リソース	ISDE タブ	選択
Framework I²C Driver	Threads	新しいスレッドをハイライトして、[New] > [Framework] > [Connectivity] > [I²C Framework Device on sf_i2c] の順に選択します。

リソース	ISDE タブ	選択
Framework I2C Bus Driver	Threads	HAL/Common をハイライトして、[New] > [Framework] > [Connectivity] > [I2C Framework Shared Bus on sf_i2c] の順に選択します。共有バスは常に共通サービスであるため、ISDE ではスレッドで共有バスを選択できません。

さらに、I2C HAL ドライバをスレッドに 1 つ以上追加して、MCU で実行できる I2C アプリケーションを作成します。

I2C フレームワークと SCI モジュールを併用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
SCI Driver for I2C on SCI	Threads	新しいスレッドをハイライトして、[New] > [Driver] > [Connectivity] > [I2C Driver on r_sci_i2c] の順に選択します。
SCI Common driver	Threads	新しいスレッドをハイライトして、[New] > [Driver] > [Connectivity] > [SCI Common] の順に選択します。このドライバの [Properties] ウィンドウで、簡易 I2C モードを有効化します。
Interrupts	Threads	チャネルを選択して、選択したチャネルの SCIn RXI、SCIn TXI、SCIn TEI および SCIn ERI 割り込みを有効にします。

I2C フレームワークと IIC モジュールを使用する場合は、以下のリソースを追加します。

リソース	ISDE タブ	選択
RIIC driver	Threads	新しいスレッドをハイライトして、[New] > [Driver] > [Connectivity] > [I2C Driver on r_riic] の順に選択します。

リソース	ISDE タブ	選択
Interrupts	Threads または ICU	チャネルを選択して、選択したチャネルの IICn RXI、IICn TXI**、** IICn TEI** および **IICn ERI** 割り込みを有効にします。

I²C フレームワークのクロックの設定

e² studio ISDE で、[Clocks] タブを使用して I²C クロックを設定します（クロックの設定を参照）。

I²C フレームワーク自体は固有のクロック設定を必要としませんが、IIC または SCI モジュールを選択した場合は、選択したモジュールの要件に従ってクロックを設定する必要があります。両方のモジュールの I²C クロックオプションは、I2C ドライバで説明されています。

I²C フレームワークのピンの設定

e² studio ISDE を使用して、[Pins] タブから UART ピンを設定します（ピンの設定を参照）。

I²C フレームワーク自体は固有のピン設定を必要としませんが、IIC または SCI モジュールを選択した場合は、選択したモジュールの要件に従ってピンを設定する必要があります。両方のモジュールの I²C ピン要件は、I2C ドライバで説明されています。

I²C フレームワークの割り込みの設定

データ転送中に各種ハードウェアイベントを通知するために、I²C 転送割り込みを BSP で有効にする必要があります。割り込みは、I²C バスを設定するために選択されたドライバとチャネルに基づいて設定する必要があります。

I²C フレームワークの SCI I²C の設定

e² studio ISDE を使用して、[Threads] タブから SCI 割り込みを設定します（割り込みの設定を参照）。

SCI I²C の割り込みを有効にするには、SCIn RXI、TXI、TEI および ERI 割り込み（n は SCI チャネル番号）のプライオリティを選択します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

I²C フレームワークの RIIC 割り込みの設定

e² studio ISDE を使用して、[Threads] タブから IIC 割り込みを設定します（割り込みの設定を参照）。

RIIC 割り込みを有効にするには、RIICn RXI、RIICn TXI、RIICn TEI および RIICn EEI 割り込み（n は RIIC チャネル番号）のプライオリティを設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

I²C フレームワークモジュールの設定

I²C 周辺機器用に I²C フレームワークを設定するには、以下のモジュールを設定する必要があります。

1) I²C バスモジュールの設定
2) I²C ドライバモジュールの設定
3) I²C フレームワークモジュールの設定

I²C バスモジュールを設定したら、各種の I²C 周辺機器（デバイス）をそのバスに関連付けることができます。バスに接続している I²C デバイスごとに、1 つの I²C ドライバモジュールと 1 つの I²C フレームワークモジュールを設定する必要があります。

以降のセクションでは、これらの各モジュールの設定方法について説明します。

I²C フレームワークドライバパラメータの設定

e² studio ISDE を使用して、g_i2c ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

表 : I²C フレームワークの設定（共通パラメータ）

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define SF_I2C_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータチェック用のコードを含めるかどうかを制御します。

I²C HAL ドライバパラメータの設定

e² studio ISDE を使用して、SCI の IIC または I²C の g_i2c ドライバパラメータを設定します（スレッドとドライバの追加を参照）。

表 : I²C ドライバモジュールの設定

ISDE プロパティ	設定	設定値	説明
Channel	channel	Integer value between 0 and 9.	デバイスが接続されている SCI または RIIC チャネル番号。
Rate	rate	Standard, Fast mode, Fast mode plus. Note: Fast mode plus is only supported for the RIIC peripheral.	デバイスの最大クロックレート。
Slave	slave	Arbitrary integer value	スレーブデバイスのアドレス。
Address mode	addr_mode	7-Bit, 10-Bit	スレーブフィールドの解釈方法を示します。

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	NULL	フレームワーク設定用のコールバックを設定する必要はありません。コールバックは、フレームワークコードにより内部的に処理されています。

SF I²C バスモジュールパラメータの設定

e² studio ISDE を使用して、g_sf_i2c_bus ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）

表 : I²C フレームワーク共有バス設定モジュールのパラメータ

ISDE プロパティ	設定	設定値	説明
Name	p_bus_name	Arbitrary symbol	バスを識別するための名前を設定します。このバス名は、フレームワーク設定で I²C 周辺機器をバスに関連付けるために使用されます。
Channel	channel	Integer value between 0 and 9	デバイスが接続されている SCI または RIIC チャネル番号。フレームワークでは、このバス設定で指定されたチャネル番号により HAL ドライバで指定されたチャネル番号が上書きされます。
I²C Implementation	p_lower_lvl_api	SCI I²C、RIIC	フレームワークで使用するローレベルインタフェースを選択します。

SF I²C フレームワーク パラメータの設定

e² studio ISDE を使用して、g_sf_i2c_device ドライバ パラメータを設定します （ドライバのスレッドへの追加とドライバの設定を参照）

I²C フレームワークは、SF_I2C_Open API で sf_i2c_cfg_t 型のポインタを受け渡しすることにより設定されます。

表 : I²C フレームワーク モジュールの設定

ISDE プロパティ	設定	設定値	説明
Bus Name	p_bus	Arbitrary symbol	I²C バス モジュール設定で設定したバスに名前を付けます。これにより、この I²C 周辺機器が指定したバスに関連付けられます。
Lower Level I²C Configuration	p_lower_lvl_cfg	Arbitrary symbol	設定した I²C ドライバの名前を付けます。ローレベル設定へのポインタは、この設定から取得されます。
Name	任意の記号	Arbitrary symbol	I²C デバイスを識別するための名前を設定します。この名前に基づいて、API、Config および Control インスタンスが作成されます。

4.1.11.4 I²C フレームワーク アプリケーションの作成

ISDE は、新しい I²C フレームワーク アプリケーションのサポートを目的として、次のファイルを自動的に生成します。

- 選択した設定オプションが適用された I²C 構造体を含む、src/ssp_gen ディレクトリの数種類のファイル。
- ユーザー アプリケーション コードのプレースホルダーを含む src ディレクトリの new_thread_entry.c ファイル。

以下で説明するアプリケーションは、RIIC 周辺機器を使用するように I²C フレームワークのインスタンスを設定します。

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_i2c_instance_t  g_sf_i2c_device =

{

    .p_ctrl  = &g_sf_ i2c _ctrl,

    .p_cfg = &g_sf_ i2c _cfg,

    .p_api = &g_sf_ i2c _on_sf_ i2c

};
```

RIIC を使用する I²C フレームワーク アプリケーションは、以下のようにして作成します。

1) RTOS を使用する Synergy プロジェクトを作成します。
2) [Threads] タブで、モジュールの追加と設定を実行します。
3) 前述のように割り込みを設定します。
4) ISDE でプロジェクト コンテンツを生成します。これにより、フレームワーク関連のヘッダー ファイルと設定ファイルが自動的に作成されます。次の構造体は、src/ssp_gen/new_thread.c ファイルの設定を利用して初期化されます。
   - i2c_cfg_t
   - sf_i2c_bus_t
   - sf_i2c_cfg_t
5) ユーザー独自のアプリケーション コードを new_thread_entry.c に追加する前に、エラーのないプロジェクトをビルドします。
6) ユーザーのアプリケーション コードで、I²C が RIIC として実装された I²C フレームワーク インスタンスを開きます。I²C フレームワークは、次のインスタンスを通じて開かれます。

```
g_sf_i2c_device.p_api->open(g_sf_i2c_device.p_ctrl, g_sf_i2c_device.p_cfg);
```

ここで、g_sf_i2c_device.p_ctrl と g_sf_i2c_device.p_ctrl は、ISDE による I²C フレームワーク設定ステップの後に自動生成されます。I²C フレームワークを開くと、I²C デバイスのハンドルがわかります。上記の例では、ハンドルは g_sf_i2c_ctrl です。このハンドルを使用して、各種の I²C 転送操作を実行します。以下に、このハンドルを使用して write 操作を行うコード例を示します。

```
 reg = FT5X06_REG_TOUCH1_XH;

err = g_sf_i2c_device.p_api->write(g_sf_i2c_device.p_ctrl, &reg, 1,

 false, TX_WAIT_FOREVER );
```

open、write、read、および reset の使用方法を示すコード例を以下にしまします。

```
void new_thread_entry(void) {

    /* TODO: add your own code here */

    /* Replace write and read source and destination addresses and buffer

    * as required by the application.

    * For example, use reg = FT5X06_REG_TOUCH1_XH and ft5x06_touch_t touch[1] for

    * an application using touch points. */

    ssp_err_t  err;

    err = g_sf_i2c_device.p_api->open(g_sf_i2c_device.p_ctrl, g_sf_i2c_device.p_cfg);

  uint8_t reg = 0x0; //address of the source - replace by actual address for the application

    uint8_t buf[10]; // read buffer - replace by actual buffer for the application

    err = g_sf_i2c_device.p_api->write(g_sf_i2c_device.p_ctrl,

     &reg, 1, false, TX_WAIT_FOREVER );

    if (SSP_SUCCESS != err)

    {

        g_sf_i2c_device.p_api->reset(g_sf_i2c_device.p_ctrl,

     TX_WAIT_FOREVER);

        return;

    }

    err = g_sf_i2c_device.p_api->read(g_sf_i2c_device.p_ctrl,

     (uint8_t*)&buf[0], 3, false, TX_WAIT_FOREVER );
```

```
    if (SSP_SUCCESS != err)

    {

        g_sf_i2c_device.p_api->reset(g_sf_i2c_device.p_ctrl,

     TX_WAIT_FOREVER);

        return;

    }

    while (1) {

    tx_thread_sleep(1);

    }

}
```

4.1.11.5 同じバス上での複数の I²C デバイスの接続

複数の I²C デバイスを同じバス上で設定するには、バスに接続する各デバイスについて、以下のモジュールを追加して設定します。

- I²C Driver on r_sci_i2c または I2C Driver on r_riic
- I²C Framework Device on sf_i2c

Note：各 I²C デバイスは、ISDE コンフィギュレータで一意の名前を使用して設定する必要があります。

4.1.11.6 I²C フレームワークでサポートされるデバイス

このドライバは、S7G2 でテストされています。I²C フレームワークは、API への変更なしに、SCI I²C または RIIC 周辺機器および HAL ドライバを使用して、すべての MCU をサポートするように設計されています。

I²C フレームワークは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.1.11.7 I²C フレームワークの制限事項

このモジュールには既知の制限事項はありません。最新のソフトウェア バージョンの詳細については、SSP のリリース ノートを参照してください。

4.1.11.8 I²C フレームワーク ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。どのドライバを選択したかに基づいて、SCI ドライバまたは RIIC HAL ドライバが抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
I²C Framework Interface	synergy/ssp/inc/framework/instances/sf_i2c_api.h
I²C Framework Driver	synergy/ssp/src/framework/sf_i2c.h
I²C HAL Interface	synergy/ssp/inc/driver/api/r_i2c_api.h
I²C on SCI Instance	synergy/ssp/inc/driver/instances/r_sci_i2c.h
I²C on SCI Driver	synergy/ssp/src/driver/r_sci_i2c
SCI common instance	synergy/ssp/scr/driver/r_sci_common.h
SCI common Driver	synergy/ssp/scr/driver/r_sci_common
IIC Instance	synergy/ssp/inc/driver/instances/r_riic.h
IIC Driver	synergy/ssp/src/driver/r_riic

4.1.12 JPEG デコード フレームワーク

JPEG デコード フレームワーク インタフェースは、JPEG デコード処理のための ThreadX 対応 API です。このフレームワークは sf_jpeg_decode に実装され、Synergy JPEG ハードウェアへのスレッドセーフなアクセスを提供します。このセクションでは、e² studio ISDE を使用して JPEG デコードフレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e² studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Graphics] > [JPEG Decode Framework on sf_jpeg_decode] を選択することで、JPEG デコードフレームワークモジュールを追加および構成できます。詳細については、以下を参照してください : e² studio ISDE による JPEG デコードフレームワークを使用するアプリケーションの作成。

API リファレンスは、次の JPEG デコードフレームワークインタフェースの説明内に記載されています : JPEG デコードフレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

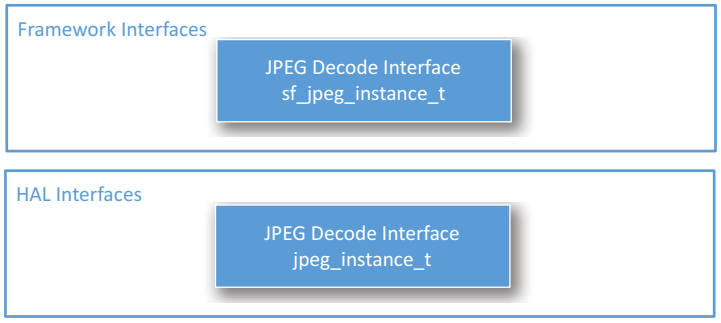

図 106: JPEG デコードフレームワーク - ブロック図

関連項目：JPEG デコード ドライバ

4.1.12.1 JPEG デコードフレームワークの機能

- Synergy JPEG ハードウェアへのスレッドセーフなアクセスを提供します。
- JPEG デコード HAL ドライバを通じた JPEG 解凍をサポートします。
- JPEG ハードウェアサポートイベントと同期するためのスレッドをサスペンド / 再開するための wait API 関数をサポートします。

4.1.12.2 e² studio ISDE による JPEG デコードフレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています（e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（スレッドとドライバの追加を参照）。

JPEG デコードフレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
JPEG Decode Framework	Threads	[Framework] > [Graphics] > [JPEG Decode Framework on sf_jpeg_decode]

リソース	ISDE タブ	選択
JPEG Decode Driver	Threads	[Driver] > [Graphics] > [JPEG Decode Driver on r_jpeg]
Interrupts	Threads	アプリケーション例については、JPEG デコードフレームワークの割り込みの設定
Clock	Clocks	アプリケーション例については、JPEG デコードフレームワークのクロックソースの設定

JPEG デコードフレームワークのクロックソースの設定

クロックソースの設定については、JPEG デコード ドライバを参照してください。

JPEG デコードフレームワークの割り込みの設定

割り込みの設定については、JPEG デコード ドライバを参照してください。

JPEG デコードフレームワークステータス

JPEG ドライバステータスについては、JPEG デコード ドライバを参照してください。

JPEG デコードフレームワークパラメータの設定

e^2 studio ISDE を使用して、JPEG デコードステータスパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定。

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

JPEG デコードフレームワークでは、次のコンポーネントの構成が必要です。

JPEG デコードフレームワーク sf_jpeg_decode_cfg_t

JPEG デコードフレームワークのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking Enable	#define BSP_CFG_PARAM_CHECKING_ENABLE	Enabled (Default), Disabled	パラメータエラーチェックを有効または無効にします。
Name	Name prefix of instances	Arbitrary C Symbol (Default: "g_sf_jpeg_decode")	JPEG デコードフレームワークモジュールインスタンスに使用する名前。

JPEG デコードフレームワークの初期化コード例

```
/* JPEG Decode Framework module configuration. Specify the pointer to
the API function pointer set and configuration in the JPEG Decode HAL module
instance. */

static const sf_jpeg_decode_cfg_t  g_sf_jpeg_decode_cfg = {



 .p_lower_lvl_jpeg_decode

     = (jpeg_decode_instance_t  const *) &g_jpeg_decode

};

/* JPEG Decode Framework module instance */

const sf_jpeg_decode_instance_t  g_sf_jpeg_decode = {

    .p_api  = &g_sf_jpeg_decode_on_sf_jpeg_decode,

    .p_ctrl = &g_sf_jpeg_decode_ctrl,

    .p_cfg = (sf_jpeg_decode_cfg_t  const *) &g_sf_jpeg_decode_cfg

};
```

JPEG デコードフレームワークのオープン

JPEG エンコードされたデータのデコードは、open を呼び出すことで開始できます。モジュールを開くには、JPEG デコードフレームワークモジュールインスタンスを使用します。このインスタンスには、API 関数のポインタ、制御ブロックへのポインタ、および e^2 studio ISDE の Synergy プロジェクトコンフィギュレータを通じて生成される静的構成が含まれています。

```
/* JPEG Decode Framework driver open */

g_sf_jpeg_decode.p_api->open(g_sf_jpeg_decode.p_ctrl,
g_sf_jpeg_decode.p_cfg);
```

JPEG デコードフレームワークのクローズ

JPEG デコードフレームワークモジュールを停止するには、close を呼び出します。

```
/* JPEG Decode Framework driver close */
g_sf_jpeg_decode.p_api->close(g_sf_jpeg_decode.p_ctrl);
```

JPEG デコードフレームワークを使用した入力バッファストリーミングモード

次の例は、JPEG エンコードされたデータを JPEG 入力バッファーストリーミングモードでデコードする方法を示しています。JPEG デコードフレームワークモジュールは、JPEG デコードフレームワークのオープンに記載の構成を使用してあらかじめ開かれていなければなりません。

```
…

/* Set the image vertical and horizontal sample. */
g_sf_jpeg_decode.p_api->imageSubsampleSet(g_sf_jpeg_decode.p_ctrl,
    JPEG_DECODE_OUTPUT_NO_SUBSAMPLE, JPEG_DECODE_OUTPUT_NO_SUBSAMPLE);

/* Set the decoding image horizontal stride. Here, assuming it is 800 in pixels */
g_sf_jpeg_decode.p_api->horizontalStrideSet(g_sf_jpeg_decode.p_ctrl, 800);

/* Set the output buffer address. */
g_sf_jpeg_decode.p_api->outputBufferSet(g_sf_jpeg_decode.p_ctrl,
                        data_buffer, output_buffer_size);

/* Get an input buffer address and the size */
input_buffer = user_get_input_buffer_address();
input_buffer_size = user_get_input_buffer_size();

/* Set the input buffer address. */
g_sf_jpeg_decode.p_api->inputBufferSet(g_sf_jpeg_decode.p_ctrl,
                        input_buffer, input_buffer_size);

/* Wait JPEG_DECODE_STATUS_IMAGE_SIZE_READY */
g_sf_jpeg_decode0.p_api->wait(g_sf_jpeg_decode0.p_ctrl,

(jpeg_decode_status_t  *)&status, 100);

/* Repeat jpeg decoding. */
while(1)

{

    g_sf_jpeg_decode0.p_api->wait(g_sf_jpeg_decode0.p_ctrl,

                (jpeg_decode_status_t  *)&status, 100);

    if(*JPEG_DECODE_STATUS_DONE* & status)

    {

        break;

    }

    else if(*JPEG_DECODE_STATUS_INPUT_PAUSE* & status)
```

```
    {

        input_buffer = user_get_input_buffer_address();
        input_buffer_size = user_get_input_buffer_size();

        /* Set input buffer address. */

        g_sf_jpeg_decode.p_api->inputBufferSet(g_sf_jpeg_decode.p_ctrl,

                    input_buffer, input_buffer_size);
         if(*SSP_SUCCESS* != err)

        {

            while(1);

        }

    }

}

…
```

JPEG デコードフレームワークを使用した出力バッファストリーミングモード

次の例は、JPEG エンコードされたデータを JPEG 出力バッファーストリーミングモードでデコードする方法を示しています。JPEG デコードフレームワークモジュールは、JPEG デコードフレームワークのオープンに記載の構成を使用してあらかじめ開かれていなければなりません。

```
...
/* Set the image vertical and horizontal sample. */
g_sf_jpeg_decode.p_api->imageSubsampleSet(g_sf_jpeg_decode.p_ctrl,
      JPEG_DECODE_OUTPUT_NO_SUBSAMPLE, JPEG_DECODE_OUTPUT_NO_SUBSAMPLE);

/* Set the decoding image horizontal stride. Here, assuming it is 800 in pixels */
g_sf_jpeg_decode.p_api->horizontalStrideSet(g_sf_jpeg_decode.p_ctrl, 800);

/* Set the output buffer address. */
g_sf_jpeg_decode.p_api->outputBufferSet(g_sf_jpeg_decode.p_ctrl,
                        data_buffer, output_buffer_size);

/* Set the input buffer address. */
g_sf_jpeg_decode.p_api->inputBufferSet(g_sf_jpeg_decode.p_ctrl,
                input_buffer, input_buffer_size);

/* Wait JPEG_DECODE_STATUS_IMAGE_SIZE_READY */
g_sf_jpeg_decode0.p_api->wait(g_sf_jpeg_decode0.p_ctrl,
                (jpeg_decode_status_t  *)&status, 100);

/* Get image size */
uint16_t horizontal_size;
uint16_t vertical_size;

g_sf_jpeg_decode0.p_api->imageSizeGet(g_sf_jpeg_decode0.p_ctrl,
                    &horizontal_size, &vertical_size);

/* Calculate the size of image in bytes. Here is the case of 2bpp(RGB565) */
uint32_t total_bytes = (uint32_t)(horizontal_size * vertical_size * 2);

while(1)

{

    uint32_t decoded_bytes = 0;

    g_sf_jpeg_decode0.p_api->wait(g_sf_jpeg_decode0.p_ctrl,
            (jpeg_decode_status_t  *)&status, 100);

    if(*JPEG_DECODE_STATUS_DONE* & status)

    {

        break;

    }

    else if(*JPEG_DECODE_STATUS_OUTPUT_PAUSE* & status)
```

```
    {

        /* Set output buffer address. */

        output_buffer = output_buffer + output_size;

        g_sf_jpeg_decode0.p_api->outputBufferSet(g_sf_jpeg_decode0.p_ctrl,
                                                 output_buffer, output_size);

        /* Check decoded bytes */
        decoded_bytes = decoded_bytes + output_size;

        if (total_bytes <= decoded_bytes)

        {

            break;

        }

    }

}
…
```

4.1.12.3 JPEG デコードフレームワークの制限事項

JPEG デコードフレームワークドライバは、JPEG エンコード処理をサポートしていません。

4.1.12.4 JPEG デコードフレームワークでサポートされるデバイス

このモジュールは S7G2 をサポートするように設計されており、S7G2 でテスト済みです。

4.1.13 メッセージングフレームワーク

メッセージングフレームワークは sf_message 上で実装され、スレッド間のメッセージを送受信する、イベント駆動型の軽量なフレームワーク API です。メッセージングフレームワークにより、アプリケーションは 2 つ以上のスレッド間でメッセージを送受信できるようになります。

このフレームワークは、ThreadX のメッセージキュープリミティブを使用してメッセージを送受信し、ThreadX RTOS メッセージキューサービス単独の場合よりも多くの利点が得られます。メッセージングフレームワーク API は純粋なソフトウェア API であり、ハードウェア周辺機器にはアクセスしません。

e^2 studio ISDE のプロジェクトコンフィギュレータには、[Messaging Configurator] タブが含まれており、スレッドがどのようにメッセージングフレームワークを使用するかを設定できます。[Messaging] タブを使用して、メッセージングフレームワーク用の独自のカスタムイベントクラスやイベント、サブスクライバーを

作成したり、タッチパネルフレームワークによって使用されるタッチイベントなどの事前設定されたイベントをカスタマイズしたりできます。

e2 studio ISDE のプロジェクトコンフィギュレータでは、[Threads] タブの [Thread Stacks] ウィンドウで [New Stack] > [Framework] > [Service] > [Messaging Framework on sf_message] を選択することで、メッセージングフレームワークモジュールを追加および構成できます。詳細については、以下を参照してください : e2 studio ISDE によるメッセージングフレームワークを使用するアプリケーションの作成。

API リファレンスは、次のメッセージングフレームワークインタフェースの説明内に記載されています : メッセージングフレームワークインタフェース。

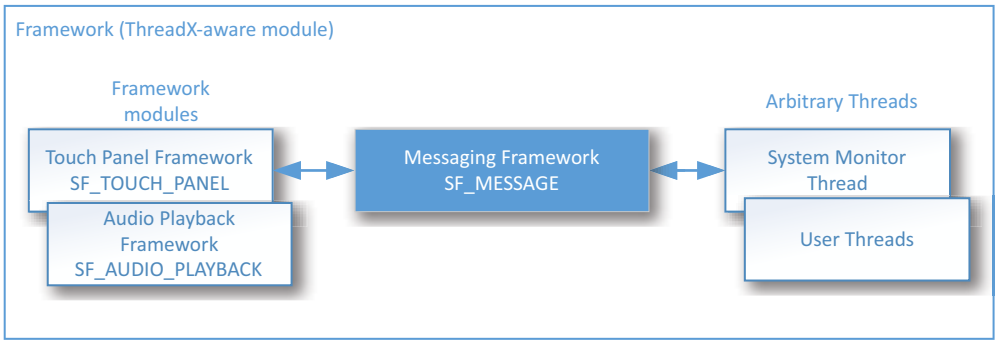

図 107: メッセージング – ブロック図

4.1.13.1 メッセージングフレームワークの機能

フレームワークは以下の機能をサポートしています。

- スレッド間通信 - 異なるデバイスを制御したりサブシステムを管理したりするアプリケーションスレッドが互いに通信できます。
- パブリッシュ / サブスクライブ方式 - フレームワークの設計は、疎結合型のメッセージング方式に基づいています。複数のスレッドが 1 つのイベントクラスを受信待ちできる設計になっています。メッセージ作成元スレッドは、イベントクラスのメッセージにサブスクライブしているスレッドを知る必要がありません。サブスクライバーは、メッセージの作成元を知る必要がありません。
- メッセージ管理 - フレームワークでは、バッファを制御するフラグやハンドシェイクのためのコールバック関数ポインタなど、各メッセージを管理するためのバッファ制御ブロックがサポートされています。
- メッセージバッファリング - フレームワークは、メッセージ用のバッファの割り当てと解放を管理します。アプリケーションは、割り当てられたバッファを利用してメッセージを書き込んだり、不要になったメッセージを破棄することができます。
- 同期通信 - フレームワークでは、ThreadX メッセージキューを使用した非同期メッセージングがサポートされていますが、メッセージ作成元とサブスクライバースレッドの間のハンドシェイクを確立するためのオプションも提供されています。ハンドシェイクは、サブスクライバースレッドから生成スレッドのユーザーコールバック関数を呼び出すことで実装されています。

- メッセージ構造の指定 - フレームワークでは、定義済みの共通メッセージヘッダーが提供されています。また、いくつかの典型的なペイロード構造体テンプレートも例として提供されています。
- メッセージの優先度 - 高優先度のメッセージを送信して、サブスクライバースレッドが、メッセージキュー内の他のメッセージの前にそのメッセージを受け取るようにすることができます。

4.1.13.2 メッセージングデータフロー

下図に、メッセージングフレームワークモジュールを利用するシステム内の、メッセージ作成元スレッドとサブスクライバースレッドの間のメッセージングデータフローの概要を示します。

参考資料

参考資料

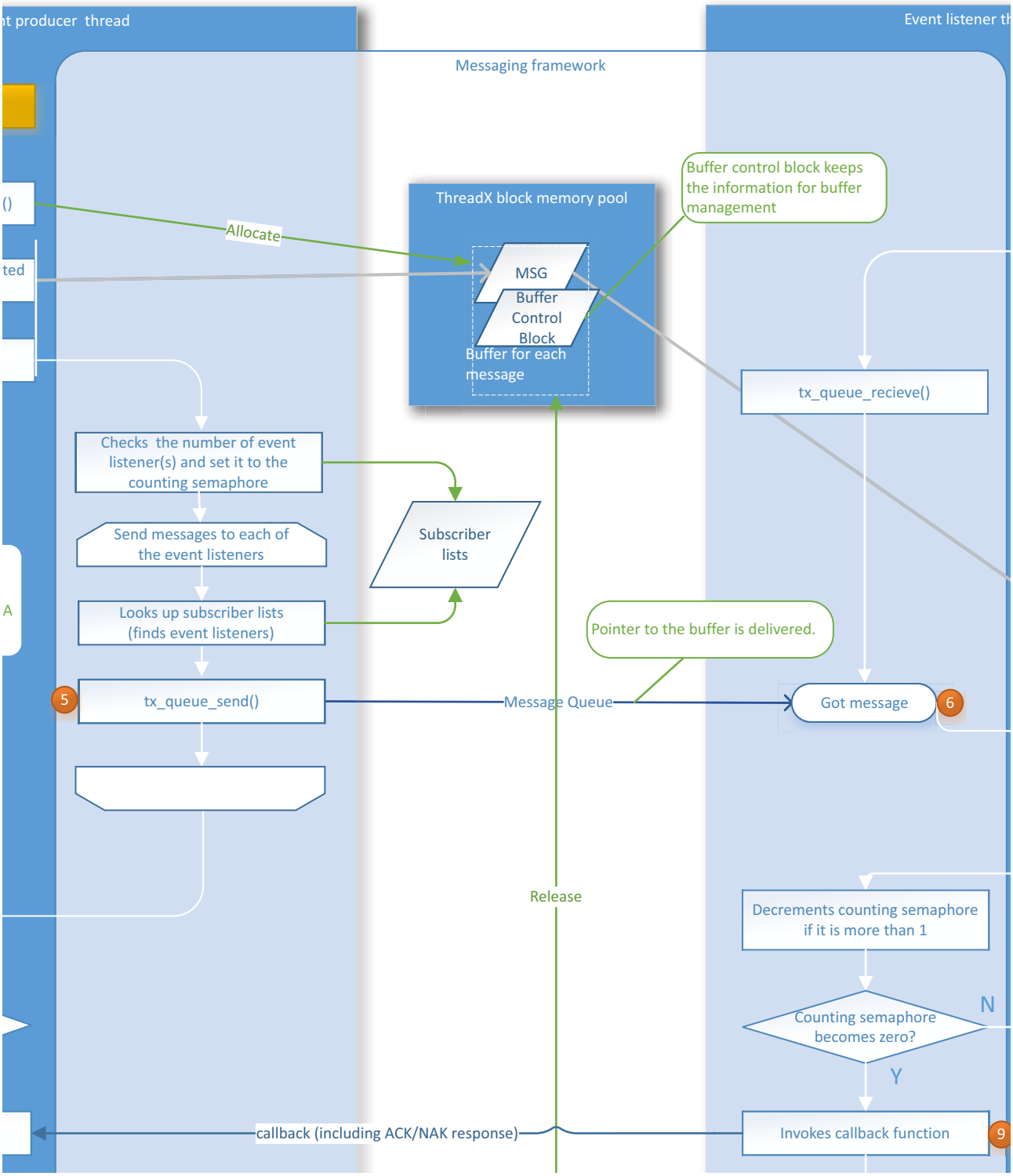
t producer thread
Event listener th
Messaging framework
()
ted
A
Allocate
ThreadX block memory pool
MSG
Buffer Control Block
Buffer for each message
Buffer control block keeps the information for buffer management
tx_queue_recieve()
Checks the number of event listener(s) and set it to the counting semaphore
Send messages to each of the event listeners
Looks up subscriber lists (finds event listeners)
Subscriber lists
Pointer to the buffer is delivered.
5
tx_queue_send()
Message Queue
Got message
6
Release
Decrements counting semaphore if it is more than 1
Counting semaphore becomes zero?
N
Y
callback (including ACK/NAK response)
Invokes callback function
9

図 108: メッセージング – データフロー

以下で、メッセージ送受信手順の各ステージについて説明します。

Note：システムのスレッドがすでに open から呼び出されており、さらにメッセージサブスクライバースレッドが pend を呼び出して、イベントクラスのメッセージを待っています。

1) イベント（イベント A）がメッセージ作成元スレッドで発生します。
2) メッセージ作成元スレッドは、bufferAcquire を呼び出して、メッセージングフレームワークモジュールが管理する ThreadX メモリプールからバッファを取得します。bufferAcquire は割り当てられたバッファのアドレスを返します。
3) メッセージ作成元が、割り当てられたバッファにメッセージを書き込みます。
4) メッセージ作成元が、post を呼び出してメッセージをポストします。
5) メッセージングフレームワークモジュールは、イベントサブスクライバーリストを検索し、ThreadX メッセージキュープリミティブを使用して、メッセージサブスクライバースレッドのメッセージキューにメッセージを送信します。フレームワークは、バッファのポインタのみを送信し、メッセージ全体を送信せず、軽量なメッセージ送受信を行うことに注意してください。
6) メッセージがメッセージサブスクライバースレッドのメッセージキューに到達し、メッセージサブスクライバースレッドが pend から戻ります。この API 関数は、メッセージが保存されているバッファアドレスをメッセージサブスクライバースレッドに返します。
7) メッセージサブスクライバースレッドは、メッセージを受信し、イベントに応じたアクションを実行します。
8) メッセージサブスクライバースレッドは、bufferRelease を呼び出して、メッセージに割り当てられたバッファを解放しようとします。そのメッセージサブスクライバースレッドが、メッセージにサブスクライブしている最後のスレッドでない場合、フレームワークはバッファを解放しません。なぜなら、すべてのサブスクライバーがメッセージを受信するまでメッセージをバッファ内に保持する必要があるためです。
9) メッセージングフレームワークモジュールは、そのメッセージサブスクライバースレッドが、メッセージサブスクライバーグループの最後のスレッドである場合、イベント作成元スレッドによって指定されたユーザーコールバック関数を呼び出します。
10) メッセージングフレームワークモジュールは、そのメッセージサブスクライバースレッドが、メッセージサブスクライバーグループの最後のスレッドである場合、バッファを解放します。バッファを解放しないオプションがあります。このオプションの詳細については、bufferAcquirebufferAcquire、SF_MESSAGE_ACQUIRE_OPTION_KEEP オプションの説明を参照してください。

メッセージ作成元とサブスクライバー

メッセージフレームワークモジュールは、パブリッシュ / サブスクライブモデルに基づくスレッド間メッセージングシステムです。メッセージは、イベント作成元スレッドによって、イベントクラスコードとともにポストされます。メッセージサブスクライバースレッドは、イベントクラスにサブスクライブする保留中のメッセージを確認できます。サブスクライバーは、フレームワークによって参照されるサブスクライバーリストに登録されます。サブスクライバーリストを使用すると、フレームワークは、複数のサブスクライバーにメッセージを配信できます。以下の図「メッセージのサブスクライブ」を参照してください。

メッセージングフレームワークモジュールシステムネットワークに参加するすべてのスレッドと、ネットワーク内のすべてのスレッドがメッセージを受信待ちできます。

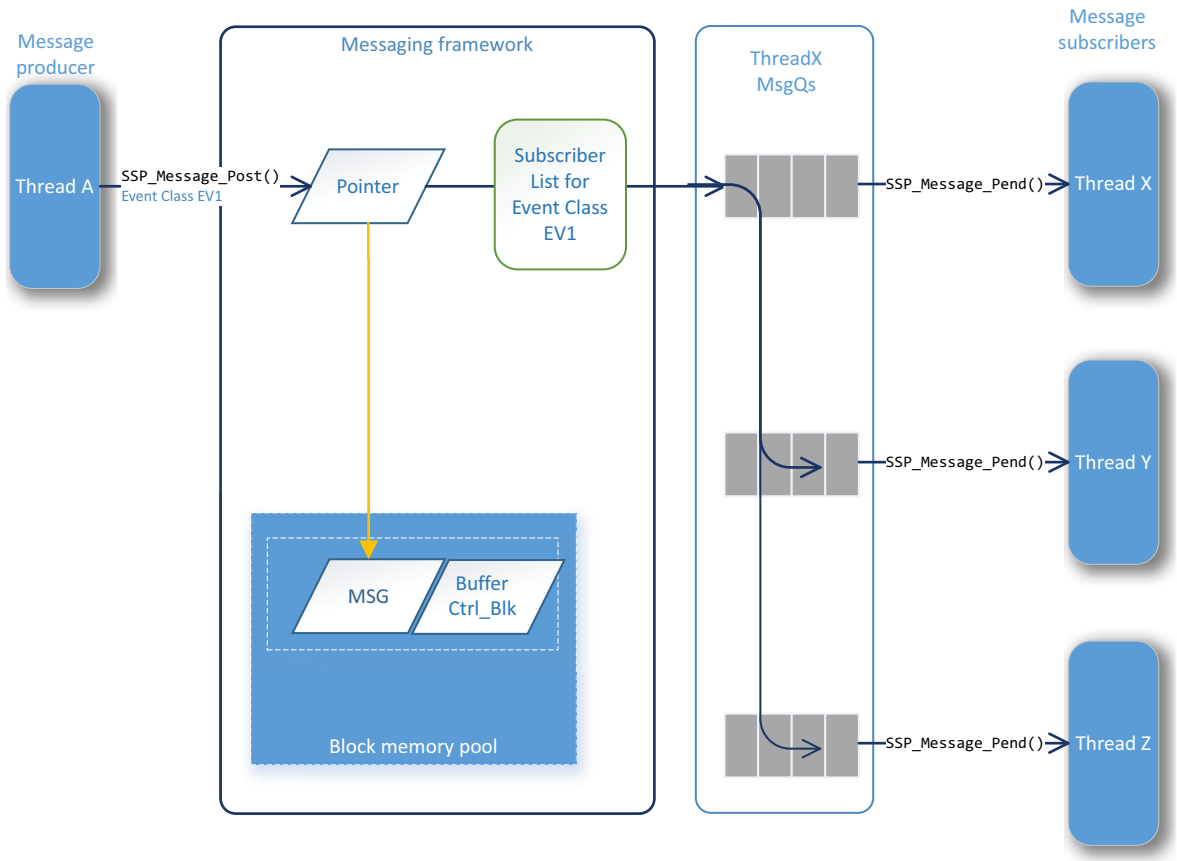

図 109:　メッセージ – サブスクライブ

4.1.13.3 イベントクラスとイベントコード

イベントクラスコード

イベントクラスコードは、メッセージングモジュールの最も重要な定義です。メッセージングモジュールは、イベントクラスコードを、メッセージ作成元をサブスクライバーに結び付けるためのメカニズムとして使用します。イベントクラスコードは、アプリケーション内で発生するイベントのクラス定義です。イベントクラスの分類はユーザー定義に依存しますが、サブシステム内で発生する可能性がある特定のイベントのグループ名であることが想定されています。たとえば、以下のイベントクラスを使用できます。

- タッチサブシステムの一部である「touch」イベントクラス。タッチイベントクラスは、Synergy プロジェクトにタッチパネルフレームワークを追加すると、プロジェクトコンフィギュレータの [Messaging] タブの [Event Class] ウィンドウに自動的にロードされます。

- 時間関連アプリケーションの機能を管理するサブシステムの一部である「time」イベントクラス。

イベントクラスコードは、sf_message_event_class_t 列挙で定義され、プレフィックス SF_MESSAGE_EVENT_CLASS_XXX を持ちます。イベントクラスコードの定義はシステムごとに異なりますが、フレームワークでは具体的なイベントクラスコードが用意されておらず、代わりに一連のイベントクラスコードが例として提供されています。イベントクラスコードとイベントコードの設定を参照してください。イベントクラスの最大数は 255 です。

アプリケーションは、イベントクラスコードを以下のように使用できます。

- メッセージ作成元スレッドは、イベントクラスコードを sf_message_header_t 型の共通メッセージヘッダーの event_b.class ビットフィールドに設定してからメッセージをポストます。
- メッセージサブスクライバースレッドは、メッセージを受信した後、メッセージヘッダーに設定されているイベントクラスコードに従ってイベント処理を分岐します。
- イベントクラスコードのサブスクライバーは、メッセージフレームワークがメッセージをサブスクライバーに配信できるように、サブスクライバーリストにグループ化および登録されている必要があります。

イベントクラスは、e2 studio ISDE で以下のとおり設定できます。

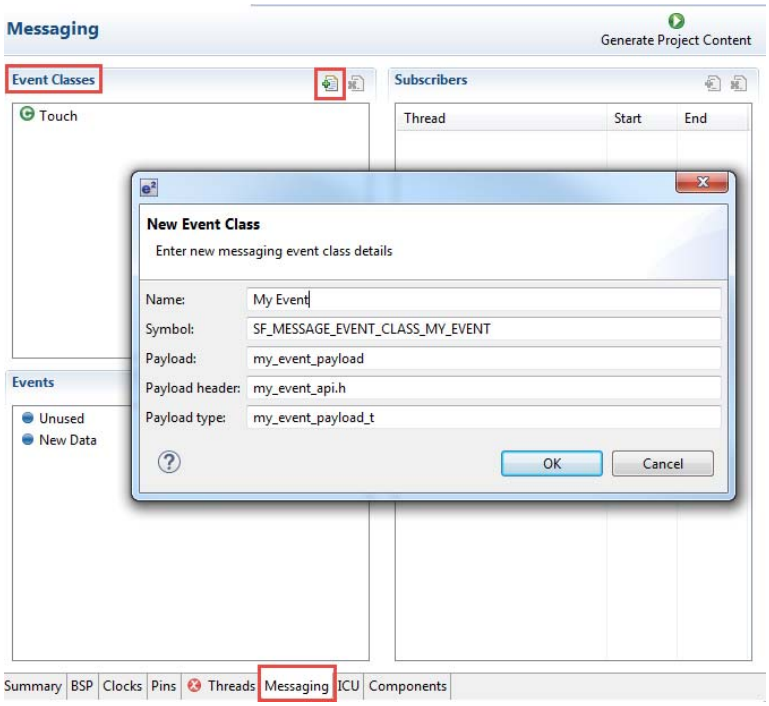

図 110: メッセージング – e2 studio ISDE イベントクラスの設定

イベントクラスインスタンス番号

イベントクラスインスタンス番号は、アプリケーションが異なるイベントクラスインスタンスを必要とするときに使用されます。たとえば、オーディオストリーミングイベントクラスでは、ストリーミングチャネル N を示すインスタンス N を使用できます。メッセージサブスクライバーは、メッセージの共通ヘッダーに

あるイベントクラスインスタンス番号が、所有する番号と一致する場合にのみメッセージを受信できます。言い換えれば、イベントクラスインスタンス番号がサブスクライバーの範囲外であるメッセージはフィルタ処理で除外され、イベントクラスサブスクライバーであっても、そのサブスクライバーには配信されません。イベントクラスインスタンス番号の最大値は 255 です。

Note：タッチパネルフレームワークには通常、1 つしかインスタンスがないため、イベントクラス番号は 0 となり、サブスクライバーリストの開始値と終了値が 0 に設定されます。

アプリケーションは、イベントクラスインスタンス番号を以下のように使用できます。

- メッセージ作成元スレッドは、イベントクラスインスタンス番号を sf_message_header_t 型の共通メッセージヘッダーの event_b.class_instance ビットフィールドに設定してからメッセージをポストます。
- サブスクライバーリスト内の各サブスクライバーインスタンスは、メッセージを受信するために、イベントクラスインスタンス番号の範囲を指定する必要があります (sf_message_subscriber_t::instance_range.start および sf_message_subscriber_t::instance_range.end)。
- イベントクラスに複数のインスタンスが不要な場合は、サブスクライバーリストのサブスクライバーインスタンスで sf_message_header_t::event_b.class_instance、sf_message_subscriber_t::instance_range.start、sf_message_subscriber_t::instance_range.end にゼロを指定します。

イベントコード

イベントコードには、イベント定義の詳細が含まれています。たとえば、オーディオ再生イベントクラスのイベントコードは、再生開始と再生停止です。もう 1 つの例は、「time」イベントクラスの「set」または「get」です。イベントコードは sf_message_event_t で列挙され、プレフィックス SF_MESSAGE_EVENT_XXX を持ちます。イベントコードの定義は、ユーザーコードとイベントクラスコードに依存します。フレームワークには、いくつかのコードが例として用意されています。イベントコードの設定については、イベントクラスコードとイベントコードの設定を参照してください。イベントクラスインスタンス番号の最大値は 65535 です。

タッチパネルフレームワークは、イベントコードとしてのみ新しいデータを使用します。

アプリケーションは、イベントコードを以下のように使用できます。

- メッセージ作成元スレッドは、イベントコードを sf_message_header_t 型の共通メッセージヘッダーの event_b.code bit ビットフィールドに設定してからメッセージをポストます。
- メッセージサブスクライバースレッドは、メッセージを受信した後、メッセージヘッダーに設定されているイベントコードに従ってアクションを実行します。
- e^2 studio ISDE では、次のとおり、[Event] ウィンドウでイベントを追加できます。

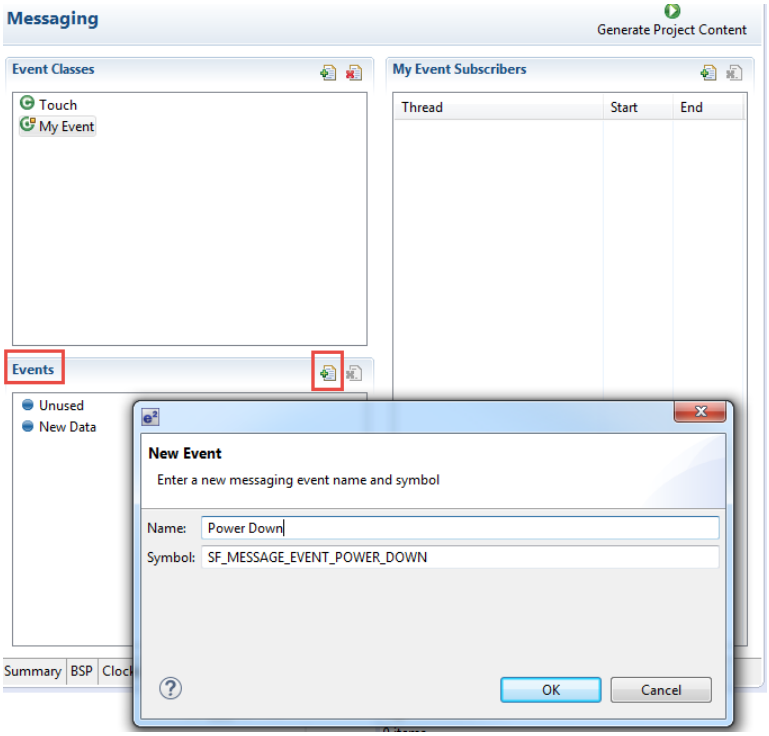

図 111: メッセージング – e² studio ISDE イベントの設定

4.1.13.4 サブスクライバーリスト

サブスクライバーリストの作成

サブスクライバーリストは、すべてのメッセージサブスクライバーのルックアップテーブルです。サブスクライバーリストはコンパイル時に設定されます。postAPI 関数が呼び出されると、メモリに静的にマップされ、フレームワークにより検索されます。フレームワークは、サブスクライバーリストを使用することで、メッセージの配信先のメッセージキューを決定できます。サブスクライバーリストは、下図に示す 2 つの構造体（sf_message_subscriber_list_t と sf_message_subscriber_t）からなります。

- サブスクライバースレッドのキューは、sf_message_subscriber_t インスタンスに登録されます。
- 同じイベントクラスコードのための上記インスタンスは、グループ化され、ポインタ配列に格納されます。
- 上記のサブスクライバーグループのポインタ配列は、sf_message_subscriber_list_t インスタンスに登録されます。
- サブスクライバーリストは、サブスクライバーグループ構造体へのポインタ配列です。サブスクライバーは、イベントクラスコードによってグループ化されます。
- 上記ポインタ配列は、NULL で終端している必要があります。

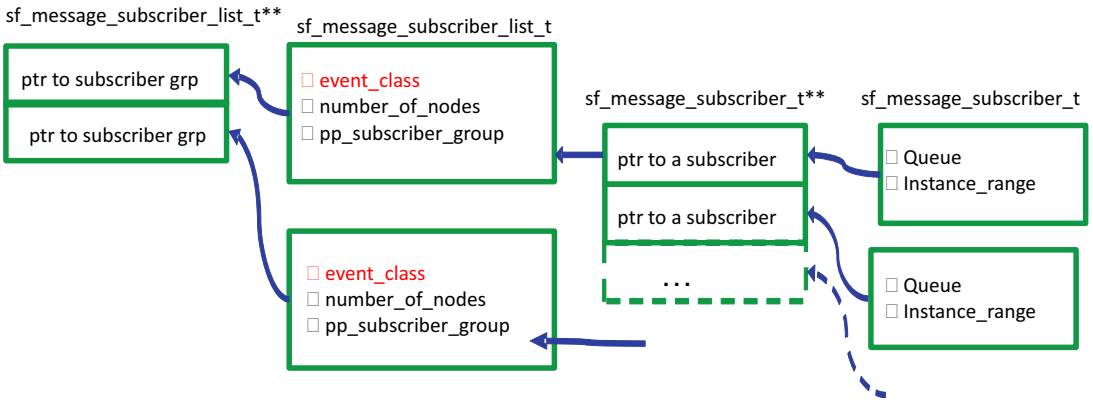

図 112:　メッセージ – サブスクライバーリスト

e² studio ISDE では、以下のとおり、[Threads] タブで名付けたスレッドのイベントクラス別にグループ化されたサブスクライバーを設定できます。以下の例では、スレッドは [Threads] タブの [Threads] ウィンドウで「マイスレッド」と名付けられています。開始値と終了値は、このスレッドが受け入れるイベントクラスのインスタンス数を反映しています。

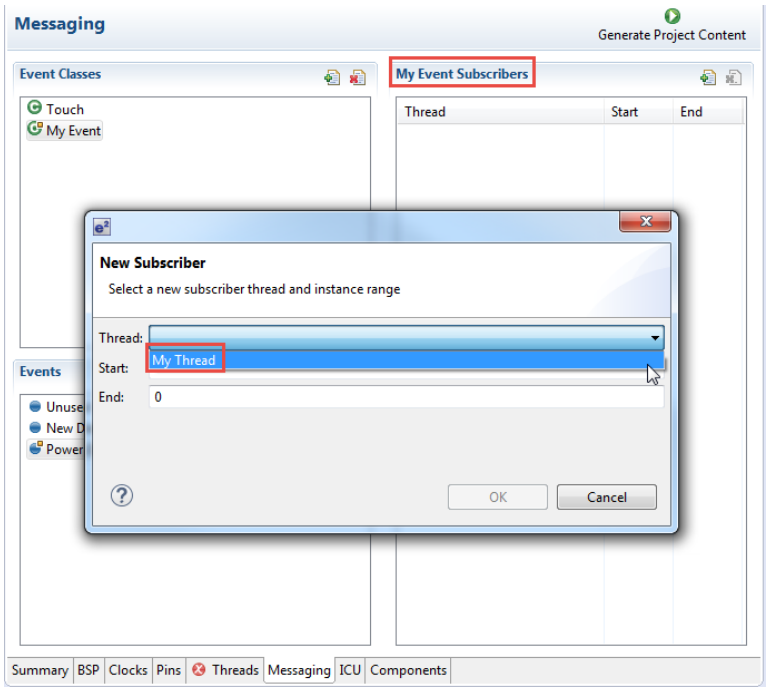

図 113:　メッセージング –e² studio ISDE サブスクライバーの設定

サブスクライバーリストの仕組み

サブスクライバーリストは、フレームワークによるメッセージ配信と検索に使用されます。

フレームワークは、sf_message_subscriber_list_t インスタンスへのポインタ配列の先頭にリストされたサブスクライバーグループから、各サブスクライバースレッドのメッセージキューの検索を開始します。サブスクライバーリストがイベントクラスコード（event_class）によりグループ化されるという点は重要です。

フレームワークが実行時に postAPI 関数のサブスクライバーリストを検索する際は、メッセージペイロードデータに含まれているメッセージヘッダー（sf_message_header_t::event_b.class）のイベントクラスコードを、サブスクライバーグループインスタンス（event_class）のものと比較します。一致する場合、フレームワークは、次のレベルに移動し、繰り返し回数が number_of_nodes に達するまでメッセージキューインスタンス（sf_message_subscriber_t::queue）を取得します。一致しない場合、フレームワークは次のサブスクライバーグループを検索し、sf_message_subscriber_list_t インスタンスへのポインタ配列に NULL が見つかるまで繰り返します。

上記の検索手順では、サブスクライバーリストの先頭にあるサブスクライバーグループのメッセージングスループットが最大になりますが、下位のサブスクライバーグループは不利となり、メッセージングスループットが低下します。

4.1.13.5 メッセージペイロード

メッセージペイロードは、メッセージ作成元とメッセージサブスクライバーが互いに通信するために使用する構築されたデータです。メッセージペイロードには、メッセージ作成元がサブスクライバーに発生したイベントについて通知するメッセージをポストできるよう、イベントクラスとイベントコードが共通ヘッダーに含まれます（sf_message_header_t 型のデータ、イベントクラスとイベントコードを参照）。

タッチパネルフレームワークモジュールやオーディオ再生モジュールなど、事前に定義された構造体が SSP によって提供されるモジュール以外では、システム固有のメッセージペイロード構造体を定義する必要があります。メッセージペイロードには、共通ヘッダーに加え、イベント処理に必要な追加データを含むことができます。

SSP 事前定義ペイロード

SSP には、以下の事前定義されたメッセージペイロード構造体が含まれています。

- sf_touch_panel_payload_t タッチパネルフレームワークモジュールの型
- sf_audio_playback_data_t オーディオ再生フレームワークモジュールの型

これらのフレームワークモジュールは、内部でメッセージフレームワークを使用して、最適なメッセージペイロード構造体を定義します。タッチパネルフレームワークモジュールからのタッチイベントメッセージをサブスクライブするアプリケーションスレッドは、ヘッダーファイル sf_touch_panel_api.h を含むことにより、事前定義されたメッセージペイロードを使用できます。同様に、オーディオイベントメッセージをオーディオ再生再生フレームワークモジュールにポストするアプリケーションスレッドは、sf_audio_playback_api.h を使用できます。

ユーザー定義ペイロード

各イベントクラスコードに対しメッセージペイロード構造体を定義する必要があります。ただし、上述の SSP 事前定義ペイロード、または共通ヘッダーのみを必要とするペイロードは例外です。新しいメッセージペイロード構造体を作成するには、共通のメッセージヘッダー（sf_message_header_t 型の構造体）を、ユーザー固有のメッセージペイロード構造体の先頭に追加します。ヘッダーのサイズは 4 バイトです。

Attention：ペイロードは、バッファサイズを超えてはなりません。

このバッファサイズによる制限はきわめて重要であり、バッファを超えて大量のデータを書き込むと、ThreadX カーネルに必要なブロックメモリプール内のデータが破壊される可能性があります。サイズの制限に違反すると、ハード故障例外が発生します。バッファサイズは buffer_size で設定できます。

以下の構造体定義は、温度メッセージペイロードの例です。

```
typedef struct st_sf_message_payload_temperature

{

    sf_message_header_t  header;

    float temperature;

} sf_message_payload_temperature_t;
```

4.1.13.6 e2 studio ISDE によるメッセージングフレームワークを使用するアプリケーションの作成

フレームワークモジュールは、e2 studio ISDE 内の SSP に組み込まれています（e2 studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します（プロジェクトの作成を参照）。
2) プロジェクトを設定します（プロジェクトの設定を参照）。
3) ドライバを追加します（ドライバのスレッドへの追加とドライバの設定を参照）。

メッセージングフレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Messaging Framework	Threads	[Thread Stacks] > [New Stack] で、[Framework] > [Service] > [Messaging Framework on sf_message] の順に選択します。メッセージングフレームワークは常に共通サービスであるため、新規スタックからのメッセージングフレームワークの選択は 1 度のみ許可されます。

メッセージングフレームワークの設定

e^2 studio ISDE を使用して、ドライバパラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

メッセージングフレームワークモジュールは論理モジュールです。ハードウェアに関連して必要な設定は、Cortex-M コア SysTick タイマのみです。ピンや割り込みなど、他のハードウェアを設定する必要はありません。

フレームワーク API 関数は、メッセージキュー、ブロックメモリプール、ミューテックス、タイムアウトのカウントに必要な SysTick サイクルおよびセマフォを使用します。フレームワークは、ThreadX カーネルの特殊な機能を使用しないため、特殊なカーネル設定なしで動作します。

表 : メッセージングフレームワークの構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking	Parameter Checking	BSP setting(Default), Enabled, Disabled	パラメータエラーチェックを有効または無効にします。
Name	Name	Arbitrary symbol(Default: " g_sf_message ")	メッセージングフレームワークモジュール制御ブロックインスタンスの名前。
Work memory size in bytes	p_work_memory_start	Arbitrary integer value(Default: 2048)	作業メモリサイズをバイト単位で指定します。小さい数を選択すると、同時に割り当てることができるバッファの数が少なくなります（総バッファ数は number_of_buffers で確認できます）。この値が、同時にポストできるピークメッセージ数よりも小さい場合、フレームワークでバッファ割り当てが失敗し、システムのパフォーマンスに影響を与えます。
Pointer to the Subscriber List array	pp_subscriber_lists	Arbitrary symbol(Default: "p_subscriber_lists")	サブスクライバーリスト配列へのポインタ名を指定します。

ISDE プロパティ	設定	設定値	説明
Block pool name	p_block_pool_name	Arbitrary symbol(Default: "sf_msg_blk_pool")	フーレムワークが制御ブロック内に作成するメモリブロックメモリの名前です。このパラメータは、デバッグ目的で有用な可能性がありますが、メモリを節約するために NULL を指定できます。

4.1.13.7 メッセージングフレームワークを使用したアプリケーションの作成

メッセージキューの作成

メッセージングコンフィギュレータは、サブスクライバーのメッセージキューを自動的に作成します。

[Generate Project Content] ボタンを押すと、メッセージングキューのコードが message_data.c ファイルの g_message_init 関数に生成されます。

```
...

void g_message_init(void)

{

    tx_queue_create (&sample_thread_message_queue, (CHAR *)
                     "Sample Thread Message Queue", 1,
                     &queue_memory_sample_thread_message_queue,
                     sizeof(queue_memory_sample_thread_message_queue));

}
```

イベントクラスとイベントの設定

独自のイベントクラスでメッセージングフレームワークを使用するには、e^2 studio ISDE のプロジェクトコンフィギュレータの [Threads] タブと [Messaging] タブを使用します。

[Threads] タブで、次の操作を実行します。

1) [Threads] ウィンドウの [Thread Stacks] パネルにメッセージングフレームワークコンポーネントを追加します。
2) [Threads] ウィンドウに新しいスレッドを追加し、一意の名前を付けます。

[Messaging] タブで、次の操作を実行します（イベントクラスコードを参照）。

1) [Event Classes] ウィンドウにクラスを追加し、サブスクライブするスレッドのイベントクラスの名前を [New Event Class] ダイアログボックスに入力します。

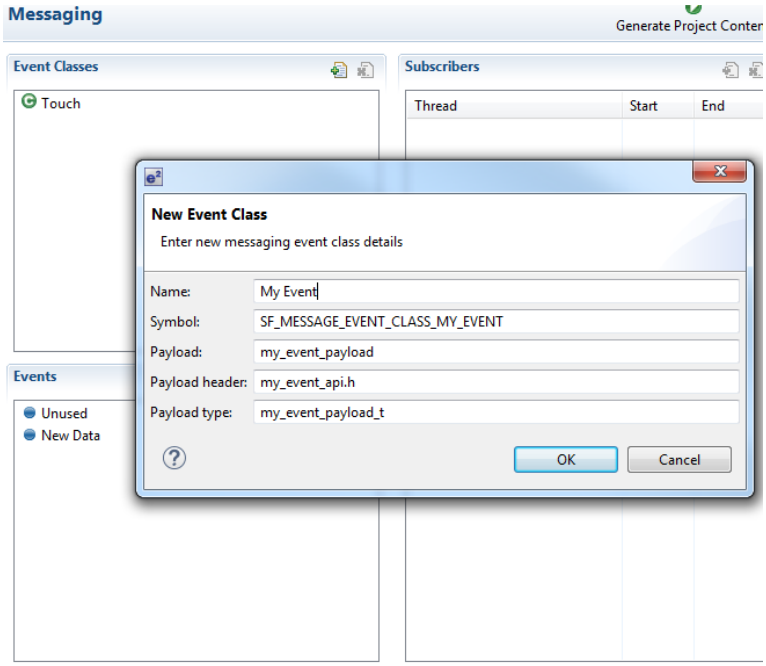

図 114: メッセージング – e² studio ISDE 新しいイベントクラスの設定

2) [Event] ウィンドウで、アプリケーションがサポートする可能性のあるイベントを追加します（イベントコードを参照）。

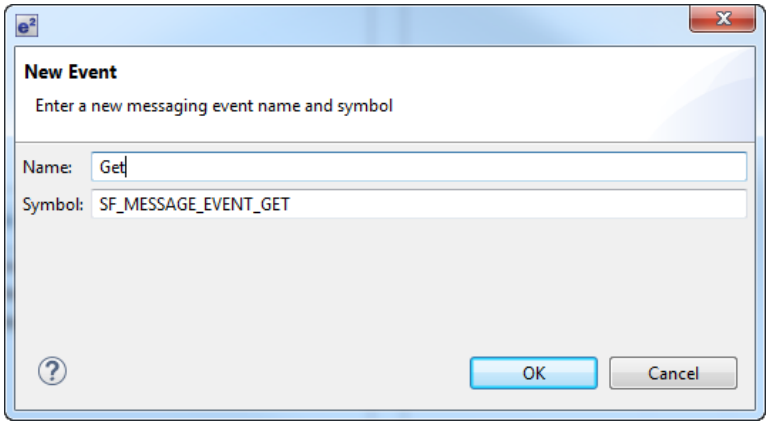

図 115: メッセージング – e² studio ISDE 新しいイベントの設定

カスタムイベントクラスコードとイベントコードは sf_message_port.h というファイルに格納されます。

オーディオ再生クラスとタッチイベントクラスは、SSP で事前定義されている 2 つのイベントクラスです。

```
typedef enum e_sf_message_event_class

{

    SF_MESSAGE_EVENT_CLASS_AUDIO, /* Audio *Playback* */

    SF_MESSAGE_EVENT_CLASS_TOUCH, /* Touch */

} sf_message_event_class_t;

typedef enum e_sf_message_event

{

    SF_MESSAGE_EVENT_UNUSED, /* Unused */

    SF_MESSAGE_EVENT_NEW_DATA, /* New Data */

    SF_MESSAGE_EVENT_AUDIO_START, /* Audio *Playback* Start */

    SF_MESSAGE_EVENT_AUDIO_STOP, /* Audio *Playback* Stop */

    SF_MESSAGE_EVENT_AUDIO_PAUSE, /* Audio *Playback* Pause */

    SF_MESSAGE_EVENT_AUDIO_RESUME, /* Audio *Playback* Resume */

} sf_message_event_t;
```

タッチイベントクラスは、新しいデータイベント SF_MESSAGE_EVENT_NEW_DATA のみを使用します。

サブスクライバーリストの設定

[Messaging] タブで、次の操作を実行します（サブスクライバーリストも参照）。

1) [Event Classes] ウィンドウで < イベントクラス名 > というイベントクラスを選択し、[<Event Class name> Subscribers] ウィンドウでサブスクライバーリストのスレッドを構成します。
2) [Threads] ダイアログボックスのドロップダウンリストからスレッドを選択します。
3) [Start] の横にイベントクラスインスタンスの開始番号を入力します。システムでこのイベントクラスに対し複数のイベントクラスインスタンスが使用されない場合、または指定する番号が不明な場合は、デフォルトの番号（0）のままにしておきます。使用可能な値は 0 ～ 255 です。
4) [End] の横にイベントクラスインスタンスの終了番号を入力します。システムでこのイベントクラスに対し複数のイベントクラスインスタンスが使用されない場合、または指定する番号が不明な場合は、デフォルトの番号（0）のままにしておきます。使用可能な値は 0 ～ 255 です。
5) [OK] をクリックします。指定したイベントのサブスクライバーがサブスクライバーリストに追加されます。

6) イベントクラスインスタンスが複数ある場合は、すべてでこの手順を繰り返します。

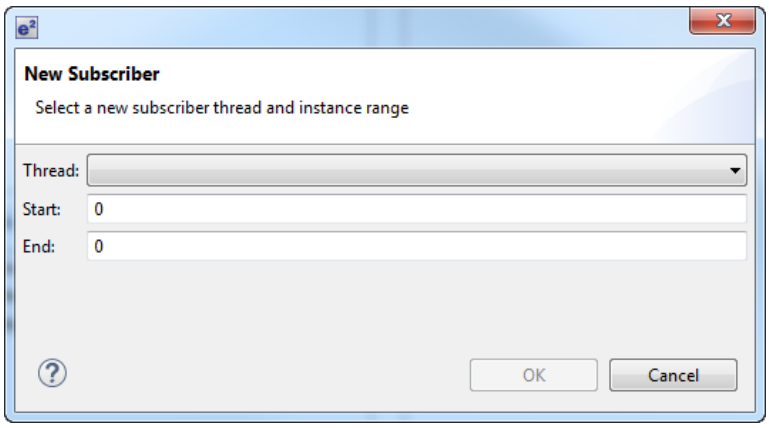

図 116: メッセージング – e² studio ISDE 新しいサブスクライバーの設定

以下のコードは、サブスクライバーリストの構造体定義の例です。

```
/* Subscriber instance for the system thread */

static sf_message_subscriber_t  g_message_system =

{

    .p_queue  = &g_system_queue,

    .instance_range.start = 0,

    .instance_range.end = 0

};

/* Pointer array to the subscribers for the SF_MESSAGE_EVENT_CLASS_SYSTEM Event Class */

static sf_message_subscriber_t  * gp_message_group_system[] =

{

    &g_message_system,

};

/* System Event Class subscriber group instance */

static sf_message_subscriber_list_t
```

```
g_message_subscriber_list_system =

{

   .event_class  = SF_MESSAGE_EVENT_CLASS_SYSTEM,

    .number_of_nodes = 1,

    .pp_subscriber_group = & gp_message_group_system

};

/* Touch Event Class subscriber group instance */

static sf_message_subscriber_list_t
```

```
g_message_subscriber_list_touch =

{

   .event_class  = (uint16_t) SF_MESSAGE_EVENT_CLASS_TOUCH,

    .number_of_nodes = 1,

    .p_subscriber_group = &g_message_group_system

};

/* Display Event Class subscriber group instance */

static sf_message_subscriber_list_t
```

```
g_message_subscriber_list_display =

{

   .event_class  = (uint16_t) SF_MESSAGE_EVENT_CLASS_DISPLAY,

    .number_of_nodes = 1,

    .p_subscriber_group = &g_message_group_system

};

/* Time event class subscriber group instance */

static sf_message_subscriber_list_t
```

```
g_message_subscriber_list_time =

{

    .event_class  = (uint16_t) SF_MESSAGE_EVENT_CLASS_TIME,

    .number_of_nodes = 1,

    .p_subscriber_group = &g_message_group_system

};

/* List of subscriber definitions */

sf_message_subscriber_list_t  * gp_message_subscriber_list [] =

{

    &g_message_subscriber_list_system,

    &g_message_subscriber_list_touch,

    &g_message_subscriber_list_display,

    &g_message_subscriber_list_time,

    NULL

};
```

イベントクラスコードとイベントコードの設定

メッセージペイロードの定義

独自のメッセージペイロード構造体を定義できます。ユーザー定義のメッセージ構造体には必ず、メンバーの 1 つとして sf_message_header_t 型の構造体を含む必要がありますが、他のメンバーは完全にユーザー定義が可能です。メッセージフレームワークでは、メッセージペイロード構造体がどこで定義されているかは認識しません。メッセージ作成元とサブスクライバーのスレッドのソースファイル内に独自のメッセージペイロード構造体を定義するファイルを含むことができます。

以下は、ユーザー定義のメッセージペイロードの例です。

```
typedef struct st_message_payload_user_defined

{

    sf_message_header_t  header;

    int data;

} sf_message_payload_user_defined_t;
```

メッセージングモジュールのオープン

sf_message_cfg_t 型設定パラメータを、システムに合わせて設定します。構成構造体用のコードは、Synergy Configuration ツールを使用して生成できます。[Threads] タブで、スレッドスタックに [Messaging Framework] コンポーネントを追加し、[Properties] ウィンドウでメッセージングフレームワークモジュールのプロパティを変更します。[Generate Project Content] ボタンを押すと、メッセージングフレームワークモジュールのコードがスレッドコードに生成されます。

コード例（common_data.c）を以下に示します。

```
extern sf_message_subscriber_list_t  * p_subscriber_lists[];

extern const sf_message_instance_t  g_sf_message_on_sf_message;

static sf_message_buffer_ctrl_t  g_sf_message_ctrl;

static uint8_t g_sf_message_work_buffer[2048];

// Configures the messaging framework

sf_message_cfg_t  g_sf_message_cfg = {

   .p_work_memory_start  = &g_sf_message_work_buffer,

    .work_memory_size_bytes = 2048,

    .buffer_size = sizeof (sf_message_payload_t),

    .pp_subscriber_lists = p_subscriber_lists,

    .p_block_pool_name = (uint8_t *)"sf_msg_blk_pool"

};

// Instance structure to use this module.

const sf_message_instance_t  g_sf_message = {

   .p_ctrl  = &g_sf_message_ctrl,
```

```
    .p_cfg = &g_sf_message_cfg,
    .p_api = &g_sf_message_on_sf_message
};
...
void g_common_init(void)
{
    g_common_init ();
      ssp_err_t  err_g_sf_message;
    /* Initializes Messaging Framework Queues */
    g_message_init ();
    /* Opens the messaging framework */
    err_g_sf_message = g_sf_message.p_api->open
      (g_sf_message.p_ctrl, g_sf_message.p_cfg);
    if (SSP_SUCCESS != err_g_sf_message)
      {
            /* Error returns, check the cause. */
           while (1)
            ;
      }
}
```

バッファの取得

1) メッセージをポストする前に、イベント作成元スレッドは、メッセージングフレームワークモジュールからメッセージ用のバッファを取得する必要があります。イベント作成元スレッドは、SF_MESSAGE_BufferAcquire を呼び出すことでバッファを取得できます。

2) API 関数が SSP_SUCCESS を返す場合、Synergy Configuration ツールで設定された [Message buffer size in bytes] のバッファは、メッセージングフレームワークによって管理されるメモリプールに割り当てられます。割り当て可能な最大数は、Synergy Configuration ツールで指定された [Work memory size in bytes] の設定によって異なります。最大数の見積もりについては、バッファ数の見積もりを参照してください。

SF_MESSAGE_BufferAcquire には、メッセージ送受信の動作を変えるためのいくつかのオプションがあります。

- バッファキープ : このオプションを使用すると、API 関数 bufferRelease により解放されないバッファをアプリケーションスレッドが保持します（true に設定した場合）。一般に、バッファは、メッセージが渡された後で bufferRelease によって解放されます。ただし、スレッド間の定期的なメッセージや繰り返されるメッセージがある場合は、バッファを何度も割り当てて解放することなく同じバッファをメッセージングに再利用できます。このオプションを有効にすると、バッファの割り当て / 解放操作のオーバーヘッドを低減してシステムスループットを向上できます。

- 待機オプション : これは、待ち時間オプションであり、すべてのバッファが取得された場合に有効です。任意の ThreadX ティック カウント、TX_WAIT_FOREVER、TX_NO_WAIT をこのオプションに設定できます。このオプションの詳細については、ThreadX ユーザー ガイドの ThreadX サービス呼び出し tx_block_allocate() の説明を参照してください。

```
// Example code

// The scenario for this example code:

// - The control block g_sf_message.p_ctrl is initiated by sf_message_api_t::open().

// - Get the address of allocated buffer on 'p_buffer'.

// - Mark buffer keep option to reuse the buffer later without release the buffer.

// - Wait 300 OS tick cycle if all the buffer has been acquired.

ssp_err_t  error;

sf_message_header_t  * p_buffer;

sf_message_acquire_cfg_t  acquireCfg =

{

    .buffer_keep  = true

};

// Gets buffer in the block memory pool managed by the Messaging Framework module

g_sf_message.p_api->bufferAcquire  (g_sf_message.p_ctrl, &p_buffer, &acquireCfg, 300);
```

バッファの解放

メッセージ サブスクライバー スレッドは、イベント作成元によってポストされたメッセージを受信した後、バッファをフレームワークに解放する必要があります。バッファの解放は bufferRelease を呼び出すことで実行します。システムに複数のイベント サブスクライバーがある場合、API 関数は複数回呼び出される可能性があるため、実際のバッファ解放は、イベント サブスクライバーの中の最後のメッセージ サブスクライバー スレッドによってのみ実行されます。たとえば、イベント クラスのサブスクライバー グループ中に 3 つのサブスクライバーがある場合、bufferRelease を呼び出す 1 番目と 2 番目のスレッドはバッファを開放しません。3 番目のスレッドのみがバッファを解放します。SF_MESSAGE_BufferAcquire により buffer keep オプションが指定されている場合、オプション SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE が API 関数の引数 option に渡される場合を除いて、バッファが解放されることはありません。API 関数の使用法については、メッセージング フレームワークのコールバックの bufferRelease も参照してください。

この API は、ユーザー コールバック関数を呼び出して、イベント作成元スレッドとメッセージ サブスクライバー スレッドの間でハンドシェイクを確立するためにも使用されます。

```
// Example code

// The scenario for this example code:

// - The control block g_sf_message.p_ctrl is initiated by sf_message_api_t::open().

// - The address of buffer 'p_buffer' has been obtained by sf_message_api_t::pend().

// - This API is called by the event listener thread.

// - The event listener thread has been receiving a message and has done event

// processing corresponding to the event.

// - Provides no buffer release option

// Performs event close processing. Invokes a user callback function if it is

// registered by the message producer thread, then releases an allocated buffer

// if this thread is the last message receiver in multiple subscriber case.

g_sf_message.p_api->bufferRelease (g_sf_message.p_ctrl,

            &p_buffer,

                    SF_MESSAGE_RELEASE_OPTION_NONE);
```

メッセージのポスト

1) bufferAcquire でバッファを取得した後、イベント作成元はメッセージ ペイロード データをバッファに書き込むことができます。

   Attention：データをバッファに書き込むのはユーザーの責任であり、バッファ サイズを超えてデータを書き込むと、メッセージング フレームワーク モジュールで致命的なエラーとなります。

2) ペイロード構造体の sf_message_header_t::event_b.class にイベント クラス コードを書き込みます。

3) ペイロード構造体の sf_message_header_t::event_b.code にイベント コードを書き込みます。この指定は必須ではありませんが、多くの場合に必要となります。

4) sf_message_header_t::event_b.class_instance にイベント クラス インスタンス番号を書き込みます。特定のイベント クラスのインスタンスが複数、システムにある場合は、0 ～ 255 の番号を指定します。イベント クラスのインスタンスがシステムに 1 つしかない場合は、0 を指定します。

5) post によってメッセージをポストします。バッファへのポインタは API に渡す際 sf_message_header_t * 型でキャストする必要があります。メッセージは、メッセージ サブスクライバー リスト中でメッセージ サブスクライバーとして登録されているメッセージ サブスクライバーに配信されます。post には、メッセージ送受信の動作を変えるためのいくつかのオプションがあります。

- **メッセージの優先度**：メッセージには、2 つのレベルのメッセージ優先度 SF_MESSAGE_PRIORITY_NORMAL と SF_MESSAGE_PRIORITY_HIGH があります。SF_MESSAGE_PRIORITY_HIGH が指定された場合、メッセージは、メッセージ サブスクライバーのメッセージ キューの先頭に格納されます。一般にこれは、緊急メッセージに使用され、メッセージ サブスクライバーは、メッセージ キューに格納済みのイベントよりも前にそのイベント処理するようになります。
- **ユーザー コールバック関数**：この関数は、メッセージング フレームワークモジュールのバッファ制御ブロックに登録されます。コールバック関数は、bufferRelease によって呼び出されます。この関数は、イベント作成元スレッドとメッセージ サブスクライバー スレッドの間でのハンドシェイクに使用できます。
- **待機オプション**：これは、待ち時間オプションであり、メッセージ サブスクライバー スレッドのメッセージ キューが一杯の場合に有効です。任意の ThreadX ティック カウント、TX_WAIT_FOREVER、および TX_NO_WAIT をこのオプションに設定できます。このオプションの詳細については、ThreadX ユーザー ガイドの ThreadX サービス呼び出し tx_queue_sennd() の説明を参照してください。

```
// Example code

// The scenario for this example code:

// - The control block g_sf_message.p_ctrl is initiated by sf_message_api_t::open().

// - The address of buffer 'p_buffer' has been obtained by

// sf_message_api_t::bufferAcquire().

// - Sends message to event listeners who are subscribing SF_MESSAGE_EVENT_CLASS_USER

// Event Class

// - Event class instance number is 0.

// - Sends SF_MESSAGE_EVENT_USER event.

// - Posts a message with normal priority.

// - Wait 300 OS tick cycle if blocked at message queue posting.

// - No user callback function registered.

sf_message_post_cfg_t  post_cfg =

{

    .priority  = SF_MESSAGE_PRIORITY_NORMAL;

   .p_callback = NULL;

};
```

```
sf_message_post_err_t  errPost; // A variable for error return

sf_message_payload_user_defined_t * pPayload; // User defined payload

// Casts the buffer to the user defined payload type.

pPayload = (sf_message_payload_user_defined_t *)p_buffer;

pPayload->p_header->event_b.class = SF_MESSAGE_EVENT_CLASS_USER;

pPayload->p_header->event_b.class_instance = 0

pPayload->p_header->event_b.code = SF_MESSAGE_EVENT_USER;

// Posts message to the messaging system.

g_sf_message.p_api->post  (g_sf_message.p_ctrl,

        (sf_message_header_t  *)pPayload,

                &post_cfg,

                &errPost,

                300);
```

保留メッセージの確認

1) メッセージング フレームワーク モジュールがオープンされた後、メッセージ サブスクライバー スレッドは、pend を呼び出すことで、メッセージを待つことができます。通常の使用では、[Thread Stacks] ペインでメッセージ サブスクライバー スレッドに対して設定した、API の第 2 引数はメッセージ キューへのポインタを指定します （[Threads] タブ、ISDE メッセージング コンフィギュレータ）が、必要に応じて他のメッセージ キューを指定することも可能です。
2) イベント作成元からメッセージが配信されると、スレッドは pend から返されます。
3) API は、API の第 3 引数を介してスレッドへのメッセージを含むバッファへのポインタを返します。
4) メッセージ サブスクライバーは、ユーザー カスタム メッセージ ペイロード構造体に上記のポインタ型のポインタをキャストし、イベント クラス コード sf_message_header_t::event_b.class とイベント コード sf_message_header_t::event_b.code、メッセージ内にあるユーザー定義の任意のデータに応じてイベントを処理します。

pend には、API 関数の動作を変更する wait_option オプションがあります。

pend の第 4 引数は待ち時間オプションであり、メッセージ サブスクライバー スレッドのメッセージ キューが空の場合に有効です。任意の ThreadX ティック カウント、TX_WAIT_FOREVER、および TX_NO_WAIT

をこのオプションに設定できます。このオプションの詳細については、ThreadX ユーザー ガイドの ThreadX サービス呼び出し tx_queue_sennd() の説明を参照してください。

```
// Example code

// The scenario for this example code:

// - The control block g_sf_message.p_ctrl is initiated by sf_message_api_t::open().

// - The message queue used for this event listener thread is 'myQueue'.

// - Wait forever until receiving message.

sf_message_header_t  * pHeader; // Common message header

sf_message_payload_user_defined_t * pPayload; // User defined payload

// Event loop

while (1)

{

        // Pends on a message.

        g_sf_message.p_api->pend  (g_sf_message.p_ctrl,

    &myQueue, // Specify the symbol of message queue you named for

                    // the message subscriber thread on the e2 studio Synergy Configurator

                    // Messaging tab.

        &pHeader,

        TX_WAIT_FOREVER);

        switch (pHeader->event_b.class)

        {

              case SF_MESSAGE_EVENT_CLASS_USER:
```

```
            pPayload = (sf_message_payload_user_defined_t *) pHeader;

            if (pPayload->p_header->event_b.code == SF_MESSAGE_EVENT_USER)

            {

            // Do processing corresponding to the received Event code

            }

            ...

            break;

        ...

        }

}
```

メッセージング フレームワークのクローズ

必要に応じてメッセージング フレームワーク モジュールをクローズできます。

```
// Example code

// The scenario for this example code:

// - The control block g_sf_message.p_ctrl is initiated by sf_message_api_t::open().

// Closes the messaging framework module

g_sf_message.p_api->close (g_sf_message.p_ctrl);
```

4.1.13.8 メッセージング フレームワークの割り込み

フレームワークは割り込みを使用しません。

4.1.13.9 メッセージング フレームワークのコールバック

メッセージング フレームワーク モジュールは、ユーザー コールバック関数の呼び出しをサポートしています。これにより、イベント作成元スレッドとメッセージ サブスクライバー スレッドは、メッセージの送受信後にハンドシェイクを行うことができます。ユーザー コールバック関数は、post を呼び出し、設定パラメータでこの関数を指定することで、イベント作成元スレッドによる登録が可能です。コールバック関数は、イベント クラスのサブスクライバー内の最後のメッセージ サブスクライバー スレッドにより、bufferRelease を呼び出すことで呼び出されます。コールバック関数 event の引数には、以下に示す列挙型の

コールバック イベントを使用できます。これにより、メッセージがメッセージ サブスクライバー スレッドによってどのように処理されたかをイベント作成元スレッドが識別できます。

イベント名	イベントの発生条件
SF_MESSAGE_CALLBACK_EVENT_ACK	イベントのすべてのメッセージ サブスクライバーが、SF_MESSAGE_RELEASE_OPTION_ACK オプションにより bufferRelease を呼び出しました。
SF_MESSAGE_CALLBACK_EVENT_NAK	イベントの 1 つ以上のメッセージ サブスクライバーが、SF_MESSAGE_RELEASE_OPTION_NAK オプションにより bufferRelease を呼び出しました。

Note：bufferRelease に指定された SF_MESSAGE_RELEASE_OPTION_NAK オプションは、複数のサブスクライバーがイベント クラスをリッスンする場合に備えてバッファ制御ブロックと ORed に保存されます。SF_MESSAGE_RELEASE_OPTION_NAK が bufferRelease オプションに設定されているメッセージ サブスクライバー スレッドがあれば、コールバック イベントは SF_MESSAGE_CALLBACK_EVENT_NAK になります。

4.1.13.10 メッセージング フレームワーク使用上の注意

メッセージング フレームワークと OS メッセージ キュー サービス

メッセージング フレームワーク モジュールは、ThreadX のプリミティブ メッセージ キューとカーネル サービスを使用し、ThreadX RTOS の機能に対するいくつかの拡張をサポートしています。このため、メッセージング フレームワーク モジュールは、ThreadX のメッセージ キュー サービスと全く同じように動作するわけではありません。ただし、メッセージング フレームワーク モジュールを使用したメッセージング システムは、アプリケーション内で ThreadX メッセージ キュー サービスと同時に使用できます （2 つのメッセージング システムが分離されている場合）。

API 呼び出しのコンテキスト

API 呼び出しは以下のコンテキストで使用できます。

- open は、1 つのスレッドからのみ呼び出すことができます。メッセージ フレームワーク制御ブロック インスタンスあたり 1 回だけ呼び出すことができます。この関数を 2 回呼び出したときの動作は不定です。
- close は、1 つのスレッドからのみ呼び出すことができます。
- bufferAcquire は、1 つのスレッドおよび 1 つの ISR から呼び出すことができます。
- bufferRelease は、1 つのスレッドからのみ呼び出すことができます。
- post は、1 つのスレッドおよび 1 つの ISR から呼び出すことができます。
- pend は、1 つのスレッドおよび 1 つの ISR から呼び出すことができます。

バッファ数の見積もり

メッセージング システムを設計する際には、作業メモリ内で割り当てることができるバッファの数を正しく見積もる必要があります。バッファ数は以下のように見積もります。

(作業メモリ サイズ) / (メッセージ バッファ サイズ + sizeof (バッファ制御ブロック) + (ThreadX 用に予約されたバイト数)) = (作業メモリ サイズ) / (メッセージ バッファ サイズ + 12 バイト + 4 バイト)

同時に割り当てることができるバッファの最大数は、システム内のメッセージ キューの深さの総数に等しくなります。そのため、理想的には、堅牢なシステムのバッファ数は、システム内のメッセージ キューの深さの合計にする必要があります。

メッセージ キューのサイズと深さの設定

メッセージング フレームワークは、メッセージ ペイロードが格納されたバッファのポインタを配信するため、メッセージ キュー上に 4 バイトのメモリ ブロックのみを必要とします。このため、メッセージ キューのサイズは 4 バイトに固定する必要があります。サイズが 4 バイトよりも大きいと、メッセージング フレームワーク API 関数によって内部的に実行される ThreadX メッセージ キュー サービスで余分なメモリがコピーされるため、パフォーマンスに悪影響を与えます。

メッセージ キューの深さは任意ですが、実行時にキューに格納されるメッセージ数を収容できるようにする必要があります。ガイドラインとして、次のように値を見積もります。

キューの深さ:(作成元からの平均メッセージ配信速度) / (サブスクライバーでの平均イベント ループ完了時間)

4.1.13.11 メッセージング フレームワークでサポートされるオペレーティング システム

本質的に、メッセージング フレームワーク モジュールは、RTOD の操作を必要とします。このモジュールは、RTOS でサポートされているメッセージ キュー、メモリ ブロック プール、およびミューテックスを使用します。メッセージング モジュールには ThreadX が必要です。

4.1.13.12 メッセージング フレームワークの制限事項

メッセージ フレームワーク モジュールの最新情報については、SSP のリリースノートを参照してください。

4.1.13.13 メッセージング フレームワークでサポートされるデバイス

このドライバは、S7G2 でテストされています。

メッセージング フレームワークは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.1.14 パワープロファイルフレームワーク

パワー プロファイル フレームワークは、より低電力のソフトウェア スタンバイ モードに MCU を配置できるように設定が可能な汎用 API です。このモジュールは sf_power_profiles に実装され、3 種類の動作モード（実行、RTC および外部割り込み）のうちの 1 つで動作するように設定できます。これらのモードは、ソフトウェア スタンバイ中に無効にするクロックと周辺機器を定義し、さらにソフトウェア スタンバイ モードの開始前と終了後の出力ピン状態も定義します。

パワー プロファイル フレームワークは、MCU の RTC、LPM、IOPORT および CGC 周辺機器を使用するもので、ローパワー動作モードにアクセスする際に使いやすいソフトウェア インタフェースとなっています。

このセクションでは、e^2 studio ISDE を使用してパワー プロファイル フレームワークを設定する方法と、API 関数をアプリケーションにインクルードする方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Service] > [Power Profiles Framework of sf_power_profiles] を選択することで、パワープロファイル フレームワーク モジュールを追加および構成できます。詳細については、以下を参照してください。e^{2}studio ISDE によるパワー プロファイル フレームワークを使用するアプリケーションの作成

API リファレンスは、次のフレームワークパワープロファイルインタフェースの説明内に記載されています : パワー プロファイルフレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

下記のブロック図は、スリープ開始 （ソフトウェアスタンバイ）モードの処理フローを示したものです。

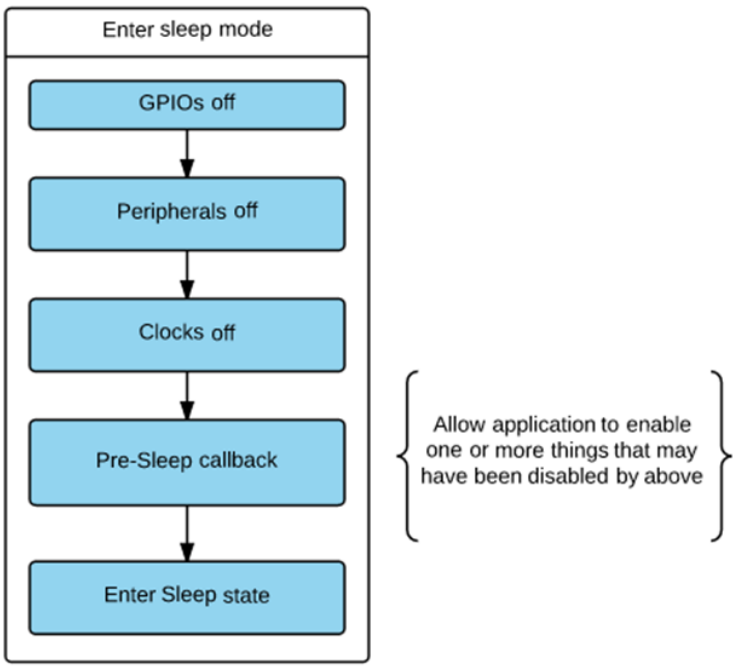

図 117:　パワープロファイルフレームワーク – スリープ処理 ブロック図

下記のブロック図は、スリープ （ソフトウェア スタンバイ）モードからの復帰に関する処理フローを示したものです。

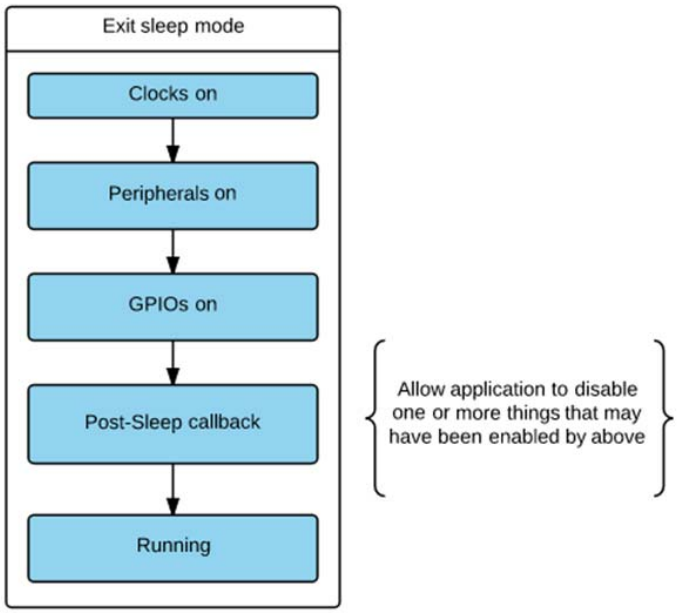

図 118: パワー プロファイル - 復帰処理 ブロック図

4.1.14.1 パワー プロファイル フレームワークの機能

パワー プロファイル フレームワークにより、事前に定義された状態でソフトウェア スタンバイ モードの開始と終了を実行できるように、システムが設定されます。

パワー プロファイル モジュールは、ユーザーが行う初期設定により動作モード （RTC または外部割り込み）が決定されるという形で動作します。パワー プロファイルを使用して、ユーザーは、2 つのピン設定表（ソフトウェア スタンバイおよび復帰）を定義できます。それらの表へのポインタは、復帰およびソフトウェア スタンバイのピン設定に関するパワー プロファイルのプロパティに記載されています。これはピン設定表の定義にあたって必須ではありません。これらのポインタの一方または両方に NULL を指定することにより、それぞれのスリープ / 復帰状態に関するピンの再設定が防止されます。

設定表により、ソフトウェア スタンバイ モード中とその後の復帰時での MCU ポート ピンの状態が定義されます。両方の表は、ともに bsp_pin_cfg.h ファイルをベースとして使用し、これに従って表の名前などを含む全体を修正することにより生成できます。

パワー プロファイル フレームワークは、スレッド環境と非スレッド環境の両方で使用できます。

RTC ドライバの RTC 実装は、RTC モードを使用する際にこのフレームワークに必須です。

LPM ドライバの LPM 実装は、両方のモードでこのフレームワークに必須です。

次を参照してください：

- RTC インタフェース と RTC ドライバ

- 低電力モードインタフェース とローパワー モードドライバ

4.1.14.2 e²studio ISDE によるパワー プロファイル フレームワークを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （ドライバのスレッドへの追加とドライバの設定を参照）。

パワー プロファイル フレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Power Profiles Framework	Threads	HAL/Common パネルから、[Framework] > [Services] > [Power Profiles Framework on sf_power_profiles] の順に選択します。このモジュールを、ThreadX アプリケーションの共通スレッドまたは RTOS 独立 HAL レベル アプリケーションに追加できます。ThreadX アプリケーションでは、このモジュールをスレッドに追加することもできます。

さらに、MCU で実行できるパワー プロファイル アプリケーションを作成するには、LPM と RTC ドライバ（オプション）が必要です。

リソース	ISDE タブ	選択
LPM Driver	Threads	[Driver] > [Power] > [Low Power Modes Driver on r_lpm]
RTC Driver	Threads	[Driver] > [Timers] > [RTC Driver on r_rtc]

パワー プロファイル フレームワークのクロックの設定

パワー プロファイル フレームワークは、固有のクロック設定を必要としません。RTC のソース クロックの仕様は、RTC HAL 使用上の注意 （RTC ドライバ用のクロックの設定）に記載されています。

パワー プロファイル フレームワークのピンの設定

e^2 studio ISDE を使用して、[Pins] タブから パワー プロファイル ピンを設定します （ピンの設定を参照）。

このアプリケーションでは、ソフトウェア スタンバイ開始前とソフトウェア スタンバイからの復帰後のポート ピン状態の設定に使用する、ソフトウェア スタンバイと復帰時のピン設定表を任意に利用できます。パワー プロファイル フレームワークの機能を参照してください。

パワー プロファイル フレームワークの割り込みの設定

どの割り込みでもソフトウェア スタンバイ モード中に MCU を復帰できるものの、パワー プロファイルは割り込みを直接には使用しません。

パワー プロファイル フレームワーク パラメータの設定

e^2 studio ISDE を使用して、SF パワー プロファイル ドライバ パラメータで SF パワー プロファイルを設定します （ドライバのスレッドへの追加とドライバの設定 を参照）。

<SF Power Profiles Framework on SF Power Profiles> を追加することにより、各構成パラメータについて選択した設定が記載された sf_power_profiles_cfg.h ファイルが synergy_cfg\framework ディレクトリに生成されます。

パワー プロファイル フレームワークのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking	SF_POWER_PROFILES_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE, Enabled or Disabled	API パラメータ チェック用のコードを含めるかどうかを制御します。
RTC Support	SF_POWER_PROFILES_RTC_SUPPORT_ENABLE	Enabled or Disabled	RTC モードを使用するには有効にする必要があります。外部モードで無効に設定すると、モジュール コード サイズが少し小さくなります。

Note：どのような構成エラーも捕捉できるように、パラメータ チェックを有効にした状態でアプリケーションを開発することを お勧めします。実稼働コードではパラメータ チェックを無効化して、コード スペースと実行時間を節約できます。

e^2 studio ISDE を使用して、g_sf_power_profiles ドライバ パラメータを設定します（ドライバのスレッドへの追加とドライバの設定を参照）。

パワー プロファイル フレームワーク設定

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	NULL or valid name	ソフトウェア スタンバイ前 / ソフトウェア スタンバイ後のコールバック関数の名前があれば指定します。
Wakeup pin config table	p_wake_ioport_pin_tbl	NULL or Name of theioport_cfg_t table that will contain Wakeup port pin settings.	この名前は、復帰後に使用するポート ピン設定に関する、ユーザーが定義した表へのポインタにあたります。
Sleep pin config table	p_sleep_ioport_pin_tbl	NULL or Name of theioport_cfg_t table that will contain Software Standby port pin settings.	この名前は、ソフトウェア スタンバイに入る直前に使用するポート ピン設定に関する、ユーザーが定義した表へのポインタにあたります。
Lower Level RTC Name	p_lower_lvl_rtc	Name of the RTC driver being used for RTC Mode.	動作モードが RTC の場合は、これはパワー プロファイル モジュールからアクセスされる RTC ドライバの名前になります。
Lower level LPM Name	p_lower_lvl_lpm	Name of the LPM driver being used	パワー プロファイル モジュールには、LPM モジュールが必要です。LPM ドライバの設定は一切不要です。
Operating mode	operating_mode	Power Profiles operating mode	RUN、RTC、または外部割り込みモード。

HAL RTC パラメータの設定

RTC モジュールは、動作モードが RTC の場合にのみ必要です。この場合、必要に応じて RTC を設定し、希望する定期的復帰間隔を生成する必要があります。パワー プロファイル モジュールは、RTC ソース クロックのクエリに RTC インスタンスを使用しますが、ユーザーが定義した設定に変更は加えません。パワー プロファイル RTC モードの使用には RTC サポートが有効に設定されていることと動作モードが RTC に設定されていることの両方が必要ですのでご注意ください。

ローパワー モジュール パラメータの設定

LPM 低電力モジュールは、パワー プロファイル フレームワークにより内部で設定されます。このため、ユーザーはモジュール名の指定を除いて、設定を行う必要はありません。

4.1.14.3 パワー プロファイル フレームワーク アプリケーションの作成

ISDE コンフィギュレータを使用して、Framework\Services から入手できるパワー プロファイル フレームワーク モジュールの追加と設定を行い、LPM と RTC を含む必要な低電力 HAL モジュールについても追加と設定を行います。

モジュールの設定が完了し、ISDE ファイルが生成されたら、以下の手順に従ってパワー プロファイル フレームワークをスレッドで使用します。パワー プロファイルは、任意にスタンドアロン （非スレッド環境）アプリケーションで使用できます。

以下の手順は、パワー プロファイル フレームワーク インスタンスのユーザー定義名が g_sf_power_profiles であることを前提とします。

ISDE により、このモジュールのインスタンス構造体が定義されます。

```
/* Instance structure to use this module. */

const sf_power_profiles_instance_t  g_sf_power_profiles =

{

    .p_ctrl  = &g_sf_power_profiles_ctrl,

    .p_cfg = &g_sf_power_profiles_cfg,

    .p_api = &g_sf_power_profiles_on_sf_power_profiles

};
```

1) パワー プロファイル フレームワーク フィールド p_callback に設定したコールバック関数の本体を定義します。MCU がソフトウェア スタンバイ モードに移行する直前と、ソフトウェア スタンバイ モードから復帰した直後に、コールバック関数によってアプリケーションに通知が行われます。コールバックの使用は任意ですが、パワー プロファイルのプロパティにコールバックを 1 つ定義している場合は、それに対する定義が必要であることに注意してください。

2) アプリケーション スレッドで、open() 関数呼び出しを使用してフレームワークを初期化します。

```
g_sf_power_profiles.p_api->open(g_sf_power_profiles.p_ctrl, g_sf_power_profiles.p_cfg);
```

3) sleep() 関数によりソフトウェア スタンバイ モードに入ります。

```
g_sf_power_profiles.p_api->sleep(g_sf_power_profiles.p_ctrl);
```

4) close() 関数を呼び出してフレームワークを閉じます。

```
g_sf_power_profiles.p_api->close(g_sf_power_profiles.p_ctrl);
```

4.1.14.4 パワー プロファイル フレームワーク使用上の注意

パワー プロファイル フレームワークを使用して、すべての RAM を保持し、パラメータを操作しながら非常に低電力を使用するソフトウェア スタンバイ モードに、ユーザーが MCU を配置することができます。外部割り込みモードは、主に RTC に加えて LOCO および Sub-clock のクロック ソースがオフになっているため、最小限の電力しか使用しないモードです。RTC モードでは、RTC は動作を許可されているため、結果的により多くの電力を使用します。

通常 MCU がソフトウェア スタンバイ モード中に動作の継続を許可している周辺機器またはクロックは、ソフトウェア スタンバイ モードに入る前にパワー プロファイル モジュールによりオフにされ、その後復帰時にオンになります。

この動作対象から除外したい動作が他にある場合は （LOCO をソフトウェア スタンバイ中も動作させる場合など）、SF_POWER_PROFILES_EVENT_PRE_SLEEP または SF_POWER_PROFILES_EVENT_POST_SLEEP コールバック イベントを使用し、特定の動作を含めることで可能となります。

4.1.14.5 パワー プロファイル フレームワークでサポートされるハードウェア実装

このドライバは、S7G2 と S3A7 でテストされています。

4.1.14.6 パワー プロファイル フレームワークでサポートされるオペレーティング システム

ThreadX と併用した場合、このフレームワークはミューテックスのような ThreadX に固有のオブジェクトを使用します。ThreadX を利用した動作はオプションです。

4.1.14.7 パワー プロファイル フレームワークの制限事項

パワー プロファイル フレームワークは、現在ソフトウェア スタンバイ以外のどのスリープ モードもサポートしていません。

4.1.15 SPI フレームワーク

SPI フレームワークは、ThreadX 対応の一連のフレームワーク API で、sf_spi に実装されます。SPI フレームワークは、チップ選択処理とそのレベルのアクティブ化を含め、SPI バス上の複数の SPI 周辺機器の統合と同期を処理します。SPI フレームワークを使用すると、1 つ以上の SPI バスを作成し、複数の SPI 周辺機器を 1 つの SPI バスに接続できます。SPI フレームワークは、単一のインタフェースを使用して、SCI SPI ドライバと RSPI ドライバの両方にアクセスします。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Connectivity] > [SPI Framework Device on sf_spi] を選択することで、外部 SPI フレームワーク モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による SPI フレームワーク デバイスを使用するアプリケーションの作成

API リファレンスは、次のフレームワーク ドライバ インタフェースの説明内に記載されています : SPI フレームワークインタフェース。

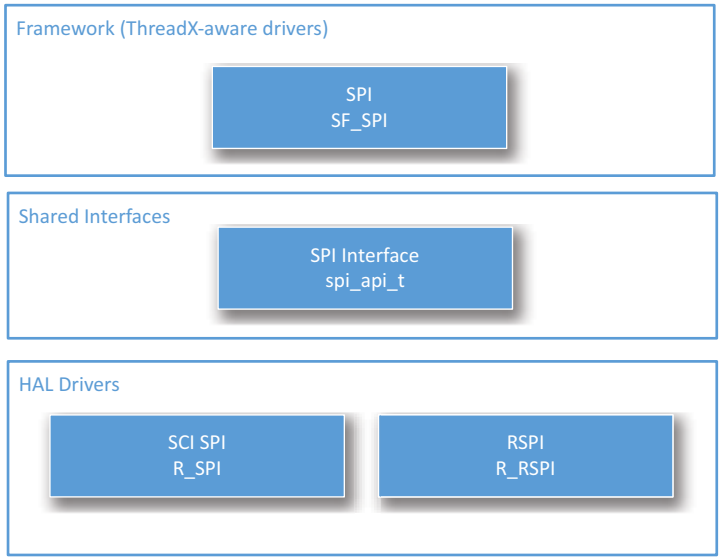

図 119: SPI フレームワーク - ブロック図

4.1.15.1 SPI フレームワークの機能

SPI フレームワークは、単一のインタフェースを使用して、SCI SPI ドライバと RSPI ドライバの両方にアクセスします。SPI フレームワーク ドライバは、SSP の階層化されたドライバ アーキテクチャに準拠しています。

フレームワーク ドライバ アーキテクチャは、「バス」と「バス上のデバイス」によるアーキテクチャを使用します。フレームワーク ドライバ アーキテクチャは、「バス」と「バス上のデバイス」によるアーキテクチャを使用します。下位では一度に 1 つのデバイスのみが設定され、残りのデバイスは、読み取りまたは書き込み操作の際に再設定されます。ユーザーは、バス、フレームワーク、バスに接続されている各デバイスのローレベル ドライバ レイヤーを設定する必要があります。共通の開始および停止手順が、すべての SPI データ転送操作 （読み取り、書き込み、および読み書き）に使用されます。開始処理の中で、フレームワーク ドライバは、再設定が必要かどうかと、デバイスにバスとの互換性があるかどうかを確認します。チップ選択は、転送開始処理の中でアサートされ、転送終了処理の中でアサート解除されます。共通の開始および停止手順が、すべての SPI データ転送操作 （読み取り、書き込み、および読み書き）に使用されます。

共通の開始および停止手順が、すべての SPI データ転送操作に使用されます。ロックすると、一定期間（ロックとロック解除の間）、デバイス用にバスを予約できるようになります。これにより、一部のニーズに対応して、デバイスが複数の読み書き操作を中断することなく完了できるようになります。これにより、一部のニーズに対応して、デバイスが複数の読み書き操作を中断することなく完了できるようになります。ロックとロック解除の間に書き込みと読み取りを行っても、チップ選択ラインは変更されません。

4.1.15.2 e² studio ISDE による SPI フレームワーク デバイスを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

参考資料

e^2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定 を参照）。
3) ドライバを追加します （ドライバのスレッドへの追加とドライバの設定 を参照）。

リソース	ISDE タブ	選択
Framework SPI Driver	Threads	新しいスレッドをハイライトして、[New] > [Framework] > [Connectivity] >[SPI Framework Device on sf_spi] の順に選択します。
Framework SPI Bus Driver	Threads	HAL/Common をハイライトして、[New] > [Framework] > [Connectivity] > [SPI Framework Shared Bus on sf_spi] の順に選択します。共有バスは常に共通サービスであるため、ISDE ではスレッドで共有バスを選択できません。

SPI フレームワークと SCI モジュールを併用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
SCI Driver for SPI on SCI	Threads	新しいスレッドをハイライトして、[New] > [Driver] > [Connectivity] > [SPI Driver on r_sci_spi] の順に選択します。
SCI Common driver	Threads	新しいスレッドをハイライトして、[New] > [Driver] > [Connectivity] > [SCI Common] の順に選択します。このドライバの [Properties] ウィンドウで、簡易 SPI モードを有効化します。
Interrupts	Threads	チャネルを選択して、選択したチャネルの SCIn RXI、TXI、TEI および ERI 割り込みを有効にします。

SPI フレームワークと RSPI モジュールを使用する場合は、以下のリソースを追加します。

リソース	ISDE タブ	選択
RSPI driver	Threads	新しいスレッドをハイライトして、[New] > [Driver] > [Connectivity] > [SPI Driver on r_rspi] の順に選択します。
Interrupts	Threads	チャネルを選択して、選択したチャネルの RSPIn SPRI、SPTI、SPII および SPEI 割り込みを有効にします。

SPI フレームワーク モジュールの設定

次のパラメータは、SPI フレームワーク操作に該当し、ボードの初期化の際に設定されます。

1) ピンの設定 – 周辺機器ピンとチップ選択 GPIO ピンの設定、出力有効化ピンとして使用するためのチップ選択設定。
2) 割り込みの設定 – SCI SPI 用の SCI RXI、TXI、TEI、および ERI 割り込みと、RSPI モジュール用の SPRI、SPTI、SPII、および SPEI 割り込みの有効化。

以下のパラメータは、SPI フレームワークの操作と SPI ハードウェアに固有であり、SPI フレームワーク インスタンスをオープンする前に設定されます。

1) SPI ドライバ パラメータ spi_cfg_t を設定します。
2) フレームワーク SPI バス パラメータ sf_spi_bus_t を設定します。
3) フレームワーク SPI デバイス パラメータ sf_spi_cfg_t を設定します。

4.1.15.3 SPI フレームワークのクロックの設定

SPI フレームワークには、個別のクロック設定は不要です。

SPI フレームワークのピンの設定

チップ選択信号を SPI デバイスとともに使用するには、チップ選択ピンとして割り当てられている I/O ポート ピンを GPIO ピン出力として設定し、ピンコンフィギュレータで有効にする必要があります : ピンの設定

SPI 周辺機器に対応する他のピンは、周辺機器ピンとして設定する必要があります （ピンの設定 を参照）。

これにより、選択したチップ選択 GPIO ピンが、関連ピンの pin_cfg フィールドで出力として設定され、その他のピンが SPI 周辺機器ピンとして設定されます。

SPI フレームワークの割り込みの設定

データ転送中に各種ハードウェア イベントを通知するために、SPI 転送割り込みを BSP で有効にする必要があります。割り込みは、SPI バスを設定するために選択されたドライバとチャネルに基づいて設定する必要があります。

SCI SPI 割り込み

Note：SCI SPI の割り込みを有効にするには、ドライバ モジュールをハイライトし、e2 studio ISDE のプロジェクト コンフィギュレータの [Threads] タブで、SCIn RXI、TXI、TEI、および ERI 割り込み （n は SCI チャネル番号）の優先度を設定します （割り込みの設定を参照）。

これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

RSPI 割り込み

Note：RSPI の割り込みを有効にするには、ドライバ モジュールをハイライトし、プロジェクト コンフィギュレータの [ICU] タブで、RSPIn SPRI、SPTI、SPII、および SPEI 割り込み （n は RSPI チャネル番号）のプライオリティを設定します （割り込みの設定を参照）。

これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

SPI フレームワーク モジュール

SPI 周辺機器用に SPI フレームワークを設定するには、以下のモジュールを設定する必要があります。

1) SPI ドライバ モジュール
2) フレームワーク SPI バス モジュール
3) フレームワーク SPI デバイス モジュール

フレームワーク SPI バス モジュールを設定したら、各種の SPI 周辺機器 （デバイス）をそのバスに関連付けることができます。バスに接続している SPI デバイスごとに、1 つの SPI ドライバ モジュールと 1 つのフレームワーク SPI デバイス モジュールを追加する必要があります。

以降のセクションでは、これらの各モジュールの設定方法について説明します。

HAL レベル SPI ドライバ パラメータの設定

e2 studio ISDE を使用して、g_spi ドライバ パラメータを設定します （ドライバのスレッドへの追加とドライバの設定を参照）

[SPI Driver on SCI_SPI] または [SPI Driver on RSPI] を追加すると、ssp_cfg_framework ディレクトリに、各構成パラメータに対して選択した設定を含む sf_spi_cfg.h ファイルが作成されます。

SPI フレームワークの設定 – sf_spi_cfg.h （共通パラメータ）

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define SF_SPI_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータ チェック用のコードを含めるかどうかを制御します。

表 : SPI ドライバ モジュールの設定

ISDE プロパティ	設定	設定値	説明
Channel	channel	Integer value between 0 and 9.	デバイスが接続されている SCI または RSPI チャネル番号。
Operating mode	operating_mode	Master, Slave	マスター デバイスまたはスレーブ デバイスとして設定します。 Note：SSP の現在のバージョンは、SPI マスター モードのみをサポートしています。
Clock Phase	clk_phase	Data sampling on even edge and data variation on odd edge. Data sampling on odd edge and data variation on even edge	奇数または偶数クロック エッジのいずれかのデータ サンプリングを選択します。
Clock Polarity	clk_polarity	High when idle, Low when idle	アイドル時のクロック レベル。
Mode Fault Error	mode_fault	Mode fault error Disable, Mode fault error Enable	モード障害エラー （マスターまたはスレーブ）のフラグを示します
Bit Order	bit_order	MSB First , LSB First	MSB ファーストと LSB ファーストのいずれかの送信順序を選択します
Bitrate	bitrate	Arbitrary integer value	送信または受信レート。ビット / 秒。
Callback	p_callback	NULL	フレームワーク設定用のコールバックを設定する必要はありません。コールバックは、フレームワーク コードにより内部的に処理されています。
Extend	p_extend	Optional Refer SPI driver interface user guide for extended configuration details.	SPI ハードウェアに依存する拡張設定。RSPI ドライバに対してのみ有効。

SPI フレームワーク共有バス モジュールのパラメータの設定

e2 studio ISDE を使用して、g_sf_spi ドライバ パラメータを設定します （ドライバのスレッドへの追加とドライバの設定を参照）

モジュール [sf_spi の SPI フレームワーク共有バス] を追加し、その中の各パラメータを設定します。

フレームワーク SPI バスのフレームワーク設定モジュールのパラメータ

ISDE プロパティ	設定	設定値	説明
Name	p_bus_name	Arbitrary symbol	バスを識別するための名前を設定します。このバス名は、フレームワーク設定で SPI 周辺機器をバスに関連付けるために使用されます。
Channel	channel	Integer value between 0 and 9	デバイスが接続されている SCI または RSPI チャネル番号。このバス設定で指定するチャネル番号は、ローレベル ドライバの open() 呼び出しで使用されます。
SPI Interface	p_lower_lvl_api	SCI SPI, RSPI	フレームワークで使用するローレベル インタフェースを選択します。

SPI フレームワーク デバイス パラメータの設定

SPI フレームワークは、SF_SPI_Open API で sf_spi_cfg_t 型のポインタを受け渡しすることにより設定されます。

SPI フレームワーク モジュールの設定

ISDE プロパティ	設定	設定値	説明
Bus Name	p_bus	Arbitrary symbol	SPI バス モジュール設定で設定されるバスに名前を付けます。これにより、この SPI 周辺機器が指定したバスに関連付けられます。
Chip Select	chip_select	Chip Select Port : 00 – 11, Chip Select Pin : 00 - 15	チップ選択ポートとチップ選択ピンです。チップ選択に使用される GPIO ポートとピンを選択します。

ISDE プロパティ	設定	設定値	説明
Chip Select Active Level	chip_select_level_active	High, Low	チップ選択信号の極性、アクティブ High または Low
Lower Level SPI Configuration Name	p_lower_lvl_cfg	Arbitrary symbol	設定する SPI ドライバの名前を付けます。ローレベル設定へのポインタは、この設定から取得されます。
Name	N/A	Arbitrary symbol	デバイスを識別するための名前を設定します。この名前は、デバイス インスタンスを作成する際に使用します。

4.1.15.4 SPI フレームワーク アプリケーションの作成

以下で説明するアプリケーションは、SCI SPI 周辺機器を使用するように SPI フレームワークのインスタンスを設定します。

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_spi_instance_t g_sf_spi_device =

{

    .p_ctrl = &g_sf_spi_ctrl,

    .p_cfg = &g_sf_spi_cfg,

    .p_api = &g_sf_spi_on_sf_spi

};
```

SCI SPI を使用する SPI フレームワーク アプリケーションは、以下のようにして作成します。

1) RTOS を使用する Synergy プロジェクトを作成します。
2) [Threads] タブで、前述のように以下の構造体を設定することで、モジュールを追加および設定します。
   - spi_cfg_t （SPI ドライバ モジュール）
   - sf_spi_bus_t （フレームワーク SPI バス モジュール）
   - sf_spi_cfg_t （フレームワーク SPI デバイス モジュール）

3) [Threads] タブで、[SCI Common] モジュールを追加し、簡易 SPI モードを有効化します。
4) [Problems] ウィンドウでエラーがないことを確認します。
5) 前述のように割り込み設定を設定します。
6) 前述のようにピンを設定します。
7) ISDE でプロジェクト コンテンツを生成します。これにより、フレームワーク関連のヘッダー ファイルと設定ファイルが自動的に作成されます。設定構造体の例を以下に示します。

```
sf_spi_bus_t g_sf_spi_bus =

{

    .p_bus_name = (uint8_t *) "g_sf_spi_bus",

    .channel = 7,

    .freq_hz_max = SF_SPI_FREQ_HZ_MAX,

    .freq_hz_min = 0,

    .p_lock_mutex = &sf_bus_mutex_g_sf_spi_bus,

    .p_sync_eventflag =&sf_bus_eventflag_g_sf_spi_bus,

    .pp_curr_ctrl = (sf_spi_ctrl_t **)
    &p_sf_curr_ctrl_g_sf_spi_bus,

    .p_lower_lvl_api = (spi_api_t *) &g_spi_on_sci,

    .device_count = 0,

};

const spi_cfg_t g_sci_spi_cfg =

{

    .channel = 7,

    .operating_mode = SPI_MODE_MASTER,

    .clk_phase = SPI_CLK_PHASE_EDGE_EVEN,

    .clk_polarity = SPI_CLK_POLARITY_HIGH,

    .mode_fault = SPI_MODE_FAULT_ERROR_DISABLE,

    .bit_order = SPI_BIT_ORDER_MSB_FIRST,
```

```
    .bitrate = 100000,

    .p_callback = NULL,

    .p_context = &g_sci_spi,

};

const sf_spi_cfg_t  g_sf_spi_device_cfg =

{

    .p_bus  = (sf_spi_bus_t  *) &g_sf_spi_bus,

    .chip_select = IOPORT_PORT_04_PIN_08,

    .chip_select_level_active = IOPORT_LEVEL_HIGH,

    .p_lower_lvl_cfg = &g_sci_spi_cfg,

};
```

8) プロジェクトをエラーなくビルドします。

9) SCI SPI として実装された SPI を使用して、SPI フレームワーク インスタンスをオープンします。SPI フレームワークは、次のインタフェースを通じてオープンします。

```
g_sf_spi_device.p_api  ->open(&g_sf_spi_device.p_cntl, &
g_sf_spi_device.p_cfg  )
```

ここで、g_sf_spi_device.p_api.p_cntl と g_sf_spi_device.p_cfg は、ISDE による SPI フレームワーク設定手順の後に自動生成されます。SPI フレームワークをオープンすると、SPI デバイスのハンドルが返されます。上の例では g_sf_spi_device.p_cntl です。このハンドルを使用して、各種の SPI 転送操作を実行します。以下に、このハンドルを使用して write 操作を行うコード例を示します。

```
length = 1;

src8[0] = ADXL345_DEVID;

error = g_sf_spi_device.p_api  ->write  (g_sf_spi_device.p_cntl,
src8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);
```

チップ選択信号を無効にせずに SPI の連続転送操作を行うには、ロックとロック解除のフレームワーク API を使用できます。ロックすると、一定期間 （ロックとロック解除の間）、デバイス用にバスを予約できるようになります。これにより、デバイスが複数の読み書き操作を中断なく完了することができます。ロックとロック解除の間は、フレームワーク API を呼び出してもチップ選択信号が切り替わりません。open、lock、write、read、および unlock の使用方法を示すコード例を以下にします。

```
/*Open the SPI Framework instance using config and control structures
generated as described above*/

error = g_sf_spi_device.p_api ->open(g_sf_spi_device.p_cntl, g_sf_spi_device.p_cfg);

if (SSP_SUCCESS != error)

return;

/*Lock the SPI bus for continuous data transfer (Required only for
continuous data transfer)*/

error = g_sf_spi_device.p_api ->lock (g_sf_spi_device.p_cntl);

if (SSP_SUCCESS != error)

return;

/*Transfer address to be read */

length = 1;

src8[0] = ADXL345_DEVID;

error = g_sf_spi_device.p_api ->write g_sf_spi_device.p_cntl, src8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREV
ER);

if (SSP_SUCCESS != error)

return;

/*Read value at the address transferred above*/

length = 1;

error = g_sf_spi_device.p_api ->read (g_sf_spi_device.p_cntl, dst8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);

if (SSP_SUCCESS != error)
```

```
return;

/*Unlock the bus, so that other peripherals can use the bus*/

error =
g_sf_spi_device.p_api ->unlock(g_sf_spi_device.p_cntl);

if (SSP_SUCCESS != error)

return;

/*Close SPI Framework instance*/

error= g_sf_spi_device.p_api ->close (g_sf_spi_device.p_cntl);

if (SSP_SUCCESS != error)

return;
```

4.1.15.5 同じバス上での複数の SPI デバイスの接続

複数の SPI 周辺機器を同じバス上で設定するには、バスに接続する各デバイスについて、以下のモジュールを追加して設定します。

- r_sci_spi 上の SPI ドライバ
- sf_spi の SPI フレームワーク デバイス

Note：各 SPI デバイスは、ISDE コンフィギュレータで一意の名前を使用して設定する必要があります

これにより、設定される周辺機器ごとに以下の構造体が作成されます。

- spi_cfg_t
- sf_spi_cfg_t

以下に、バス （上記セクションで作成済みのバス）に接続される 2 番目のデバイス用に、ISDE によって作成される一連の構造体の例を示します。

```
/*Structures with different name*/
const spi_cfg_t  g_sci_spi_2_cfg =
{
    .channel  = 8,
    .operating_mode = SPI_MODE_MASTER,
    .clk_phase = SPI_CLK_PHASE_EDGE_EVEN,
    .clk_polarity = SPI_CLK_POLARITY_LOW,
    .mode_fault = SPI_MODE_FAULT_ERROR_DISABLE,
    .bit_order = SPI_BIT_ORDER_MSB_FIRST,
    .bitrate = 100000,
    .p_callback = NULL,
    .p_context = &g_sci_spi_2,
};
/*Second peripheral linked to same bus**/*
const sf_spi_cfg_t  g_sf_spi_2_cfg =
{
    .p_bus  = (sf_spi_bus_t  *) &g_sci_bus_bus,
    .chip_select = IOPORT_PORT_04_PIN_08,
    .chip_select_level_active = IOPORT_LEVEL_LOW,
    .p_lower_lvl_cfg = &g_sci_spi_2_cfg,
};
```

2 番目のデバイス用の SPI フレームワーク インスタンスをオープンするには、新しい設定と制御構造体を使用します。

4.1.15.6 SPI フレームワークの制限事項

このモジュールには既知の制限事項はありません。最新のソフトウェア バージョンの詳細については、SSP のリリース ノートを参照してください。

4.1.15.7 SPI フレームワーク ファイル

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
SPI Framework Instance	synergy/ssp/inc/framework/instances/sf_spi.h
SPI Framework Driver	synergy/ssp/src/framework/sf_spi
SPI HAL Interface	synergy/ssp/inc/driver/api/r_spi_api.h
SPIonSCI Instance	synergy/ssp/inc/driver/instances/r_sci_spi.h
SPIonSCI Driver	synergy/ssp/src/driver/r_sci_spi
RSPI Instance	synergy/ssp/inc/driver/instances/r_rspi.h
RSPI Driver	synergy/ssp/src/driver/r_rspi

SPI フレームワークでサポートされるデバイス

このドライバは、S7G2 でテストされています。

SPI フレームワークは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

SPI フレームワークは、API への変更なしに、SPI モードの SCI または SPI 周辺機器を使用して、MCU をサポートするように設計されています。High または Low チップ選択アクティブ レベルを使用した任意の SPI デバイスをサポートできます。出力として設定されている任意の GPIO ラインをチップ選択信号として使用できます。

4.1.16 スレッド監視フレームワーク

スレッド監視フレームワークは、ThreadX RTOS のスレッド監視に適した汎用 API です。このフレームワークは sf_thread_monitor に実装され、MCU の WDT および IWDT 周辺機器を使用します。このセクションでは、e^2 studio ISDE を使用してスレッド監視フレームワークを設定する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Framework] > [Services] > [Thread Monitor Framework on sf_thread_monitor] を選択することで、スレッド

監視フレームワーク モジュールを追加および構成できます。詳細については、以下を参照してください : e2 studio ISDE によるスレッド監視フレームワークを使用するアプリケーションの作成

API リファレンスは、次のフレームワーク スレッド監視インタフェースの説明内に記載されています : スレッド監視フレームワークインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

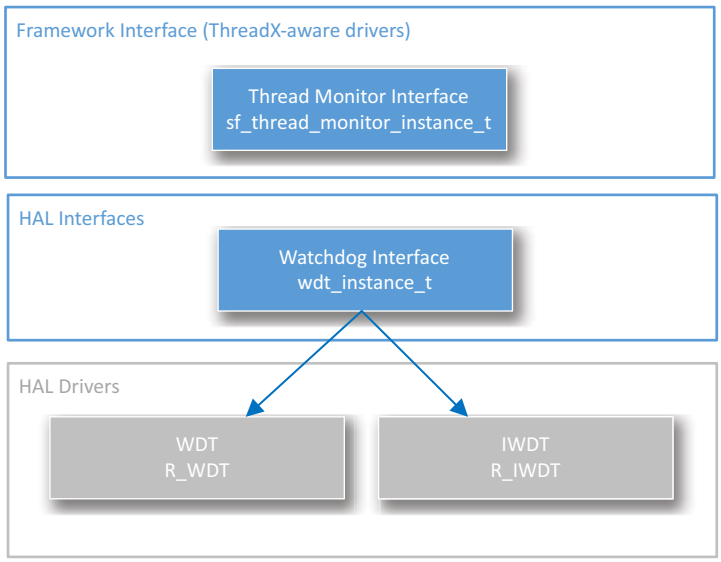

図 120: スレッド監視フレームワーク – ブロック図

4.1.16.1 フレームワーク スレッド監視の機能

スレッド監視フレームワーク インタフェースは、ウォッチドッグ タイマを利用して RTOS スレッドを監視します。スレッド監視は、監視対象のスレッドが想定外の動作を行った場合、強制的にウォッチドッグによるマイクロコントローラのリセットを実行します。

スレッド監視は、スレッドがスレッド監視にカウンタ変数と、このカウンタ変数の予測される最小値と最大値を登録する、という形で動作します。監視されているスレッドは、動作中にカウンタ変数をインクリメントします。ウォッチドッグ タイムアウト期間の半分の期間で、スレッド監視は登録されたスレッドのカウンタ変数をチェックします。最小値から最大値の範囲外となった場合、ウォッチドッグ タイマはマイクロコントローラをリセットできるようになります。すべてが予測範囲内の場合は、ウォッチドッグ タイマがリフレッシュされ、カウンタ変数はクリアされてゼロになります。

プロファイリング モードでは、登録されたスレッドのカウンタの最小値および最大値を決定できます。プロファイリング モード中は、ウォッチドッグ タイマが常にリフレッシュされるため、デバイスはリセットされません。

このフレームワーク モジュールは、WDT と IWDT HAL モジュールの両方をサポートしています。

下の図は、スレッド監視の動作に関するフローチャートを示しています。

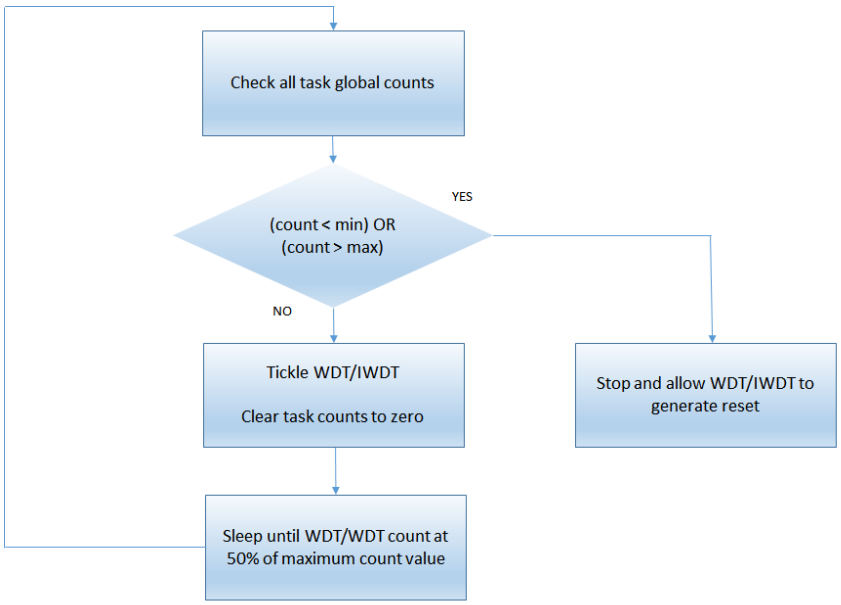

図 121: スレッド監視 - 操作

下の図は、WDT/IWDT がリセットされた場合を示しています。有効なリフレッシュ期間は、カウント期間の中心にあたる 50%、すなわち 50% のカウント値の両側 25% にわたるものであることに注意してください。

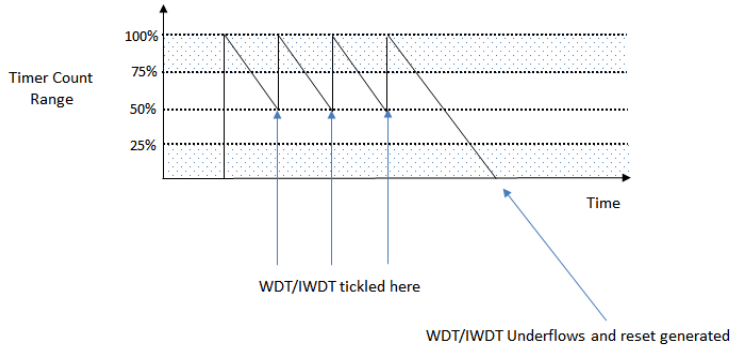

図 122: スレッド監視 - ウォッチドッグ ティクル

4.1.16.2 スレッド監視フレームワークの WDT モジュールの選択

スレッド監視は、API への変更なしに、WDT または IWDT 周辺機器および HAL ドライバを使用して、すべての MCU をサポートするように設計されています。

どのウォッチドッグ タイマを実装するかを決定する際には、次の点を考慮してください。

- IWDT には、安全性を向上させる独自のクロック ソースがあります。
- WDT は、重要なアプリケーションから開始が可能です。

4.1.16.3 e² studio ISDE によるスレッド監視フレームワークを使用するアプリケーションの作成

フレームワークは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （ドライバのスレッドへの追加とドライバの設定を参照）。

スレッド監視フレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Thread Monitor Framework	Threads	[Framework] > [Services] > [Thread Monitor Framework on sf_thread_monitor]

さらに、MCU で実行できるスレッド監視アプリケーションを作成するには、WDT または IWDT HAL ドライバが必要です。

リソース	ISDE タブ	選択
WDT Driver	Threads	[Driver] > [Monitoring] > [Watchdog Driver on r_wdt]
IWDT	Threads	[Driver] > [Monitoring] > [Watchdog Driver on r_iwdt]

スレッド監視フレームワークの WDT クロックの設定

e² studio ISDE で、[Clocks] タブを使用して WDT クロックを設定します （クロックの設定を参照）。

初期設定では、WDT は PCLKB 周波数に基づいて動作します。PCLKB 周波数を設定するには、実行時に e² studio ISDE クロックの設定 または CGC インタフェース のクロック コンフィギュレータを使用します。

PCLKB が 60 MHz で動作している状態での WDT の最大タイムアウト期間は約 2.24 秒です。

IWDT クロックが 15 kHz で動作している場合の最大タイムアウト期間は、35 秒をわずかに下回ります。

スレッド監視フレームワークのピンの設定

スレッド監視は、動作にあたってピンを必要としません。

スレッド監視フレームワークの割り込みの設定

スレッド監視は割り込みを使用しません。WDT/IWDT は、リセットを生成できるように設定する必要があります。

スレッド監視フレームワーク パラメータの設定

<WDT のスレッド監視フレームワーク > を追加すると、ssp_cfg_framework ディレクトリに、各構成パラメータに対して選択した設定を含む sf_thread_monitor_cfg.h ファイルが作成されます。

スレッド監視設定

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define THREAD_MONITOR_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE 0 1	API パラメータ チェック用のコードを含めるかどうかを制御します。
Maximum Number of Monitored Threads	#define THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS	Integer value	監視可能な最大スレッド数。

Attention：特定のデバイスで JLink を使用して動作させている場合は、このデバッグ ハードウェアを使用していると WDT/IWDT のカウントは実行されないため、デバッグ モード サポートが必要です。通常では、スレッド監視スレッドは WDT/IWDT カウンタと同期されますが、JLink を使用して動作している場合はこのような同期化が省略されます。

Note：スレッド監視スレッドに高いプライオリティ （ThreadX で低い数）を割り当てます。スレッド監視の動作中に遅延が発生すると、有効なリフレッシュ ウィンドウの範囲外でウォッチドッグがリフレッシュされることがあり、これはマイクロコントローラがリセットされる原因となります。スレッド監視スレッドは長時間にわたる動作はしないため、システムのパフォーマンスに影響を及ぼしません。

スレッド監視設定

ISDE プロパティ	設定	設定値	説明
Peripheral device	p_lower_lvl_wdt	wdt_instance_t	ウォッチドッグタイマ周辺機器のインスタンス構造体へのポインタ。
Profiling_Mode	profiling_mode_enabled	true/false	プロファイリングモード有効の要否。
Thread Monitor Thread Priority	priority	Integer value > 0	スレッド監視の内部スレッドのプライオリティ。

WDT/IWDT パラメータの設定

スレッド監視と併用している WDT または IWDT の設定パラメータを、スレッド監視と正しく連携できるように適切に設定する必要があります。これらの設定パラメータは、sf_thread_monitor_cfg_t::p_lower_level_wdt::p_cfg が指す構造体に含まれています。スレッド監視により、ローレベル ウォッチドッグはカウントの約 50% でリフレッシュされるため、このウィンドウ内でリフレッシュできるようにウォッチドッグを設定する必要があります。

スレッド監視 WDT/IWDT 設定

ISDE プロパティ	設定	設定値	説明
Start Mode	start_mode	wdt_start_mode_t	予測される WDT の開始モード。OFS0 の内容を反映している必要があります。
Start Watchdog After Configuration	autostart	true or false	ウォッチドッグがオープン時またはその後のリフレッシュ時に開始するかを制御します。
Timeout	timeout	wdt_timeout_t	リフレッシュされなかった場合に、ウォッチドッグがデバイスをリセットするまでのクロック サイクル数。
Clock Division Ratio	clock_division	wdt_clock_division_t	WDT/IWDT のクロック供給を行うためのクロック除算値。
Window Start Position	window_start	wdt_window_start_t	IWDR および WDR については、75% に設定する必要があります。

ISDE プロパティ	設定	設定値	説明
Window End Position	window_end	wdt_window_end_t	IWDR および WDR については、25% に設定する必要があります。
Reset Control	reset_control	wdt_reset_control_t	ウォッチドッグは、リセットを生成するように設定する必要があります。
Stop Control	stop_control	wdt_stop_control_t	低電力モード時に WDT/IWDT を停止するかどうか。
NMI Callback	p_callback	wdt_callback_args_t	WDT/IWDT が MCU をリセットするように設定されている場合は無効。

4.1.16.4 スレッド監視フレームワーク使用上の注意

スレッド監視期間

内部的に、スレッド監視はウォッチドッグ タイマのリセット期間の半分の速度で動作します。これにより、スレッド監視はウォッチドッグ カウント値の 50% で動作するため、有効なリフレッシュ ウィンドウの範囲内に十分収まります。スレッド監視は、ローレベル ウォッチドッグ ドライバにクエリを実行することにより、内部的にリセット期間を計算します。

4.1.16.5 スレッド監視アプリケーションの作成

以下で説明するアプリケーションは、WDT 周辺機器を使用するようにスレッド監視のインスタンスを設定します。WDT は、クロック除算値が 8192 に、タイムアウト期間が 16384 サイクルにそれぞれ設定されています。PCLKB が 60 MHz で動作している状態では、タイムアウトは約 2.2 秒となります。このタイムアウト期間は、下記のように計算されます。

このタイムアウト期間は、下記のように計算されます。

クロック分割比 = PCLKB/8192

タイムアウト周期 = 16384 サイクル

WDT クロック周波数 = 60MHz / 8192 = 7.324 kHz

サイクル時間 = 1 / 7.324 kHz = 136.53 マイクロ秒

タイムアウト = 136.53 マイクロ秒 x 16384 サイクル = 2.23 秒

監視される各スレッドは自身をスレッド監視に登録し、その際予測されるカウントの最小値と最大値も登録します。監視対象のスレッドがループを実行した場合は、毎回スレッド監視 API コールを実行してそのスレッドのカウントをインクリメントします。

プロファイリング モードでは、カウントの最小値と最大値は、このカウントに基づいて更新されます。非プロファイリング （通常）モードでは、カウント値と最小値および最大値との比較が実行されます。カウン

ト値がこのような想定された値の範囲から外れている場合、WDT はデバイスをリセットできるようになります。

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、ISDE により以下のスレッド監視インタフェースにインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const sf_thread_monitor_instance_t  g_sf_thread_monitor =

{

    .p_ctrl = &g_sf_thread_monitor,

    .p_cfg  = &g_sf_thread_monitor,

    .p_api  = &g_sf_thread_monitor

};
```

WDT を使用するスレッド監視アプリケーションは、以下のようにして作成します。

1) 前述のように、sf_thread_monitor_cfg_t と wdt_cfg_t の構造体を設定して、モジュールを構成します。ISDE により、src/ssp_gen/< thread_name > .c ファイルに構成構造体が自動的に作成されます。
2) ウォッチドッグが WDT として実装されているスレッド監視インスタンスを開きます。スレッド監視は、以下によってオープンされます。

```
g_sf_thread_monitor.p_api->open(g_thread_monitor.p_ctrl, g_thread_monitor.p_cfg)
```

g_thread_monitor.p_ctrl がスレッド監視の一部で、かつ WDT 設定手順後に ISDE により g_thread_monitor.p_cfg が自動生成された場合にオープンされます。スレッド監視がオープンすると、自動開始モードでなければ WDT が開始されます。下記の例は、ISDE により生成されたものです。

```
ssp_err_t  sf_thread_monitor_err_wdt;

sf_thread_monitor_err_wdt =
g_sf_thread_monitor.p_api->open(&g_ctrl_thread_monitor_wdt, &g_sf_thread_monitor_cfg_wdt);
```

3) 下の説明のようにスレッドを監視対象とします。
4) スレッド監視を、前述の説明のようにプロファイリング モードで動作するように設定します。
5) 監視対象のスレッドの最小値を高い数に、最大値をゼロに設定します。
6) スレッド プロファイリングが正確な値となるのに必要であれば、デバッガを使用してアプリケーションを実行します。
7) 実行を休止します。すべての監視対象スレッドの最小値と最大値をメモします。
8) スレッドの最小値と最大値を、プロファイリング中に取得した結果に基づいた値に設定します。
9) プロファイリングモードを無効にします。

10) アプリケーションを実行します。監視対象スレッドの監視が開始され、想定外の動作が発生するとデバイスがリセットされます。

4.1.16.6 スレッド監視 アプリケーションのスレッドの設定

スレッド監視により監視されるスレッドが監視モジュールと連携するには、実装されている必要があります。監視するスレッドがスレッド監視に登録されていると、予測された最小および最大カウント値が、これらの値を含む sf_thread_monitor_counter_min_max_t 型の構造体へのポインタを通じて渡されます。

下記の例は、これらの変数の作成および初期化の例です。

```
ssp_err_t  sf_thread_monitor_err_wdt;

sf_thread_monitor_counter_min_max_t  min_max_values;

min_max_values.minimum_count  = 2;

min_max_values.maximum_count  = 10;
```

スレッドのカウンタと最小値および最大値は、下図のように g_sf_thread_monitor.p_api->threadRegister() の順に呼び出してスレッド監視に登録する必要があります。

```
ssp_err_t  sf_thread_monitor_err_wdt;

sf_thread_monitor_err_wdt =
g_sf_thread_monitor.p_api->threadRegister(&g_ctrl_thread_monitor_wdt, &min_max_values);
```

監視対象のスレッドのループが完了するたびに、カウンタの値は下記のように g_sf_thread_monitor.p_api->countIncrement() の順に呼び出しが行われて更新されます。

```
sf_thread_monitor_err_wdt = g_sf_thread_monitor.p_api->countIncrement(&g_ctrl_thread_monitor_wdt);
```

今後このスレッドは、スレッド監視モジュールにより監視されます。

4.1.16.7 スレッド監視 フレームワークでサポートされるハードウェア実装

このドライバは、S7G2 でテストされています。

4.1.16.8 スレッド監視 フレームワークでサポートされるオペレーティング システム

スレッド監視はその性質上、RTOS による監視対象スレッドの操作と作成が必要です。スレッド モニターには、Express Logic 製の ThreadX が必要です。

4.1.16.9 スレッド監視 フレームワーク 制限

スレッド監視には、クローズ API コールがあります。ただし、WDT と IWDT を使用している場合は、これらを停止することはできません。このため、スレッド監視がクローズされた場合は、ウォッチドッグをリフ

レッシュするために他の対策を実行する必要があり、このような対策を実行しなければデバイスはリセットされます。

4.1.17 タッチ パネル フレームワーク

4.1.17.1 タッチ パネル フレームワーク

タッチ パネル フレームワークは、タッチ コントローラからデータを読み取り、メッセージング フレームワークのタッチ イベントにサブスクライブされたキューに対してタッチ メッセージを作成します。タッチ イベントには、位置データ （X および Y 座標）や、タッチ イベントの種類 （上、下、移動、保持、または無効）が含まれます。タッチ パネル フレームワークは、外部割り込みを使用する I2C ベースのタッチパネル実装をサポートしています。

e2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Stacks] ペインで [New] > [Framework] > [Input] > [Touch Panel Framework on sf_touch_panel_i2c] を選択することで、タッチ パネル フレームワーク モジュールを追加および構成できます。詳細については、以下を参照してください : e2 studio ISDE によるタッチ パネル フレームワークを使用するアプリケーションの作成。

API リファレンスは、次のタッチ パネル フレームワーク インタフェースの説明内に記載されています : タッチパネルフレームワークインタフェース

SSP レイヤー	タッチパネルフレームワークコンポーネント
API リファレンス : フレームワーク レイヤー	タッチパネルフレームワークインタフェース , I2C タッチパネルフレームワーク
API リファレンス : HAL インタフェース	I2C インタフェース , 外部 IRQ フレームワークインタフェース , 外部 IRQ インタフェース
API リファレンス : HAL レイヤー	-
Board Support Package	-

タッチ パネル フレームワークは、内部的にスレッドを作成し、タッチ コントローラをリードします。

4.1.17.2 e2 studio ISDE によるタッチ パネル フレームワークを使用するアプリケーションの作成

ドライバは、e2 studio ISDE の SSP に組み込まれています （e2 studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （ドライバのスレッドへの追加とドライバの設定を参照）。

タッチ パネル フレームワークを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
Touch Panel Framework	Threads	[Framework] > [Input] > [Touch Panel Framework on sf_touch_panel_i2c]
Messaging Framework	Threads	[Framework] > [Services] > [Messaging Framework on sf_message]
External IRQ Framework	Threads	[Framework] > [Input] > [External IRQ Framework on sf_external_irq]
External IRQ Driver	Threads	[Driver] > [Input] > [External IRQ Driver on r_icu]
I^2C Driver	Threads	[Driver] > [Connectivity] > [I2C Driver on r_iic or r_sci_i2c]

また、タッチ パネル フレームワークには、タッチ チップ ドライバに対する依存関係があります。

次に、フレームワークとドライバのリソースを設定します。

1) 外部 IRQ フレームワーク モジュールを設定します。イベントを「Semaphore Put」に設定します。
2) 外部 IRQ ドライバを設定します。ハードウェアとタッチ コントローラ チップの使用に従い、IRQ プライオリティ、チャネル、トリガ エッジとフィルタを設定します。DK-S7G2 （タッチ チップ SX8654）では、以下を選択します。

   - チャネル : 7
   - トリガ : Falling

   SK-S7G2 （タッチ チップ SX8654）では、以下を選択します。

   - チャネル : 9
   - トリガ : Falling

   PE-HMI1-S7G2 （タッチ チップ FT5x06）では、以下を選択します。

   - チャネル : 12
   - トリガ : Falling

   ボードの共通設定 :

   - デジタル フィルタ設定 : Any
   - 初期化後に割り込みを有効にする : True
   - コールバック : NULL

3) I^2C ドライバを設定します。DK-S7G2 では、r_sci_i2c を設定します。
   - チャネル : 7
   - 速度 : Standard
   - スレーブ アドレス : 0x48
   - アドレス モード : 7-bit

   SK-S7G2 では、r_riic を選択します。
   - チャネル : 2
   - 速度 : Standard
   - スレーブ アドレス : 0x48
   - アドレス モード : 7-bit

   PE-HMI1-S7G2 では、r_riic を選択します。
   - チャネル : 1
   - 速度 : 高速モード
   - スレーブ アドレス : 0x38
   - アドレス モード : 7-bit

   ボードの共通設定 :
   - コールバック : NULL

4) タッチ パネル フレームワーク モジュールを設定します。DK-S7G2 の場合 :
   - タッチ チップ : g_sf_touch_panel_i2c_chip_sx8654
   - 幅のサイズ ピクセル : 480
   - 高さのサイズ ピクセル : 272
   - リセット ピン : IOPORT_PORT_07_PIN_11

   SK-S7G2 の場合 :
   - タッチ チップ : g_sf_touch_panel_i2c_chip_sx8654
   - 幅のサイズ ピクセル : 240
   - 高さのサイズ ピクセル : 320
   - リセット ピン : IOPORT_PORT_06_PIN_09

   PE-HMI1-S7G2 の場合 :
   - タッチ チップ : g_sf_touch_panel_i2c_chip_ft5x06
   - 幅のサイズ ピクセル : 800

- 高さのサイズ ピクセル : 480
- リセット ピン : IOPORT_PORT_10_PIN_02

ボードの共通設定 :

- スレッドのプライオリティ : Any

SX8654 および FT5x06 のタッチ チップ ドライバは SSP のビルトイン ドライバです。

5) タッチ イベント メッセージ サブスクライバーを設定します。タッチ パネル フレームワークをプロジェクトに追加すると、e^2 studio ISDE はタッチ イベント クラスと新しいデータ イベントをプロジェクト コンフィギュレータの [Messaging] タブに自動的に生成します。プロジェクトでタッチ イベント メッセージ サブスクライバーを設定するには、次の手順に従ってください。

- e^2 studio ISDE Synergy コンフィギュレータの [Messaging] タブの [Event Class] ペインで、タッチ イベント クラスを選択します。
- [Messaging] タブの [Touch Subscribers] ペインでサブスクライバー スレッドのチェックボックスを選択します （スレッドがペインに表示されていない場合は、[Touch Subscribers] ペインの右上にある「新規」アイコンをクリックしてスレッドを追加します)。

4.1.17.3 カスタム タッチ パネル チップ ドライバ

カスタム タッチ チップ ドライバを作成するには、SSP の既存のドライバ コードと以下の指示、擬似コードを参照してタッチ パネル フレームワークに接続します。

- タッチ チップ ドライバ インスタンスを実装し、ドライバをアタッチするタッチ パネル フレームワークを有効化します。
- payloadGet 関数を実行し、タッチ パネル コントローラ デバイスから I^{2}C インタフェースを介してタッチ イベントとタッチ座標を取得します。
- reset 関数を実行し、関連付けられた GPIO ピント I^{2}C インタフェースを使用してタッチ パネル コントローラ デバイスをリセットします。

```
/* Template for Touch Chip Driver Instance */

const sf_touch_panel_i2c_chip_t
```

```
g_sf_touch_panel_i2c_chip_xxxxx =

{

    .payloadGet  = xxxxx_payload_get,

    .reset = xxxxx_reset

};

/* Pseudo code for payloadGet function */

ssp_err_t  xxxxx_payload_get (

    sf_touch_panel_ctrl_t  * const p_ctrl,

    sf_touch_panel_payload_t  * const p_payload)

{

    /* The following is an overview of the I2C touch function.

     * Refer to the existing touch chip driver code for details.

     */

    /* Get I2C interface and the control block from p_ctrl.

     * p_i2c_api = p_ctrl->p_lower_lvl_ctrl->p_lower_lvl_i2c->p_api;

     * p_i2c_ctrl = p_ctrl->p_lower_lvl_ctrl->p_lower_lvl_i2c->p_ctrl;

     */

    /* Get External IRQ interface and the control block from p_ctrl.

     * p_irq_api = p_ctrl->p_lower_lvl_ctrl->p_lower_lvl_irq->p_api;

     * p_irq_ctrl = p_ctrl->p_lower_lvl_ctrl->p_lower_lvl_irq->p_ctrl;

     */

    /* Call the wait API of the external IRQ interface to get interrupt from touch chip.

     * p_irq_api->wait(pirq_ctrl, wait_option);

     */
```

```
    /* Call the read API of the I2C Interface to read touch event and coordinate from touch chip.
     * p_i2c_api->read(pi2c_ctrl, …);
     */
    /* Set the obtained touch event and coordinate to p_payload.
     * p_payload->event_type = SF_TOUCH_PANEL_EVENT_XXXXX;
     * p_payload->x = <coordinate x>;
     * p_payload->y = <coordinate y>;
     */
    /* Store the touch event and coordinate in p_ctrl->last_payload,
     * which can be referred in the next touch event processing.
     * p_ctrl->last_payload.event_type = p_payload->event_type;
     * p_ctrl->last_payload.x = p_payload->x;
     * p_ctrl->last_payload.y = p_payload->y;
     */
    /* Return the error code. */
}
/* Pseudo code for reset function */
ssp_err_t xxxxx_chip_reset (sf_touch_panel_ctrl_t * const
p_ctrl)
{
    /* The following is an overview of the I2C touch reset function.
     * Refer the existing touch chip driver code for details.
     */
    /* Get I2C interface and the control block from p_ctrl.
     * p_i2c_api = p_ctrl->p_lower_lvl_ctrl->p_lower_lvl_i2c->p_api;
```

```
       * p_i2c_ctrl = p_ctrl->p_lower_lvl_ctrl->p_lower_lvl_i2c->p_ctrl;

       */

      /* Reset touch chip by setting GPIO reset pin low. */

      g_ioport_on_ioport.pinWrite(p_ctrl->p_lower_lvl_ctrl->pin, IOPORT_LEVEL_LOW);

    /* Delay a certain time. */

      /* Call reset API of I2C peripheral to issue reset to the touch chip.

       * p_i2c_api->reset(pi2c_ctrl);

       */

      /* Release touch chip from reset */

      g_ioport_on_ioport.pinWrite(p_ctrl->p_lower_lvl_ctrl->pin, IOPORT_LEVEL_HIGH);

/* Return the error code */

}
```

4.1.17.4 タッチ パネル フレームワーク アプリケーションの作成

タッチ パネル フレームワークの使用例 :

1) タッチ メッセージを保留にします。
2) タッチ メッセージを解析します。

次の例は、g_sf_touch_panel_queue という名前のキューにポストされたタッチ メッセージを受信する、g_sf_message という名前のメッセージング フレームワーク インスタンスを使用するアプリケーションの使用例です。

```
while(1)

{

sf_message_header_t * p_message = NULL;

g_sf_message.p_api->pend(g_sf_message.p_ctrl, &g_sf_touch_panel_queue, (sf_message_header_t **) &p_messa
ge, TX_WAIT_FOREVER);

switch (p_message->event_b.class)

{

case SF_MESSAGE_EVENT_CLASS_TOUCH:

{

    switch (p_message->event_b.code)

    {

    case SF_MESSAGE_EVENT_NEW_DATA:

        {

            sf_touch_panel_payload_t * p_touch_payload = (sf_touch_panel_payload_t *) p_message;

       switch (p_touch_payload->event_type)

            {

                case SF_TOUCH_PANEL_EVENT_DOWN:

                    break;

                case SF_TOUCH_PANEL_EVENT_UP:

                    break;

                case SF_TOUCH_PANEL_EVENT_HOLD:

                case SF_TOUCH_PANEL_EVENT_MOVE:

                    break;

                case SF_TOUCH_PANEL_EVENT_INVALID:

                    break;
```

```
                    default:

                        break;

                }

            }

        }

        default:

        break;

        }

}

err = g_sf_message.p_api->bufferRelease(g_sf_message.p_ctrl, (sf_message_header_t  *) p_message, SF_
MESSAGE_RELEASE_OPTION_ACK);

}
```

4.1.17.5 タッチ パネル フレームワークの制限事項

最新のソフトウェア バージョンの詳細については、SSP のリリース ノートを参照してください。このモジュールを使用するうえで、以下の制限事項が適用されます。

- ユーザー コールバックは、I^2C バス通信手順では使用できません。
- reset API は、以下の条件が満たされた場合にのみ使用できます : (1) stop API が呼び出されている。(2) タッチ イベントが発生し、タッチ パネル フレームワークがタッチ イベント メッセージをポストしている。

4.1.17.6 タッチ パネル フレームワークでサポートされるデバイス

このフレームワークは、以下のファミリでテストされています。

- S7G2

タッチ パネル フレームワークは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7

参考資料

4.2 HAL レイヤー

ADC ドライバ
クロック精度測定回路ドライバ
CAN ドライバ
CGC ドライバ
CRC ドライバ
暗号インタフェース
CTSU ドライバ
DAC ドライバ
ディスプレイ ドライバ
データ操作回路ドライバ
ELC ドライバ
外部 IRQ ドライバ
フラッシュ ドライバ
FMI ドライバ
I2C ドライバ
I2S ドライバ
入力キャプチャ ドライバ
I/O ポート ドライバ
JPEG デコード ドライバ
Key Matrix ドライバ
ローパワー モードドライバ
低電圧検出ドライバ
PDC ドライバ
QSPI ドライバ
RTC ドライバ
SD/MMC ドライバおよび SDIO ドライバ
セグメント LCD ドライバ
SPI ドライバ
タイマ ドライバ
転送ドライバ
UART ドライバ

ウォッチドッグドライバ

4.2.1 ADC ドライバ

ADC ドライバはアナログ / デジタル変換用の汎用 API です。ADC ドライバは、r_adc に実装され、MCU で利用可能な ADC12 および ADC14 をサポートします。このセクションでは、e^2 studio ISDE を使用して ADC ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Analog] > [ADC Driver on r_adc] を選択することで、ADC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による ADC ドライバを使用するアプリケーションの作成。

API リファレンスは、次の HAL ADC インタフェースの説明内に記載されています : ADC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

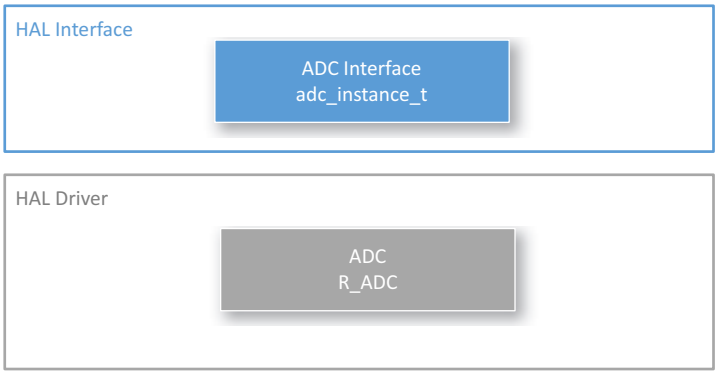

図 123: ADC ドライバ - ブロック図

4.2.1.1 ADC ドライバの機能

ADC ドライバは、ユーザーの設定に従って MCU 上の ADC を制御します。MCU 上の両方の ADC ユニットにアクセスでき、adc_mode_t でのシングル スキャン、連続スキャン、連続 スキャン、グループスキャン モードをサポートしています。スキャンが完了し、コールバックを使用できる場合 （割り込みが有効）、ADC ドライバはコールバックを呼び出します。その際、ユニットを示す adc_callback_args_t と、イベント adc_cb_event_t が引数として渡されます。

割り込みが有効になっていない場合、スキャンが完了したかどうかをポーリングするためのスキャン ステータスの確認が API でサポートされ、変換後の ADC の結果を読み取るための関数が用意されています。

グループモードの動作

ドライバはグループ モードの動作をサポートしています。このモードでは、グループ A とグループ B の 2 つのグループのいずれかにチャネルを割り当てることができます。スキャンを開始するためのトリガが各グ

ループに割り当てられます。グループ モードではハードウェア トリガのみを使用できるのに対し、通常モードでは、ソフトウェア トリガまたは外部トリガを使用できます。優先度構成パラメータ adc_group_a_t を使用すると、以下のことを指定できます。

- あるグループのトリガが、他のグループの進行中のスキャンを中断できるかどうか。
- スキャンが中断されたときに、スキャンを再開または再起動するか、単に現在のスキャンを中止して次のトリガを待つかどうか。

ADC ユニットは、通常割り込みとグループ B 割り込みの 2 つの割り込みをサポートしています。通常割り込みは、シングル スキャン モードまたは連続スキャン モードでスキャンが完了するときにトリガされます。グループ B 割り込みは、グループ B スキャンが完了するときにトリガされます。グループ モードでは、通常割り込みは、同じベクトルであってもグループ A 割り込みと呼ばれ、グループ A スキャンが完了するときに生成されます。

4.2.1.2 e² studio ISDE による ADC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

ADC インタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
ADC Driver	Threads	[Driver] > [Analog] > [ADC Driver on r_adc]
Interrupts	Threads	S12AD > S12ADn >S12ADn ADI および S12ADn GBADI （n = 0 または 1）
ADC Clock	Clocks	アプリケーション例については、ADC クロックの設定
ADC Pins	Pins	アプリケーション例については、ADC ピンの設定

アプリケーションの DAC 割り込みでデータ転送またはイベント トリガが必要な場合に、次のリソースはオプションです。

リソース	ISDE タブ	選択
Data Transfer Controller Driver	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dtc]
DMA Controller	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dmac]
Event Link Controller	Threads	ISDE には、デフォルトで ELC の 1 つのインスタンスが含まれます。

ADC クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して ADC クロックを設定します （クロックの設定を参照）。

ADC は PCLKC をそのクロック ソースとして使用します （ADCLK）。このクロックを設定する際の唯一の制限は、最大 ADC クロックよりも小さい値を設定することです。また、PCLKC クロックと PCLKB クロックの比率には、ユーザーズマニュアルで規定されている制限があります。

ADC の変換時間は、PCLKC の設定に依存します。

PCLKB と PCLKC の周波数を設定するには、e^2 studio ISDE のクロック コンフィギュレータを使用します（クロックの設定を参照）。

実行時にクロック周波数を変更するには、CGC インタフェースを使用します。

ADC ピンの設定

e^2 studio ISDE を使用して、[Pins] タブから ADC ピンを設定します （ピンの設定を参照）。

ADC を使用するには、アナログ入力を受信するチャネルのポート ピンを、ピン コンフィギュレータで入力ピンとして設定する必要があります。ピン コンフィギュレータは、pin_cfg フィールドで関連するピンの ADC ピン設定を適切に構成します。

ADC 割り込みの設定

e^2 studio ISDE を使用して、[ICU] タブから ADC 割り込みを設定します （割り込みの設定を参照）。

以下の状況では、BSP の選択したユニットに対して、ADC スキャン完了割り込みを有効にする必要があります。

- 通常のスキャンが完了したときに割り込みを取得する。
- グループ スキャンが完了したときに割り込みを取得する。

Note：割り込みを有効にするには、S12AD > S12ADn > S12AD0 ADI （通常 / グループ A 割り込み）、S12AD > S12ADn > S12AD0 GBADI （グループ B 割り込み）（n は ADC ユニット番号）の優先度を、e^2 studio ISDE のプロジェクト コンフィギュレータ の [ICU] タブで設定します （割り込みの設定を参照）。

これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_S12ADCn_ADI と BSP_IRQ_CFG_S12ADCn_GBADI に、選択した優先度が設定されます。

これらの割り込みが BSP で有効になっている場合、対応する割り込みサービス ルーチン （ISR）が ADC ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、そのコールバック関数を呼び出します。

ADC パラメータの設定

Use the e^2 studio ISDE を使用して、ADC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定。

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

Note : ADC ドライバには、open 呼び出しで使用するユニット レベルの設定 adc_cfg_t と、scanCfg で使用するチャネル設定 adc_channel_cfg_t の、2 つの設定が必要です。

ADC ユニットの設定

ISDE プロパティ	設定	設定値	説明
Unit	unit	0 or 1	使用する ADC ユニットを指定します。S7G2 には、0 と 1 の 2 つのユニットがあります。
Mode	mode	adc_mode_t	この ADC ユニットを使用するモードを指定します。
Resolution	resolution	adc_resolution_t	このユニットの変換解像度を指定します。
Alignment	alignment	adc_alignment_t	変換結果のアラインメントを指定します。
Add Average Count	add_average_count	adc_add_t	このユニット内のいずれかのチャネルに対して、加算または平均化を行う必要があるかどうかを指定します。実際のチャネルはチャネル マスク add_mask を使用して指定します。

ISDE プロパティ	設定	設定値	説明
Clearing	clearing	adc_clear_t	変換結果を読み取った後で、結果レジスタを自動的にクリアするかどうかを指定します。 Note：これが有効になっている場合、デバッガーを使用して結果レジスタをウォッチすると、常に 0 になります。
Trigger	trigger	adc_trigger_t	このユニットに使用するトリガ タイプを指定します。グループモードを mode で使用する場合、このフィールドは、グループ A トリガを設定するために使用されます。 Note：グループ モードで有効な唯一のオプションは ELC トリガです。
Trigger Group B	trigger_group_b	adc_trigger_t	グループ B トリガを指定します。このオプションは、mode でグループ モードが選択されている場合のみ有効です。
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、ADC スキャンが完了するたびに割り込みサービス ルーチン （ISR）から呼び出されます。 Attention：コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
Name	p_context	User-defined	-

ISDE プロパティ	設定	設定値	説明
-	p_extend	-	-

この表において、「プロパティ」は ISDE 内での [Properties] タブ名を表します。

ADC チャネルの設定

ISDE プロパティ	設定	設定値	説明
Scan Mask	scan_mask	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels.	通常の動作モードでは、このビットマスク フィールドを使用して、その ADC ユニットで有効なチャネルを指定します。たとえば、0x101 に設定されている場合、チャネル 0 と 2 が有効になります。グループ モードでは、このフィールドはグループ A に属するチャネルを指定するために使用されます。
Scan Mask Group B	scan_mask_group_b	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels.	このフィールドは、グループ モードでのみ有効です。グループ B に属するチャネルを指定するために使用されます。 Attention：同じチャネルをグループ A と B で指定しないようにしてください。
Priority Group A	priority_group_a	adc_group_a_t	進行中のグループ B のスキャンをグループ A のゴリがーで割り込むことができるかどうか、グループ A のトリガで中断するかどうか、または一時停止してグループ A のスキャンを許可し、グループ A のスキャンが完了したらすぐに再開するかを決定します。 Note：のフィールドは、グループ モードでのみ有効です。

ISDE プロパティ	設定	設定値	説明
Add Mask	add_mask	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels.	このフィールドは、add_average_count が有効になっている場合にのみ有効です。平均化または合計するチャネル結果を決定するために使用します。
Sample Hold Mask	sample_hold_mask	Use #define ADC_MASK_CHANNEL_0 , ADC_MASK_CHANNEL_1 , ADC_MASK_CHANNEL_2 which are defined in r_adc.h. Use (ADC_MASK_xxx|ADC_MASK_xxx) for multiple channels. Only channels 0, 1 and 2 are supported.	チャネル 0、1、2 のどれが、sample_hold_states で指定されたサンプル アンド ホールド状態の更新値を使用するかを決定します。このフィールドは、チャネル 0、1、および 2 のデフォルトのサンプル アンド ホールド カウント値を変更する場合にのみ設定する必要があります。
Sample Hold States	sample_hold_states	4-255	チャネル専用のサンプル アンド ホールド回路用の、更新されたサンプル アンド ホールド カウントを指定します。このフィールドは、sample_hold_mask が 0 でない場合にのみ有効です。チャネル 0、1、2 のみに専用のサンプル アンド ホールド回路があります。 Note：値をサンプリングするデフォルトの状態数（24）を変更するにはこのフィールドを使用します。各状態は、1/ADCLK 時間に等しくなります。

4.2.1.3 ADC ドライバ使用上の注意

サンプル状態カウントの設定

サンプル状態カウントの設定を変更するには、sampleStateCountSet を呼び出します。

この呼び出しを使用すると、必要に応じて各アナログ チャネルのサンプル状態カウントを個別に設定でき、すべてのチャネルに適用されます。

同様の設定は、sample_hold_states を通じて行うこともできますが、sample_hold_states はチャネル 0、1、および 2 の専用のサンプル アンド ホールド回路に適用される点が異なります。

サンプル状態カウント設定をデフォルト値から変更する必要があるのは、入力信号のインピーダンスが高すぎて十分なサンプリング時間が確保できないか、ADCLK が低すぎる場合に、サンプリング時間を長くする必要がある場合のみです。

指定したチャネルのサンプル状態カウントを変更するには、チャネル番号 adc_sample_state_reg_t を設定し、状態数 num_states を設定します。有効なサンプル状態カウントは 5 ～ 255 です。

複数のチャネルのサンプル状態カウントを変更するには、チャネルごとに引数を変えて sampleStateCountSet を繰り返し呼び出します。

ADC を使用したデータ転送のトリガ

ADC スキャンが完了したときにデータの転送をトリガするには、activation_source に ELC_EVENT_S12ADn_ADI および ELC_EVENT_S12ADn_GBADI （n は ADC チャネル番号）を設定してデータ転送を設定します。転送 API とともに使用する ADC 1 ユニット固有の情報を取得するには、転送 API で infoGet 関数呼び出しを使用します。

詳細については、転送インタフェース を参照してください。

ADC を使用した ELC イベントのトリガ

ADC ユニットは、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

ADC での温度センサーの使用

ADC HAL ドライバは、オンチップ温度センサーからのデータの読み取りをサポートしています。センサーから返された値は、以下の式を使用してユーザー アプリケーションによってセ氏またはカ氏に変換される必要があります。

T = （Vs – V1）/ スロープ + T1、ここで

- T: 測定温度 （°C）
- Vs: 温度センサーによる温度測定時の電圧出力 （V）
- T1: 1 地点で実験的に測定された温度 （°C）
- V1: 温度センサーによる T1 測定時の電圧出力 （V）
- T2: 別の地点で実験的に測定された温度 （°C）
- V2: 温度センサーによる T2 測定時の電圧出力 （V）
- スロープ : 温度センサーの温度勾配 （V/°C）、スロープ = （V2 – V1）/（T2 – T1）

スロープ値は、各デバイスのユーザーズマニュアルの電気特性、TSN 特性から取得できます。

スロープは、S7G2 では正、S3A7 では負です。

4.2.1.4 ADC アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const adc_instance_t  g_adc =

{

    .p_ctrl  = &g_adc_ctrl,

    .p_cfg = &g_adc_cfg,

    .p_api = &g_adc_on_adc

};
```

ADC HAL インタフェースを使用して ADC アプリケーションを作成するには、以下の手順を実行します。

1) 選択した ADC モジュールの ADCn 上の g_adc ADC ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが制御構造体 adc_ctrl_t に加えて、src/ssp_gen フォルダ内に自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

   I :ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) ADC インスタンスを開きます。ADC インタフェースを通じて ADC ドライバが呼び出されます。

```
g_adc.p_api->open(g_adc.p_ctrl, g_adc.p_cfg)
```

   g_adc.p_ctrl と g_adc.p_cfg は、ADC の設定ステップの後で自動生成されます。

4) 以下のインタフェースを呼び出してチャネルを設定します。

```
g_adc.p_api->scanCfg(g_adc.p_ctrl, g_adc.p_channel_cfg)
```

   g_adc_channel_cfg が自動生成されます。

5) 次のインタフェースを呼び出してスキャンを開始します。

```
g_adc.p_api->scanStart(g_adc.p_ctrl)
```

ハードウェア トリガを使用する場合、この呼び出しにより、ADC ユニットのハードウェア トリガによる起動が可能となります。ソフトウェア トリガを使用する場合、この呼び出しにより ADC スキャンが開始されます。

6) 次のインタフェースを呼び出して、チャネル 0 の変換結果を読み取ります。

```
g_adc.p_api->read(g_adc.p_ctrl, ADC_REG_CHANNEL_0, &my_result)
```

my_result は、ユーザーが作成した変数であり、読み取られた ADC の変換結果がここに格納されます。

7) 次のインタフェースを呼び出して、ADC スキャンを停止します。

```
g_adc.p_api->scanStop(g_adc.p_ctrl)
```

これにより、外部トリガまたはハードウェア トリガによる ADC のトリガが行われなくなります。また、進行中のソフトウェア トリガ スキャンがあれば強制的に停止されます。

8) 次のインタフェースを呼び出して、ADC インスタンスをクローズします。

```
g_adc.p_api->close(g_adc.p_ctrl))
```

シンプルな ADC アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src/ ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。

4.2.1.5 ADC の制限事項

ADC ドライバとその実装の制限事項については、SSP のリリース ノートを参照してください。

4.2.1.6 ADC ドライバでサポートされるデバイス

ADC インタフェースは、ADC12 ペリフェラル ブロックを使用して、S7G2 でテストされています。

ADC ドライバは、ADC14 周辺機器ブロックを使用して、S3A7 および S124 でテストされています。

4.2.1.7 ADC ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL ADC Interface API	synergy/ssp/inc/driver/api/r_adc_api.h

モジュール	ディレクトリ
ADC Instance	synergy/ssp/inc/driver/instances/r_adc.h
ADC Driver	synergy/ssp/src/driver/r_adc

4.2.2 クロック精度測定回路ドライバ

クロック精度測定回路ドライバ（CAC ドライバ）は r_cac に実装されます。このドライバには、参照信号入力に基づいてクロック周波数を監視できるクロック周波数測定回路に接続する API が含まれています。参照信号は、外部から供給されるクロック ソース、またはいずれかの内部クロック ソースの可能性があります。割り込みリクエストは、完了した測定、検出された周波数エラー、またはカウンタ オーバーフローによってオプションで生成されることがあります。外部から供給される参照クロックではデジタル フィルタが利用でき、除算値は内部提供の測定と参照クロックの両方で利用できます。参照クロックのエッジ検出オプションは、上昇、下降、またはその両方で構成できます。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [監視] > [r_cac 上のクロック精度測定回路ドライバ] を選択することで、クロック精度測定回路ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による CAC ドライバを使用するアプリケーションの作成

API リファレンスは、次の CAC インタフェースの説明内に記載されています : CAC インタフェース。

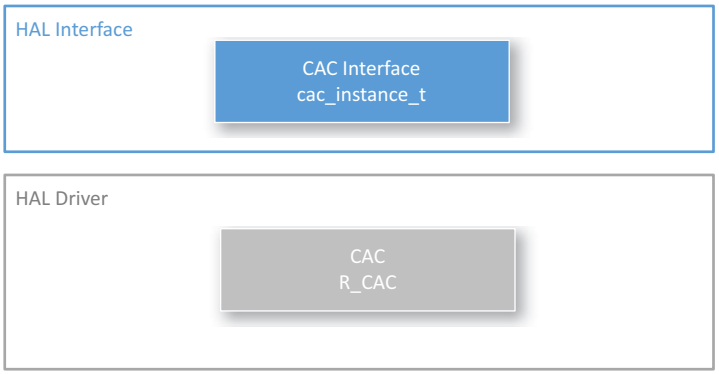

図 124: CAC - ブロック図

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

4.2.2.1 CAC 関連の用語

BSP 関連の用語	意味
Filename_xxxx.c	'xxxx' は MCU の種類を表します。たとえば、S7G2 を参照する場合は Filename_S7G2.c になります。
HLD	ハイレベル ドライバ
LLD	ローレベル ドライバ
Callback Function	この用語はイベント発生時に呼び出される関数を意味します。CAC の場合、CAC 測定が完了したとき、または周波数やオーバーフロー エラーに直面したときに呼び出されるユーザー定義関数です。コールバック構造体は、コールバックの原因となるイベント （測定完了、周波数エラー、カウンタ オーバーフロー）を提供するコールバックで渡されます。

4.2.2.2 CAC ドライバの機能

次のクロックの周波数を測定できます。

- メイン クロック発振器のクロック出力 （メイン クロック）
- サブクロック発振器のクロック出力 （サブクロック）
- 高速オンチップ発振器のクロック出力 （HOCO クロック）
- 低速オンチップ発振器のクロック出力 （LOCO クロック）
- 中速オンチップ発振器のクロック出力 （MOCO クロック）
- IWDT 専用オンチップ発振器のクロック出力 （IWDTCLK クロック）
- 周辺モジュール クロック （PCLKB）

測定クロックは参照クロックを使って監視されます。参照クロックは、CACREF 入力ピンで提供される外部クロックか、次のいずれかの内部クロックにすることができます。

- メイン クロック発振器のクロック出力 （メイン クロック）
- サブクロック発振器のクロック出力 （サブクロック）
- 高速オンチップ発振器のクロック出力 （HOCO クロック）
- 中速オンチップ発振器のクロック出力 （MOCO クロック）
- 低速オンチップ発振器のクロック出力 （LOCO クロック）
- IWDT 専用オンチップ発振器のクロック出力 （IWDTCLK クロック）

- 周辺モジュール クロック （PCLKB）

測定をリクエストすると、参照クロックで最初に検出された有効なエッジでカウントが始まり、次の有効なエッジで終わります。有効なエッジは、上昇、下降、またはその両方で設定できます。カウントは、クロックを 1、4、8、または 32 で分割可能な分周器回路を経由した後に、参照クロックの各サイクルでインクリメントされます。内部で提供される参照クロックも、クロックを 32、128、1024、または 8192 に分割可能な分周器回路を経由します。外部から提供される参照クロックは分周器回路を経由しませんが、設定されている場合はデジタル フィルタを経由することがあります。

たとえば、サブクロックが測定クロック （32 kHz）として指定されており、除算値が 1 に指定されている場合、カウンタは 32 kHz のレートでインクリメントします。1 kHz の参照クロックが提供されている場合、参照クロックの 1 サイクル後には、カウンタが 32 になることが予想されます。ここで CAC の上限および下限の設定が確認されます。CAC 測定のセットアップの一部には、測定の上限と下限の仕様が含まれます。測定が完了すると、CAC はカウンタの内容と測定の制限を比較します。カウンタ ≤ 上限およびカウンタ ≥ 下限の場合、測定がエラーなしで完了し、測定された周波数は定義された制限内で運用されていたことになります。カウンタがこれらの条件を満たさなかった場合、周波数エラーが示されます。

測定の完了は、API 呼び出しを行ってドライバをポーリングするか、次のいずれかの条件でトリガされるコールバック関数を設定して識別することができます。

- クロック測定完了 （MENDF）
- クロック周波数エラー （FERRF）
- クロック カウンタ オーバーフロー （OVFF）

4.2.2.3 e² studio ISDE による CAC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

CAC インタフェースを使用するアプリケーションでは、次のリソースが使用されます。

リソース	ISDE タブ	選択
CAC Driver	Threads	[Driver] > [監視] > [r_cac 上のクロック精度測定回路]
Interrupts	Threads	[CAC] > [CAC 周波数エラー]
Interrupts	Threads	[CAC] > [CAC 測定終了]
Interrupts	Threads	[CAC] > [CAC オーバーフロー]

CAC 割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから CAC 割り込みを設定します （割り込みの設定を参照)。あるいは、[CAC プロパティ] タブから割り込みを有効化できます。

いずれの割り込みも有効化しません。CAC ステータスを取得するために、読み取り関数を使って結果の完了をポーリングできます。ユーザーがコールバック関数を定義する場合、コールバックは有効にされている割り込みイベントのみで行われます。

コールバック関数には、コールバック イベント （CAC_EVENT_FREQUENCY_ERROR, CAC_EVENT_MEASUREMENT_COMPLETE or CAC_EVENT_COUNTER_OVERFLOW）のソースを示すイベント情報を含む構造体が受け渡されます。

割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブか、[CAC プロパティ] タブから、それぞれの CAC 割り込みの優先度を設定します。これにより、synergy_cfg/ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_CAC_FREQUENCY_ERROR、BSP_IRQ_CFG_CAC_MEASUREMENT_END、および BSP_IRQ_CFG_CAC_OVERFLOW に、選択した優先度が設定されます。

CAC 割り込みが BSP で有効になっている場合、対応する ISR が CAC ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

CAC パラメータの設定

CAC API はビルドおよびランタイム構成設定の両方を使用します。ビルド時設定はパラメータのチェックで利用できます。この機能を無効にすることにより、CAC API の全体的な ROM フットプリントを小さくできます。CAC ビルド時構成は、ユーザーの CAC 設定に基づいて ISDE によって生成される r_cac_cfg.h によって制御されます。

e2 studio ISDE を使用して、CAC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

CAC の設定

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define CAC_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE, Enabled or Disabled	API パラメータ チェック用のコードを含めるかどうかを制御します。
Name	g_cac	Instance name	このインスタンスを特定します
Continuous Measurement Operation	continuous_mode	Enabled or Disabled	有効な場合、完了後に測定が継続して再開されます

ISDE プロパティ	設定	設定値	説明
Measurement Complete Interrupt	mei_interrupt_enabled	Enabled or Disabled	有効にすると、測定の完了後に、ICU で CAC MEASUREMENT END 割り込みが有効な場合に、CAC ドライバは割り込みを生成できます。
OverFlow Interrupt	ovf_interrupt_enabled	Enabled or Disabled	有効にすると、CAC オーバーフローが生じ、ICU で CAC OVERFLOW 割り込みが有効な場合に、CAC ドライバは割り込みを生成できます。
Frequency Error Interrupt	ferr_interrupt_enabled	Enabled or Disabled	有効にすると、周波数エラーが生じ、ICU で CAC FREQUENCY ERROR 割り込みが有効な場合に、CAC ドライバは割り込みを生成できます。
Upper Limit Threshold	cac_upper_limit	0-0xFFFF	測定完了で可能な範囲の上限
Lower Limit Threshold	cac_lower_limit	0 – 0xFFFF, must be < Upper limit threshold	測定完了で可能な範囲の最低値
Reference clock source	cac_ref_clock .clock	cac_clock_source_t	参照クロック ソース
Reference clock divider	cac_ref_clock .divider	cac_ref_divider_t	参照クロック分周器
Reference clock edge detect	cac_ref_clock .edge	cac_ref_edge_t	参照クロック エッジ検出
Reference clock digital filter	cac_ref_clock .digfilter	cac_ref_digfilter_t	参照クロック デジタル フィルタ
Measurement clock source	cac_meas_clock .clock	cac_clock_source_t	測定クロック ソース
Measurement clock divider	cac_meas_clock .divider	cac_meas_divider_t	測定クロック分周器
Callback	p_callback	cac_callback_args_t	-

I：どのような構成エラーも捕捉できるように、パラメータ チェックを有効にした状態でアプリケーションを開発することをお勧めします。実稼働コードではパラメータ チェックを無効化して、コードスペースと実行時間を節約できます。

4.2.2.4 CAC アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const cac_instance_t  g_cac =

{ .p_ctrl  = &g_cac_ctrl, .p_cfg = &g_cac_cfg, .p_api = &g_cac_on_cac };
```

CAC を使用して CAC アプリケーションを作成するには、次の手順を実行します。

1) モジュールを上の説明のように設定します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます （以下の構造体が含まれます : cac_cfg_t および cac_ctrl_t、cac_instance_t）。

2) 設定時に CAC に付けた名前を使用して、CAC インスタンスをオープンします。CAC インタフェースを通じて適切なドライバが呼び出されます。g_cac という CAC インタフェースの場合、次のようになります。

```
g_cac.p_api->open(g_cac.p_ctrl, g_cac.p_cfg)
```

g_cac.p_ctrl と g_cac.p_cfg は、CAC の設定ステップの後で自動生成されます。

3) 次を使用して測定を開始します。

```
g_cac.p_api->startMeasurement(g_cac.p_ctrl)
```

4) 測定完了またはエラーを探すために読み取り機能を使ってポーリングを行います。

```
uint8_t cac_status;

uint16_t cac_counter;

ssp_err_t  err;

do

{

    err = g_cac.p_api->read(g_cac.p_ctrl, &cac_status, &cac_counter)

}

while ((cac_status == 0) && (CAC_SUCCESS == err));
```

5) 測定操作を停止します。

```
g_cac.p_api->stopMeasurement(g_cac.p_ctrl)
```

6) オーバフロー、測定終了、および周波数エラーの中断フラグをリセットします。

```
g_cac.p_api->reset(g_cac.p_ctrl)
```

4.2.2.5 CAC ドライバでサポートされるデバイス

CAC API は S3A7 および S7G2 の両方に基づいたボードで使用できます。

4.2.3 CAN ドライバ

コントロール エリア ネットワーク （CAN）ドライバは、CAN ネットワークのための汎用 API です。CAN ドライバは、MCU で利用可能な CAN 周辺機能をサポートします。このセクションでは、e^2 studio ISDE を使用して CAN ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [Connectivity] > [r_can 上の CAN ドライバ] を選択することで、CAN ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による CAN ドライバを使用するアプリケーションの作成。

API リファレンスは、次の HAL CAN インタフェースの説明内に記載されています : CAN インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

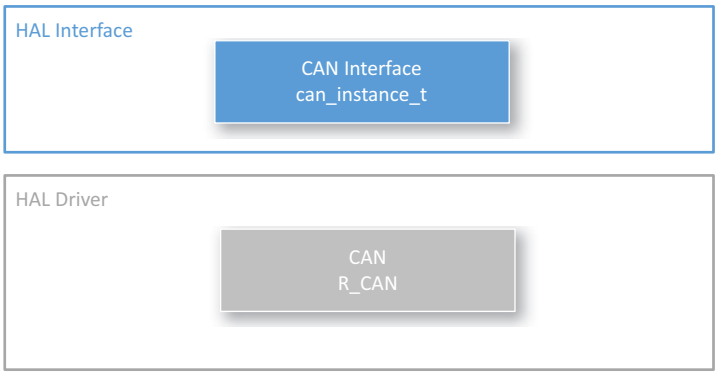

図 125: CAN ドライバ - ブロック図

4.2.3.1 CAN ドライバの機能

この CAN ドライバは、ユーザーの設定に従って MCU 上の CAN 周辺機器を制御します。API は open、close、read、write、control、information の各関数、can_api_t を提供します。このドライバでは、CAN の仕様で定義されているとおり、ビット タイミングの構成 can_bit_timing_cfg_t が可能です。これは、標準または拡張 ID フレーム can_id_mode_t で最大 32 の送信または受信メールボックスで構成できます。受信メールボックスは、データまたはリモート CAN フレーム can_frame_type_t をキャプチャするように構成できます。ユーザー コールバック関数は定義可能で、送信または受信、エラー割り込み受信の際にコールバックが呼び出されるようにドライバを設定できます。このコールバックは、チャネル、メールボックス、イベント can_event_t を示す引数 can_callback_args_t を提供します。

4.2.3.2 e^2 studio ISDE による CAN ドライバを使用するアプリケーションの作成

ドライバは、e^2 studio ISDE の SSP に組み込まれています （e^2 studio ISDE ユーザーガイドを参照）。

e^2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

CAN インタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
CAN Driver	Threads	[Threads] > [Driver] > [Connectivity] > [r_can 上の CAN ドライバ]
CAN Interrupts	Threads	CANn ERROR、CANn MAILBOX RX、CANn MAILBOX TX を有効化
CAN Clock	Clocks	アプリケーション例については、CAN クロックの設定
CAN Pins	Pins	アプリケーション例については、CAN ピンの設定

CAN クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して CAN クロックを設定します （クロックの設定を参照）。

CAN 周辺機器は、CANMCLK （メイン クロック発振器）または PCLKB （S7G2 または S3A7 のみ）をクロック ソース （fCAN、CAN システム クロック）として使用します。PCLKB をデフォルトの 60 MHz で、Synergy をデフォルト （S7G2 DK）の CAN 構成で使用すると、CAN ビット レートは 500 kbit となります。

PCLKB の周波数を設定するには、e^2 studio ISDE のクロック コンフィギュレータを使用します （クロックの設定を参照）。

実行時にクロック周波数を変更するには、CGC インタフェース を使用します。

CAN ソース クロックは CAN パラメータの設定 で設定する必要があります。

CAN ピンの設定

e2 studio ISDE を使用して、[Pins] タブから CAN ピンを設定します （ピンの設定を参照）。

[周辺機器] で [CAN] を選択し、続いて [CAN0] でチャネル 0、または [CAN1] （S7G2 と S3A7 のみ）でチャネル 1 を選択します。チャネルの動作モードを有効化され、PC ボードのレイアウトに一致する CRXn ピンと CTXn ピンが選択されている必要があります。ピン コンフィギュレータは、pin_cfg フィールドで関連するピンの CAN ピン設定を適切に構成します。

CAN パラメータの設定

e2 studio ISDE を使用して、CAN ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定。

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

この表において、「プロパティ」は ISDE 内での [Properties] タブ名を表します。

CAN チャネルの設定

ISDE プロパティ	設定	設定値	説明
Parameter Checking Enable	-	Default (BSP)	パラメータ エラー チェックを有効または無効にします。デフォルト （BSP）、有効または無効。
CANn ERROR	-	Disabled	エラー割り込みの優先度 0 ～ 15 を指定します （必須）。
CANn MAILBOX RX	-	Disabled	受信割り込みの優先度 0 ～ 15 を指定します （必須）。
CANn MAILBOX TX	-	Disabled	送信割り込みの優先度 0 ～ 15 を指定します （必須）。
Name	-	g_can	CAN モジュールのインスタンス名を指定します。
Channel	channel	0	使用する CAN チャネルを指定します。0 または 1 (S7G2 のみ)。

ISDE プロパティ	設定	設定値	説明
Baud Rate Prescaler	p_bit_timing :: baud_rate_prescaler	5	ボー レート プリスケーラー （0 ～ 1023）を指定します。CAN ビット レートの計算を参照してください。
Time Segment 1	p_bit_timing :: time_segment_1	15 Time Quanta	タイム セグメント 1 の値を指定します （4 ～ 16）。CAN ビット レートの計算を参照してください。
Time Segment 2	p_bit_timing :: time_segment_1	8 Time Quanta	タイム セグメント 2 の値を指定します （2 ～ 8）。CAN ビット レートの計算を参照してください。
Synchronization Jump Width	p_bit_timing :: synchronization_jump_width	3 Time Quanta	同期ジャンプ幅の値を指定します （1 ～ 4）。CAN ビット レートの計算を参照してください。
Clock Source	p_extend	CANMCLK	CAN クロック ソース、CANMCLK または PCLKB（S7G2 および S3A7 のみ）
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、割り込みが発生するたびに割り込みサービス ルーチン（ISR）から呼び出されます。 Attention : コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

ISDE プロパティ	設定	設定値	説明
Overwrite/Overrun mode	message_mode	Overwrite Mode	データが時間内に読み取られなかった場合に、受信メールボックスが上書きされるかオーバーランされるかを選択します。
Standard or Extended ID Mode	id_mode	Standard ID Mode	ドライバが CAN 標準 ID、拡張 ID のどちらを使用するかを選択します。
Number of Mailboxes	mailbox_count	32 Mailboxes	4、8、16、32 の中からメールボックス数を選択します。
Mailbox [0] ID … Mailbox [31] ID	p_mailbox[N] :: mailbox_id	0	メールボックス 0 の受信 ID を選択します （標準 ID を使用する場合は 0 ～ 0x7ff、拡張 ID を使用する場合は 0 ～ 0x1FFFFFFF）。メールボックスが送信型に設定されている場合、値は使用されません。
Mailbox [0] Type … Mailbox [31] Type	p_mailbox[N] :: mailbox_type	N = 0: Transmit Mailbox; N = 1 …31: Receive Mailbox	メールボックスが受信に使用されるか、送信に使用されるかを選択します。
Mailbox [0] Frame Type … Mailbox [31] Frane Type	p_mailbox[N] :: frame_type	N = 0: Remote Mailbox; N = 1 …31: Data Mailbox	メールボックスがデータフレームのキャプチャに使用されるか、リモート フレームのキャプチャに使用されるかを選択します (受信のみ)。
Mailbox 0-3 Group Mask	-	0x1FFF FFFF	メールボックス 0 ～ 3 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。
Mailbox 4 - 7 Group Mask	-	0x1FFF FFFF	メールボックス 4 ～ 7 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。

ISDE プロパティ	設定	設定値	説明
Mailbox 8 - 11 Group Mask	-	0x1FFF FFFF	メールボックス 8 ～ 11 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。
Mailbox 12 - 15 Group Mask	-	0x1FFF FFFF	メールボックス 12 ～ 15 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。
Mailbox 16 - 19 Group Mask	-	0x1FFF FFFF	メールボックス 16 ～ 19 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。
Mailbox 20 - 23 Group Mask	-	0x1FFF FFFF	メールボックス 20 ～ 23 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。
Mailbox 24 - 27 Group Mask	-	0x1FFF FFFF	メールボックス 24 ～ 27 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。
Mailbox 28 - 31 Group Mask	-	0x1FFF FFFF	メールボックス 28 ～ 31 のマスクを選択します。メールボックス グループ マスクの設定を参照してください。

4.2.3.3 CAN ビット レートの計算

タイム クォンタム （Tq）は CAN 通信クロック、fCANCLK の 1 ビットタイムです。これは CAN ビットタイムではなく、CAN 周辺機器の内部クロック周期です。周波数はボー レート プリスケーラーの値と CAN ソース クロック fCAN （CANMCLK または PCLKB）によって決定されます。

1 ビット タイムは複数のタイム クォンタム （Tqtot）に分割されます。1 タイム クォンタムは fCANCLK の周期と同じです。各ビットレート レジスタには、1 CAN ビット周期を構成する合計 TQ のうち、一定の数の Tq が付与されます。

デフォルトの ISDE ビットレット設定 （S7G2 DK テンプレート）は、fCAN が 60 MHz の場合、500 Kbps です （PCLKB を使用）。

ビットレート レジスタ設定の計算式 :

fCAN = （fPCLKB または fCANMCLK）

ボー レート プリスケーラーは CAN 周辺クロックをスケール ダウンします。

fCANCLK = fCAN/ ボー レート プリスケーラー = 60 MHz （デフォルト）/ 5 （デフォルト）= 12 MHz

1 タイム クォンタムは CAN クロックの 1 クロック周期です。

Tq =1/fCANCLK

Tqtot は 1 CAN ビット タイムあたりの CAN 周辺クロック サイクルの合計数で、「タイム セグメント」と「SS」（常に 1）の合計です。

Tqtot = TSEG1 + TSEG2 + SS （TSEG1 > TSEG2）= 15 + 8 + 1 = 24 （デフォルト）

この場合、ビットレートは次のようにして計算します。

ビットレート = fCANCLK/Tqtot = 12 MHz / 24 = 500 Kbps

重要な注意 :

1) 実行時にユーザー アプリケーションがメインのクロック発振器 （CANMCLK または XTAL）を起動する必要があります。すでに起動されていない場合 （たとえば MCU クロック ソースとして使用されていない場合）は、CGC インタフェースを使用します。
2) S7G2 および S3A7、S124 では、CAN モジュールに対し、クロック ソースがメインのクロック発振器 （CANMCLK）である場合、次の制限が満たされる必要があります : fPCLKB >= fCANCLK （fCAN / ボー レート プリスケーラー）。S7G2 および S3A7 では、クロック ソースが PCLKB の場合、周辺モジュール クロックのソースがすべての CAN モジュールにおいて PLL でなければなりません。
3) また、S3A7 では、CAN モジュールを使用する場合、PCLKA と PCLKB のクロック周波数比が 2:1 である必要があります。他の設定では操作は保証されません。
4) S124 では、CAN モジュールを使用する場合、ICLK と PCLKB のクロック周波数比が 2:1 である必要があります。他の設定では操作は保証されません。
5) SJW （同期ジャンプ幅）は多くの場合、バス管理者から供給されます。1 <= SJW <= 4 を選択します。

4.2.3.4 メールボックス グループ マスクの設定

4 つのメールボックスのグループごとに、合計 8 つのメールボックス グループ マスクがあります。これらのマスクでは、複数の ID を受信できるようにメールボックスを設定できます。マスクがすべて 1 に設定されている場合 （標準 ID では 0x7ff、拡張 ID では 0x1FFFFFFF）、そのグループ内のメールボックスは ID のビットをマスクせず、メッセージがキャプチャされる前にメールボックス ID のすべてのビットがそのメールボックス ID に一致しなければなりません。マスクのビットの中に 0 に設定されているものがある場合、これらのビットはメールボックス ID の同じビットと一致する必要はありません。たとえば、メールボックス ID 1 が 0x7ff に設定され、メールボックス 0 ～ 3 のグループ マスクが 0x7ff に設定されている場合、メールボックス 1 は ID が 0x7ff であるメッセージのみをキャプチャします。メールボックス 0 ～ 3 のグループ マスクが 0x7fe に設定されている場合、メールボックス 1 は ID が 0x7f のメッセージに加え、ID が 0x7fe のメッセージもキャプチャします。

4.2.3.5 CAN ドライバ使用上の注意

CAN アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const can_instance_t  g_can =

{

    .p_ctrl  = &g_can_ctrl,

    .p_cfg = &g_can_cfg,

    .p_api = &g_can_on_can

};
```

CAN HAL インタフェースを使用して CAN アプリケーションを作成するには、以下の手順を実行します。

1) 選択した r_can 上の CAN モジュール g_can CAN ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが制御構造体 can_ctrl_t に加えて、src/synergy_gen フォルダ内に自動生成されます。

2) アプリケーション コードを hal_entry.c に追加します。

   I :ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) CAN インスタンスをオープンします。他の関数を呼び出す前にチャネルを開く必要があります。CAN ドライバは CAN インタフェースを介して呼び出されます。

```
g_can.p_api->open(g_can.p_ctrl, g_can.p_cfg)
```

   g_can.p_ctrl と g_can.p_cfg は、CAN の設定ステップの後で自動生成されます。open 関数が呼び出されると CAN バスが動作し、バスに不具合がない限り、CAN メッセージを送受信できるようになります。

4) write 関数は、以下のとおり、インタフェースを使用して CAN メッセージを送信するために使用されます。

```
ssp_err_t  err; // Return code value.

can_frame_t  frame; // Create a transmit frame.

frame.id  = id; // Set destination ID.

frame.data_length_code  = 8; // Set data length.

frame.data[0] = 0x01; // Fill the frame with up to 8 bytes.

frame.data[1] = 0x02;

frame.data[2] = 0x03;

frame.data[3] = 0x04;

frame.data[4] = 0x05;

frame.data[5] = 0x06;

frame.data[6] = 0x07;

frame.data[7] = 0x08;

frame.type  = CAN_FRAME_TYPE_DATA; // Sending a data frame.

err = g_can.p_api->write(g_can.p_ctrl, mailbox_number, &frame)

// Check the return code for success, might need to retry.
```

ユーザー コールバックは、フレーム送信後に *CAN_EVENT_TX_COMPLETE* イベントで呼び出されます。

```
/* Simple user created function, not included or generated by the ISDE.*/

void can_callback (can_callback_args_t * p_args)

{

  switch (p_args->event)

  {

    case CAN_EVENT_RX_COMPLETE: // Receive complete event.

      rx_flag = true; // Could add code here to read data.

      break;

    case CAN_EVENT_TX_COMPLETE: // Transmit complete event.

      tx_flag = true; // Could add code here to transmit more data.

      break;

    case CAN_EVENT_ERR_BUS_OFF: // Bus error event.

      bus_error_flag = true; // Should check for bus error at some point.

      break;

    case CAN_EVENT_MAILBOX_OVERWRITE_OVERRUN:

      overrun_flag = true;

      break;

    default:

      break;

  }

}
```

5) read 関数は、受信メールボックスからのデータを受信するために使用されます。

```
err = g_can.p_api->read (g_can.p_ctrl, mailbox, &frame);
```

この関数は、ユーザー コールバック関数内から呼び出すことができます。リターン コードを確認する必要があります。

6) control 関数は、ループバック テストの実行など、診断の目的で使用されます。

```
can_mode_t  mode;

mode = *CAN_MODE_LOOPBACK_EXTERNAL*;

err = g_can.p_api->control(g_can.p_ctrl, *CAN_COMMAND_MODE_SWITCH*, &mode);
```

7) 以下を呼び出して、CAN インスタンスを閉じます。

```
g_can.p_api->close(g_can.p_ctrl));
```

シンプルな CAN アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src/ ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。

4.2.3.6 CAN の制限事項

CAN ドライバとその実装の制限事項については、SSP のリリース ノートを参照してください。

4.2.3.7 CAN ドライバでサポートされるデバイス

CAN インタフェースは S7G2 と S3A7 でテストされています。

4.2.3.8 CAN ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL CAN Interface API	synergy/ssp/inc/driver/api/r_can_api.h
CAN Instance	synergy/ssp/inc/driver/instances/r_can.h
CAN Driver	synergy/ssp/src/driver/r_can

4.2.4 CGC ドライバ

CGC ドライバ （クロック発生回路ドライバ）は r_cgc に実装され、MCU のクロック制御機能をプログラミングする他もの汎用 API が含まれています。このドライバは、MCU 上で使用可能なクロック発生回路周辺機器をサポートしています。このセクションでは、e^2 studio ISDE を使用して CGC ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e2 studio ISDE のプロジェクト コンフィギュレータで、CGC ドライバ モジュールはデフォルトですべてのプロジェクトに追加されています。CGC ドライバを構成するには、[Threads] タブの [Modules] ペインでこれを選択します。詳細については、以下を参照してください : e2 studio ISDE による CGC ドライバを使用するアプリケーションの作成

API リファレンスは、次の CGC インタフェースの説明内に記載されています : CGC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

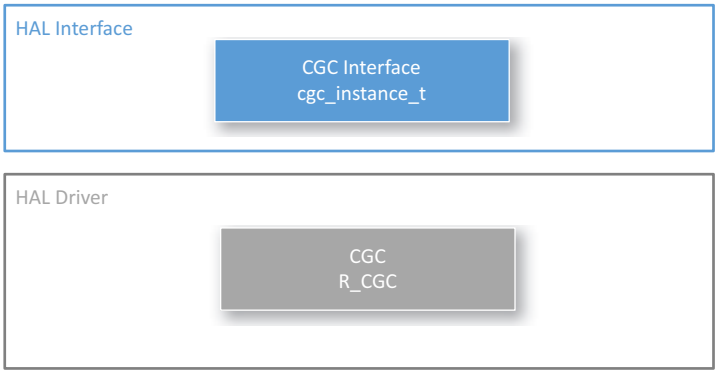

図 126: CGC モジュール - ブロック図

4.2.4.1 CGC モジュールの機能

CGC ドライバを使用して、MCU 上で以下のようにクロックを設定できます。

- 使用可能な任意のクロック （HOCO、MOCO、LOCO、メイン クロック、PLL、サブ オシレーター）をシステム クロック ソースとして設定します。
- 内部クロックを設定します （ICLK、PCLK など）。
- クロックのオンとオフを切り替えます。
- 出力クロックを設定します。
- 発振停止検出機能を設定します。

クロック発生回路周辺機器の特長は、次の発振器とクロック発振器です。

- メイン オシレーター入力 （最大 24 MHz）
- 32.768 kHz サブクロック オシレーター
- HOCO （高速オンチップ オシレーター）。最大 64 MHz で動作 （デバイスのバージョンに依存）
- MOCO （中速オンチップ オシレーター）。8 MHz で動作
- LOCO （低速オンチップ オシレーター）、32.768 kHz で動作

- PLL 回路出力。24 MHz ～ 240 MHz の間で動作 （デバイスに依存）

MCU には、6 つの内部クロック ドメインがあります。各ドメインには独立した除数がありますが、システム クロック制御レジスタで選択したクロック入力に依存します。以下の除数があります。

- ICLK - コア クロック。CPU、DMAC、ROM、および RAM 用 （最大 32/48/240 MHz）
- PCLKA - 周辺機器クロック。EtherC、EDMAC、USB2.0 HS、QSPI、および SCIF を含むモジュール用 （最大 32/48/120MHz）
- PCLKB - 周辺機器クロック （最大 32/60 MHz）
- PCLKC - 周辺機器クロック。ADC 用 （最大 64/60 MHz）
- PCLKD - 周辺機器クロック。ADC 用 （最大 64/120 MHz）
- FCLK - フラッシュ メモリ用クロック ソース （最大 32/60 MHz）

また、一部の MCU では、制御可能な外部クロック出力がサポートされており、そのいくつかには独立した除数があります。以下の除数があります。

- CLKOUT – CLOCKOUT/BUZZER クロック （最大 32 MHz）（独立したクロック セレクターと除数）
- BCLK - 外部バス コントローラへの外部バス クロック （最大 16/120 MHz）
- SDCLK – SDRAM クロック （最大 120 MHz）
- UCLK – USB クロック （最大 120 MHz）（Synergy バージョン 2 用の独立した除数 / セレクター）
- LCDSRCCLK – LCD クロック （独立したクロック セレクター。ただし除数なし）

4.2.4.2 e² studio ISDE による CGC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

プロジェクト ** の構成中に、MCU のコア コンポーネントである CGC** の r_cgc CGC ドライバがすでに追加されています。

CGC モジュールの設定

CGC パラメータは、CGC の操作に固有であり、アプリケーションを作成するときにデフォルト値に構成されます。

CGC クロックの設定

CGC クロック周波数は、e² studio ISDE で設定可能であり、コンフィギュレータの [Clocks] タブで選択します （クロックの設定を参照）。

無効な選択肢を選択すると、赤く表示されます。

CGC ピンの設定

e2 studio ISDE ピン コンフィギュレータを使用して、BCLK などの各種クロック出力の出力ピンを設定します （ピンの設定を参照）。

CGC パラメータの設定

e2 studio ISDE を使用して、CGC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

ISDE は、次の値とともにファイル r_cgc_cfg.h を生成します。

I :cgc_clock_cfg_t および cgc_system_clock_cfg_t 構造体を使用する CGC 関数は、ISDE の [Clocks] タブで行った設定に基づき、BSP によって内部的に呼び出されます （クロックの設定を参照）。

CGC のビルド時構成 （共通パラメータ）

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define CGC_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータ エラー チェックを有効または無効にします。
Main Oscillator Clock Source	#define CGC_CFG_MAIN_OSC_CLOCK_SOURCE	External Oscillator (1), Crystal or Resonator (0)	共振器または水晶を使用する場合は 0 を設定します。外部オシレーター入力を使用する場合は 1 を設定します。

ISDE プロパティ	設定	設定値	説明
Main Oscillator Wait Time	#define CGC_CFG_MAIN_OSC_WAIT	3 cycles, 35 cycles, 67 cycles, 131 cycles, 259 cycles, 547 cycles, 1059 cycles, 2147 cycles, 4291 cycles, 8163 cycles	これらいずれかの値を設定します。この遅延は CGC_CFG_MAIN_OSC_CLOCK_SOURCE が 0 に設定されてレゾネータ / クリスタルが使用されていることを示した場合にのみ構成されます。この遅延は、#define CGC_CFG_MAIN_OSC_CLOCK_SOURCE が 0 に設定されて発振子 / 水晶が使用されていることを示す場合にのみ構成されます。メイン クロック発振安定化時間は、オシレーターの製造元が推奨する安定化時間よりも長いかそれと等しい長さに設定します。
Oscillator Stop Detect	CGC_CFG_OSC_STOP_DET_USED	Enabled (Default), Disabled	有効にすることで、R_CGC_OscStopDetect 関数のコードが生成されます。この機能を使用するには、コールバック ポインタとともにこの関数を呼び出す必要があります。
Subclock Drive	#define CGC_CFG_SUBCLOCK_DRIVE	Middle (4.4 pf), Standard (12.5 pf) (Default)	この設定は、サブクロック発振器ドライブ キャパシタンスを水晶パラメータ #define CGC_CFG_SUBCLOCK_DRIVE に基づいて一致させるためのものです。

4.2.4.3 CGC スレッドの設定

このドライバは HAL ドライバであり、ThreadX RTOS に依存しません。

4.2.4.4 CGC アプリケーションの作成

CGC ドライバは、MCU のコア機能の 1 つであり、一般にリセット後に設定され、その後は変更されません。ただし、電力を節約したり、速度を高めるためなど、クロック設定を変更する関数を作成できます。

4.2.4.5 CGC ドライバ ファイル

プロジェクトを作成すると、ISDE はデフォルトで CGC API ファイルを追加します。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL CGC Interface API	synergy/ssp/inc/driver/api/r_cgc_api.h
CGC Instance	synergy/ssp/inc/driver/instances/r_cgc.h
CGC Driver	synergy/ssp/src/driver/r_cgc

4.2.4.6 CGC ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

4.2.5 CRC ドライバ

CRC ドライバは r_crc に実装され、メモリ内のデータ ブロック上またはシリアル ポートを介したデータ ストリーム上の 8、16、および 32 ビットの CRC 値を、さまざまな種類の業界標準の多項式を利用して計算するための汎用 API です。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [監視] > [r_crc 上の CRC ドライバ] を選択することで、CRC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による CRC ドライバを使用するアプリケーションの作成

API リファレンスは、次の HAL CRC インタフェースの説明内に記載されています : CRC インタフェース。

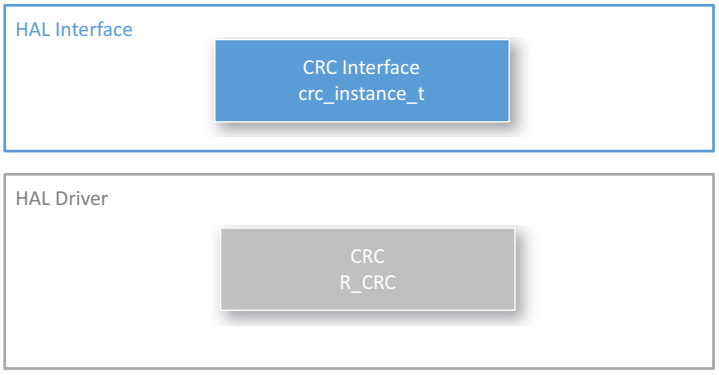

図 127: CRC ドライバ - ブロック図

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

4.2.5.1 CRC ドライバの機能

CRC ドライバは、以下の機能をサポートしています。

- メモリ内のデータ ブロックに対して CRC 計算を実行します。
- シリアル ポートを介して送信または受信されたデータ ストリームに対して CRC 計算を実行します。
- 計算を実行するシリアル ポートの数およびデータの方向を指定します。

4.2.5.2 e² studio ISDE による CRC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

CRC ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
CRC Driver	Threads	[Driver] > [監視] > [r_crc 上の CRC ドライバ]

CRC ドライバ用のクロックの設定

CRC ブロックにはクロック設定はありません。

CRC 割り込みの設定

CRC ブロックは割り込みを生成しません。

CRC のコールバック ISR 関数の定義

コールバックはありません。

CRC パラメータの設定

ISDE プロパティ	設定	設定値	説明
CRC Polynomial	polynomial	crc_polynomial_t	計算に使用する多項式を指定します
Bit Order	bit_order	crc_bit_order_t	計算のビット順序を指定します

4.2.5.3 CRC アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

CRC の例

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const crc_instance_t g_crc =

{

    .p_ctrl = &g_crc_ctrl,

    .p_cfg = &g_crc_cfg,

    .p_api = &g_crc_on_crc

};
```

CRC HAL インタフェースを使用して CRC アプリケーションを作成するには、以下の手順を実行します。

1) CRC の選択した CRC モジュール g_crc CRC ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが制御構造体 crc_ctrl_t に加えて、src/ssp_gen フォルダ内に自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

I :ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) CRC インスタンスをオープンします。CRC ドライバは、次の CRC インタフェースを通じて呼び出されます。

```
g_crc.p_api->open(g_crc.p_ctrl, g_crc.p_cfg)
```

g_crc.p_ctrl と g_crc.p_cfg は、ADC の設定ステップの後で自動生成されます。

4) 以下を呼び出して、CRC 値を計算します。

```
g_crc.p_api->calculate(g_crc.p_ctrl, p_input_buffer, num_bytes, crc_seed, p_crc_result)
```

この呼び出しでは次のパラメータが使用されます。

- input_buffer: データ値の配列へのポインタ。
- num_bytes: 配列内のバイト数 （要素数ではありません）。
- crc_seed: CRC 計算のシード値。
- crc_result: CRC 計算の計算値。

5) CRC インスタンスをクローズするには、以下を呼び出します :

```
g_crc.p_api->close(g_crc.p_ctrl)
```

シンプルな CRC アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src/ ディレクトリ内にあり、アプリケーションコードを格納するために ISDE で生成されます。

4.2.5.4 CRC の制限事項

CRC インタフェースのその他の制限事項については、SSP のリリース ノートを参照してください。

4.2.5.5 CRC ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL CRC Interface API	synergy/ssp/inc/driver/api/r_crc_api.h
CRC Instance	synergy/ssp/inc/driver/instances/r_crc.h
CRC Driver	synergy/ssp/src/driver/r_crc

4.2.5.6 CRC ドライバでサポートされるデバイス

このドライバは、CRC ペリフェラル ブロックを使用して、S7G2 および S3A7 でテストされています。

4.2.6 暗号インタフェース

SCE ドライバは r_sce に実装され、基本的な暗号操作を提供します。このセクションでは、e^2 studio ISDE を使用して SCE ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [グラフィックス] [暗号化]

を選択し、希望する暗号化サブモジュールを選択することで、SCE ドライバ モジュールを追加および構成できます。

R_SCE モジュールの暗号化 API 関数にアクセスするには、R_SCE として実装されている暗号化インタフェース API を使用します。暗号化モジュールのドライバ アーキテクチャを下図に示します。

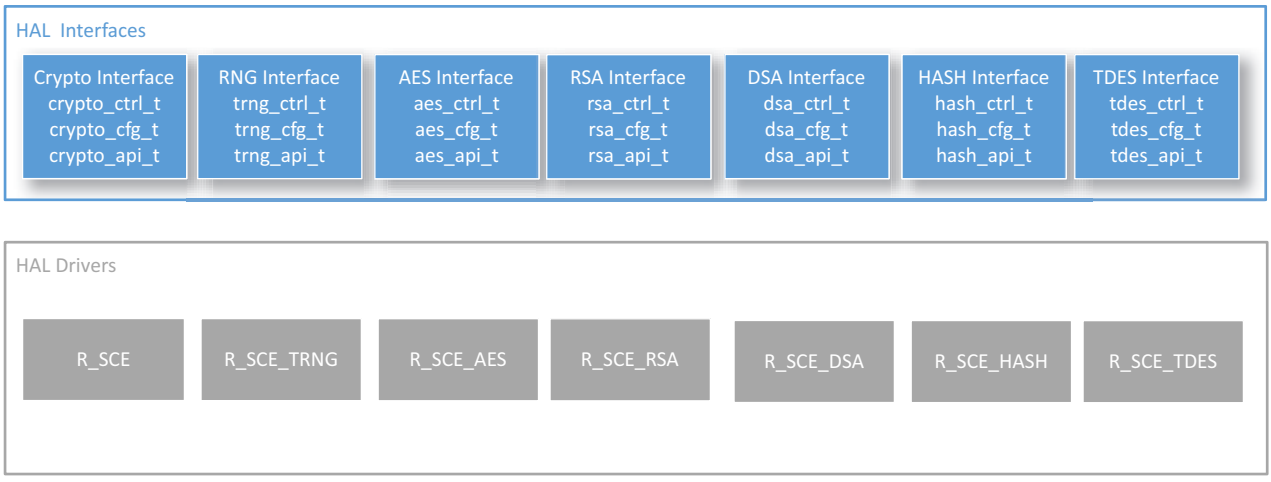

図 128: 暗号化 – ブロック図

4.2.6.1 HAL レイヤー

このドライバは、以下の暗号化プリミティブ関数を提供します。

- 乱数生成
- AES または Triple DES アルゴリズムを使用したデータ暗号化および復号化
- RSA または DSA アルゴリズムを使用した署名の生成と検証
- SHA1 または SHA224、SHA256 アルゴリズムを使用したメッセージ ダイジェストの計算

	S7G2	S124	注記：
TRNG - サポートされる関数	乱数の生成と読み取り	乱数の生成と読み取り	乱数生成
AES - サポートされる関数	暗号化、復号化	暗号化、復号化	AES 標準に基づく共通キー暗号化
AES - キー サイズ	128 ビット、192 ビット、256 ビット	128 ビット、256 ビット	
AES - チェーン モード	ECB、CBC、CTR、GCM、XTS	ECB、CBC、CTR、GCM	
RSA - サポートされる関数	署名生成、署名検証、公開キー暗号化、秘密キー復号化	N/A	秘密キーの操作に CRT キーと標準キーをサポート
RSA - サポートされるキー サイズ	1024 ビット、2048 ビット		
DSA - サポートされる関数	署名生成、署名検証	N/A	
DSA - サポートされるキー サイズ	(1024, 128) ビット、(2048, 224) ビット、(2048, 256) ビット	N/A	
HASH - サポートされるメソッド	SHA1、SHA224、SHA256	N/A	メッセージ ダイジェスト アルゴリズム

4.2.6.2 暗号化でサポートされるデバイス

このドライバは、S7G2 と S3A7 でテストされています。

4.2.6.3 サポートされている暗号化ハードウェア実装

HAL SCE ドライバ モジュールは、MCU の SCE 周辺機器にアクセスします。

4.2.6.4 暗号化アプリケーションを作成するための準備

ドライバは、SSP に組み込まれています。e^2 studio ISDE では、ボードを選択し、アプリケーションのプロジェクトを作成できます。プロジェクトの設定の際に、コンポーネント r_sce をアプリケーションに追加します。

4.2.6.5 e^2 studio ISDE を使用する暗号化アプリケーションの作成

暗号化ドライバは、e^2 studio ISDE に組み込まれています （e^2 studio ISDE ユーザーガイドを参照）。このインタフェースでは、次の暗号化アルゴリズムを設定できます。

- 乱数生成方式モジュール
- AES モジュール
- HASH モジュール
- RSA モジュール
- DSA モジュール

e^2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

SCE で暗号化ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
SCE Driver	Threads	[Driver] > [暗号化] > [r_sce 上の SCE ドライバ]
SCE TRNG Driver	Threads	[Driver] > [暗号化] > [r_sce_trng 上の SCE TRNG ドライバ]
SCE AES Driver	Threads	[Driver] > [暗号化] > [r_sce_aes 上の SCE AES ドライバ]
SCE RSA Driver	Threads	[Driver] > [暗号化] > [r_sce_rsa 上の SCE RSA ドライバ]

リソース	ISDE タブ	選択
SCE DSA Driver	Threads	[Driver] > [暗号化] > [r_sce_dsa 上の SCE DSA ドライバ]
SCE HASH Driver	Threads	[Driver] > [暗号化] > [r_sce_hash 上の SCE HASH ドライバ]

乱数生成 （TRNG）の設定

乱数生成では、基礎となるハードウェアが、前回生成した乱数とは異なる固有の 16 バイトの乱数を生成する最大試行回数を設定できます。最大試行回数に達した場合、読み取り API は呼び出し側に対しエラーを返します。それ以外の場合はサクセス コードが返され、呼び出し側が供給するデータ バッファに生成された乱数が転送されます。

TRNG の設定

ISDE プロパティ	設定	設定値	説明
max. attempts	nattempts	User defined, default is 2	新たに生成された乱数が前回生成された乱数とは異なる場合の、最大試行回数を設定します

AES モジュール インタフェース （AES）の設定

AES モジュールは、ユーザーが指定したキーの長さおよびチェーン モードを使用するよう設定できます。

AES の設定

ISDE プロパティ	設定	設定値	説明
Key length		User defined, default is 128 bits. Allowed values for S7G2 device: 128, 192, or 256 bits. Allowed values for S3A7 device: 128 or 256 bits	ドライバのこのインスタンスにより暗号化 / 復号化操作に用いられるキーの長さ
Chaining mode		User defined, default is CBC. Allowed values for S7G2 device: ECB, CBC, CTR, GCM, XTS. Allowed values for S3A7 device: ECB, CBC, CTR, GCM.	ドライバのこのインスタンスにより暗号化 / 復号化操作に用いられる、ブロック暗号チェーン モード

RSA モジュール インスタンス （RSA）の設定

RSA モジュールは、ユーザーが指定したキーの長さを使用するよう設定できます。

RSA の設定

ISDE プロパティ	設定	設定値	説明
Key length		User defined, default is 2048 bits. Allowed values for S7G2 device: 1024 or 2048 bits. Allowed values for S3A7 device: Not available.	ドライバのこのインスタンスにより署名 / 検証 / 暗号化 / 復号化操作に用いられるキーの長さ

DSA モジュール インスタンス （DSA）の設定

DSA モジュールは、ユーザーが指定したキーの長さを使用するよう設定できます。

DSA の設定

ISDE プロパティ	設定	設定値	説明
Key length		User defined, default is (2048, 256) bits. Allowed values for S7G2 device: (1024, 160), (2048, 224) or (2048, 256) bits Allowed values for S3A7 device: Not available.	ドライバのこのインスタンスにより署名 / 検証操作に用いられるキーの長さ。

HASH モジュール インスタンス （HASH）の設定

HASH の設定

ISDE プロパティ	設定	設定値	説明
Algorithm		User defined, default is SHA256. Allowed values for S7G2 device: SHA1, or SHA256. Allowed values for S3A7 device: Not available.	メッセージ データ上のメッセージ ダイジェスト / ハッシュの計算に用いられるアルゴリズム。

4.2.6.6 暗号化ドライバの例

r_sce_trng 上の TRNG ドライバを使った乱数生成例

Synergy 構成の [Threads] タブにある [Modules] ペインで、r_sce_trng 上の TRNG ドライバを追加します。ISDE で生成されたファイルを使ってアプリケーション コードを追加します。ISDE によって生成されたソース ファイルでプロジェクトを作成すると、インスタンス構造体が構成されます。

```
/* Instance structure to use this module. */

const crypto_instance_t  g_sce =

{ .p_ctrl  = &g_sce_ctrl, .p_cfg = &g_sce_cfg, .p_api = &g_sce_crypto_api };

const trng_instance_t  g_sce_trng =

{ .p_ctrl  = &g_sce_trng_ctrl, .p_cfg = &g_sce_trng_cfg, .p_api = &g_trng_on_sce  };
```

この例では、TRNG インタフェースで乱数バイトを設定する、および読み取る方法を示しています。

```
uint32_t rand_data\[32\];

/* Open the secure *crypto* engine driver */

g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

/* Open the TRNG driver */

g_sce_trng.p_api->open(g_sce_trng.p_ctrl, g_sce_trng.p_cfg);

/* read random data */

g_sce_trng.p_api->read(g_sce_trng.p_ctrl, rand_data, 16);
```

r_sce_aes 上の AES ドライバを使った AES 例

Synergy 構成の [Threads] タブにある [Modules] ペインで、r_sce_aes 上の AES ドライバを追加します。ISDE で生成されたファイルを使ってアプリケーション コードを追加します。ISDE によって生成されたソース ファイルでプロジェクトを作成すると、インスタンス構造体が構成されます。

```
/* Instance structure to use this module. */

const crypto_instance_t g_sce =

{ .p_ctrl = &g_sce_ctrl, .p_cfg = &g_sce_cfg, .p_api =
&g_sce_crypto_api };

const aes_instance_t g_sce_aes_0 =

{ .p_ctrl = &g_sce_aes_0_ctrl, .p_cfg = &g_sce_aes_0_cfg, .p_api =
&g_aes128cbc_on_sce };
```

この例では、128 ビット キー データおよび CBC モード用に構成された AES を使って、データを設定および暗号化する方法を示しています。

```
uint32_t msg_data[32]; /* original message data */

uint32_t enc_data[32]; /* encrypted data */

uint32_t dec_data[32]; /* decrypted data */

uint32_t key_data[4]; /* key material to encrypt or decrypt */

uint32_t ive_data[4]; /* initialization vector data for encryption */

uint32_t ivd_data[4]; /* initialization vector data for decryption */

/* Open the secure crypto engine driver */

g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

/* Open the AES driver */

g_sce_aes_0.p_api->open(g_sce_aes_0.p_ctrl, g_sce_aes_0.p_cfg);

/* encrypt data */

g_sce_aes_0.p_api->encrypt(g_sce_aes_0.p_ctrl, key_data, ive_data, 32, msg_data,
enc_data);

/* decrypt data */

g_sce_aes_0.p_api->decrypt(g_sce_aes_0.p_ctrl, key_data, ivd_data, 32, enc_data,
dec_data);
```

l :encrypt() 関数と decrypt() 関数は、データのパディングをサポートしません。これらの関数は、16 バイトの倍数であるデータ長で動作します。データ パディングは、ユーザーのアプリケーションにより処理され

る必要があります。AES GCM モードでは、16 バイトの倍数ではない認証データへのサポートが必要となる場合があります。これをサポートするため、AES GCM モードに対し zeroPaddingEncrypt() 関数と zeroPaddingDecrypt() 関数 API が提供されています。

r_sce_hash 上の HASH ドライバを使った HASH 例

Synergy 構成の [Threads] タブにある [Modules] ペインで、r_sce_hash 上の HASH ドライバを追加します。ISDE で生成されたファイルを使ってアプリケーション コードを追加します。ISDE によって生成されたソース ファイルでプロジェクトを作成すると、インスタンス構造体が構成されます。

```
/* Instance structure to use this module. */

const crypto_instance_t g_sce =

{ .p_ctrl = &g_sce_ctrl, .p_cfg = &g_sce_cfg, .p_api = &g_sce_crypto_api };

const hash_instance_t g_sce_hash_0 =

{ .p_ctrl = &g_sce_hash_0_ctrl, .p_cfg = &g_sce_hash_0_cfg, .p_api =
&g_sha256_hash_on_sce };
```

この例では、SHA256 アルゴリズムを使ったデータのハッシュの設定および計算方法を示しています。

```
uint32_t sha256InitialValue[8] =

{

0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,

0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19

};

/* Open the secure crypto engine driver */

g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

/* Open the HASH driver */

g_sce_hash_0.p_api->open(g_sce_hash_0.p_ctrl, g_sce_hash_0.p_cfg);

/* hash data */

memcpy(hashout_data, sha256InitialValue, sizeof(hashout_data));

g_sce_hash_0.p_api->hashUpdate(g_sce_hash_0.p_ctrl, message_data, 16, hashout_data);
```

r_sce_rsa 上の RSA ドライバを使った RSA 例

Synergy 構成の [Threads] タブにある [Modules] ペインで、r_sce_rsa 上の RSA ドライバを追加します。ISDE で生成されたファイルを使ってアプリケーション コードを追加します。ISDE によって生成されたソース ファイルでプロジェクトを作成すると、インスタンス構造体が構成されます。

```
/* Instance structure to use this module. */

const crypto_instance_t g_sce =

{ .p_ctrl = &g_sce_ctrl, .p_cfg = &g_sce_cfg, .p_api = &g_sce_crypto_api };

const rsa_instance_t g_sce_rsa_0 =

{ .p_ctrl = &g_sce_rsa_0_ctrl, .p_cfg = &g_sce_rsa_0_cfg, .p_api =
&g_rsa2048_on_sce };
```

この例では、2048 ビット キー データ用に構成された RSA アルゴリズムを使って、ハッシュ済みのデータを設定および署名する方法を示しています。

```
uint32_t key_data[128]; // 64-bytes of private key, and 64-bytes of modulus

uint32_t hashed_data[64];

uint32_t signed_data[64];

/* Open the secure crypto engine driver */

g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

/* Open the RSA driver */

g_sce_rsa_0.p_api->open(g_sce_rsa_0.p_ctrl, g_sce_rsa_0.p_cfg);

/* sign data from hashed_data using key_data and output to
signed_data buffer */

g_sce_rsa_0.p_api->sign(g_sce_rsa_0.p_ctrl, key_data, NULL, 64, hashed_data,
signed_data);
```

r_sce_dsa 上の DSA ドライバを使った DSA 例

Synergy 構成の [Threads] タブにある [Modules] ペインで、r_sce_dsa 上の DSA ドライバを追加します。ISDE で生成されたファイルを使ってアプリケーション コードを追加します。ISDE によって生成されたソース ファイルでプロジェクトを作成すると、インスタンス構造体が構成されます。

```
/* Instance structure to use this module. */

const crypto_instance_t  g_sce =

{ .p_ctrl  = &g_sce_ctrl, .p_cfg = &g_sce_cfg, .p_api = &g_sce_crypto_api };

const dsa_instance_t  g_sce_dsa_0 =

{ .p_ctrl  = &g_sce_dsa_0_ctrl, .p_cfg = &g_sce_dsa_0_cfg, .p_api =
&g_dsa2048_256_on_sce  };
```

この例では、2048 ビット キー データ用に構成された DSA アルゴリズムを使って、ハッシュ済みのデータを設定および署名する方法を示しています。

```
uint32_t dsa_domain_params[(256 + 2048 + 2048)/32];

uint32_t dsa_private_key_data[8];

uint32_t dsa_public_key_data[64];

uint32_t padded_hash[8];

uint32_t signed_data[16];

/* Open the secure *crypto* engine driver */

g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

/* Open the RSA driver */

g_sce_dsa_0.p_api->open(g_sce_dsa_0.p_ctrl, g_sce_dsa_0.p_cfg);

/* sign data from padded_hash using dsa_privatekey_data

and output to signed_data buffer */

g_sce_dsa_0.p_api->hashSign(g_sce_dsa_0.p_ctrl, dsa_private_key_data,
dsa_domain_params, 8, padded_hash, signed_data);

/* verify signed data using public key */

g_sce_dsa_0.p_api->hashVerify(g_sce_dsa_0.p_ctrl, dsa_public_key_data,
dsa_domain_params, 8, signed_data, padded_hash);
```

4.2.7 CTSU ドライバ

このセクションでは、e2 studio ISDE を使用して CTSU ドライバを設定およびプログラムする方法について説明します。

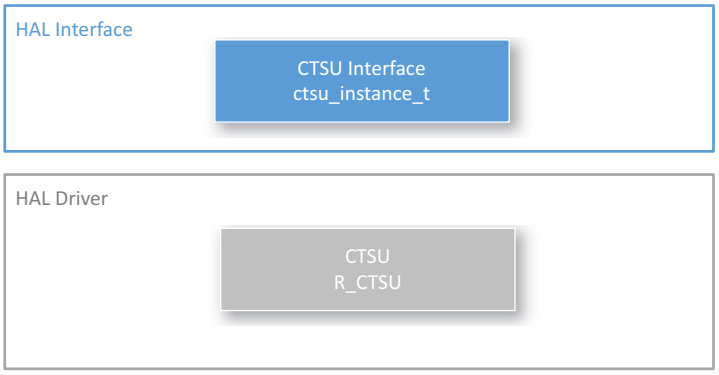

図 129: CTSU ドライバ - ブロック図

CTSU ドライバは、静電容量タッチ センシング アプリケーション用に r_ctsu に実装された汎用ドライバであり、MCU 上の CTSU 周辺機器をサポートしています。このドライバは、このモジュールで初期化または動作のために使用される必要な構造体を生成する、Workbench 6 ツールと組み合わせて使用するように設計されています。

e2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [入力] > [r_ctsu 上の CTSU ドライバ] を選択することで、CTSU ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e2 studio ISDE による CTSU ドライバを使用するアプリケーションの作成

静電容量タッチ センサーの調整には、Workbench 6 ツールを使用します。このツールは以下から別途ダウンロードできます。http://renesassynergy.com

API リファレンスは、次の CTSU インタフェースの説明内に記載されています : CTSU インタフェース

4.2.7.1 CTSU ドライバの機能

CTSU ドライバは、CTSU 周辺デバイスを初期化して任意の構成済み （かつ有効化済み）チャネルのキャパシタンスの変化を検出し、必要なフィルタリングを実行し、ボタンのホイールやスライダなど上位のウィジェット レイヤーで使用できるさまざまなデータを生成するために使用されます。こうしたレイヤーで要求されるさまざまな種類のデータをサポートするため、この実装では上位レベルのレイヤーで多様な種類の処理済みデータを必要性に基づいて読み取れるよう、Read() 関数が提供されています。またドライバには、各スキャンが完了したときおよび新しい処理データが使用可能になったときに呼び出されるコールバックも用意されています。このコールバックを使用して、上位レイヤーでデータを読み取ることができます。

CTSU ドライバを利用すると、[相互] や [セルフ キャパシタンス] を含む、サポートされているすべての動作モードで CTSU チャネルを設定できます。ドライバは設定済みチャネルのスキャン、DTC を使用したデータの移動、フィルタリングの実行、ドリフト補償、自動調整などを実行し、各繰り返し処理が完了する

たびにコールバック関数によってユーザーに通知を行います。またこのドライバにより、ユーザーは独自のフィルタリングや自動調整アルゴリズムを設定し、それをプロセスに統合することも可能になります。ドライバでは一度に 1 つの構成しかサポートされませんが、アプリケーションの必要に応じて複数のチャネル設定でドライバを再開することができます。

4.2.7.2 e² studio ISDE による CTSU ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

CTSU ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
CTSU Driver	Threads	[Driver] > [入力] > [r_ctsu 上の CTSU ドライバ]
Interrupts	Threads	CTSU WRITE、CTSU READ、CTSU END
CTSU Clock	Clocks	アプリケーション例については、CTSU クロックの設定
CTSU Pins	Pins	アプリケーション例については、CTSU ピンの設定

CTSU クロックの設定

CTSU クロック速度を、静電容量タッチ調整ツール Workbench 6 で選択したクロック速度と同じ値に設定します。

CTSU ピンの設定

ピンをタッチ センサー ピン TSxx として設定し、TSCAP 機能ピンを有効にします。

CTSU 割り込みの設定

[ICU] タブで 3 つの CTSU 割り込みをすべて同じプライオリティに設定して、これらの割り込みを有効にします。どのような値を設定しても構いません。

CTSU パラメータの設定

e² studio ISDE を使用して、CTSU ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

Workbench によって生成された CTSU 構成を含むファイルまたはフォルダが ${PROJECT_ROOT}\src フォルダに存在し、そのファイル / フォルダがビルドに含まれていることを確認します。

SSP を使用して CTSU パラメータを設定するには、以下の手順を実行します。

1) r_ctsu_api.h で、構造体型 ctsu_cfg_t の詳細を参照します。
2) ctsu_cfg_t 型のローカル変数を作成し、以下のように設定します。

CTSU の共通 / ビルド時構成パラメータ

ISDE プロパティ	設定	設定値	説明
Parameter Checking	-	Enabled (Default), Disabled	パラメータ エラー チェックを有効または無効にします。
Offset Adjustment	-	Enabled (Default), Disabled	オフセット調整を有効または無効にするのに使用します。ボードの調整を行う場合以外は、有効にしておいてください。(レートは、センサー値調整のランタイム レートにより決定されます)
Drift Compensation	-	Enabled (Default), Disabled	ドリフト補正を有効または無効にするのに使用します。ボードの調整を行う場合以外は、有効にしておいてください。ドリフト補正により、ボードやサーフェスの老化あるいは温度や環境の変化に応じて、タッチの存在の有無を決定するベースライン値が経時的に変化するよう設定できます。ドリフト補正レートは、*定常状態のドリフト補正レート*および*スタートアップ ドリフト補正レート*、*チャネル解放補正レート*により決定されます。

ISDE プロパティ	設定	設定値	説明
Drift Compensation Methods	-	Alternate 1	ドリフト補正のアルゴリズムを提供します。現在サポートされているのは Alternate 1 のみです。その他のオプションは、今後のリリースでサポートされる可能性があります。ドリフト補正方法 Alternate 1 を使うと、定常状態、スタートアップ、チャネル解放の各イベントに対し、異なるドリフト補正レートを用いることができます。
Dynamic Memory Usage	-	Enabled, Disabled (Default)	チャネルが動的に有効あるいは無効であるアプリケーションのメモリの保存に使用されます。動的メモリの使用は現在サポートされていません。
Perform auto-tune and drift compensation only when all channels are untouched	-	Enabled (Default), Disabled	すべてのチャネルがタッチされていない場合にのみ、自動調整とドリフト補正を実行します。
DTC Usage	-	Enabled (Default), Disabled	CTSU を最小化する CPU オーバーヘッドからデータを出し入れするために DTC を使用することを可能にします。このオプションは現在無効にすることができません。
Maximum Active Channels	-	-	アプリケーションによって使用されるチャネルの最大数を指定します。[セルフキャパシタンス] モードでは、使用されるチャネル数を指します。[相互キャパシタンス] モードでは、Rx チャネルと Tx チャネルの乗算の結果です。たとえば、Rx チャネル 4 つと Tx チャネル 3 つの場合、最大アクティブ チャネル数は 12 です。

ISDE プロパティ	設定	設定値	説明
Steady state drift compensation rate	-	Defaults to 500	チャネルが定常状態にある場合のドリフト補正レート（N スキャンごとに適用）を決定します。（スキャンレートに依存します）
Startup drift compensation rate	-	Defaults to 5	ドライバの初期化実行時におけるドリフト補正レート（N スキャンごとに適用）を決定します。（定常状態ドリフト補正レートよりも低く設定する必要があります）
Channel release compensation rate	-	Defaults to 500	チャネルが押された状態から開放された状態に移行する際のドリフト補正レート（N スキャンごとに適用）を決定します。（定常状態ドリフト補正レート以下に設定する必要があります）
Default filter depth	-	Defaults to 14	ドライバが提供するソフトウェア センサー カウント フィルタで用いられます。このデフォルト フィルタは、比較的一般的な「漏れ積分回路」ソフトウェア フィルタを改良したもので、深さ 4 は約 16 のフィルタ定数と同等となります。入力が 16 以上の定数値である場合、フィルタ出力はフィルタの 16 サイクル / 繰り返し後に定数入力値の 68 ～ 69% に達します。
Runtime rate of tuning of sensor values	-	Defaults to 800	オフセット調整のレート。（ドリフト補正を使用する場合、この値は定常状態ドリフト補正レートの 2 倍に設定されます）

ISDE プロパティ	設定	設定値	説明
DTC Usage for CTSU	-	Defaults to true	今回のリリースでは、r_ctsu を使用するには、DTC が必要となります。今後のリリースでは、DTC なしでの r-ctsu データ転送がサポートされる可能性があります。

CTSU モジュール構成パラメータ

ISDE プロパティ	設定	設定値	説明
Transfer Instance for DTC Read	p_lower_lvl_transfer_read	-	データ読み取りのための CTSU ドライバを使用した転送インスタンスの名前
Transfer Instance for DTC Write	p_lower_lvl_transfer_write	-	構成データの書き込みのための CTSU ドライバを使用した転送インスタンスの名前
CTSU Hardware Configuration	p_ctsu_hw_cfg	-	Workbench によって生成された構成構造体の名前
Processing Functions	p_ctsu_functions	-	ユーザーがデフォルトの内部処理機能に置き換えて使用するあらゆる処理機能。使用しない場合は NULL に設定できます。
Callback	p_callback	-	スキャンが完了するたびに呼び出されるユーザーコールバック関数。新しい処理済みデータが使用可能になったときにも呼び出されます。使用しない場合は NULL に設定できます。
Processing Option	ctsu_soft_option	-	完了した前回のデータ後にスキャンが開始されるかどうかや、自動較正がスキャンの間に実行されるかどうかなどの指定に使用することはできません。通常はデフォルト設定を使用してください。

ISDE プロパティ	設定	設定値	説明
Closing Option	ctsu_close_option	-	クローズ時にレジスタ ステータスをリセットするか保持するか （電力 / メモリは多く消費しますが、短時間で再有効化できます）を指定するのに使用します。

4.2.7.3 CTSU アプリケーションの作成

1) Synergy プロジェクトを生成します。
2) Synergy 設定 を開きます。
3) [Pins] タブの [CTSU] セクションに移動し、タッチ センサー ピン TSxx として使用する GPIO ピンを有効化します。
4) TSCAP ピンも有効化されていることを確認します。
5) MCU のクロック速度が、外部の CapTouch 調整ツール Workbench 6 で選択されているものと一致することを確認します。
6) [Driver] > [転送] > [r_dtc 上の転送ドライバ] から DTC 転送モジュールを追加します。構成は不要です。
7) CTSU HAL ドライバを HAL モジュールへ、または [Driver] > [転送] > [r_ctsu 上の転送ドライバ] からスレッドへ追加します。
8) Workbench によって生成された CTSU 構成を含むファイルまたはフォルダが ${PROJECT_ROOT}\src フォルダに存在し、そのファイル / フォルダがビルドに含まれていることを確認します。
9) [ICU] タブで 3 つの CTSU 割り込みをすべて同じプライオリティに設定して、これらの割り込みを有効にします。どのような値を設定しても構いません。
10) r_ctsu 上の CTSU ドライバの [Properties] タブを使用して、任意の設定で使用されるチャネルの最大数 （相互モードの場合はチャネルの組み合わせ）を調整します。
11) ユーザー定義関数にコールバック フィールドを設定します。
12) DTC 転送名フィールドを DTC モジュールの名前に設定します。
13) [使用される CTSU 構成] フィールドを、Workbench によって生成された構成の名前に設定します。
14) [Generate Project Content] をクリックして、必要な構造体を生成します。
15) hal_data.h （または thread_name.c）を開いて、このドライバを使用するように生成された構造体を確認します。SSP モジュールの使用方法の概要については、SSP ユーザー マニュアルを参照してください。

16) Name.api->open を呼び出します。この際、適切に初期化された引数が ctsu_cfg_t 構造体インスタンスを通して提供されていることを確認してください。これはメインの while ループに入る前に行う必要があります。

17) メインの while ループから周期的に Name.api->scan を呼び出し続け、新しいスキャンを開始します。このタスクはタイマを使用して定期的に実行すると便利です。それにより、スキャン レートが決定されます。

18) 各スキャンの終了時には、ユーザー コールバックが呼び出されます。コールバック内で、コールバックのイベントを確認して Name.api->update を呼び出し、データの処理を実行します。

19) 処理が完了すると、ユーザー コールバックが再度、データが処理されたことを示す異なる引数で呼び出されます。

4.2.7.4 CTSU ドライバの制限事項

ドライバは [相互容量] モードおよび [セルフ キャパシタンス マルチスキャン] モードのみをサポートします。ドライバは、いくつかの使用するチャネルが動的に変更されるアプリケーションでの効率的なメモリ使用が可能な動的メモリ割り当てをサポートしません。SSP のリリースノートも参照してください。

4.2.7.5 CTSU ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
CTSU HAL Interface	synergy/ssp/inc/driver/spi/r_ctsu_api.h
CTSU HAL Instance	synergy/ssp/inc/driver/instances/r_ctsu.h
CTSU HAL Driver	synergy/ssp/src/driver/r_ctsu

4.2.7.6 CTSU ドライバでサポートされるデバイス

このドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S7G2
- S3A7
- S124

CTSU フレームワークは、API への変更なしに、CTSU 周辺デバイスを使用するすべての MCU をサポートするように設計されています。

4.2.8 DAC ドライバ

DAC ドライバは、デジタルからアナログへの変換に使用する汎用 API で、r_dac に実装されています。DAC ドライバは、MCU 上で利用可能なデュアルチャネル 12 ビット D/A コンバータ （DAC12）をサポートしています。このセクションでは、e^2 studio ISDE を使用して DAC ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [アナログ] > [r_dac 上の DAC ドライバ] を選択することで、DAC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による DAC ドライバを使用するアプリケーションの作成

API リファレンスは、次の DAC インタフェースの説明内に記載されています : DAC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

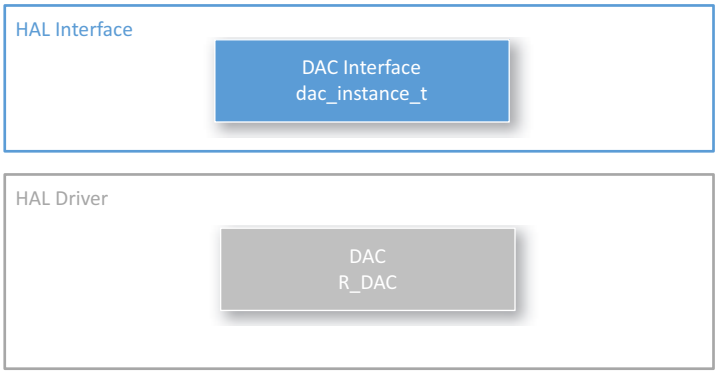

図 130: DAC ドライバ - ブロック図

4.2.8.1 DAC ドライバの機能

このドライバは、デュアルチャネル 12 ビット D/A コンバータ （DAC12）を設定し、正および負の基準電圧の間の 4096 段階の電圧レベルのいずれかを出力します。オプションは以下のとおりです。

- 16 ビット入力データ レジスタ用に、左詰めまたは右詰めの 12 ビット値形式を設定します。
- 出力の増幅器を有効または無効にします。
- 外部または内部基準電圧を選択します。
- アナログ / デジタルコンバータ （ADC）モジュールを使用して、同期干渉防止モードで動作します。

4.2.8.2 e^2 studio ISDE による DAC ドライバを使用するアプリケーションの作成

ドライバは、e^2 studio ISDE の SSP に組み込まれています （e^2 studio ISDE ユーザーガイドを参照）。

参考資料

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

DAC ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
DAC Driver	Threads	[Driver] > [アナログ] > [r_dac 上の DAC ドライバ]
DAC Clock	Clocks	アプリケーション例については、DAC クロックの設定
DAC Pins	Pins	アプリケーション例については、DAC ピンの設定

アプリケーションの DAC 割り込みでデータ転送またはイベント トリガが必要な場合に、次のリソースはオプションです。

リソース	ISDE タブ	選択
Data Transfer Controller Driver	Threads	[HAL/Common Modules] > [新規] > [Driver] > [転送] > [r_dtc 上の転送ドライバ]
DMA Controller	Threads	[HAL/Common Modules] > [新規] > [Driver] > [転送] > [r_dmac 上の転送ドライバ]
Event Link Controller	Threads	ISDE には、デフォルトで ELC の 1 つのインスタンスが含まれます。

DAC クロックの設定

DAC には個別のクロック設定は不要です。

DAC ピンの設定

e2 studio ISDE を使用して、[Pins] タブから DAC ピンを設定します （ピンの設定を参照）。

DAC を使用するには、アナログ入力を受信するチャネルのポート ピンを、ピン コンフィギュレータでアナログ ピンとして設定する必要があります。ピン コンフィギュレータは、pin_cfg フィールドで関連するピンの DAC ピン設定を適切に構成します。

DAC 割り込みの設定

DAC には個別の割り込み設定は不要です。

DAC ドライバ パラメータの設定

e^2 studio ISDE を使用して、DAC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

DAC 固有の唯一の初期化手順は以下のとおりです。

1) DAC モジュール ストップ ビットをゼロにクリアします。
2) DAC チャネル出力有効化に 1 を設定します。

DAC モジュール ストップ ビットは、open を呼び出した際に、ドライバのインスタンス カウンタがゼロの場合にゼロにクリアされます。ドライバのインスタンス カウンタはゼロに初期化され、チャネルの open が正常に戻るときにインクリメントされて、チャネルの close が呼び出されるとデクリメントされます。DAC モジュール ストップ ビットに 1 が設定されるのは、ドライバのインスタンス カウンタが、close 呼び出しで 0 にデクリメントされたときです。

DAC チャネル出力有効に 1 が設定されるのは、チャネルの write が、open が正常に呼び出された後で初めて呼び出されるときです。open 呼び出しでは、dac_ctrl_t 構造体要素に 0 が書き込まれます channel_started。channel_started をゼロにクリアして write を呼び出すと、そのチャネルの DAC 出力有効ビットに 1 が設定されます。チャネルの DAC 出力有効は、close および stop を呼び出したときにゼロにクリアされます。

SSP を使用して DAC パラメータを設定するには、以下の手順を実行します。

1) r_dac_api.h で、構造体型 dac_cfg_t の詳細を参照します。
2) dac_cfg_t 型のローカル変数を作成し、以下のように設定します。

表 : DAC の共通 / ビルド時設定パラメータ

ISDE プロパティ	設定	設定値	説明
Parameter Checking	#define DAC_CFG_PARAM_CHECKING_ENABLE	Enabled (Default), Disabled	パラメータ エラー チェックを有効または無効にします。

表 : DAC モジュール設定パラメータ

ISDE プロパティ	設定	設定値	説明
Channel	channel	0, 1	出力 DA0 の場合は 0、出力 DA1 の場合 1 を設定します。

ISDE プロパティ	設定	設定値	説明
Synchronize with ADC	ad_da_synchronized	Enabled (true), Disabled (false). Default: Disabled	アナログ / デジタルコンバータ （ADC）モジュールを使用した干渉防止同期の場合は true を設定します。アナログ モジュール間の電源干渉が問題にならないか、DAC モジュールによる非同期変換が望ましい場合は false を設定します。
Data Format	data_format	Right Justified (0) or Left Justified (1); Default: 1	12 ビット データ 値をビット 11 から 0 にロードする（右詰め）場合は 0 を設定します。12 ビット データ値をビット 15 から 4 にロードする （左詰め）場合は 1 を設定します。
Output Amplifier	output_amplifier_enabled	Enabled (true) or Disabled (false)	出力増幅器ハードウェア機能が望ましい場合は true を設定します。出力増幅器ハードウェア機能をバイパスする場合は false を設定します。

DAC 電圧基準の設定

DAC の制限事項を参照してください。1 つまたは複数の内部または外部電圧基準を選択します。DAC VREF 制御レジスタのデフォルトのリセット値 （0）は、電圧基準が選択されていない有効な動作モードに対応します。

DAC 出力の設定

DAC の制限事項を参照してください。ピン機能制御レジスタの ASEL ビットのデフォルトのリセット値（0）は、DA0 と DA1 を出力として有効にします。

ピン機能制御レジスタの ASEL フィールドに 1 を設定すると、DA0 と DA1 で共有されるピンが、アナログ / デジタルコンバータ （ADC）モジュール用のアナログ入力に切り替えられ、DA0 と DA1 の出力が無効になります。この状態では DAC 出力の操作が無効となり、推奨されません。

DAC イベント トリガーと DAC 有効化の設定

DAC の制限事項を参照してください。制御レジスタの DAE ビットのデフォルトのリセット値 （0）では、各チャネルを個別にトリガーできます。

制御レジスタの DAE ビットがクリアされるかゼロにリセットされると、ビット DAOE0 および DAOE1 が、それぞれのチャネルの変換を有効および無効にします。制御レジスタの DAE ビットに 1 が設定されると、両方の DAC チャネル変換が同時に有効になります。

4.2.8.3 DAC スレッドの設定

このドライバは HAL ドライバであり、ThreadX RTOS に依存しません。

4.2.8.4 DAC アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const dac_instance_t  g_dac =

{

    .p_ctrl  = &g_dac_ctrl,

    .p_cfg = &g_dac_cfg,

    .p_api = &g_dac_on_dac

};
```

DAC ドライバを使用してアプリケーションを作成するには、以下の手順を実行します。

1) 選択した DAC DACn 上の g_dac DAC ドライバに対し、ISDE の [Properties] タブを使用して DAC モジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと構成ファイルに加え、上記で説明した制御構造体 dac_ctrl_t が src/ssp_gen フォルダに自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

   l :ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) open 関数を呼び出して、DAC チャネル インスタンスを作成します。

```
g_dac.p_api->open(g_dac.p_ctrl, g_dac.p_cfg)
```

   . g_dac.p_ctrl および g_dac.p_cfg の各構造体は、ADC の設定ステップの後で自動生成されます。

4) DAC チャネルの write 関数を、構造体 dac_ctrl_t と、アナログ電圧への変換のためのデータ値を指定して呼び出します :

```
g_dac.p_api->write(g_dac_p_ctrl, val)
```

5) DAC チャネルの

```
g_dac.p_api->write(g_dac_p_ctrl, val)
```

関数を、アナログ電圧に変換する新しい値ごとに繰り返し呼び出します。

6) DAC チャネルを停止するため、関数

```
g_dac.p_api->close(g_dac_p_ctrl)
```

または

```
g_dac.p_api->stop(g_dac_p_ctrl)
```

関数への入力パラメータとして使用されます。

- g_dac.p_api->close(g_dac_p_ctrl) を使用すると、チャネル インスタンスが終了されるため、g_dac.p_api->open(g_dac.p_ctrl, g_dac.p_cfg) を呼び出して再開する必要があります。
- g_dac.p_api->stop(g_dac_p_ctrl) を使用すると、チャネル インスタンスはアクティブのままになるため、g_dac.p_api->start(g_dac_p_ctrl) 関数を呼び出して再開することができます。

7) DAC チャネル インスタンスを終了し、他の用途のためにチャネルを解放するため、

```
g_dac.p_api->close(g_dac_p_ctrl)
```

関数を呼び出します。

4.2.8.5 DAC の制限事項

DAC モジュールのピン設定は実装されていません。現在、DA0 と DA1 の出力は、ピン設定制御レジスタの ASEL フィールドのリセット値によって有効になります。DAC 出力の設定を参照してください。

DAC モジュールの電圧基準選択は実装されていません。現在、D/A VREF 制御レジスタ （DAVREFCR）のリセット値によって基準は選択されませんが、これは正しい状態です。DAC 電圧基準の設定を参照してください。

変換をトリガーする DAC 入力イベントの設定は、現在実装されていません。制御レジスタの DAE ビットのデフォルトのリセット値 （0）では、各チャネルを個別にトリガーできます。

DAC モジュール変換の同期のためのイベント信号入力は、現在実装されていません。DAC イベント トリガーと DAC 有効化の設定を参照してください。

4.2.8.6 DAC ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL DAC Interface API	synergy/ssp/inc/driver/api/r_dac_api.h

モジュール	ディレクトリ
DAC Instance	synergy/ssp/inc/driver/instances/r_dac.h
DAC Driver	synergy/ssp/src/driver/r_dac

4.2.8.7 DAC ドライバでサポートされるデバイス

HAL DAC ドライバ モジュールは、MCU の DAC 周辺機器にアクセスします。

このドライバは、S7G2 でテストされています。

DAC ドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S7G2
- S3A7

4.2.9 ディスプレイ ドライバ

ディスプレイ ドライバは、r_glcd に実装された LCD ディスプレイの制御に使用される汎用 API です。このドライバは、MCU 上で利用可能なグラフィック LCD コントローラ（GLCDC）の周辺機器をサポートしています。このセクションでは、e^2 studio ISDE を使用してディスプレイ ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [グラフィックス] > [r_glcd 上のディスプレイ ドライバ] を選択することで、ディスプレイ ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE によるディスプレイ ドライバを使用するアプリケーションの作成

API リファレンスは、次のディスプレイ インタフェースの説明内に記載されています : ディスプレイインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

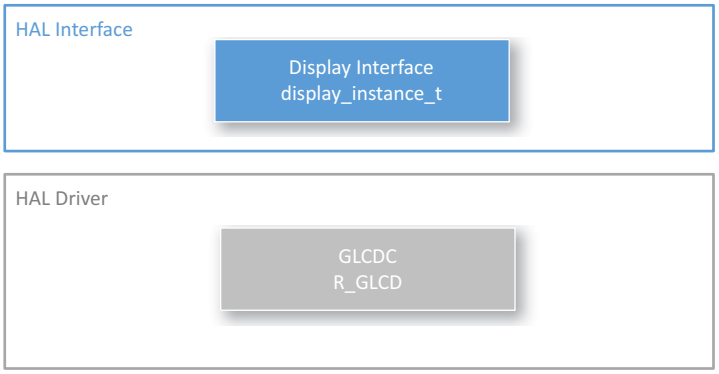

図 131:　ディスプレイ ドライバ - ブロック図

4.2.9.1 ディスプレイ ドライバの機能

- RGB インタフェース （最大 24 ビット）と同期信号 （HSYNC、VSYNC、およびデータ有効 （オプション））を使用して LCD パネルをサポートしています。
- 入力グラフィックス プレーン用の各種の色形式をサポートしています （RGB888、ARGB888、RGB565、ARGB1555、ARGB4444、CLUT8、CLUT4、CLUT1）。
- 512 ワード（32 ビット / ワード）の入力グラフィックス プレーンのための CLUT（カラー ルックアップ テーブル）の使用をサポートしています。
- 出力用の各種の色形式をサポートしています （RGB888、RGB666、RGB565、シリアル RGB）。
- 背景プレーン上に 2 つのグラフィックス プレーンを入力し、画面上でブレンドできます。
- パネルにドット クロックを生成します。クロック ソースは内部または外部 （LCD_EXTCLK）から選択できます。
- 明るさ調整、コントラスト調整、ガンマ補正をサポートしています。
- フレーム バッファの切り替えまたはアンダーフロー検出を処理するための、GLCDC 割り込みをサポートしています。

下図に、GLCDC ドライバ モジュールを使用したグラフィックス データの流れの概要を示します。このドライバは、グラフィックス フレーム画像データのメモリからの読み取り （最大 2 フレーム）と、これら画像のモノクロ背景画面上でのブレンドをサポートしています。このドライバは CLUT メモリをサポートしており、CLUT 用のグラフィック フレーム形式を指定します。

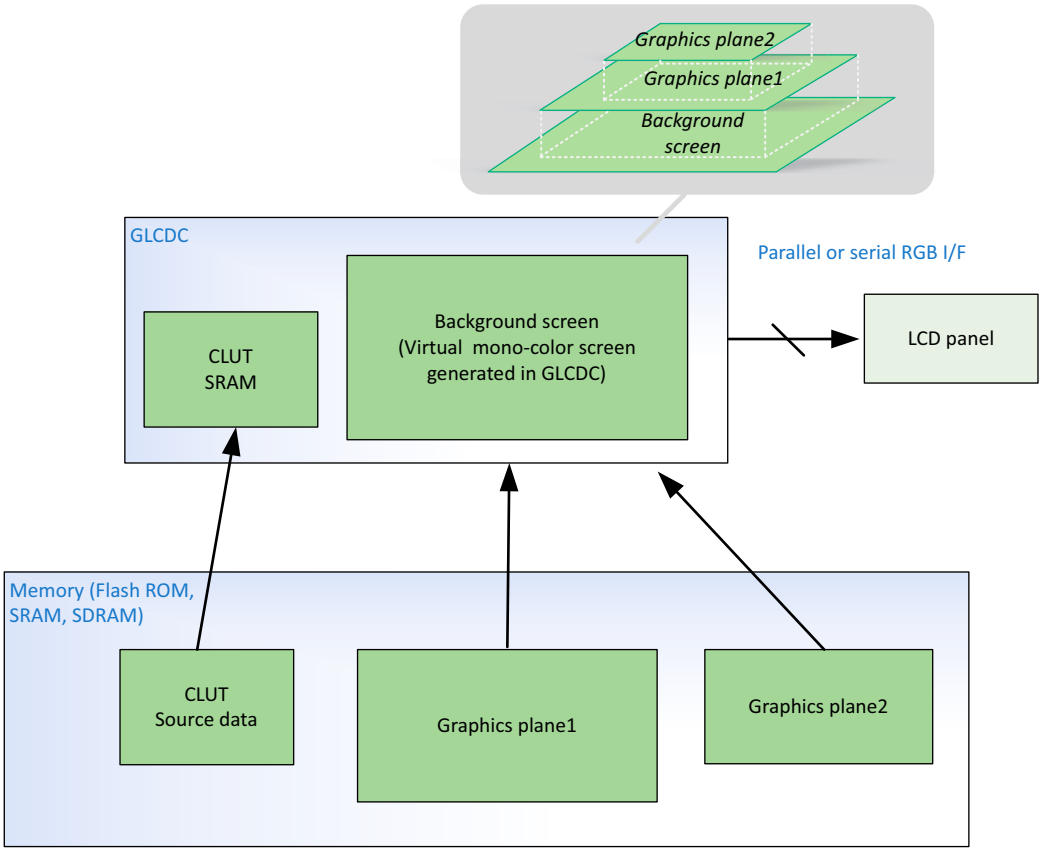

図 132: GLCDC データ フロー

下の図は、ピンポン フレーム バッファを備えた表示システムを示しています。表示システム内では 3 つ以上のフレーム バッファを使用して、単一のフレーム バッファを持つ表示システムで発生する切れ目問題を回避することをお勧めします。そのような設計では、GLCDC ハードウェアがグラフィックス フレーム画像をいずれかのフレーム バッファから読み込んでいる間、画像描画エンジン （DRW や JPEG など）、CPU、または DMAC/DTC が同時にグラフィックス フレーム画像を別のフレーム バッファに転送することができます。このドライバは、layerChange による実行時のフレーム バッファ切り替えをサポートしています。

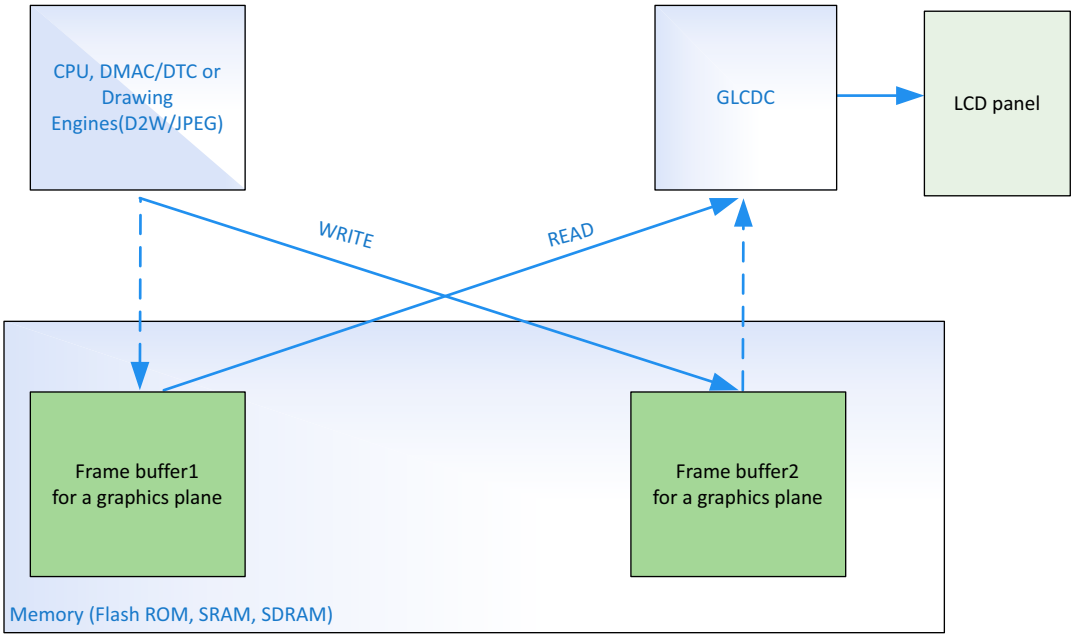

図 133: GLCDC ディスプレイ - GLCDC を使用した一般的なピンポン バッファ システム

4.2.9.2 e² studio ISDE によるディスプレイ ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

次のリソースは、ディスプレイ ドライバを使用するアプリケーションで必要となります。

リソース	ISDE タブ	選択
GLCDC Driver	Threads	[Driver] > [グラフィックス] > [r_glcd 上のディスプレイ ドライバ]
Interrupts	Threads	アプリケーション例については、GLCDC 割り込みの設定

リソース	ISDE タブ	選択
Clock	Clocks	アプリケーション例については、GLCDC ピクセル クロックの設定
Pins	Pins	アプリケーション例については、GLCDC ピンの設定

GLCDC ピクセル クロックの設定

GLCDC モジュールは、以下のいずれかのクロック ソースからピクセル クロックを生成できます。ソース クロックの選択は、e^2 studio ISDE の Synergy 構成で行うことができます。クロックの設定

- 内部クロック ソース （PLLOUT; 240 MHz）
- LCD_EXTCLK ピンからの外部クロック ソース

I :S7G2 WS1 （Working Sample1）チップと WS2 （Working Sample2）チップ以降では、内部クロックが異なります。WS1 チップは PCLKB （最大 60 MHz）を使用していますが、WS2 チップ以降では PLLOUT （最大 240 MHz）が使用されています。

GLCDC ピンの設定

e^2 studio ISDE ピン コンフィギュレータを使用して、GLCDC ピンを設定します （ピンの設定を参照）。

Note：S7G2 PE-HMI1 ボード上で GLCDC モジュールを使用するには、PORT10 ピン 3 （PA03）とピン 5（PA05）を、出力レベル HIGH の IOPORT ピンとして設定してください。ピン PA03 は DISP 信号 （ディスプレイ オン / オフ）を制御し、ピン PA05 は LCD パネルのバックライトを制御します。詳細については、S7G2 PE-HMI1 ボードの回路図を参照してください。

GLCDC 割り込みの設定

以降のセクションで説明するように、複数の GLCDC 割り込みを設定することもできます。

ライン検出割り込み

ライン検出割り込みは、GLCDC がすべてのラインの LCD パネルへの出力を完了し、ブランク期間になったことを示すために使用されます。この割り込みは、2 つ以上のフレームを使用したフレーム バッファを使用するグラフィック システムで、フレーム バッファの切り替えを処理するために使用します。

Note：割り込みを有効にするには、e^2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、GLCDC > GLCDC LINE DETECT 割り込みの優先度を設定します。

layer1 または layer2 バッファ アンダーフローの割り込み

GLCDC layer1 または layer2 バッファ アンダーフローの割り込みを使用すると、システムのメモリ帯域幅不足を検出できます。バッファ アンダーフローが発生するのは、メモリ （SDRAM や SRAM など）から

GLCDC 内部ライン バッファへのグラフィックス データ転送が、他のデータ転送によってブロックされ、GLCDC ライン バッファから LCD パネル インタフェースへのデータ転送に対して十分でない場合です。この割り込みが発生しないようにグラフィックス システムを設計する必要があります。

I：割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、GLCDC > GLCDC UNDERFLOW1 （グラフィックス プレーン 1 の場合）または GLCDC UNDERFLOW1（グラフィックス プレーン 2 の場合）割り込みの優先度を設定します。

GLCDC コールバック

ユーザー コールバック関数を open で登録できます。ユーザー コールバック関数が指定されている場合、割り込みが発生するたびに割り込みサービス ルーチン （ISR）からコールバック関数が呼び出されます。コールバック関数の引数 event は、グラフィックス システムで発生したイベントをユーザーが識別できるように、以下に示す列挙値を受け取ることができます。DISPLAY_EVENT_LINE_DETECTION イベントは、画面を更新するためにフレーム バッファを切り替えるために使用でき、DISPLAY_EVENT_GRn_UNDERFLOW イベントは、アンダーフローが発生した場合のエラー処理に使用できます。

イベント名	割り込み名	イベントの条件
DISPLAY_EVENT_LINE_DETECTION	ライン検出	GLCD が、アクティブ ビデオ領域内の最後のラインを出力した場合
DISPLAY_EVENT_GR1_UNDERFLOW	グラフィックス 1 アンダーフロー	グラフィックス 1 プレーン用のデータの読み取り中に GLCD がアンダーフローした場合
DISPLAY_EVENT_GR2_UNDERFLOW	グラフィックス 2 アンダーフロー	グラフィックス 2 プレーン用のデータの読み取り中に GLCD がアンダーフローした場合

I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

GLCDC パラメータの設定

e2 studio ISDE を使用して、GLCDC HAL ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

参考資料

GLCDC には、以下のコンポーネントの設定が必要です。

- 表示 :display_cfg_t
- GLCDC:glcd_cfg_t

GLCDC ビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking Enable	#define BSP_CFG_PARAM_CHECKING_ENABLE	Enabled (Default), Disabled	パラメータ エラー チェックを有効または無効にします。
Name	Name prefix of instances. Arbitrary symbol (Default: "g_display")	Arbitrary symbol (Default: "g_display")	GLCDC モジュール制御ブロック インスタンスに使用する名前。この名前は、他の可変インスタンスのプレフィックスにも使用されます。
Name of display callback function to be defined by user	Name of display callback function	Arbitrary symbol (Default: "user_display_callback")	名前は有効な C シンボルである必要があります。

GLCDC 入力パラメータ

ISDE プロパティ	設定	設定値	説明
Input - Panel clock source select	clksrc	Internal clock (GLCDCLK) (Default), external clock (LCD_EXTCLK)	システムに応じてパネル クロック ソースを選択します。

ISDE プロパティ	設定	設定値	説明
Input - Graphics screen1/2	display_input_cfg_t :: input[0].p_base and display_input_cfg_t :: input[1].p_base	Used (Default), Not used	グラフィックス画面 N を使用する場合は「使用」を指定します。これにより、グラフィックス画面 1 用の「display_fb_background」と、グラフィックス画面 2 用の「display_fb_foreground」が、ISDE により自動生成されます。いずれのグラフィックス画面も使用しない場合は「未使用」を指定します。その場合、フレーム バッファは作成されません。「未使用」を指定した場合、フレーム バッファに対するメモリ読み取りアクセスがなく、バス帯域幅の消費量が減ることに注意してください。
Input – Section where Graphics screen1/2 frame buffer allocated	Section name for framebuffers	Arbitrary strings started with "." (Default: ".sdram")	フレーム バッファを割り当てるセクション名を指定します。これは、「入力 - グラフィックス画面 1」に「使用」が設定されている場合に有効です。
Input – Graphics screen1/2 frame buffer name	-	fb_foreground/fb_background	フレーム バッファのカスタム名
Input - Graphics screen1/2 input horizontal size	hsize	Arbitrary integer value (Default: 800)	幅のピクセル数を指定します。デフォルト値は、800x480 ピクセルの画像のサイズになります。
Input - Graphics screen1/2 input vertical size	vsize	Arbitrary integer value (Default: 480)	高さのピクセル数を指定します。デフォルト値は、800x480 ピクセルの画像のサイズになります。

ISDE プロパティ	設定	設定値	説明
Input - Graphics screen1/2 input horizontal stride (not bytes but pixels)	hstride	Arbitrary integer value (Default: 800)	水平ラインのメモリ ストライドを指定します。この値は、実際のバイト数ではなくピクセル数で指定する必要があります。一般に、このパラメータは、パラメータ「入力幅」と同じ数に設定します。デフォルト値は、800x480 ピクセルの画像のサイズになります。
Input - Graphics screen1/2 input	display_in_format_tformat	32-bit ARGB8888, 32-bit RGB888, 16-bit RGB565 (Default), 16-bit ARGB1555, 16-bit ARGB4444, CLUT8, CLUT4, CLUT1	グラフィックス画面の入力形式を指定します。CLUT 形式を選択する場合は、clut を使用して CLUT データを書き込んでから start を実行する必要があります。デフォルト設定では、RGB565 形式の画像がサポートされます。
Input - Graphics screen1/2 input line descending	line_descending_enable	On, Off (Default)	画像データが、フレームバッファの一番下のラインから一番上のラインに下降する場合に「オン」を指定します。通常は「オフ」です。
Input - Graphics screen1/2 input lines repeat	lines_repeat_enable	On, Off (Default)	LCD パネルのサイズよりも小さいラスター画像を繰り返し読み取ることが期待される場合は「オン」を指定します。通常は「オフ」です。詳細については、ライン リピート機能の説明を参照してください。
Input - Graphics screen1/2 input lines repeat times	lines_repeat_times	Arbitrary integer value between 0 and 65536 (Default: 0)	フレームに繰り返し読み込まれるラスター画像の繰り返し回数を指定します。
Input - Graphics screen1/2 layer coordinate X	x display_layer_t::coodinate	Arbitrary Integer value between 0 and the maximum horizontal pixel size (Default: 0)	グラフィックス画面の背景画面からの水平オフセットをピクセル単位で指定します。

ISDE プロパティ	設定	設定値	説明
Input - Graphics screen1/2 layer coordinate Y	y display_layer_t::coodinate	Arbitrary Integer value between 0 and the maximum vertical pixel size (Default: 0)	グラフィックス画面の背景画面からの垂直オフセットをピクセル単位で指定します。
Input - Graphics screen1/2 layer background color alpha	display_color_t :: byte.abg_color	Arbitrary alpha value between 0 and 255 (Default: 255) for the background color (the region outside of the active video region in the graphics screen).	アルファ値に基づいて、グラフィックス画面 2 （前景グラフィックス画面）がグラフィックス画面 1 （背景グラフィックス画面）にブレンドされるか、グラフィックス画面 1 がモノクロ背景画面にブレンドされます。
Input - Graphics screen1/2 layer background color Red/Green/Blue	display_color_t :: byte.r display_layer_t:: bg_color display_color_t :: byte.g, bg_colordisplay_color_t :: byte.b bg_color	Arbitrary (R, G, B) color with a value between 0 and 255 (Default: 255) for the background color (the region outside of active video region in the graphics screen).	グラフィックス画面 N の背景色を指定します。
Input - Graphics screen1/2 layer fading control	display_fade_control_tfade_control	On, Off (Default)	グラフィックス画面のフェードインを行うには「オン」を指定します。透明の画面が徐々に不透明に変化します。グラフィックス画面のフェードアウトを行うには「オフ」を指定します。不透明の画面が徐々に透明に変化します。この処理は、GLCDC ハードウェアによってアクセラレートされ、いったん開始すると停止できないことに注意してください。遷移ステータスは、statusGet で監視できます。
Input - Graphics screen1/2 layer fade speed	fade_speed	Arbitrary integer value (Default: 0)	フェード遷移が完了するフレーム数を指定します。

GLCDC 出力パラメータ

ISDE プロパティ	設定	設定値	説明
Output - Horizontal total cycles	total_cychtiming	Arbitrary integer value between 24 and 1024 (Default: 1024)	水平ラインの合計サイクル数を指定します。システムの LCD パネル シートのデータ シートに定義されているサイクル数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Horizontal active video cycles	display_cychtiming	Arbitrary integer value between 16 and 1016 (Default: 800)	水平ラインのアクティブ ビデオ サイクル数を指定します。システムの LCD パネル シートのデータ シートに定義されているサイクル数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Horizontal back porch cycles	back_porchhtiming	Arbitrary integer value between 6 and 1006 (Default: 46)	水平ラインのバック ポーチ サイクル数を指定します。バック ポーチは、Hsync サイクルの始点から開始します。つまり、バック ポーチ サイクルには Hsync サイクルが含まれます。システムの LCD パネル シートのデータ シートに定義されているサイクル数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Horizontal sync signal cycles	sync_widthhtiming	Arbitrary integer value between 0 and 1023 (Default: 20)	Hsync 信号アサーション サイクル数を指定します。システムの LCD パネル シートのデータ シートに定義されているサイクル数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。

ISDE プロパティ	設定	設定値	説明
Output - Horizontal sync signal polarity	sync_polarityhtiming	Low active (Default), High active	システムに合わせて、Hsync 信号の極性を選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Vertical total lines	total_cycvtiming	Arbitrary integer value between 20 and 1024 (Default:525)	1 フレームの合計ライン数を指定します。システムの LCD パネル シートのデータ シートに定義されているライン数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Vertical active video lines	display_cycvtiming	Arbitrary integer value between 16 and 1020 (Default: 480)	1 フレームのアクティブ ビデオ ライン数を指定します。システムの LCD パネル シートのデータ シートに定義されているライン数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Vertical back porch cycles	back_porchvtiming	Arbitrary integer value between 3 and 1007 (Default: 23)	1 フレームのバック ポーチ ライン数を指定します。バック ポーチは、Vsync ラインの始点から開始します。つまり、バック ポーチ ラインには Vsync ラインが含まれます。システムの LCD パネル シートのデータ シートに定義されているライン数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。

ISDE プロパティ	設定	設定値	説明
Output - Vertical sync signal lines	sync_widthvtiming	Arbitrary integer value between 0 and 1023 (Default:10)	1 フレームの Vsync 信号アサーション ライン数を指定します。システムの LCD パネル シートのデータ シートに定義されているライン数を設定します。デフォルト値は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Vertical sync signal polarity	display_output_cfg_t :: vtiming.sync_polarity	Low active (Default), High active	システムに合わせて、Vsync 信号の極性を選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Format	display_out_format_tformat	24-bits RGB888 (Default), 18bits RGB666, 16bits RGB565, 8bits Serial RGB	LCD パネルに合ったグラフィックス画面の出力形式を指定します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Endian	display_endian_tendian	Little endian (Default), Big endian	LCD パネルへの出力信号のデータ エンディアンを選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Color order	display_color_order_tcolor_order	RGB (Default), BGR	LCD パネルへの出力信号のデータ オーダーを選択します。赤と青の順序は、必要に応じて入れ替えることができます。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Data Enable Signal Polarity	display_signal_polarity_tdata_enable_polarity	Low active, High active(Default)	システムに合わせて、データ有効信号の極性を選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。

ISDE プロパティ	設定	設定値	説明
Output - Sync edge	display_sync_edge_tsync_edge	Rising edge(Default), Falling edge	システムに合わせて、同期信号の極性を選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネルに一致します。
Output - Background color alpha channel	display_color_t :: byte.abg_color	Arbitrary alpha value between 0 and 255 (Default: 255) for the background color (the region outside of the active video region in the graphics screen).	-
Output - Background color R/G/B channel	display_color_t :: byte.r display_color_t :: byte.g, display_color_t :: byte.b, bg_color	Arbitrary (R, G, B) color with the value between 0 and 255 (Default: 255) for the background color (the region outside of active video region in the background screen).	背景画面の背景色を指定します。

GLCDC カラー ルックアップ テーブル （CLUT）パラメータ

ISDE プロパティ	設定	設定値	説明
CLUT	CLUT buffer generation	Used (default), Not used	グラフィックス画面の入力形式に CLUT 形式を選択する場合は、「使用」を指定します。その場合、CLUT ソース データ用の「CLUT_buffer」という名前のバッファが、ISDE で自動生成されるソース ファイル中に生成されます。

ISDE プロパティ	設定	設定値	説明
CLUT buffer size	Size of CULT buffers in bytes. The name of CUT buffer is fixed as: CLUT_buffer	Arbitrary integer value (Default:256)	CLUT ソース データ バッファのエントリ数を指定します。各エントリは 4 バイト （1 ワード）を消費します。このパラメータで指定された CLUT ソース データのワードは、ISDE で自動生成されるソース ファイル中に生成されます。

GLCDC TCON ピン構成パラメータ

ISDE プロパティ	設定	設定値	説明
TCON - Hsync pin select	glcd_tcon_pin_ttcon_hsync	Not used, LCD_TCON0 (Default), LCD_TCON1, LCD_TCON2	システムに合わせて、Hsync 信号に使用する TCON ピンを選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネル用です。
TCON - Vsync pin select	glcd_tcon_pin_ttcon_vsync	Not used, LCD_TCON0, LCD_TCON1 (Default), LCD_TCON2	システムに合わせて、Vsync 信号に使用する TCON ピンを選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネル用です。
TCON - DataEnable pin select	glcd_tcon_pin_ttcon_de	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2 (Default)	システムに合わせて、DataEnable 信号に使用する TCON ピンを選択します。デフォルト設定は、S7G2 PE-HMI1 ボード上の LCD パネル用です。
TCON - Panel clock division ratio	glcd_panel_clk_div_tclock_div_ratio	1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8 (Default), 1/9, 1/12, 1/16, 1/24, 1/32	クロック ソースの除算値を選択します。ピクセル クロックのソース クロックについては、この表の下部にある注記を参照してください。

GLCDC 色補正パラメータ

ISDE プロパティ	設定	設定値	説明
Color correction - brightness	enable	On, Off (Default)	明るさ制御を行う場合は「オン」を指定します。「オフ」を指定した場合、以下の設定は出力色に影響を与えません。
Color correction - Brightness R/G/B channel	r, g, b	Arbitrary integer value between 0 and 1023 (Default: 512)	出力色レベルは次のように計算されます : 出力色レベル = 入力色レベル +/ - 512。R、G、B チャネルのそれぞれの値を設定します。
Color correction - Contrast	enable	On, Off (Default)	コントラスト制御を行う場合は「オン」を指定します。「オフ」を指定した場合、以下の設定は出力色に影響を与えません。
Color correction - Contrast (gain) R/G/B channel	r, g, b	Arbitrary integer value between 0 and 255 (Default: 128)	出力色レベルは次のように計算されます : 出力色レベル = 入力色レベル x (/128)。R、G、B チャネルのそれぞれの値を設定します。
Color correction – Gamma correction	enable, r, g, b	On, Off (Default)	各チャネルの R/G/B の制御。赤チャネルのガンマ補正を行う場合は「オン」を指定します。「オフ」を指定した場合、ゲインとしきい値の設定は出力色に影響を与えません。
Color correction - Gamma gain R/G/B[0]…R/G/B[15]	gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM], r, g, b	Arbitrary integer value between 0 and 2047 (Default: 0)	ガンマ補正カーブ上のエリア N 内の赤チャネルのゲイン値を設定します。エリア N のゲイン設定は、色レベルが （(ガンマしきい値 R[N-1]）<<2）と （(ガンマしきい値 R[N]）<<2）の間にある入力データに適用されます。出力値は次のように計算されます : 出力色レベル = 入力色レベル / 1024 （/128）。

ISDE プロパティ	設定	設定値	説明
Color correction - Gamma threshold R/G/B[0]…R/G/B[15]	threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM] r, g, b	Arbitrary integer value between 0 and 1023 (Default: 0)	ガンマ補正カーブ上のエリア N 内の赤チャネルのしきい値を設定します。エリア N のゲイン設定は、色レベルがガンマしきい値 R[N-1] とガンマしきい値 R[N] の間にある入力データに適用されます。出力値は次のように計算されます : 出力色レベル = 入力色レベル / 1024 （/128）。

GLCDC ディザリング パラメータ

ISDE プロパティ	設定	設定値	説明
Dithering	dithering_on	On, Off (default)	ディザリング有効。出力形式 RGB666 または RGB565 を選択する場合に、ディザ効果を適用しバンディングを減らす場合は「オン」を指定します。ディザリングは変換時に適用できます。「オフ」を指定した場合、以下のディザリング設定は出力に影響を与えません。ディザ効果の詳細については、ユーザーズマニュアルの出力制御ブロック パネル ディザ補正レジスタ （OUT_PDTHA）を参照してください。
Dithering - Mode	glcd_dithering_mode_tdithering_mode	Truncate (Default), Round off, 2x2 Pattern	ディザ モードを指定します。詳細については、ユーザーズマニュアルの出力制御ブロック パネル ディザ補正レジスタ （OUT_PDTHA）を参照してください。

ISDE プロパティ	設定	設定値	説明
Dithering - Pattern A/B/C/D	glcd_dithering_pattern_tdithering_pattern_A, glcd_dithering_pattern_tdithering_pattern_B, glcd_dithering_pattern_tdithering_pattern_C,glcd_dithering_pattern_tdithering_pattern_D	Pattern 00, Pattern 01, Pattern 10, Pattern 11	2X2 パターン モードのディザ パターンを指定します。詳細については、ユーザーズマニュアルの出力制御ブロック パネル ディザ補正レジスタ（OUT_PDTHA）を参照してください。
Misc - Correction Process Order	glcd_correction_proc_order_tcorrection_proc_order	Brightness and Contrast then Gamma (Default) or Gamma, then Brightness and Contrast	必要に応じて色補正処理の順序を指定します。

I :S7G2 WS2 チップ以降のピクセル クロックのソース クロックは PLLOUT（最大 240MHz）ですが、S7G2 WS1 チップの場合は PCLKB （最大 60MHz）になります。

ユーザー定義のコールバック関数例の表示

初期化コード例の表示

```
/* Display frame buffer */

uint8_t g_display_fb_background[2][800*480*2] __attribute__((aligned(64), section(".sdram")));

/* Display device extended configuration */

static const glcd_cfg_t  g_display_extend_cfg = {

  .tcon_hsync  = GLCD_TCON_PIN_0,

  .tcon_vsync = GLCD_TCON_PIN_1,

  .tcon_de = GLCD_TCON_PIN_2,

  .correction_proc_order = GLCD_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA,

  .clksrc = GLCD_CLK_SRC_INTERNAL,

  .clock_div_ratio = GLCD_PANEL_CLK_DIVISOR_8,

  .dithering_mode = GLCD_DITHERING_MODE_TRUNCATE,

  .dithering_pattern_A = GLCD_DITHERING_PATTERN_11,

  .dithering_pattern_B = GLCD_DITHERING_PATTERN_11,

  .dithering_pattern_C = GLCD_DITHERING_PATTERN_11,

  .dithering_pattern_D = GLCD_DITHERING_PATTERN_11
```

```
};
/* Display control block instance */
display_ctrl_t g_display_ctrl;
/* Display interface configuration */
const display_cfg_t g_display_cfg =
{
/* Input1 (Graphics1(background) screen) configuration */
.input[0] = {
  .p_base = (uint32_t *)&g_display_fb_background[0],
  .hsize = 800,
  .vsize = 480,
  .hstride = 800,
  .format = DISPLAY_IN_FORMAT_16BITS_RGB565,
  .line_descending_enable = false,
  .lines_repeat_enable = false,
  .lines_repeat_times = 0
},
/* Input2 (Graphics2(foreground) screen) configuration (Not used) */
.input[1] = {
  .p_base = NULL,
  .hsize = 800,
  .vsize = 480,
  .hstride = 800,
  .format = DISPLAY_IN_FORMAT_16BITS_RGB565,
```

```
    .line_descending_enable = false,
    .lines_repeat_enable = false,
    .lines_repeat_times = 0
},
/* Input1 (Graphics1(background) screen) layer configuration */
.layer[0] = {
    .coordinate = {.x = 0, .y = 0 },
    .bg_color = {
    .byte = {.a = 255, .r = 255, .g = 255, .b = 255}
    },
    .fade_control = DISPLAY_FADE_CONTROL_NONE,
    .fade_speed = 0
},
/* Input2 (Graphics1(background) screen) layer configuration (Not used) */
.layer[1] = {
        .coordinate = {.x = 0, .y = 0 },
        .bg_color = {
            .byte = {.a = 255, .r = 255, .g = 255, .b = 255}
    },
    .fade_control = DISPLAY_FADE_CONTROL_NONE,
```

```
    .fade_speed = 0
},
/* Output configuration */
.output = {
    .htiming = {
        .total_cyc = 1024,
        .display_cyc = 800,
        .back_porch = 46,
        .sync_width = 20,
        .sync_polarity = DISPLAY_SIGNAL_POLARITY_LOACTIVE
    },
    .vtiming = {
        .total_cyc = 525,
        .display_cyc = 480,
        .back_porch = 23,
        .sync_width = 10,
        .sync_polarity = DISPLAY_SIGNAL_POLARITY_LOACTIVE
```

```
 },
.format = DISPLAY_OUT_FORMAT_24BITS_RGB888,
 .endian = DISPLAY_ENDIAN_LITTLE,
 .color_order = DISPLAY_COLOR_ORDER_RGB,
 .data_enable_polarity = DISPLAY_SIGNAL_POLARITY_HIACTIVE,
 .sync_edge = DISPLAY_SIGNAL_SYNC_EDGE_RISING,
 .bg_color = {
      .byte = {.a = 255, .r = 0, .g = 0, .b = 0}
 },
 .brightness = {.enable = false, .r = 512, .g = 512, .b = 512},
 .contrast = {.enable = false, .r = 128, .g = 128, .b = 128},
 .p_gamma_correction =
(display_gamma_correction_t *)(&g_display_gamma_cfg),
 .dithering_on = false
 },
 /* Display device callback function pointer */
```

```
    .p_callback = NULL,

    .p_context = (void *) &g_display,

    .p_extend = (void *) (&g_display_extend_cfg)

};

/* Instance structure to use this module. */

const display_instance_t g_display = {

    .p_ctrl = &g_display_ctrl,

    .p_cfg = (display_cfg_t *) &g_display_cfg,

    .p_api = (display_api_t *) &g_display_on_glcd

};
```

表示の開始

フレーム バッファの画像の表示を開始するには、open を呼び出します。CLUT 形式を使用する場合は、clut を呼び出して CLUT SRAM を初期化します。e^2 studio ISDE の Synergy コンフィギュレータを通じて生成された CLUT 構成構造体 （display_clut_cfg_t 型）のインスタンスを変更して使用することができます。

```
/* Display driver open */

g_display.p_api->open (g_display.p_ctrl, g_display.p_cfg);

/* Display driver start */

g_display.p_api->start (g_display.p_ctrl);
```

表示の停止

フレーム バッファの画像の表示を停止するには、close を呼び出します。

```
/* Display driver close */

g_display.p_api->close (g_display.p_ctrl);
```

ランタイム レイヤー設定の変更

ランタイム設定（display_runtime_cfg_t 型）パラメータは ISDE の Synergy コンフィギュレータを通じて生成し、GLCDC レイヤー画面の更新に利用することができます。実行時設定を指定して、layerChange を呼び出します。GLCDC ハードウェアがグラフィックス画面を更新している間は、この関数がエラー コード SSP_ERR_INVALID_UPDATE_TIMING ですぐに返り、更新要求が破棄される場合があります。こうした場

合は、次の Vsync 信号がアサートされるまで待ってから、再度この関数を呼び出してください。画面のオンとオフを切り替える際に、layerChange を使用してフェードイン / フェードアウト効果で画面を遷移させる場合は、遷移が完了するまで待ってから、次の実行時画面の更新を行ってください。フェード遷移のステータスを確認するには、statusGet をポーリングし、DISPLAY_FADE_STATUS_NOT_UNDERWAY が返されることを確認します。

```
display_runtime_cfg_t g_display_runtime_cfg_bg =

{

.input =

{

    .p_base = (uint32_t *) &g_display_fb_background[0],

    .hsize = 800,

    .vsize = 480,

    .hstride = 800,

    .format = DISPLAY_IN_FORMAT_16BITS_RGB565,

    .line_descending_enable = false,

    .lines_repeat_enable = false,

    .lines_repeat_times = 0

},

.layer =

{

    .coordinate = {.x = 0, .y = 0},

    .bg_color = {

        .byte = {
```

```
                .a = 255,

                .r = 255,

                .g = 255,

                .b = 255

            }

        },

        .fade_control = DISPLAY_FADE_CONTROL_NONE,

        .fade_speed = 0

        }

    };
```

ランタイム色補正設定の変更

LCD パネルの色補正を実行できます （display_correction_t 型)。明るさまたはコントラストは変更できますが、ガンマ補正の設定は変更できないことに注意してください。上記の設定で、correction を呼び出します。GLCDC ハードウェアがグラフィックス画面を更新している間は、API がエラー コード SSP_ERR_INVALID_UPDATE_TIMING ですぐに返り、更新要求が破棄されることに注意してください。

```
display_correction_t  display_correction;

display_correction.brightness.enable  = false;

display_correction.contrast.enable  = true;

display_correction.contrast.g  = 128;

while (SSP_SUCCESS != g_display.p_api->correction(g_display.p_ctrl, &display_correction)) {

/* Wait until the function returns SSP_SUCCESS */

}
```

ディスプレイ ユーザー コールバック関数例

GLCDC モジュールは、open の構成 p_callback に渡されたユーザー定義のコールバック関数を呼び出すことができます。ユーザーはこのコールバック関数を利用して、ディスプレイ デバイスのブランキング期間の開始タイミングを知って追加処理を実行したり、各レイヤーで発生したバッファ アンダーフローを検出したりできます。

```
void user_callback_glcdc(display_callback_args_t * p_arg)
{
    switch (p_arg->event)
    {
    case DISPLAY_EVENT_LINE_DETECTION:
        /* Display device goes into blanking period.
        * Do operation.
        */
        break;
    case DISPLAY_EVENT_GR1_UNDERFLOW:
        /* Layer1 line buffer underflow happen.
        * Do operation.
        */
        break;
    case DISPLAY_EVENT_GR2_UNDERFLOW:
        /* Layer2 line buffer underflow happen.
        * Do operation.
        */
        break;
    default:
        break;
    }
}
```

4.2.9.3 LCD タイミング

画面のフォーマット

次の図は、GLCDC モジュールの LCD 画面フォーマットと LCD タイミング パラメータの関係を示したものです。このモジュールは、LCD パネル設定用の汎用タイミング パラメータを備えているため、さまざまな LCD パネルをサポートします。

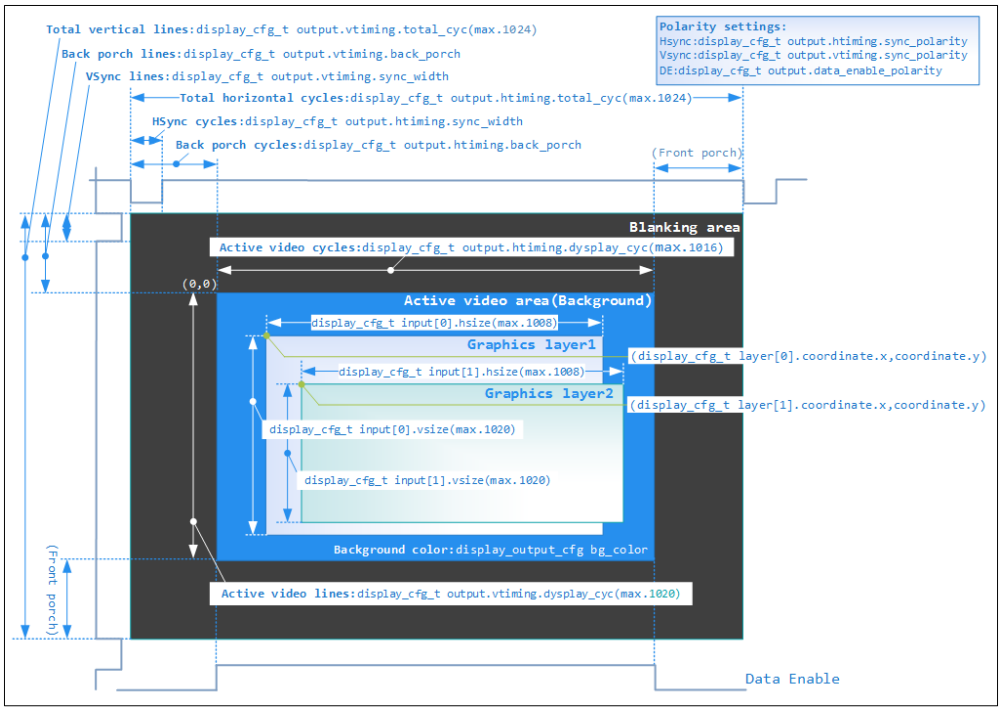

図 134: GLCDC 画面のフォーマット

フロント ポーチ期間

GLCDC モジュールは、垂直 / 水平フロント ポーチ サイクル / ラインに対する設定を備えていません。これらのサイクル / ラインは、合計水平サイクル / 垂直ライン設定に含まれている必要があります。このモジュールでは、以下で示す GLCDC のハードウェア仕様に基づき、バック ポーチ サイクル / ラインを設定する必要があります。一般的な LCD パネルが備えたバック ポーチ サイクル / ラインは以下で示す数よりも多いため、これはモジュールの真の制限ではありません。

- 水平バック ポーチ サイクル数 >= 3
- 垂直バック ポーチ ライン数 >= 2

パラメータ設定例

PE-HMI1 v2.0 ボード （LXD Research & Display, LLC, M7504A）:

以下の例では、60 Hz の LCD パネル リフレッシュ レートを生成するため、水平合計サイクル、垂直合計ライン、パネル クロック分周比を調整しています。LCD パネルの記号については、M7504A データ シートを参照してください。

参考資料

ISDE プロパティ	設定値
Panel clock source select	Internal clock
Graphics screen1/2 input horizontal size	800（thd）
Graphics screen1/2 input vertical size	480（tvd）
Graphics screen1/2 input horizontal stride	800（thd）
Horizontal total cycles	976（th）
Horizontal active video cycles	800（thd）
Horizontal back porch cycles	46（thp + thb）
Horizontal sync signal cycles	20（thp）
Horizontal sync signal polarity	Low active
Vertical total lines	512（tv）
Vertical active video lines	480（tvd）
Vertical back porch cycles	23（tvp + tvb）
Vertical sync signal lines	10（tvp）
Vertical sync signal polarity	Low active
Output Format	24bits RGB888
Data Enable Signal Polarity	High active
Sync edge	Rising edge
TCON - Hsync pin select	LCD_TCON0
TCON - Vsync pin select	LCD_TCON1
TCON - DataEnable pin select	LCD_TCON2
TCON - Panel clock division ratio	1/8

DK-S7G2 v3.0 ボード （LXD Research & Display, LLC, M7190A）：

参考資料

以下の例では、60 Hz の LCD パネル リフレッシュ レートを生成するため、水平合計サイクル、垂直合計ライン、パネル クロック分周比を調整しています。LCD パネルの記号については、M7190A データ シートを参照してください。

ISDE プロパティ	設定値
Panel clock source select	Internal clock
Graphics screen N input horizontal size	480（thd）
Graphics screen N input vertical size	272（tvd）
Graphics screen N input horizontal stride	480（thd）
Horizontal total cycles	582（th）
Horizontal active video cycles	480（thd）
Horizontal back porch cycles	43（thp + thb）
Horizontal sync signal cycles	41（thp）
Horizontal sync signal polarity	Low active
Vertical total lines	286（tv）
Vertical active video lines	272（tvd）
Vertical back porch cycles	12（tvp + tvb）
Vertical sync signal lines	10（tvp）
Vertical sync signal polarity	Low active
Output Format	16bits RGB565
Data Enable Signal Polarity	High active
Sync edge	Rising edge
TCON - Hsync pin select	LCD_TCON1
TCON - Vsync pin select	LCD_TCON2
TCON - DataEnable pin select	LCD_TCON0
TCON - Panel clock division ratio	1/24

SK-S7G2 v2.0 ボード （ILI Technology Corp., IL9341C）：

以下の例では、水平合計サイクルと垂直合計ラインを、約 76.8 Hz の LCD パネル リフレッシュ レートのパネルに許容される最大値に設定しています。LCD パネルの記号については、LIL9314V データ シートを参照してください。

ISDE プロパティ	設定値
Panel clock source select	Internal clock
Graphics screen N input horizontal size	256（See note）
Graphics screen N input vertical size	320（VAdr）
Graphics screen N input horizontal stride	256（See note）
Horizontal total cycles	290（HAdr + Hsync（16）+ HBP（24）+ HFP（16））
Horizontal active video cycles	240（HAdr）
Horizontal back porch cycles	40（Hsync（16）+ HBP（24））
Horizontal sync signal cycles	16（Hsync）
Horizontal sync signal polarity	Low active
Vertical total lines	330（VAdr + Vsync（4）+ VBP（2）+ VFP（4））
Vertical active video lines	320（VAdr）
Vertical back porch cycles	6（Vsync + VBP）
Vertical sync signal lines	4（Vsync）
Vertical sync signal polarity	Low active
Output Format	16bits RGB565
Data Enable Signal Polarity	High active
Sync edge	Rising edge
TCON - Hsync pin select	LCD_TCON2
TCON - Vsync pin select	LCD_TCON1
TCON - DataEnable pin select	LCD_TCON0

ISDE プロパティ	設定値
TCON - Panel clock division ratio	1/32

I：パネルのパラメータは 240 ピクセルに設定する必要がありますが、入力幅および水平ストライドは、意図的に 256 ピクセルに設定されています。これは、GLCDC ハードウェアに合わせ、水平ラインを 64 バイトにする必要があるためです。ライン開始点での 240 ピクセルが有効であれば、ラインの残りのピクセル（16 ピクセル）は考慮されません。

4.2.9.4 CLUT

GLCD モジュールは、色形式が ARGB1555、CLUT8、CLUT4、または CLUT1 の場合に使用される CLUT をサポートしています。clut は、CLUT0/CLUT1 SRAM （GLCDC ハードウェア内部で実装）を、グラフィックスの前景画面または背景画面それぞれに対して更新できます。

I a:CLUT を使用するカラー フォーマットを選択する場合は、必ず clut を呼び出してから start を実行してください。そうしないと、CLUT0 と CLUT1 が不明な状態となり、グラフィックスが正しく表示されません。

また、実行時に clut を呼び出して、CLUT SRAM を更新することもできます。API は CLUT データのソースを、現在使用されていない CLUT SRAM にコピーすることに注意してください （各 CLUT SRAM はピンポン バッファで構成されます）。CLUT データ更新の完了後、API は、画像が切れるのを避けるため、GLCDC ハードウェアが読み取る CLUT SRAM を次のフレームから自動的に切り替えます。

4.2.9.5 ライン繰り返しモード

ライン繰り返しは、メモリが十分にないシステムで特に重要なモードです。このモードでは、GLCDC モジュールは LCD パネルの画面サイズよりも少ないピクセルのラスター画像を読み込み、ラスターを繰り返し画面に表示します。下図は、背景グラフィックス プレーンに小さいラスター画像を繰り返し読み込むことで構築された画面の例です。

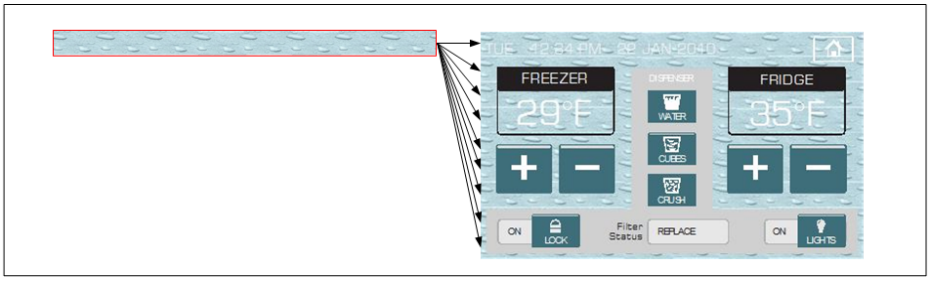

図 135: GLCDC ライン繰り返しモード

I：このモードを有効にするには、GLCD モジュールのプロパティ「入力 - グラフィックス画面 N 入力ライン繰り返し」（N = 1 または 2）に Synergy 設定で「オン」を設定し、ラスター画像を読み込む繰り返し回数を「入力 グラフィックス画面 1/2 入力ライン繰り返し回数」で指定します。ラスター画像の水平ピクセル サイズを「入力 グラフィックス画面 N 入力幅」と「入力 グラフィックス画面 N 入力水平 ストライド」に指定し、ラスター画像の垂直ピクセル サイズを「入力 グラフィックス画面 N 入力高さ」に指定します。

4.2.9.6 ガンマ補正

ガンマ補正は、LCD パネルの色特性をフラットな特性に変更するために使用します。下図に、GLCDC モジュールで設定可能なガンマ補正カーブを示します。このモジュールは、（R, G, B）色のそれぞれについて、入力色レベルに対して 16 個のしきい値をサポートしており、16 のエリアそれぞれに対してしきい値で割ったゲイン レベルを定義します。

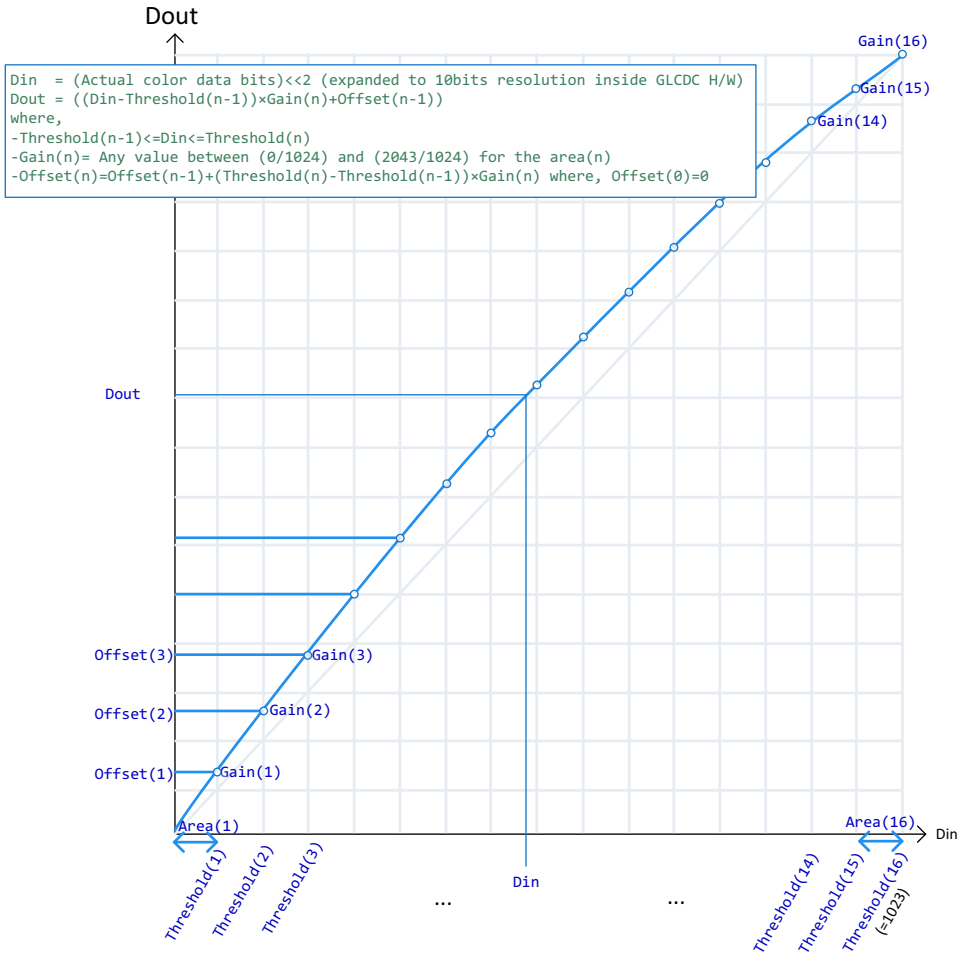

図 136: GLCDC ガンマ補正カーブ

I：（R, G, B）チャネルのそれぞれについてガンマ補正を有効にするには、GLCD モジュールのプロパティ「色補正 - ガンマ補正 （R,G, B)」に ISDE で「オン」を設定します。しきい値 （合計 16 個）は、「色 補正 - ガンマ補正しきい値 （R, G, B）[n]」（n=[0..15]）に設定します。各エリアのゲイン値は、「色補正 - ガンマ 補正ゲイン （R, G, B）[n]」（n=[0..15]）に設定します。

4.2.9.7 ディスプレイ ドライバの制限事項

- r_glcd 上のディスプレイ ドライバは、RGB インデックス彩度キーをサポートしていません。
- r_glcd 上のディスプレイ ドライバは、イベント リンク機能をサポートしていません。

4.2.9.8 ディスプレイ ドライバでサポートされるデバイス

このモジュールは、次のファミリをサポートするように設計されており、またこれらのファミリでテストも実施されています。

- S7G2

4.2.10 データ操作回路ドライバ

データ操作回路ドライバ （DOC ドライバ）は r_doc として実装され、MCU で入手可能なデータ操作回路（DOC）用の API を含みます。このセクションでは、e^2 studio ISDE を使用して DOC ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [監視] > [r_doc 上のデータ操作回路ドライバ] を選択することで、データ操作回路ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による DOC ドライバを使用するアプリケーションの作成

API リファレンスは、次の DOC インタフェースの説明内に記載されています : DOC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

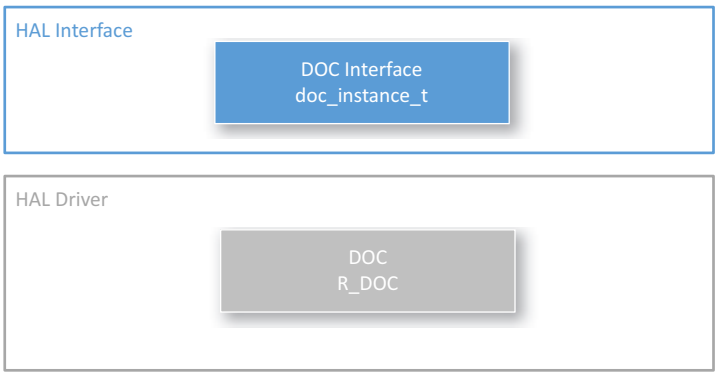

図 137: DOC ドライバ - ブロック図

4.2.10.1 データ操作回路ドライバの機能

このドライバは、ユーザーの設定に従って MCU 上の DOC を制御します。DOC モジュールの周辺機器は 16 ビット データを比較するために使用され、doc_event_t で以下のイベントを検出できます。

- データ値の不一致または一致
- 加算演算のオーバフロー
- 減算演算のアンダーフロー

設定されたイベントが発生し、コールバックを使用できる場合 （割り込みが有効）、ドライバはコールバックを呼び出します。その際、イベント doc_event_t を示すイベント doc_callback_args_t が引数として渡されます。

割り込みが有効になっていない場合は、比較、加算、または減算演算のステータスをポーリングするための DOS ステータスの確認が API でサポートされます。

4.2.10.2 e² studio ISDE による DOC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

DOC ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
DOC Driver	Threads	[Driver] > [監視] > [r_doc 上のデータ操作回路ドライバ]
Interrupts	Threads	[DOC] > [DOC DOPCF]

DOC モジュールがポーリングされるだけの場合、そのモジュールの割り込みは無効化したままで構いません。

DOC クロックの設定

DOC モジュールには、個別のクロック設定は不要です。

DOC ピンの設定

DOC モジュールには、個別のピン設定は不要です。

DOC 割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから DOC 割り込みを設定します （割り込みの設定を参照）。

I：割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、DOC >DOC DOPCF の優先度を設定します （割り込みの設定を参照）。

これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_DOC_INT に、選択した優先度が設定されます。

この割り込みが BSP で有効になっている場合、対応する割り込みサービス ルーチン （ISR）が DOC ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、そのコールバック関数を呼び出します。

DOC パラメータの設定

e2 studio ISDE を使用して、DOC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

I :DOC ドライバには、doc_cfg_t の設定が必要です。

DOC の設定

ISDE プロパティ	設定	設定値	説明
Event	event	doc_event_t	DOC 割り込みをトリガーするイベントを指定します。

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、設定された DOC イベントが発生すると、この関数が割り込みサービス ルーチン（ISR）から呼び出されます。 I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
Name	p_context	User-defined	-

4.2.10.3 DOC ドライバ使用上の注意

比較データ

比較データの初期設定は、write を呼び出すことで DOC に書き込まれます。API write は、DOC DODSR および DODIR レジスタに書き込みます。比較対象のデータが変化しない場合は、比較が必要になるたびにそのデータを書き込み直す必要はありません。比較対象の値は、inputRegisterWrite を呼び出して DOC に書き込むことができます。API inputRegisterWrite は DOC DODIR レジスタにのみ書き込みを行います。

4.2.10.4 DOC アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const doc_instance_t  g_doc =

{

    .p_ctrl  = &g_doc_ctrl,

    .p_cfg = &g_doc_cfg,

    .p_api = &g_doc_on_doc

};
```

DOC ドライバを使用して DOC アプリケーションを作成するには、次の手順を実行します。

1) r_doc 上の DOC モジュール g_doc データ操作回路ドライバに対し、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが制御構造体 doc_ctrl_t に加えて、src/ssp_gen フォルダ内に自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

   I :ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) DOC インスタンスをオープンします。DOC ドライバは、次の DOC ドライバを通じて呼び出されます

```
g_doc.p_api->open(g_doc.p_ctrl, g_doc.p_cfg)
```

   g_doc.p_ctrl と g_doc.p_cfg は、ADC の設定ステップの後で自動生成されます。

4) 型が doc_data_t の変数を作成し、そこに比較値を追加します。比較演算中、1 つの値を無変化のままにする場合は、その値を dodsr に書き込みます。以下に例を示します。

```
doc_data_t  g_doc_values;

  g_doc_values.dodir  = 0x1000;

  g_doc_values.dodsr  = 0x1000;

  g_doc.p_api->write(g_doc.p_ctrl, &g_doc_values)
```

5) その後、比較 / 加算 / 減算データを DOC DODIR レジスタに書き込むには、以下を呼び出します :

```
g_doc.p_api->inputRegisterWrite(g_doc.p_ctrl, value)
```

value はユーザーが作成した変数または値です。

6) DOC 操作のステータスを読み取るには、以下を呼び出します :

```
g_doc.p_api->statusGet(g_doc.p_ctrl, &my_Status)
```

my_status はユーザーが作成した変数であり、読み取られた DOC ステータスがここに格納されます。

7) DOC のステータス フラグをクリアするには、以下を呼び出します :

```
g_doc.p_api->statusClear(g_doc.p_ctrl)
```

8) DOC インスタンスをクローズするには、以下を呼び出します :

```
g_doc.p_api->close(g_doc.p_ctrl)
```

シンプルな DOC アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。

4.2.10.5 DOC ドライバの制限事項

DOC インタフェースとその実装の制限事項については、SSP のリリース ノートを参照してください。

4.2.10.6 DOC DOC ドライバでサポートされるデバイス

DOC ドライバは、S7G2 と S3A7 でテストされています。

4.2.10.7 DOC ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL DOC Interface API	synergy/ssp/inc/driver/api/r_doc_api.h
DOC Instance	synergy/ssp/inc/driver/instances/r_doc.h
DOC Driver	synergy/ssp/src/driver/r_doc

4.2.11 ELC ドライバ

r_elc に実装された ELC ドライバは、イベント要求を使用してさまざまな周辺機器を CPU の介入なしに連結する汎用 API です。

e2 studio ISDE のプロジェクト コンフィギュレータで、ELC ドライバ モジュールはデフォルトですべてのプロジェクトに追加されています。ELC ドライバを構成するには、[Threads] タブの [Modules] ペインでこれを選択します。詳細については、以下を参照してください : e2 studio ISDE による ELC ドライバを使用するアプリケーションの作成

ELC ドライバの API リファレンスは、次の HAL ELC インタフェースの説明内に記載されています : ELC インタフェース。

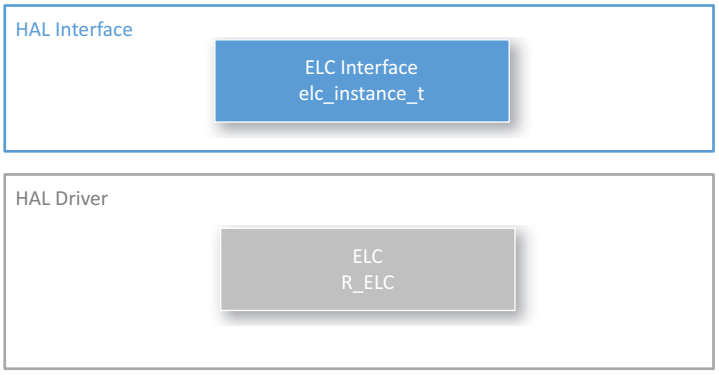

図 138: ELC - ブロック図

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

4.2.11.1 ELC ドライバの機能

ELC ドライバは、以下の機能をサポートしています。

- 2 つのブロック間にイベント リンクを作成します。
- その 2 つのブロック間のイベント リンクを切断します。
- CPU に割り込む 2 つのソフトウェア イベントのうち 1 つを生成します。

4.2.11.2 e2 studio ISDE による ELC ドライバを使用するアプリケーションの作成

ドライバは、e2 studio ISDE の SSP に組み込まれています （e2 studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

ISDE ではプロジェクト構成の際、すべてのプロジェクトに ELC ドライバが自動的に追加されます。

ELC ドライバ用のクロックの設定

ELC ブロックにはクロック設定はありません。

ELC 割り込みの設定

ELC ブロックは割り込みを生成しません。

ELC のコールバック ISR 関数の定義

コールバックはありません。

ELC パラメータの設定

設定する ELC パラメータはありません。

4.2.11.3 ELC アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ELC の例

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const elc_instance_t  g_elc =

{

    .p_ctrl = &g_elc_ctrl,

    .p_cfg  = &g_elc_cfg,

    .p_api = &g_elc_on_elc

};
```

ELC ドライバを使用して ELC アプリケーションを作成するには、以下の手順を実行します。

1) ELC の選択した ELC モジュール g_elc ELC ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが src/ssp_gen フォルダ内に自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

   I :ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) ELC インスタンスをオープンします。
4) ELC ドライバは、次の ELC インタフェースを通じて呼び出されます。

```
g_elc.p_api->init(g_elc.p_cfg)
```

   g_elc.p_cfg は、ELC の設定ステップの後で自動生成されます。

5) 以下を呼び出すことで、1 つのイベント リンクを作成します。

```
g_elc.p_api->linkSet(peripheral, signal)
```

   この関数呼び出しでは次のパラメータが使用されます。

   - peripheral: イベント信号を受信する周辺機器ブロック。
   - signal: イベント信号。

6) 以下を呼び出して、ソフトウェア イベントを生成します。

```
g_elc.p_api->softwareEventGenerate(event_num)
```

この関数呼び出しでは次のパラメータが使用されます。event_num: 生成されるソフトウェア イベント番号。

シンプルな ELC アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。

4.2.11.4 ELC の制限事項

ELC ドライバのその他の制限事項については、SSP のリリース ノートを参照してください。

4.2.11.5 ELC HAL ドライバ ファイル

プロジェクト設定中に、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL ELC Interface API	synergy/ssp/inc/driver/api/r_elc_api.h
ELC Instance	synergy/ssp/inc/driver/instances/r_elc.h
ELC Driver	synergy/ssp/src/driver/r_elc

4.2.11.6 ELC ドライバでサポートされるデバイス

このドライバは、ELC ペリフェラル ブロックを使用して、S7G2 および S3A7 でテストされています。

4.2.12 外部 IRQ ドライバ

外部 IRQ ドライバは、プッシュボタンその他の外部割り込みを使用するアプリケーション用に外部割り込みをプログラミングするための汎用 API です。r_icu 上に実装されたドライバは、MCU 上で利用可能な割り込みコントローラ ユニット （ICU）周辺機能をサポートしています。このセクションでは、e^2 studio ISDE を使用して外部 IRQ ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [入力] > [r_icu 上の外部 IRQ ドライバ] を選択することで、外部 IRQ ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による外部 IRQ ドライバを使用するアプリケーションの作成

外部 IRQ ドライバの API リファレンスは、次の外部 IRQ インタフェースの説明内に記載されています : 外部 IRQ インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

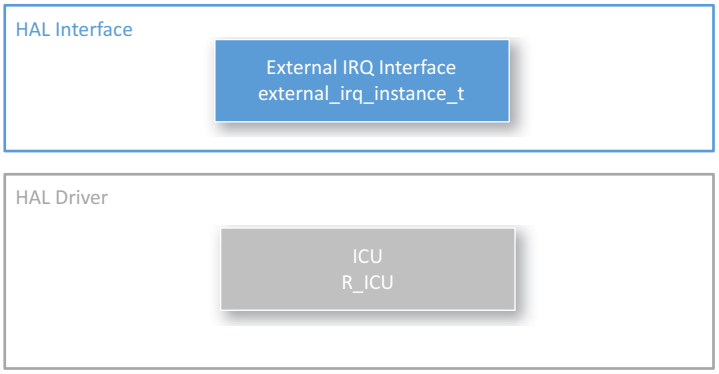

図 139: 外部 IRQ ドライバ - ブロック図

4.2.12.1 外部 IRQ ドライバの機能

このドライバは、ICU （割り込みコントローラ ユニット）の外部 IRQ 入力を設定します。このドライバは、外部 IRQ 入力の以下の機能をサポートしています。

- 割り込みの生成の有効化と無効化。
- IRQ イベントが発生したときのコールバックの呼び出し
- IRQ ノイズ フィルタの有効化と無効化

4.2.12.2 e² studio ISDE による外部 IRQ ドライバを使用するアプリケーションの作成

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

このドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

外部 IRQ ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
HAL External IRQ Driver	Threads	[Driver] > [入力] > [r_icu 上の外部 IRQ ドライバ]。[Properties] ウィンドウで [チャネル] プロパティを変更することにより、外部割り込みピンを選択できます。
Interrupts	Threads	[ポート] > [PORTn] > [PORTn IRQ]（n = 選択したチャネル）

アプリケーションの外部割り込みでデータ転送またはイベント トリガーが必要な場合に、次のリソースはオプションです。

リソース	ISDE タブ	選択
Data Transfer Controller Driver	Threads	DTC イベント PORT0 IRQ の g_transfer 転送モジュール
DMA Controller	Threads	DMAC イベント PORT0 IRQ の g_transfer 転送モジュール
Event Link Controller	Threads	ISDE には、デフォルトで ELC の 1 つのインスタンスが含まれます。

外部割り込みドライバ用のクロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して外部割り込み用のクロックを設定します （クロックの設定を参照）。

ICU は、誤ったトリガーを最小化するために、外部 IRQ 入力上のノイズをフィルタで除去するためのデジタル フィルタを備えています。ノイズ フィルタは PCLKB を使用して動作します。

PCLKB 周波数を設定するには、実行時に e^2 studio ISDE クロックの設定または CGC インタフェースのクロック コンフィギュレータを使用します。

ノイズ フィルタ用の ICU の除算器は、最大 PCLKB/64 です。IRQ ピンは、この速度でサンプリングされ、設定されているイベント （ローレベルなど）が 3 つの連続するサンプルで有効な場合、信号が受け入れられ、IRQ がトリガーされます。

外部割り込みピンの設定

e^2 studio ISDE で、[Pins] タブを使用して外部割り込みピンを設定します （ピンの設定を参照）。

外部 IRQ ピンとして使用する I/O ポート ピンは、ピン コンフィギュレータで IRQ ピンとして設定されている必要があります。e^2 studio ISDE により、関連するピンが pin_cfg フィールドで適切に設定されます。

ICU 割り込みの設定

e^2 studio ISDE を使用して、[ICU] タブから外部割り込みを設定します （割り込みの設定を参照）。

以下の状況では、BSP で選択した使用チャネルに対する ICU IRQ 割り込みを有効にする必要があります。

設定されたイベントが IRQ で発生したときに割り込みを取得するため

I ：割り込みを有効にするには、e^2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、ICU PORTn 割り込み（n は IRQ チャネル番号）の優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_PORTn_IRQ に、選択した優先度が設定されます。

ICU IRQn 割り込みが BSP で有効になっている場合、対応する ISR が EXTERNAL_IRQ ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

I a:irq が TRANSFER_IRQ_END に設定された DTC 周辺機器と共に使用した場合、割り込みはスキップされることがあります。

External_IRQ パラメータの設定

e^2 studio ISDE を使用して、HAL 外部 IRQ ドライバのパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

ICU の設定

ISDE プロパティ	設定	設定値	説明
Channel	channel	0-15 for S7G2, 0-5 for S3A7, 0-7 for S124	使用するハードウェア IRQ チャネルを指定します。
Trigger condition	trigger	external_irq_trigger_t	-
Digital Filtering	filter_enable	true/false	デジタル フィルタ有効 / 無効。
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	pclk_div	external_irq_pclk_div_t	ノイズ フィルタ サンプリング期間を設定します。
Interrupt enabled after initialization	autostart	true, false	-

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、IRQn がトリガーされるたびに割り込みサービス ルーチン （ISR）から呼び出されます。 I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
-	p_context	User-defined	コールバックがユーザー定義データを受け取ることができるようにします。
-	p_extend	User-defined	ハードウェアに依存する設定。

4.2.12.3 外部 IRQ ドライバ使用上の注意

ICU を使用した DMAC/DTC のトリガー

設定した external_irq イベントが発生したときに、DMAC または DTC 周辺機器を使用してデータの転送をトリガするには、activation_source に ELC_EVENT_PORTn_IRQ （n は IRQ チャネル番号）を設定して DMAC/DTC 転送を設定します。

ICU を使用した ELC イベントのトリガー

ICU は、elc_peripheral_t にリストされている他の周辺機器の起動をトリガーできます。

4.2.12.4 外部 IRQ ドライバ アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const external_irq_instance_t  g_external_irq =

{

    .p_ctrl  = &g_external_irq_ctrl,

    .p_cfg = &g_external_irq_cfg,

    .p_api = &g_external_irq_on_external_irq

};
```

ICU を使用して External_IRQ アプリケーションを作成するには、以下の手順を実行します。

1) 上記の説明に従い、構造体 external_irq_cfg_t を設定してモジュールを構成します。モジュールを構成すると、ISDE により、関連するヘッダーと設定ファイルが src/ssp_gen フォルダ内に自動生成されます。

2) ICU として実装された external_irq を使用して、external_irq インスタンスをオープンします。ICU ドライバは、external_irq インタフェースを通じて呼び出されます。

```
g_external_irq.p_api->open(g_external_irq.p_ctrl, g_external_irq.p_cfg)
```

構造体 g_external_irq.p_ctrl と g_external_irq.p_cfg は、external_irq 設定ステップの後に自動生成されます。

3) IRQ が open 呼び出しの中で有効化されていない場合は、下記コードを通じて有効化できます。

```
g_external_irq.p_api->enable(g_external_irq.p_ctrl)
```

4) ノイズ フィルタリングは、必要に応じて有効化および無効化できます。リセット後に較正が必要な場合は、この関数の後、アプリケーションから

```
g_external_irq.p_api->filterEnable(g_external_irq.p_ctrl)
```

または

```
g_external_irq.p_api->filterDisable(g_external_irq.p_ctrl)
```

5) トリガー条件は必要に応じて変更できますが、誤ったインベントを避けるために、ドライバのインスタンスをクローズしてから行うことをお勧めします。以下を使用してください。

```
g_external_irq.p_api->triggerSet(g_external_irq.p_ctrl)
```

6) 次のインタフェースを呼び出して、external_irq インスタンスをクローズします。以下を使用してください。

```
g_external_irq.p_api->close(g_external_irq.p_ctrl)
```

4.2.12.5 外部 IRQ の制限事項

外部 IRQ ドライバとその実装の制限事項については、SSP のリリース ノートを参照してください。

4.2.12.6 外部 IRQ ドライバ ファイル

プロジェクト設定中に、ISDE により次の表に記載されているファイルが ssp ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL External IRQ Interface API	synergy/ssp/inc/driver/api/r_external_irq_api.h
ICU Instance	synergy/inc/driver/instances/r_icu.h
ICU Driver	synergy/src/driver/r_icu

4.2.12.7 外部割り込みドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

外部 IRQ ドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.13 フラッシュ ドライバ

フラッシュ ドライバには、MCU 内に存在するデータおよび ROM フラッシュ （ユーザー ROM）の両方を読み取る、書き込む、および消去するための API が含まれます。これに加え、より高度な関数も使用可能です。

S3A7 と S7G2 は異なるフラッシュ技術を使用します。この結果、フラッシュ ドライバが異なる 2 つのフラッシュ モジュール （r_flash_hp と r_flash_lp）に実装されます。高性能フラッシュ モジュール （Flash_HP）は S7G2 部分をプログラミングするために使用される API です。ローパワー フラッシュ モジュール （FLASH _LP）は S3A7 をプログラミングするために使用される API です。この 2 つを相互に交換することはできません。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [ストレージ] > [r_flash_hp 上のフラッシュ ドライバ]、または [新規] > [Driver] > [ストレージ] > [r_flash_lp 上のフラッシュ ドライバ] を選択することで、フラッシュ ドライバ モジュールを追加および構成

できます。詳細については、以下を参照してください：e² studio ISDE によるフラッシュ ドライバを使用するアプリケーションの作成

フラッシュ ドライバ API は、以下で入手可能なフラッシュ インタフェースの説明内に記載されています：フラッシュインタフェース。

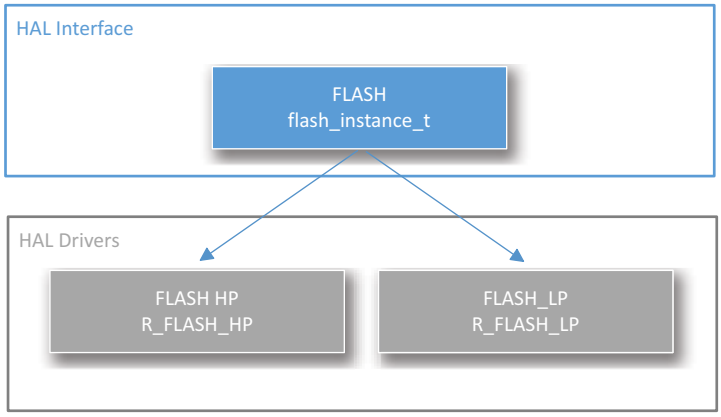

図 140: フラッシュ ドライバ - ブロック図

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

4.2.13.1 フラッシュ ドライバ関連の用語

BSP 関連の用語	意味
Filename_xxxx.c	'xxxx' は MCU の種類を表します。たとえば、S7G2 を参照する場合は Filename_S7G2.c になります。
HLD	ハイレベルドライバ
LLD	ローレベルドライバ
Callback Function	この用語はイベント発生時に呼び出される関数を意味します。フラッシュの場合、これは、バックグラウンド（BGO）モードで実行されるデータ フラッシュ操作が完了したときに呼び出されるユーザー定義関数です。コールバック構造体は、コールバックの原因となるイベント（消去完了、書き込み完了など）を提供するコールバックで渡されます。

4.2.13.2 フラッシュ ドライバの機能

フラッシュ API は、アプリケーションが MCU 内に存在するデータおよび ROM フラッシュ エリアの両方を読み取り、書き込み、および消去することを可能にします。使用可能なフラッシュ メモリの量は MCU パーツごとに異なりますが、API を通じて使用可能な機能は下記のとおりです。

- ROM フラッシュのブロック化消去、読み取り、書き込みおよびブランク チェック。
- データおよびコード フラッシュのブロック化および非ブロック化消去、読み取り、書き込みおよびブランク チェック。
- 非ブロック化データ フラッシュ操作の完了時に使用されるコールバック関数。
- コード フラッシュの指定されたエリアだけを消去または書き込み可能にする、ROM フラッシュ用のアクセス ウィンドウ （書き込み保護）。
- スタートアップ プログラムを最初に消去することなく安全に再書き込みできるようにする、ブート ブロック スワッピング用のスワップ エリア。

4.2.13.3 e^2 studio ISDE によるフラッシュ ドライバを使用するアプリケーションの作成

ドライバは、e^2 studio ISDE に組み込まれています （e^2 studio ISDE ユーザーガイドを参照）。

e^2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

Flash_LP によってフラッシュ ドライバを使用するアプリケーションでは、次のリソースが使用です。

リソース	ISDE タブ	選択
Flash Driver	Threads	[Driver] > [Storage] > [r_flash_lp 上のフラッシュドライバ]
Interrupts	Threads	[FCU] > [FCU FRDYI] （フラッシュ対応割り込み）

Flash_HP によってフラッシュ ドライバを使用するアプリケーションでは、次のリソースが使用です。

リソース	ISDE タブ	選択
Flash Driver	Threads	[Driver] > [Storage] > [r_flash_hp 上のフラッシュドライバ]

リソース	ISDE タブ	選択
Interrupts	Threads	[FCU] > [FCU FRDYI] （フラッシュ対応割り込み）
Interrupts	Threads	[FCU] > [FCU FIFERR]（フラッシュエラー割り込み）

フラッシュ対応割り込みの設定

e2 studio ISDE を使用して、[ICU] タブからフラッシュ対応割り込みを構成します （割り込みの設定を参照）。

フラッシュ対応割り込みを有効にする必要があるのは、ユーザーがデータ フラッシュ BGO （バックグラウンド モード）操作を使用する予定のある場合だけです。このモードでは、アプリケーションはデータ フラッシュ操作を開始した後、ユーザー提供のコールバック関数を使用してその完了またはエラーの通知を非同期的に受けることができます。コールバック関数には、コールバック イベント（FLASH_EVENT_ERASE_COMPLETE）のソースを示すイベント情報が含まれる構造体が渡されます。

割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、FCU > FRDYI 割り込みの優先度を設定します。これにより、synergy_cfg/ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_FCU_FRDYI に、選択した優先度が設定されます。

BSP で FLASH FRDYI 割り込みが有効になっている場合、対応する ISR がフラッシュ ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

Note：フラッシュ HP は追加のフラッシュ エラー割り込みをサポートしており、FLASH HP で BGO モードが有効になっている場合は、FRDYI および FIFERR 割り込みの両方に優先度を与える必要があります。

フラッシュ ドライバ パラメータの設定

フラッシュ API ではビルド時および実行時の構成設定の両方を使用します。ビルド時構成設定は、パラメータ チェックとコード フラッシュ （ROM）プログラミングの有効化 / 無効化の両方で使用できます。これらの機能のいずれかまたは両方を無効化することにより、フラッシュ API の全体的な ROM フットプリントを小さくできます。フラッシュのビルド時構成は、ユーザーのフラッシュ設定に基づいて ISDE によって生成される r_flash_cfg.h によって制御されます。

e2 studio ISDE を使用して、フラッシュ ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

フラッシュ ドライバの設定

ISDE プロパティ	設定	設定値	説明
Parameter Checking	FLASH_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE、有効または無効	API パラメータ チェック用のコードを含めるかどうかを制御します。
Code Flash Programming Enable	FLASH_CFG_PARAM_CODE_FLASH_PROGRAMMING_ENABLE	Enabled or Disabled	コード フラッシュ プログラミングを有効にするかどうかを制御します。無効化すると、API が使用する ROM の量を小さくできます。
Data Flash Background Operation	data_flash_bgo	Enabled or Disabled	有効にすると、データ フラッシュを参照するフラッシュ API 呼び出しが、操作をバックグラウンドで継続させて即座に戻ることが可能になります。
Callback	p_callback	NULL or name of user supplied callback function	データ フラッシュ BGO 操作が完了またはエラーになったときに呼び出されるコールバック関数。ユーザー コールバック関数を open で登録できます。Attention : コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないでください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

Note : どのような構成エラーも捕捉できるように、パラメータ チェックを有効にした状態でアプリケーションを開発することをお勧めします。実稼働コードではパラメータ チェックを無効化して、コード スペースと実行時間を節約できます。

4.2.13.4 フラッシュ ドライバ使用上の注意

フラッシュ API により、オンチップ フラッシュ エリアのプログラミングと消去のプロセスをより容易に実行できます。コード (ユーザー ROM) およびデータ フラッシュ エリアの両方がサポートされています。この API を最も単純な形式で使用して、ブロック化消去およびプログラム操作を実行できます。" ブロック化 " という用語は、プログラムまたは消去関数が呼び出されたときに、操作が完了するまで関数が戻らないことを意味します。この API はコードとデータ フラッシュの両方のブロック化をサポートしていますが、BGO 操作はデータ フラッシュ操作でのみ使用できます。コード フラッシュ操作が実行中のとき、ユーザー

がコード フラッシュ エリアにアクセスすることはできません。コード フラッシュ操作が進行中のときにコード フラッシュ エリアにアクセスしようとすると、フラッシュ コントロール ユニットがエラー状態に遷移します。

データ フラッシュ BGO の使用上の注意

データ フラッシュ BGO を使用しているとき、ユーザー ROM、RAM、および外部メモリに引き続きアクセスできます。これは、データ フラッシュ操作の間、データ フラッシュにアクセスしないことだけに気を付ける必要があることを意味します。これには、データ フラッシュにアクセスする可能性のある割り込みが含まれます。

コード フラッシュの使用上の注意

BGO モードはコード フラッシュでサポートされていないため、コード フラッシュ操作が完了する前に戻ることを心配する必要はありません。ただし、ベクトル テーブルはデフォルトでユーザー ROM （コード フラッシュ）内に存在します。ROM 操作中に割り込みが発生すると、割り込みの開始アドレスをフェッチするために ROM へのアクセスが行われ、エラーが発生します。

最も単純な回避策は、コード フラッシュ操作中の割り込みを無効化することです。別のオプションは、ベクトル テーブルを RAM にコピーし、それに応じて VTOR （ベクトル テーブル オフセット レジスタ）を更新し、割り込みサービス ルーチンが RAM 外で実行されるようにすることです。

フラッシュ クロック （FCLK）

フラッシュ回路は FCLK をそのクロックとして使用します。FCLK は <= 4 MHz である必要があり、このクロック レートが Flash Open() 関数を呼び出した後で変化した場合、updateFlashClockFreq() を呼び出してフラッシュ API にその変化を伝える必要があります。

ブランク チェック

コードまたはデータ フラッシュの内容がブランクかどうかをチェックするための API 関数が存在します。フラッシュ （コードまたはデータ）には、最初に消去しない限り書き込みができないことに注意してください。BlankCheck 関数は、指定されたエリアがブランクで書き込み可能かどうかを判定します。ほとんどすべての場合、フラッシュの内容を 0xFF と比較するだけではエリアがブランクかどうかを判定するのに十分ではないことに注意してください。唯一の例外はフラッシュ HP コード フラッシュです。フラッシュ HP コード フラッシュ内の 0xFF は確実にブランクを示します。Renesas は、すべての場合に BlankCheck API 関数を使用することを強くお勧めします。

フラッシュ ステータス

アプリケーションがフラッシュの " 準備完了 " ステータスをクエリできるようにする API 関数 （statusGet）が存在します。これは、ユーザーがコールバック関数を使用しないことを選択したためにデータ フラッシュ操作の完了が非同期的に通知されないデータ フラッシュ BGO 操作で有用です。この場合、データ フラッシュは BGO モードで動作するように構成されているため、操作 （たとえば消去）が開始されると、呼び出しは即座に戻り、操作はバックグラウンドで実行されます。statusGet() 関数を呼び出すことにより、ユーザーは、操作が安全に完了したかエラーを生成し、別のフラッシュ操作に安全に移行できるようになったことを判定できます。

ブロックのスワップ

スタートアップ エリア ブロックとして使用されるブロック （デフォルト （ブロック 0）または代替 （ブロック 1））をユーザーが選択することを可能にする API 関数 （startupAreaSelect）。提供されているパラ

メータは、どのブロックがアクティブ スタートアップ ブロックになり、そのアクションが次のリセットの後にすぐに （しかし一時的に）実行されるのか、恒常的に実行されるのかを決定します。

一時的な切り替えのメリットは限定的のようです。しかし、ブロック 0 が書き込み保護となるようなアクセス ウィンドウがある場合、アクセス ウィンドウに触らずに、一時的な切り替えを行い、ブロックを更新して、元に切り替えることもできます。

4.2.13.5 フラッシュ ドライバ アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module (Flash_LP). */

const flash_instance_t  g_flash_lp =

{ .p_ctrl  = &g_flash_lp_ctrl, .p_cfg = &g_flash_lp_cfg, .p_api = &g_flash_on_flash_lp };
```

Flash_LP を使用するフラッシュ アプリケーションを作成するには、次の手順に従います （手順は Flash_HP の場合と類似しています）。

1) モジュールを上の説明のように設定します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます （以下の構造体が含まれます : flash_cfg_t および flash_ctrl_t、flash_instance_t）。

2) 設定時にフラッシュに付けた名前を使用して、フラッシュ インスタンスをオープンします。フラッシュ インタフェースを通じて適切なドライバが呼び出されます。g_flash_lp というフラッシュ インタフェースの場合は以下のようになります。

```
g_flash_lp.p_api->open(g_flash_lp.p_ctrl, g_flash_lp.p_cfg)
```

フラッシュの設定手順後に、g_flash_lp.p_ctrl と g_flash_lp.p_cfg が自動生成されます。

3) 次を使用してデータ フラッシュ ブロックを消去します : #define FLASH_DF_BLOCK_0 0x40100000

```
g_flash_lp.p_api->erase(g_flash_lp.p_ctrl, FLASH_DF_BLOCK_0, 1)
```

4) 次のようにしてデータ フラッシュ ブロック 0 に 64 バイトを書き込みます。

```
g_flash_lp.p_api->write(g_flash_lp.p_ctrl, write_buffer, FLASH_DF_BLOCK_0, 64)
```

5) 次のようにしてブロックをブランク チェックします。

```
g_flash_lp.p_api->blankCheck(g_flash_lp.p_ctrl, FLASH_DF_BLOCK_0, FLASH_DATA_BLOCK_SZ, &blank_check_result
)
```

6) データ フラッシュ ブロック 0 から 64 バイトを読み取ります。

```
g_flash_lp.p_api->read(g_flash_lp.p_ctrl, read_buffer, FLASH_DF_BLOCK_0, 64)
```

7) コード フラッシュ ブロック 40 ～ 79 だけが書き込み可能であるアクセス ウィンドウを配置します。

```
g_flash_lp.p_api->accessWindowSet(g_flash_lp.p_ctrl, FLASH_CF_BLOCK_40, FLASH_CF_BLOCK_80)
```

8) 構成された任意のアクセス ウィンドウを削除します。

```
g_flash_lp.p_api->accessWindowClear(g_flash_lp.p_ctrl)
```

9) スタートアップ エリアを一時的にブロック 1 に切り替えます。

```
g_flash_lp.p_api->startupAreaSelect(g_flash_lp.p_ctrl, FLASH_STARTUP_AREA_BLOCK1, true)
```

4.2.13.6 フラッシュ ドライバでサポートされるデバイスとボード

- ローパワー フラッシュ API を任意の S3A7 ベースのボードで使用します。
- 高性能フラッシュ API を、任意の S7G2 ベースのボードで使用します。

4.2.14 FMI ドライバ

4.2.14.1 FMI ドライバ

FMI ドライバは r_fmi 上に実装されており、Factory MCU Information Flash Table からレコードを読み取るための汎用 API を含みます。このセクションでは、e^2 studio ISDE を使用して FMI ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [System] > [FMI Driver on r_fmi] を選択することで、FMI ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による FMI ドライバを使用するアプリケーションの作成

API リファレンスは、次の FMI インタフェースの説明内に記載されています : FMI インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

4.2.14.2 FMI ドライバの機能

このドライバは、MCU の FMIFRT （Factory MCU Information Flash Root Table）を読み取って、フラッシュ内のテーブルの先頭アドレスを取得します。このドライバにより、テーブルの製品情報レコードへの呼び出し側のポインタが設定されます。

4.2.14.3 e^2 studio ISDE による FMI ドライバを使用するアプリケーションの作成

ドライバは、e^2 studio ISDE の SSP に組み込まれています （e^2 studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

FMI ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
FMI Driver	Threads	r_fmi 上の FMI ドライバ

4.2.14.4 FMI ドライバ アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const fmi_instance_t  g_fmi =

{

    .p_api  = &g_FMI_on_FMI

};
```

FMI ドライバを使用して FMI アプリケーションを作成するには、以下の手順を実行します。

1) 選択した FMI モジュール g_fmi FMI ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが src/ssp_gen フォルダ内に自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

   Note：ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。
3) FMI インスタンスをオープンします。この FMI ドライバは、次の FMI インタフェースを通じて呼び出されます。

```
g_fmi.p_api->productInfoGet(fmi_product_info_t **pp_product_info);
```

シンプルな FMI アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src/ ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。

4.2.14.5 FMI ドライバの制限事項

FMI HAL インタフェースとその実装の制限事項については、SSP のリリース ノートを参照してください。

4.2.14.6 FMI ドライバでサポートされるデバイス

FMI ドライバは、FMIFRT 周辺レジスタを使用して S7G2（WS2）でテストされています。これは、現在その Factory MCU Information Table 内のデータを使用してプログラムされている MCU は S7G2 （WS2）のみであるためです。

4.2.14.7 FMI ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL FMI Interface API	synergy/ssp/inc/driver/api/r_fmi_api.h
FMI Instance	synergy/ssp/inc/driver/instances/r_fmi.h
FMI Driver	synergy/ssp/src/driver/r_fmi

4.2.15 I²C ドライバ

I²C ドライバは、I²C プロトコルを使用した通信用の汎用 API です。このドライバは、MCU で利用可能な IIC および SCI 周辺機器をサポートするため、r_riic と r_sci_i2c に実装されています。このセクションでは、e² studio ISDE を使用して I²C ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e² studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Connectivity] > [r_riic 上の I²C ドライバ] または [New] > [Driver] > [Connectivity] > [r_sci_i2c 上の I²C ドライバ] を選択することで、I²C ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e² studio ISDE による I2C ドライバを使用するアプリケーションの作成。

API リファレンスは、次の I²C インタフェースの説明内に記載されています : I2C インタフェース。

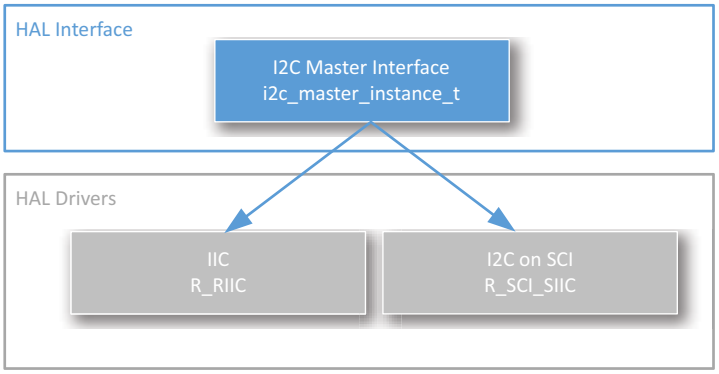

図 141: I²C ドライバ - ブロック図

4.2.15.1 I²C ドライバの機能

I²C ドライバは、マスター モードの I²C 通信を設定します。このドライバでは、以下のことが可能です。

- ドライバの初期化。
- スレーブ デバイスからの読み取り。
- スレーブ デバイスへの書き込み。
- I²C 周辺機器のリセット。

ドライバは、コールバックへのサポートも提供します。コールバック関数は、次のイベント i2c_event_t で呼び出されます。

- 転送中断
- 転送完了
- 受信完了

コールバック構造体 i2c_callback_args_t には、送信または受信したバイト数も設定されます。

4.2.15.2 I²C モジュールの選択

両方の I²C モジュールが、最大ビット レート 400 kHz の I²C 高速モードをサポートしています。IIC 周辺機器と RIIC HAL ドライバは、ビット レートが 1 MHz の後続モード プラスをサポートしています。このドライバは、両方の実装について、マスター モードのみをサポートしています。

4.2.15.3 e² studio ISDE による I²C ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

参考資料

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

I2C と SCI モジュールを使用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
SCI Driver for I2C on SCI	Threads	新しいスレッドまたは HAL/ 共通をハイライトして、[New] > [Driver] > [Connectivity] > [I2C ドライバ r_sci_i2c] の順に選択します。
SCI Common driver	Threads	SCI 共通モジュールがスレッドに自動的に追加されます。このドライバの [Properties] ウィンドウで簡易 I2C モードを有効化します。
Interrupts	Threads	選択したモジュールのプロパティから、選択したチャネルに対し SCIn RXI および SCIn TXI、SCIn TEI、SCIn ERI 割り込みを設定します。

I2C フレームワークと IIC モジュールを使用する場合は、以下のリソースを追加します。

リソース	ISDE タブ	選択
IIC driver RIIC	Threads	新しいスレッドまたは HAL/ 共通をハイライトして、[新規] > [Driver] > [Connectivity] > [I2C ドライバ r_riic] の順に選択します。
Interrupts	Threads	選択したモジュールのプロパティから、選択したチャネルの RIICn EEI、RIICn RXI、RIIC TXI** および **RIIC TEI 割り込みを有効化します。

アプリケーションの I2C 割り込みでデータ転送またはイベント トリガーが必要な場合に、次のリソースはオプションです。IIC モジュールについては、DMA/DTC 転送はサポートされていません。

リソース	ISDE タブ	選択
Data Transfer Controller Driver	Threads	新しいスレッドまたは HAL/ 共通をハイライトして、[新規] > [Driver] > [転送] > [r_dtc 上の転送ドライバ] の順に選択します。
DMA Controller	Threads	新しいスレッドまたは HAL/ 共通をハイライトして、[新規] > [Driver] > [転送] > [r_dmac 上の転送ドライバ] の順に選択します。
Event Link Controller	Threads	ISDE には、デフォルトで ELC の 1 つのインスタンスが含まれます。

I²C ドライバ用の IIC クロックの設定

e² studio ISDE で、[Clocks] タブを使用して IIC クロックを設定します （クロックの設定を参照）。

IIC は PCLKB をそのクロック ソースとして使用します。PCLKB 周波数を設定するには、実行時に e² studio ISDE または CGC インタフェースのクロック コンフィギュレータを使用します。

実際の I²C 転送レート i2c_rate_t は、選択された転送レートに応じて、ドライバにより内部で計算および設定されます。選択された内部レートを実現できないような設定が PCLKB で行われた場合は、ドライバを初期化するときにエラーが返されます。

I²C ドライバ用の IIC ピンの設定

e² studio ISDE を使用して、[Pins] タブから IIC ピンを設定します （ピンの設定を参照）。

IIC との出力比較を使用するには、出力ピンとして使用する I/O ポート ピンが、ピン コンフィギュレータで IIC 周辺機器ピンとして設定されている必要があります。

これにより、pin_cfg フィールドで関連するピンのピン構成が適切に設定されます。

I²C ドライバの IIC 割り込みの設定

e² studio ISDE を使用して、[Threads] タブから IIC 割り込みを設定します （割り込みの設定を参照）。

選択した使用チャネルの RIIC エラー （EEI）、受信バッファ フル （RXI）、送信バッファ エンプティ （TXI）、および送信完了 （TEI）割り込みは、ユーザーがコールバックを使用するかどうかにかかわらず、BSP で有効にする必要があります。

I :e² studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、そのチャネルのすべての割り込みの優先度を同じレベルに設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_RIICn_RXI、BSP_IRQ_CFG_RIICn_TEI、BSP_IRQ_CFG_RIICn_TXI、BSP_IRQ_CFG_RIICn_EEI に、選択した優先度が設定されます。

割り込みが BSP で有効になっている場合、対応する ISR が RIIC ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

I a: 割り込みを異なる優先度に設定すると、適切に動作しなくなる可能性があります。

IIC 周辺機器の I²C ドライバの設定

e² studio ISDE を使用して、IIC 周辺機器の I²C ドライバのパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定。

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

I : すべてのパラメータは、I²C ドライバ構成構造体 i2c_cfg_t に設定されます。

IIC 周辺機器の I²C ドライバ構成

ISDE プロパティ	設定	設定値	説明
Channel	channel	0、1、および 2	この設定で使用する IIC チャネルを指定します。
Rate	rate	i2c_rate_t	IIC レート計算を参照してください。
Slave Address	slave	ユーザー定義	I²C マスターが通信するスレーブ デバイスのアドレスを設定します。7 ビット アドレスと 10 ビット アドレスの両方がサポートされています。
Address Mode	addr_mode	i2c_addr_mode_t	-

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	ユーザー定義	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、i2c_event_t で定義されている条件のそれぞれに対し、割り込みサービス ルーチン （ISR）から呼び出されます。 I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
-	p_context	ユーザー定義	-
-	p_extend	-	-

I²C ドライバ用の I²C モードの SCI クロックの設定

e² studio ISDE で、[Clocks] タブを使用して SCI クロックを設定します （クロックの設定を参照）。

I²C ドライバ用の I²C モードの SCI ピンの設定

e² studio ISDE を使用して、[Pins] タブから IIC ピンを設定します （ピンの設定を参照）。

I²C ドライバ用の I²C モードの SCI 割り込みの設定

e² studio ISDE を使用して、[Threads] タブから IIC 割り込みを設定します （割り込みの設定を参照）。

SCI I²C の割り込みを有効にするには、SCI > SCIn RXI、TXI、TEI および ERI 割り込み （n は SCI チャネル番号を意味します）の優先度を選択します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

I²C モード （SIIC）の SCI 周辺機器の I²C ドライバの設定

I：すべてのパラメータは、I²C ドライバ構成構造体 i2c_cfg_t に設定されます。

SCI 上の I²C の設定

ISDE プロパティ	設定	設定値	説明
Channel	channel	0 ～ 9	この設定で使用する SCI チャネルを指定します。SCI は以下のチャネルを持っています : Series S7 : 0 1 2 3 4 5 6 7 8 9; Series S3 : 0 1 2 3 4 - - - - 9; Series S1 : 0 1 - - - - - - - 9
Rate	rate	i2c_rate_t	標準、高速および高速プラス。
Slave Address	slave	User defined	スレーブ デバイスのアドレス。
Address Mode	addr_mode	i2c_addr_mode_t	7 ビットまたは 10 ビット アドレシング モード。
Callback	p_callback	User defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、i2c_event_t で定義されている条件のそれぞれに対し、割り込みサービス ルーチン （ISR）から呼び出されます。Attention : コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないでください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
-	p_context	ユーザー定義	-
-	p_extend	-	-

4.2.15.4 I²C ドライバ使用上の注意

IIC レート計算

RIIC ドライバは、内部ボー レート設定を、要求された転送レート i2c_rate_t （i2c_cfg_t に設定され、open に渡されます）に基づいて計算します。

現在の PCLKB 設定で達成可能な最も近いボー レート （要求されたレートより小さいか等しいもの）が計算されて使用されます。

有効なクロック レートを計算できなかった場合は、エラーが返されます。

IIC を使用した DMAC/DTC のトリガ

DMAC/DTC を IIC モジュールとともに使用しないでください。使用すると、内部の状態マシンが壊れる可能性があります。

IIC を使用した ELC イベントのトリガ

IIC は、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

4.2.15.5 IIC 周辺機器の I²C ドライバ アプリケーションの作成

ISDE によって生成されたヘッダー ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const i2c_master_instance_t  g_i2c =

{

 .p_ctrl  = &g_i2c_master_ctrl,

 .p_cfg = &g_i2c_master_cfg,

 .p_api = &g_i2c_master_on_riic

};
```

IIC を使用して I²C アプリケーションを作成するには、以下の手順を実行します。この

```
g_i2c.p_api->open()
```

関数は最初に呼び出す必要があります。残りの呼び出しは、アプリケーションの要件に応じて任意の順序で使用できます。

1) 構造体 i2c_cfg_t を設定してモジュールを構成します。モジュールを設定すると、モジュール関連のヘッダーと設定ファイルが自動的に生成されます。

2) IIC によって実装された I²C を使用して、I²C インスタンスをオープンします。IIC ドライバは、I²C インタフェースを通じて呼び出されます。

```
g_i2c.p_api->open (g_i2c.p_ctrl, g_i2c.p_cfg)
```

p_ctrl と p_cfg は、I²C の設定手順の後に自動生成される制御および構成構造体のインスタンスです。

3) 以下を呼び出して、スレーブ デバイスへの書き込みを初期化します

```
g_i2c.p_api->write(g_i2c.p_ctrl, &write_buffer,
num_bytes_to_write, false)
```

g_i2c.p_ctrl は、open 呼び出しで使用した同じ制御インスタンスです。最後の引数は、書き込み後に再開状態とするか （true に設定）、停止状態とするか （false に設定）を指定します。再開状態が一般に使用されるのは、マスターが、連続する読み取り / 書き込み呼び出しの間でバスを解放したくない場合です。

4) 以下を呼び出して、スレーブ デバイスからの読み取りを初期化します

```
g_i2c.p_api->read(g_i2c.p_ctrl,&write_buffer,
num_bytes_to_write, false)
```

g_i2c.p_ctrl は、open 呼び出しで使用した同じ制御インスタンスです。最後の引数は、読み取り後に再開状態とするか （true を設定）、停止状態とするか （false を設定）を指定します。

5) IIC 周辺機器をリセットする必要がある場合は、次を呼び出してリセットします。

```
g_i2c.p_api->reset(g_i2c.p_ctrl)
```

g_i2c.p_ctrl は、open 呼び出しで使用した同じ制御インスタンスです。

6) IIC チャネルをクローズするには、次を呼び出します。

```
g_i2c.p_api->close(&g_i2c, p_ctrl)
```

g_i2c.p_ctrl は、open 呼び出しで使用した同じ制御構造体です。

I：すべてのデバイスに対しクロック レートが同じに保たれる限り、I²C（SCI および IIC）HAL ドライバは、同一バス上の複数の I²C デバイスをサポートできます。つまり、クロック レートが同じであれば、同一バス内で複数のデバイスを開くことができます。デバイス間でクロック レートが異なる場合は、一度に開くことのできるデバイスは 1 つのみです。

4.2.15.6 I²C ドライバの制限事項

I²C インタフェースは、SCI 上の I²C と IIC 周辺機器の両方について、マスター モードのみを実装しています。スレーブ モードはサポートされていません。

4.2.15.7 I²C ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL I²C Interface API	synergy/ssp/inc/driver/api/r_i2c_api.h
I²C on SCI Instance	synergy/ssp/inc/driver/instances/r_sci_i2c.h
I²C on SCI Driver	synergy/ssp/src/driver/r_sci_i2c
IIC Instance	synergy/ssp/inc/driver/instances/r_riic.h
IIC Driver	synergy/ssp/src/driver/r_riic

4.2.15.8 I²C ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

I²C ドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.16 I2S ドライバ

I2S ドライバは r_ssi に実装されており、I2S プロトコルを使用したシリアル オーディオ通信用の汎用 API を含みます。このドライバは、MCU 上で I2S マスター モードにある SSI 周辺機器をサポートします。このセクションでは、e² studio ISDE を使用して I2S ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e² studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [Connectivity] > [r_ssi 上の I2S ドライバ] を選択することで、I2S ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e² studio ISDE による I2S ドライバを使用するアプリケーションの作成

API リファレンスは、次の I2S インタフェースの説明内に記載されています : UART インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

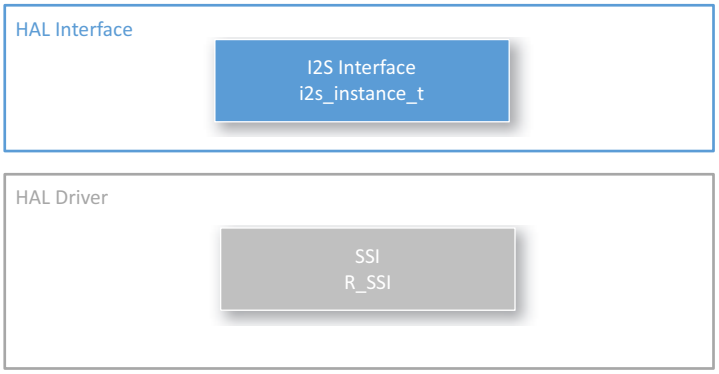

図 142: I2S ドライバ - ブロック図

4.2.16.1 I2S ドライバの機能

I2S ドライバは、汎用 I2S オーディオ シリアル通信プロトコルをサポートします。これは、マスター モードでの非圧縮オーディオ データの送信および受信に用いられます。

I2S マスター モードの SSI 周辺機器で使用される I2S ドライバは、標準的な I2S プロトコルに加えて、次の機能をサポートしています。

- 全二重 I2S 通信 （SSI チャネル 0 のみ）
- 割り込み駆動型のデータ送信と受信
- DTC 転送モジュールとの統合

4.2.16.2 e² studio ISDE による I2S ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

I2S と SCI モジュールを使用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
I2S Driver for I2S on SCI	Threads	[Driver] > [通信] > [r_ssi 上の I2S ドライバ]
(Recommended) DTC Transfer Module for Transmission/Reception	Threads	[Driver] > [転送] > [r_dtc 上の転送ドライバ]; 名前が送信 / 受信用の SSI プロパティ DTC 転送モジュールと一致していることを確認してください。
(Optional) GPT Timer Module Used to Generate Audio Clock	Threads	[Driver] > [転送] > [r_gpt 上のタイマドライバ]; 名前がオーディオ クロック作成に使用される SSI プロパティ GPT タイマ モジュールと一致していることを確認してください。
Interrupts	Threads	[SSI] > [SSIx RXI] および / または [SSIx TXI] 割り込みを、選択されたチャネルに対して設定します。[SSIx INT] 割り込みを、選択されたチャネルに対して設定します。

SSI での I2S ドライバ用のクロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して SSI クロック上の I2S を設定します （クロックの設定を参照）。

SCI モジュールは、次のクロックを使用します。

- PCLKB
- AUDIO_CLK ピンに対する外部クロック入力 （SSI での I2S ドライバ用のピンの設定を参照）

SSI での I2S ドライバ用のピンの設定

e^2 studio ISDE を使用して、[Pins] タブを使用して SSI ピン上の I2S を設定します （ピンの設定を参照）。

選択したチャネルに対し、SSI RX および / または TX ピンを設定します（[Pins] タブ > [周辺機器] > [SSI] > [SSIn] > [SSITXD0/SSIRXD0]）。チャネル 0 の場合は、これらのピンのうちいずれかまたは両方を有効化します。チャネル 1 の場合は、SSIDATA1 ピンを有効化します。

ワード選択およびクロック ピンを設定します （[Pins] タブ > [周辺機器] > [SSI] > [SSIn] > [SSITWSn および SSISCKn]）。このピンは大抵、どちらとも必要となります。必要なピンについては、使用する I2S デバイスのデータシートを確認してください。

オーディオ クロック ピンを SSI 用に設定します （[Pins] タブ > [周辺機器] > [SSI] > [SSI0_SSI1_AUDIO_CLK]）。外部オーディオ クロックをこのクロック入力ピンに接続します。オーディオ クロックの作成に GPT タイマを使用する場合は、GPT タイマ出力ピンを設定し （[Pins] タブ > [周辺機器] > [GPT1] > [GPTn] > [GTIOCx]）、使用する GPT 出力ピンをオーディオ クロック入力ピンに接続します。

SSI での I2S ドライバの割り込みの設定

e2 studio ISDE を使って、[ICU] タブまたは [Threads] タブの [Properties] ウィンドウから SSI 割り込みを設定します （割り込みの設定を参照）。

チャネル 0 でのオーディオ データ受信を有効化するには、SSI > SSI0 RXI 割り込みを有効化します。チャネル 0 でのオーディオ データ送信を有効化するには、SSI > SSI0 TXI 割り込みを有効化します。チャネル 0 でのオーディオ データ送信および受信を両方有効化するには、SSI > SSI0 TXI と SSI > SSI0 RXI 割り込みを両方とも有効化します。チャネル 1 での送信または受信を有効化するには、SSI > SSIn TXI RXI 割り込みを有効化します。すべての場合に、SSI > SSIn INT 割り込みを有効化してください。

割り込みを有効化するには、e2 studio ISDE のプロジェクトコンフィギュレータの [ICU] タブ、あるいは e2 studio ISDE のプロジェクト コンフィギュレータの [Threads] タブにある [Properties] タブで割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応するマクロに、選択した優先度が設定されます。

割り込みが BSP で有効になっている場合、対応する ISR が I2S ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

SSI での I2S ドライバ用のパラメータの設定

e2 studio ISDE を使用して、I2S HAL ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

この表において、「**プロパティ**」は ISDE 内での [Properties] タブ名を表します。

SSI 構成上の I2S ドライバ

ISDE プロパティ	設定	設定値	説明
Channel	channel	0-1 for S7G2	物理ハードウェア チャネル。
Audio Clock Frequency (Hertz)	audio_clk_freq_hz	User input value Default: 2822400	I2S クロックの作成に用いられる入力オーディオ クロック周波数。以下の 1 ～ 128 の倍数である必要があります : (sampling_freq_hz * word_length_in_bits)
Sampling Frequency (Hertz)	sampling_freq_hz	User input value Default: 44100	オーディオ データのサンプリング周波数。
Data Bits	pcm_width	8 bits, 16 bits, 18 bits, 20 bits, 22 bits, 24 bits, 32 bits Default 16 bits i2s_pcm_width_t	オーディオ データのビット深度 （オーディオ データのサンプル 1 点のビット サイズ）。

ISDE プロパティ	設定	設定値	説明
Word Length	word_length	8 bits, 16 bits, 24 bits, 32 bits Default: 16 bits i2s_word_length_t	オーディオ データのワード長、少なくともビット深度と同じサイズである必要がります （データ ビット フィールド）。
WS Continue Mode	ws_continue	Enabled, Disabled Default: Enabled	WS 続行モードを有効化すると、周辺機器がアイドル状態になった場合でも、ワード選択ラインの出力が継続して行われます。周辺機器がアイドル状態になった場合にワード選択ラインの出力を停止するには、これを無効化します。
Callback	p_callback	User-defined, called with argumentsi2s_callback_args_t	ユーザー コールバック関数は open で登録する必要があります。全送信データの送信後に送信 FIFO 最高値に達した場合、あるいは受信が完了した （要求されたバイト数が受信された）場合に、割り込みサービス ルーチン （ISR）からコールバックが呼び出されます。 I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

4.2.16.3 I2S ドライバ使用上の注意

SSI 上の I2S のハードウェア フロー制御

プレースホルダー

4.2.16.4 SSI 上の I2S を使った I2S ドライバ アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const i2s_instance_t  g_i2s =

{

  .p_ctrl  = &g_i2s_ctrl,

  .p_cfg = &g_i2s_cfg,

  .p_api = &g_i2s_on_sci

};
```

I2S を使用して SSI アプリケーションを作成するには、以下の手順を実行します。

1) モジュールを上の説明のように設定します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます （以下の構造体が含まれます : i2s_cfg_t）。

2) 設定時に I2S ドライバに付けた名前を使用して、I2S インスタンスをオープンします。I2S インタフェースを通じて適切なドライバが呼び出されます。g_i2s という I2S ドライバの場合は以下のようになります。

```
g_i2s.p_api->open(g_i2s.p_ctrl, g_i2s.p_cfg)
```

タイマの設定手順後に、g_i2s.p_ctrl と g_i2s.p_cfg が自動生成されます。

3) I2S バスにオーディオ データを書き込む場合は、write API を呼び出します。1024 バイトのバッファ（バッファ プロトタイプ : uint8_t p_src [1024]）を書き込むための書き込み API の呼び出しは、次のようになります。

```
g_i2s.p_api->write(g_i2s.p_ctrl, &p_src[0], 1024)
```

イベント I2S_EVENT_TX_EMPTY のコールバックを待ってから、p_src バッファの解放または次のバッファの書き込みを行ってください。

4) I2S バスからオーディオ データを読み取るには、read API を呼び出します。1024 バイトのバッファ（バッファ プロトタイプ : uint8_t p_dest[1024]）を読み取るための読み取り API の呼び出しは、次のようになります。

```
g_i2s.p_api->read(g_i2s.p_ctrl, &p_dest[0], 1024)
```

イベント I2S_EVENT_RX_FULL のコールバックを待ってから、p_dest buffer バッファ内のデータへのアクセスまたは次のバッファの読み取りを行ってください。

5) 必要に応じて、i2s_api_t から他の API を使用します。

4.2.16.5 I2S ドライバの制限事項

このドライバは、割り込みベースの操作をサポートしていますが、ポーリング I2S モードはサポートしていません。

SSI ハードウェアは、8 バイトの倍数のみでの読み取り / 書き込みをサポートします。read および write API は、8 の倍数の長さで呼び出すことが強く推奨されます。送信の長さが 8 バイトの倍数ではない場合、これを 8 の倍数にするため送信の末尾にパディング 0 が追加されます。受信の長さが 8 バイトの倍数ではない場合、要求された数値を上回る直近の 8 バイトの倍数が受信され、余ったバイトは破棄されます。

4.2.16.6 I2S ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL I2S Interface API	synergy/ssp/inc/driver/api/r_i2s_api.h
I2S on SSI Instance	synergy/ssp/inc/driver/instances/r_ssi.h
I2S on SSI Driver	synergy/ssp/src/driver/r_ssi.c

I2S ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

I2S ドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.17 入力キャプチャ ドライバ

入力キャプチャ ドライバは、入力キャプチャ アプリケーション用の汎用 API で、MCU 上の GPT をサポートします。このため、入力キャプチャ ドライバは r_gpt_input_capture 上に実装することができます。このセクションでは、e^2 studio ISDE を使用して入力キャプチャ ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [タイマ] > [r_gpt_input_capture 上の入力キャプチャ ドライバ] を選択することで、入力キャプチャ ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による入力キャプチャ ドライバを使用するアプリケーションの作成

API リファレンスは、次の 入力キャプチャ インタフェースの説明内に記載されています : 入力キャプチャ インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

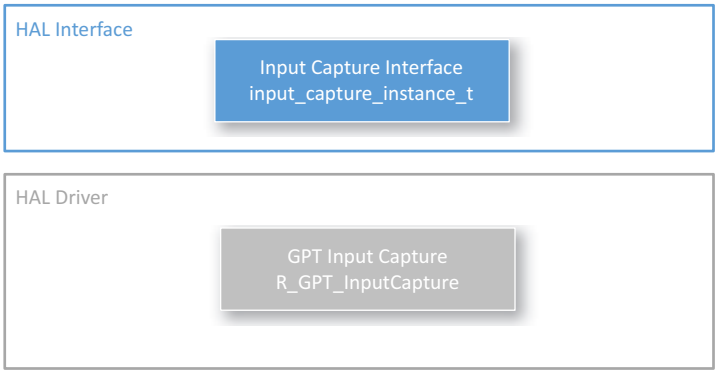

図 143:　入力キャプチャ ドライバ - ブロック図

4.2.17.1 入力キャプチャ ドライバの機能

入力キャプチャ ドライバは、パルス幅を測定するためのタイマを設定します。測定がキャプチャされた、またはカウンタ オーバーフローが発生した場合、測定データとともに CPU 割り込みからコールバックが呼び出されます。

4.2.17.2 e² studio ISDE による入力キャプチャ ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

GPT で入力キャプチャ ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
GPT Input Capture Driver	Threads	[Driver] > [タイマ] > [r_gpt_input_capture 上の入力キャプチャ ドライバ]。1 つ以上の入力キャプチャ ドライバを追加する場合、プロパティ エディターの [名前] フィールドで各モジュールに固有の名前を作成するようにします。
Interrupts	Threads	[GPT] > [GPTn] > [GPTn COUNTER OVERFLOW] オーバーフロー割り込みおよび [GPT] > [GPTn] > [GPTn CAPTURE COMPARE A] キャプチャ割り込み。
GPT Clock	Clocks	アプリケーション例については、GPT 入力キャプチャ クロックの設定
GPT Pins	Pins	アプリケーション例については、GPT 入力キャプチャ ピンの設定

GPT 入力キャプチャ クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して GPT クロックを設定します （クロックの設定を参照）。

GPT タイマは、PCLKA 周波数に基づいてクロック供給されます。PCLKA 周波数を設定するには、実行時に e^2 studio ISDE クロックの設定または CGC インタフェースのクロック コンフィギュレータを使用します。

最大期間は、タイマの種類 （32 ビットまたは 16 ビット）に依存します。

GPT 入力キャプチャ ピンの設定

e^2 studio ISDE を使用して、[Pins] タブから GPT ピンを設定します （ピンの設定を参照）。

GPT で入力キャプチャを使用するには、入力ピンとして使用する I/O ポート ピンが、ピン コンフィギュレータで GPT 周辺機器 （GTIOC）ピンとして設定されている必要があります （ピンの設定を参照）。

ピン コンフィギュレータでは、関連するピンのために IOPORT_CFG_PERIPHERAL_PIN と IOPORT_PERIPHERAL_GPT1 が pin_cfg フィールドで設定されます。

GPT 入力キャプチャ割り込みの設定

e^2 studio ISDE を使用して、[ICU] タブから GPT 割り込みを設定します （割り込みの設定を参照）。

選択したチャネルについて、GPT > GPTn COUNTER OVERFLOW 割り込み （n は GPT チャネル番号）および GPT > GPTn CAPTURE COMPARE A 割り込み （n は GPT チャネル番号）が BSP 内で有効化されている必要があります。

割り込みを有効化するには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブ、あるいは e2 studio ISDE のプロジェクト コンフィギュレータの [Threads] タブにある [Properties] タブで割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h 内の対応するマクロ SP_IRQ_CFG_GPTn_COUNTER_OVERFLOW および BSP_IRQ_CFG_GPTn_CAPTURE_COMPARE_A が選択した優先度に設定されます。

BSP で割り込みが有効化されており、入力キャプチャ ドライバでチャネルが使用されている場合は、対応する ISR が入力キャプチャ ドライバ内で定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

入力キャプチャ ドライバ用の GPT 入力キャプチャ パラメータの設定

e2 studio ISDE を使用して、r_gpt_input_capture に実装された入力キャプチャ ドライバのパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

この表において、「**プロパティ**」は ISDE 内での [Properties] タブ名を表します。

GPT 入力キャプチャ構成

ISDE プロパティ	設定	設定値	説明
Channel	channel	0-13 for S7G2, 0-9 for S3A7, 0-7 for S124	物理ハードウェア チャネル。
Mode	mode	Pulse Width input_capture_mode_t	信号エッジから反対側のエッジまでの入力を測定します
Signal Edge	edge	Rising, Falling Default: Rising input_capture_signal_edge_t	立ち上がりまたは立ち下がりエッジで測定を開始し、反対側のエッジで測定を停止します。
Repetition	repetition	One Shot, Periodic Default One Shot input_capture_repetition_t	信号測定をキャプチャし、その後 enable が呼び出されるまでキャプチャを無効化する （ワン ショット）か、継続的に測定をキャプチャします （周期）
Auto Start	autostart	True, false Default: true	設定後に測定を有効化するには、true に設定します。enable を呼び出すまで測定を無効状態にするには false に設定します。

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	User-defined, called with arguments input_capture_callback_args_t	ユーザー コールバック関数は open で登録する必要があります。このコールバックは、タイマ期間が経過するたびに割り込みサービス ルーチン （ISR）から呼び出されます。 I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
Input Capture Signal Pin	signal	GTIOCA, GTIOCB Default: GTIOCA gpt_input_capture_signal_t	測定の開始をトリガする入力ピンを選択します。
GTIOCx Signal Filter	signal_filter	None, PCLK/1, PCLK/4, PCLK/16, PCLK/64 Default: None gpt_input_capture_signal_filter_t	ノイズフィルタは、PCLK をいずれかの値で割った間隔で外部信号をサンプリングします。3 つ連続したサンプルが同じレベル （高または低）の場合、そのレベルは信号の観測値として受け渡されます。
Clock Divider	clock_divider	None, PCLK/1, PCLK/4, PCLK/16, PCLK/64, PCLK/256, PCLK/1024 Default: None gpt_input_capture_clock_divider_t	測定カウンタのスケールに用いられるクロック分周器。

ISDE プロパティ	設定	設定値	説明
Enable Level	enable_level	None, Low, High Default: None gpt_input_capture_signal_filter_t	それぞれの GPT チャネルは、2 つの I/O ピン（GTIOCA と GTIOCB）を備えています。そのうち 1 つを入力キャプチャ信号ピンとして選択する必要があります。もう 1 方の GPT I/O ピンは、キャプチャを有効化するためのハードウェア イネーブル信号として用いることができます。[なし] を選択した場合はキャプチャが常に有効化され、[Low] を選択した場合はイネーブル入力ピンが Low の間に限りキャプチャが有効化され、[High] を選択した場合はイネーブル入力ピンが High の間に限りキャプチャが有効化されます。
Enable Filter	enable_filter	None, PCLK/1, PCLK/4, PCLK/16, PCLK/64 gpt_input_capture_signal_filter_t	イネーブル フィルタは、PCLK をいずれかの値で割った間隔でイネーブル信号をサンプリングします。3 つ連続したサンプルが同じレベル （高または低）の場合、そのレベルは信号の観測値として受け渡されます。

GPT 入力キャプチャ信号

入力キャプチャ測定は、入力キャプチャ信号ピン （GTIOCA/GTIOCB）で入力キャプチャ信号エッジ （立ち下がりまたは立ち下がり）が検出され、かつイネーブル条件を満たした場合に開始されます。イネーブル条件はイネーブル レベルにより定義され、無効化する （なし）ことも、入力キャプチャ イネーブル ピン（GTIOCA/GTIOCB）のレベル （Low か High）を指定することもできます。入力キャプチャ イネーブル ピンは、入力キャプチャ信号ピンとして使用されないピンです。

測定カウントの時間への変換

測定が完了したら、未加工のカウント データとオーバーフロー数がコールバック関数としてユーザーに返されます。

未加工の測定データは、必要に応じ、コールバックまたはユーザーのアプリケーション内で論理的な時間単位に変換できます。未加工のデータを変換するには、現在の PCLKD クロック周波数、オーバーフロー数、

最大カウンタ値、測定カウントを考慮する必要があります。測定カウントとオーバーフロー数は、コールバック引数 input_capture_callback_args_t 内に記載されます。

現在の PCLKD 周波数を取得するには、systemClockFreqGet API を使用する方法が推奨されます。この PCLKD 周波数は、次の表において、clk_freq_hz として示されます。

S7G2 （全チャネル）および S3A7 （全チャネル）、S124 （チャネル 0）の最大カウンタ値は 0xFFFFFFFF です。S124 （チャネル 1-7）の最大カウンタ値は 0xFFFF です。以下の表において、この最大カウンタ値は max_counts として示されます。

入力キャプチャの時間計算

使用する時間単位	式
ナノ秒 （ns）	time_ns = ((overflows * max_counts) + counter) * 1000000000 / clk_freq_hz
マイクロ秒 （us）	time_ns = ((overflows * max_counts) + counter) * 1000000 / clk_freq_hz
ミリ秒 （ms）	time_ns = ((overflows * max_counts) + counter) * 1000 / clk_freq_hz
秒 （s）	time_ns = ((overflows * max_counts) + counter) / clk_freq_hz

4.2.17.3 入力キャプチャ ドライバ アプリケーションの作成

```
 /* Instance structure to use this module. */

const input_capture_instance_t  g_input_capture =

{

    .p_ctrl  = &g_input_capture_ctrl,

    .p_cfg = &g_input_capture_cfg,

    .p_api = &g_input_capture_on_gpt

};
```

GPT を使用して入力キャプチャ ドライバ アプリケーションを作成するには、次の手順を実行します。

1) モジュールを上の説明のように設定します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます （以下の構造体が含まれます : input_capture_cfg_t および gpt_input_capture_extend_t）。

2) 設定時に入力キャプチャ ドライバに付けた名前を使用して、入力キャプチャ インスタンスをオープンします。入力キャプチャ インタフェースを通じて適切なドライバが呼び出されます。g_input_capture というタイマの場合は以下のようになります。

```
g_input_capture.p_api->open(g_input_capture.p_ctrl, g_input_capture.p_cfg)
```

タイマの設定手順後に、g_input_capture.p_ctrl と g_input_capture.p_cfg が自動生成されます。

3) オートスタートが False に設定された場合は、次を呼び出して測定を有効化します。

```
g_input_capture.p_api->enable(g_input_capture.p_ctrl)
```

4) 必要に応じて、input_capture_api_t から他の API を使用します。

4.2.17.4 内部キャプチャ ドライバでサポートされるデバイス

このドライバは、S7G2 と S3A7 でテストされています。

タイマ インタフェースは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S124

4.2.17.5 入力キャプチャ ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL Input Capture Interface API	synergy/ssp/inc/driver/api/r_input_capture_api.h
GPT Input Capture Instance	synergy/ssp/inc/driver/instances/r_gpt_input_capture.h
GPT Input Capture Driver	synergy/ssp/src/driver/r_gpt_input_capture/r_gpt_input_capture.c

4.2.18 I/O ポート ドライバ

I/O ポート ドライバは r_ioport に実装されており、I/O ピン制御用の汎用 API を含みます。このドライバは、MCU 上で利用可能な I/O ポート周辺機器をサポートしています。このセクションでは、e^2 studio ISDE を使

用して I/O ポート ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e2 studio ISDE のプロジェクト コンフィギュレータで、I/O ポート ドライバ モジュールはデフォルトですべてのプロジェクトに追加されています。I/O ポート ドライバを構成するには、[Threads] タブの [Modules] ペインでこれを選択します。詳細については、以下を参照してください : e2 studio ISDE による I/O ポート ドライバを使用するアプリケーションの作成

API リファレンスは、次の I/O ポート インタフェースの説明内に記載されています : I/O ポートインタフェース。

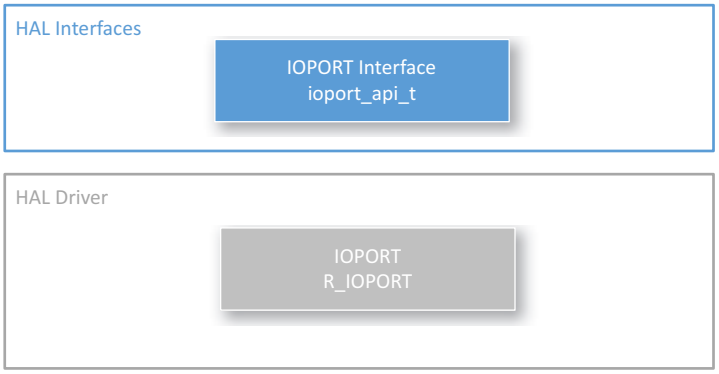

図 144: I/O ポート ドライバ - ブロック図

I :I/O ポート ドライバを使用して、実行時の I/O ピンを設定することができます。e2 studio ISDE では、[Pins] タブを通じてすべてのピン関連の設定を行うことができます （ピンの設定を参照）。

4.2.18.1 I/O ポート ドライバの機能

このドライバは、1 つ以上の I/O ピンを設定します。ピンの方向は、以下に示すその他のいくつかのオプションと共に設定できます。

- プルアップ
- NMOS/PMOS
- ドライブ強度
- イベント エッジ トリガー （立ち下がり、立ち上がり、または両方）
- ピンを IRQ ピンとして使用するかどうか

- ピンをアナログ ピンとして使用するかどうか
- ピンを周辺機器ピンとして使用するかどうかと、その周辺機器

ドライバは以下の機能も提供します。

- ポート上の 1 つ以上のピンの方向の変更
- ポート上の 1 つ以上のピンへの書き込み
- ポート上の 1 つ以上のピンからの読み取り
- イベント出力データの設定
- イベント入力データの読み取り

I：使用可能なオプションは MCU ごとに異なり、ピンごとに異なる場合があります。

4.2.18.2 e² studio ISDE による I/O ポート ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

プロジェクトの設定の際に、ISDE は **r_ioport** モジュールを、アプリケーションに自動的に追加します。

I/O ポート ドライバの設定

ピンの構成は、e² studio ISDE の [Pins] タブを使って行います。[Pins] タブは、デバイス上のピンの構成用のグラフィカル インタフェースを示します。個々のピンの構成は ioport_pin_cfg_t で、複数のピンの構成は ioport_cfg_t で設定されます。

I/O ポート ドライバ用のピンの設定

I/O ポート ドライバで、実行時のすべてのピン設定を行うことができます。

I/O ポート ドライバ用のクロックの設定

IOPORT ドライバには、固有のクロック構成は不要です。

I/O ポート ドライバ用の割り込みの設定

IOPORT ドライバには、割り込み構成は不要です。

I/O ポート パラメータの設定

e² studio ISDE を使用して、I/O ポート ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

I/O ポート ドライバでは、この g_ioport ドライバに関して [Properties] ウィンドウで設定できる設定可能オプションは 1 つのみです。

ISDE プロパティ	設定	設定値	説明
Parameter Checking	IOPORT_CFG_PARAM_CHECKING_ENABLE	Enabled (Default), Disabled	パラメータ エラー チェックを有効または無効にします。

アプリケーション起動時の個々のピンの初期設定は、IOPORT の設定構造体 ioport_pin_cfg_t で設定します。複数のピンの設定は、ioport_cfg_t で設定します。これらの構造は、プロジェクトのピン設定構造体により決定されます。ピン構成の初期設定は ISDE のピン構成により [Pins] タブを使って決定されます。

I/O ポート設定

ISDE プロパティ	設定	設定値	説明
See ISDE Pin configurator	pin_cfg	Integer value of ioport_cfg_options_t values	ピンの PFS （ピン機能選択）レジスタに書き込まれる値
See ISDE Pin configurator	pin	ioport_port_pin_t	設定データを書き込むピン。
See ISDE Pin configurator	number_of_pins	Integer value	ioport_cfg_t のこのインスタンスによって設定するピンの数。
See ISDE Pin configurator	p_pin_cfg_data	Pointer to ioport_pin_cfg_t	ピン設定構造体の配列へのポインタ。

4.2.18.3 I/O ポート アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
 /* Instance structure
to use this module. */

const ioport_instance_t  g_ioport =

{

    .p_api  = &g_ioport_on_ioport,

    .p_cfg = NULL

};
```

I/O ポート アプリケーションを作成するには、以下の手順を実行します。

1) 上記のように、構造体 ioport_pin_cfg_t および ioport_cfg_t を設定してモジュールを構成します。ISDE を使用するとこの設定が自動化され、g_bsp_pin_cfg が ssp_cfg\bsp\bsp_pin_cfg.h に作成されます。この設定は、起動時にピンを初期化するために使用されます。

2) 以下を呼びだすことでピンを初期化します。

```
g_ioport.p_api->init(g_ioport.p_ctrl)
```

これにより手順 1 で作成した ioport_cfg_t の構造体へのポインタを渡します。これでピンが設定されます。

3) GPIO ピンの方向を変更するには、以下を使用します。

```
g_ioport.p_api->pinDirectionSet()
```

4) 出力 GPIO ピンの出力レベルを設定するには、以下を使用します。

```
g_ioport_on_ioport.pinWrite()
```

5) 入力ピンの入力レベルを読み取るには、以下を使用します。

```
g_ioport_on_ioport.pinRead()
```

I/O ポートの例

次の 3 つのピンを設定します。

```
ioport_pin_cfg_t g_pin_cfg_data[] =

{

    {

        /* Port 1 */

        .pin = IOPORT_PORT_01_PIN_00,

        .pin_cfg = IOPORT_CFG_PORT_DIRECTION_OUTPUT, // GPIO output pin

    },

    {

        .pin = IOPORT_PORT_02_PIN_00,

        .pin_cfg = IOPORT_CFG_PULLUP_ENABLE, // GPIO input with pull-up enabled

    },

    {

        .pin = IOPORT_PORT_02_PIN_01,

        .pin_cfg = (IOPORT_CFG_PERIPHERAL_PIN | IOPORT_PERIPHERAL_GPT0),

    },

}

ioport_cfg_t g_pin_cfg =

    {

    .number_of_pins =
    (sizeof(g_pin_cfg_data))/(sizeof(ioport_pin_cfg_t)),

    .p_pin_cfg_data = &g_pin_cfg_data[0]

};

Set pins to specified levels:

ioport_level_t level;

/* Configure pins */
```

```
g_ioport.p_api->init(g_ioport.p_cfg);

/* Set pin 15 on port 3 as an output */

g_ioport.p_api->pinDirectionSet(IOPORT_PORT_03_PIN_15, IOPORT_DIRECTION_OUTPUT);

/* Set pin 15 on port 3 high */

g_ioport.p_api->pinWrite(IOPORT_PORT_03_PIN_15, IOPORT_LEVEL_HIGH);

/* Read level on port 2 pin 0 */

g_ioport.p_api->pinRead(IOPORT_PORT_02_PIN_00, &level);

/* Change pin 15 on port 3 to be an IRQ pin, falling edge triggered */

g_ioport.p_api->pinCfg(IOPORT_PORT_03_PIN_15, (IOPORT_CFG_IRQ_ENABLE | IOPORT_CFG_EVENT_FALLING_EDGE)
);
```

4.2.18.4 I/O ポート ドライバの制限事項

IOPORT HAL ドライバとその実装の制限事項については、SSP のリリース ノートを参照してください。

4.2.18.5 I/O ポート ドライバ ファイル

ISDE では、すべてのプロジェクトについて、次のファイルが自動的に含まれます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL IOPORT Interface API	synergy/ssp/inc/driver/api/r_ioport_api.h
IOPORT Instance	synergy/ssp/inc/driver/instances/r_ioport.h
IOPORT Driver	synergy/ssp/src/driver/r_ioport

4.2.18.6 I/O ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

I/O ポート ドライバは、API への変更なしに、すべての Synergy ファミリをサポートするように設計されています。

4.2.19 JPEG デコード ドライバ

JPEG デコード ドライバは、JPEG デコード処理のための汎用 API で、r_jpeg に実装されています。このドライバは、Synergy JPEG コーデック周辺機器をサポートします。このセクションでは、e2 studio ISDE を使用して JPEG デコード ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Stacks] ペインで [新規スタック] > [Driver] > [グラフィックス] > [r_jpeg 上の JPEG デコード ドライバ] を選択することで、JPEG デコード ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e2 studio ISDE による外部 JPEG デコード ドライバを使用するアプリケーションの作成。

API リファレンスは、次の JPEG デコード インタフェースの説明内に記載されています : JPEG デコードインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

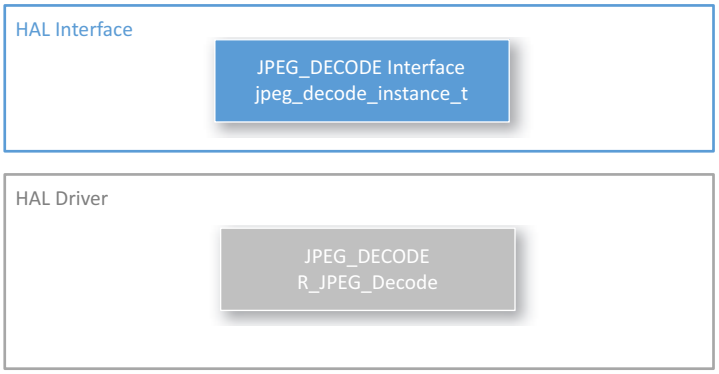

図 145: JPEG ドライバ - ブロック図

4.2.19.1 JPEG デコード ドライバの機能

- JPEG 解凍をサポート。
- JPEG デコーダが完了するまでアプリケーションが待機できるようにするポーリング モードをサポート。
- ユーザーが指定したコールバック関数を使用した割り込みモードをサポート。
- 水平および垂直サブサンプル値、水平ストライド、デコードされたピクセル フォーマット、入力および出力データ フォーマット、色空間などのパラメータを設定。
- イメージをデコードする前にそのサイズを取得。
- デコードされたイメージ フレームの保管のため、コード化されたデータを入力バッファーおよび出力バッファーに格納する操作をサポート。

- JPEG デコーダー モジュールへのコード化されたデータのストリーム転送をサポート。この機能により、アプリケーションは、ファイルやネットワークからのコード化された JPEG イメージの読み取りを、イメージ全体をバッファーリングすることなく実行できます。
- デコードするイメージ行数を設定。この機能により、アプリケーションは、デコードされたイメージの処理を、フレーム全体をバッファーリングすることなくオンザフライで実行可能。
- 入力値のデコードされたフォーマットとして YCbCr444、YCbCr422、YCbCr420、YCbCr411 をサポート。
- 出力フォーマットとして ARGB8888、RGB565 をサポート。
- JPEG イメージのサイズ、高さ、および幅が要件を満たさない場合にエラーを返すことが可能。

4.2.19.2 e² studio ISDE による外部 JPEG デコード ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

JPEG デコード ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
JPEG Decode Driver	Threads	[Driver] > [グラフィックス] > [r_jpeg 上の JPEG デコード ドライバ]
Interrupts	Threads	アプリケーション例については、JPEG デコード割り込みの設定
Clock	Clocks	アプリケーション例については、JPEG デコード ドライバのクロック ソースの設定
Pins	Pins	なし

JPEG デコード ドライバのクロック ソースの設定

e² studio ISDE で、[Clocks] タブを使用して JPEG クロックを設定します （クロックの設定を参照）。

JPEG デコード モジュールは、周辺機器モジュール クロック A （PCLKA）を使用して内部論理を実行します。ソース クロックの選択は、e² studio ISDE の Synergy 構成の [Clocks] タブで行うことができます。

JPEG デコード割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから JPEG 割り込みを設定します （割り込みの設定を参照）。

以降のセクションで説明するように、複数の JPEG デコード割り込みを設定することもできます。

解凍プロセス割り込み （JEDI）

JPEG 解凍プロセス割り込みは、次のイベントが検出されたときに発生します。

1) 現在の解凍プロセスが正常に完了した。
2) 解凍プロセス中にエラーが発生した。
3) イメージのサイズとピクセル フォーマットが正常に読み出された。

I：割り込みを有効にするには、e2 studio ISDE のプロジェクトコンフィギュレータの [ICU] または [Threads] タブで、JPEG > JPEG JEDI 割り込みの優先度を設定します。

データ転送インタフェース （JDTI）

JPEG データ転送割り込みは、次のイベントが検出されたときに発生します。

1) JPEG コードされたデータがすべて正常に完了した。
2) linesDecodedGet で指定された出力イメージ データ行数が転送された。
3) inputBufferSet で指定された入力イメージ データ行数が転送された。

I：割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、JPEG > JPEG JDTI の優先度を設定します。

JPEG デコード ステータス

JPEG デコード モジュールの制御ブロックにはステータス フラグがあります。ユーザーは statusGet を通じて現在のステータスを取得できます。ステータスの定義については以下の表をご覧ください。また、モジュールで特定のイベントが発生すると、ユーザー コールバック関数を通じてステータスがレポートされます。JPEG デコード コールバックを参照してください。

イベント名	イベントの発生条件
JPEG_DECODE_STATUS_FREE	JPEG デコード モジュールが閉じられた。

イベント名	イベントの発生条件
JPEG_DECODE_STATUS_IDLE	JPEG デコード モジュールが開いているが、動作していない。
JPEG_DECODE_STATUS_RUNNING	JPEG デコード操作が実行されている。
JPEG_DECODE_STATUS_DONE	JPEG デコード操作が正常に完了した。
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG デコードが追加の入力データを待つために一時停止した。
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG デコードが、ユーザーにより指定された行数をデコードした後に一時停止した。
JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG デコードが、デコードするデータのイメージ サイズを取得して、一時停止した。
JPEG_DECODE_STATUS_ERROR	JPEG デコード モジュールでエラーが発生した。
JPEG_DECODE_STATUS_HEADER_PROCESSING	JPEG デコード モジュールが JPEG ヘッダーを処理中である。

JPEG デコード コールバック

ユーザー コールバック関数を open で登録できます。ユーザー コールバック関数が指定されている場合、割り込みが発生するたびに割り込みサービス ルーチン （ISR）からコールバック関数が呼び出されます。コールバック関数の引数 status は、デコーディング プロシージャにおいて発生したイベントをユーザーが識別できるように、以下に示す列挙値を受け取ることができます。

イベント名	イベントの発生条件
JPEG_DECODE_STATUS_ERROR	JPEG デコード モジュールでエラーが発生した。
JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG デコードが、デコードするデータのイメージ サイズを取得して、一時停止した。
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG デコードが追加の入力データを待つために一時停止した。
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG デコードが、ユーザーにより指定された行数をデコードした後に一時停止した。
JPEG_DECODE_STATUS_DONE	JPEG デコード操作が正常に完了した。

I a: ユーザー コールバック関数は ISR から呼び出されるため、ブロッキング呼び出しを使用することや、長

時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

JPEG デコード ドライバ パラメータの設定

e^2 studio ISDE を使用して、JPEG デコード ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定。

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

JPEG デコードには、次のコンポーネントの設定が必要です。

- JPEG デコード :jpeg_decode_cfg_t

JPEG ドライバ デコードのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter Checking Enable	#define BSP_CFG_PARAM_CHECKING_ENABLE	Enabled (Default), Disabled	パラメータ エラー チェックを有効または無効にします。
Name	インスタンス名のプレフィックス。	Arbitrary C Symbol (Default: "g_jpeg_decode")	JPEG デコード モジュール インスタンスに使用する名前。
Byte Order for Input Data Format	input_data_format	Normal byte order - (1)(2)(3)(4)(5)(6)(7)(8)(Default);Byte Swap - (2)(1)(4)(3)(6)(5)(8)(7); Word Swap - (3)(4)(1)(2)(7)(8)(5)(6); Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5); Longword Swap - (5)(6)(7)(8)(1)(2)(3)(4); Longword-Byte Swap - (6)(5)(8)(7)(2)(1)(4)(3); Longword-Word Swap -(7)(8)(5)(6)(3)(4)(1)(2); Longword-Word-Byte Swap - (8)(7)(6)(5)(4)(3)(2)(1)	入力データのバイト順序を指定します。順序は 8 バイトごとに指定に従ってスワップされます。

ISDE プロパティ	設定	設定値	説明
Byte Order for Output Data Format	output_data_format	Normal byte order - (1)(2)(3)(4)(5)(6)(7)(8)(Default);Byte Swap - (2)(1)(4)(3)(6)(5)(8)(7); Word Swap - (3)(4)(1)(2)(7)(8)(5)(6); Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5); Longword Swap - (5)(6)(7)(8)(1)(2)(3)(4); Longword-Byte Swap - (6)(5)(8)(7)(2)(1)(4)(3); Longword-Word Swap -(7)(8)(5)(6)(3)(4)(1)(2); Longword-Word-Byte Swap - (8)(7)(6)(5)(4)(3)(2)(1)	出力データのバイト順序を指定します。順序は 8 バイトごとに指定に従ってスワップされます。
Pixel Format	pixel_format	ARGBB888 or RGB565 (Default)	出力データ フォーマットを指定します。
Alpha value to be applied to decoded pixel data	alpha_value	Value must be an integer greater than or equal to 0 and less than or equal to 255 (Default: 255)	出力データ フォーマットのアルファ値を指定します (ARGB8888 フォーマットに対してのみ有効)。
Name of user callback function	p_callback	Arbitrary C Symbol (Default: NULL)	ユーザー コールバック関数の名前を指定します。

JPEG のユーザー定義のコールバック関数

JPEG デコード モジュールは、open の構成 p_callback に渡されたユーザー定義のコールバック関数を呼び出します。このコールバック関数を使用して、JPEG デコード モジュールのステータスを取得し、JPEG デコーディングのシーケンスを開始することができます。

```
volatile _Bool g_input_buffer_pause_flag = false;
volatile _Bool g_output_buffer_pause_flag = false;
volatile _Bool g_decode_done_flag = false;
volatile _Bool g_image_size_is_ready_flag = false;
volatile _Bool g_decode_error_flag = false;

void user_jpeg_callback (jpeg_decode_callback_args_t *p_args)
{
    if (p_args->status & JPEG_DECODE_STATUS_IMAGE_SIZE_READY)
  {
        g_image_size_is_ready_flag = true;
    }
    if (p_args->status & JPEG_DECODE_STATUS_INPUT_PAUSE)
  {
        g_input_buffer_pause_flag = true;
    }
    if (p_args->status & JPEG_DECODE_STATUS_OUTPUT_PAUSE)
  {
        g_output_buffer_pause_flag = true;
    }
    if (p_args->status & JPEG_DECODE_STATUS_DONE)
  {
        g_decode_done_flag = true;
    }
    if (p_args->status & JPEG_DECODE_STATUS_ERROR)
  {
        g_decode_error_flag = true;
    }
}
```

JPEG モジュールを初期化する

```
/* JPEG Decode module control block */
static jpeg_decode_ctrl_t  g_jpeg_decode_ctrl;

/* JPEG Decode module configuration */
static const jpeg_decode_cfg_t  g_jpeg_decode_cfg = {
    .input_data_format  = JPEG_DECODE_DATA_FORMAT_NORMAL,
   .output_data_format = JPEG_DECODE_DATA_FORMAT_NORMAL,
    .pixel_format = JPEG_DECODE_PIXEL_FORMAT_RGB565,

    .alpha_value = 255,
    .p_callback = user_jpeg_callback
};

/* JPEG Decode module instance */
const jpeg_decode_instance_t  g_jpeg_decode = {
    .p_api  = (jpeg_decode_api_t  const *)&g_jpeg_decode_on_jpeg_decode,
   .p_ctrl = &g_jpeg_decode_ctrl,
    .p_cfg = &g_jpeg_decode_cfg
};
```

JPEG モジュールを開く

JPEG エンコードされたデータのデコードは、open を呼び出すことで開始できます。モジュールを開くには、JPEG モジュール インスタンスを使用します。このインスタンスには、API 関数のポインタ、制御ブロックへのポインタ、および e^2 studio ISDE のプロジェクト コンフィギュレータを通じて生成可能な静的構成が含まれています。

```
/* JPEG Decode driver open */
g_jpeg_decode.p_api->open(jpeg_decode.p_ctrl, jpeg_decode.p_cfg);
```

JPEG モジュールを閉じる

JPEG デコード モジュールを停止するには、close を呼び出します。

```
/* JPEG Decode driver close */
g_jpeg_decode.p_api->close(jpeg_decode.p_ctrl);
```

JPEG 入力バッファ ストリーミング モード

次の例は、JPEG エンコードされたデータを JPEG 入力バッファー ストリーミング モードでデコードする方法を示しています。JPEG デコード モジュールは、JPEG モジュールを開く で記述した構成を使用してあらかじめ開かれていなければなりません。

```
    ...

/* Set the vertical and horizontal sample. */

g_jpeg_decode0.p_api->imageSubsampleSet(g_jpeg_decode0.p_ctrl,
*JPEG_DECODE_OUTPUT_NO_SUBSAMPLE*,
*JPEG_DECODE_OUTPUT_NO_SUBSAMPLE*);

/* Set the decoding image horizontal stride. Here, assuming it is 800
in pixels */
g_jpeg_decode0.p_api->horizontalStrideSet(g_jpeg_decode0.p_ctrl,
800);

/* Set output buffer address. */

g_jpeg_decode0.p_api->outputBufferSet(g_jpeg_decode0.p_ctrl,
output_buffer, output_buffer_size);

/* Get an input buffer address and the size */
input_buffer = user_get_input_buffer_address();
input_buffer_size = user_get_input_buffer_size();

/* Set the input buffer address. */

g_jpeg_decode0.p_api->inputBufferSet(g_jpeg_decode0.p_ctrl,
input_buffer, input_buffer_size);

while(1)

{   /* Wait until the JPEG hardware being ready to get image size */

    if(true == g_image_size_is_ready_flag)

    {

        break;

    }

}

while(1)

{

    /* Wait until either of flags is set. Decode done or Input buffer pause. */

    if(true == g_decode_done_flag)
```

```
    {

        break;

    }

    if(true == g_input_buffer_pause_flag)

    {

        input_buffer = user_get_input_buffer_address();
        input_buffer_size = user_get_input_buffer_size();

        /* Set input buffer address. */

        input_buffer = input_buffer + input_size;

        err = g_jpeg_decode0.p_api->inputBufferSet(g_jpeg_decode0.p_ctrl,
input_buffer, input_buffer_size);

    }

}

    …
```

JPEG 出力バッファ ストリーミング モード

次の例は、JPEG エンコードされたデータを JPEG 出力バッファー ストリーミング モードでデコードする方法を示しています。JPEG デコード モジュールは、JPEG モジュールを開く で記述した構成を使用してあらかじめ開かれていなければなりません。

```
…

/* Set the vertical and horizontal sample. */

g_jpeg_decode0.p_api->imageSubsampleSet(g_jpeg_decode0.p_ctrl,
*JPEG_DECODE_OUTPUT_NO_SUBSAMPLE*,
*JPEG_DECODE_OUTPUT_NO_SUBSAMPLE*);

/* Set the decoding image horizontal stride. Here, assuming it is 800
in pixels */
g_jpeg_decode0.p_api->horizontalStrideSet(g_jpeg_decode0.p_ctrl,
800);

/* Set output buffer address. */

g_jpeg_decode0.p_api->outputBufferSet(g_jpeg_decode0.p_ctrl,
output_buffer, output_buffer_size);

/* Set the input buffer address. */

g_jpeg_decode0.p_api->inputBufferSet(g_jpeg_decode0.p_ctrl,
                                                input_buffer, input_buffer_size);

while(1)

{   /* Wait until the JPEG hardware being ready to get image size */

      if(true == g_image_size_is_ready_flag)

      {

            break;

      }

}

uint16_t horizontal_size;

uint16_t vertical_size;

g_jpeg_decode0.p_api->imageSizeGet(g_jpeg_decode0.p_ctrl,

&horizontal_size, &vertical_size);

/* Calculate the size of image in bytes. Here is the case of 2bpp(RGB565) */

uint32_t total_bytes = (uint32_t)(horizontal_size * vertical_size * 2);
```

```
uint32_t decoded_bytes = 0;

while(1)

{

    /* Wait until either of flags is set. Decode done or Output buffer pause. */

    if(true == g_decode_done_flag)

    {

        break;

    }

    if(true == g_output_buffer_pause_flag)

    {

        /* Set input buffer address. */

        output_buffer = output_buffer + output_size;

        g_jpeg_decode0.p_api->outputBufferSet(g_jpeg_decode0.p_ctrl,
                    output_buffer, output_size);

        decoded_bytes = decoded_bytes + output_size;

        if (total_bytes <= decoded_bytes)

        {

            break;

        }

        g_output_buffer_pause_flag = false;

    }

}

…
```

4.2.19.3 JPEG デコード ドライバの制限事項

JPEG デコード ドライバは、JPEG エンコード処理をサポートしていません。

4.2.19.4 JPEG デコード ドライバでサポートされるデバイス

モジュールは S7G2 ファミリでサポートされています。

4.2.20 Key Matrix ドライバ

Key Matrix ドライバは r_kint に実装されており、キー イベント検出用の汎用 API を含みます。このドライバ、MCU 上で利用可能なキー割り込み関数周辺機能をサポートしています。このセクションでは、e^2 studio ISDE を使用して Key Matrix ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [入力] > [r_kint 上の Key Matrix ドライバ] を選択することで、Key Matrix ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による Key Matrix ドライバを使用するアプリケーションの作成

API リファレンスは、次の Key Matrix インタフェースの説明内に記載されています : Key Matrix インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

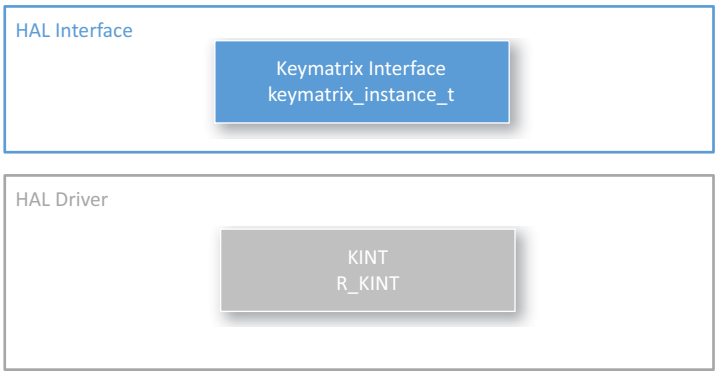

図 146: Key Matrix ドライバ - ブロック図

4.2.20.1 KINT ドライバの機能

このドライバは、いずれかのキー 割り込み （KINT）チャネル keymatrix_channels_t で立ち上がりまたは立ち下がりエッジを検出するよう、KINT 周辺機器を設定します。設定されたピンのいずれかで該当するイベントが検出されると、モジュールで割り込みが生成されます。次に、この割り込みによって、ビットマスク

keymatrix_channels_t を通じてエッジが検出されたチャネルを指定するコールバック引数 keymatrix_callback_args_t でユーザー コールバックが呼び出されます。

いずれか 1 つのチャネルでエッジが検出されれば割り込みが生成されますが、他のいずれかのピンでもエッジが検出された場合には、その時点でトリガーされたすべてのピンのビットマスクがコールバックで返されます。そのため、コールバックが呼び出される前に他のピンでもエッジが検出された場合、ピンごとのエッジ検出に対して必ずしも割り込みが生成されるとは限りません。コールバックが呼び出された後で新しいエッジが検出されると、割り込みが再度トリガーされ、新たにコールバックが呼び出されます。

このモジュールを使用すると、押された実際のキーを示す 2 つのチャネル上のエッジを使用した、マトリックス キーパッドを実装できます。また、このモジュールは 1 つの入力ピン上のエッジを検出する単一入力としても使用できます。

4.2.20.2 e² studio ISDE による Key Matrix ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

次のリソースは、Key Matrix ドライバを使用するアプリケーションで必要となります。

リソース	ISDE タブ	選択
Key Matrix Driver	Threads	[Driver] > [入力] > [r_kint 上の KeyMatrix]
Interrupts	Threads	[KEY] > [KEYINT] > [KEYINT KR]
Pins	Pins	アプリケーション例については、Key Matrix ドライバ用のピンの設定

アプリケーションの Key Matrix 割り込みでデータ転送またはイベント トリガが必要な場合に、次のリソースはオプションです。

リソース	ISDE タブ	選択
Data Transfer Controller Driver	Threads	DTC イベント PORT0 IRQ の g_transfer 転送モジュール
DMA Controller	Threads	DMAC イベント PORT0 IRQ の g_transfer 転送モジュール

リソース	ISDE タブ	選択
Event Link Controller	Threads	ISDE には、デフォルトで ELC の 1 つのインスタンスが含まれます。

Key Matrix ドライバ用のクロックの設定

Key Matrix ドライバには、固有のクロック構成は不要です。

Key Matrix ドライバ用のピンの設定

e2 studio ISDE を使用して、[Pins] タブから KINT ピンを設定します （ピンの設定を参照）。

KINT 周辺機器を使用するには、入力ピンとして使用する I/O ポート ピンが、ピン コンフィギュレータで KINT 周辺機器ピンとして設定されている必要があります。

e2 studio ISDE により、pin_cfg フィールドで関連するピンの適切なピン設定が構成されます。

Key Matrix ドライバ用の割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから KEYINT 割り込みを設定します （割り込みの設定を参照）。

このモジュールが動作するためには、open 呼び出しでコールバックを使用するかどうかにかかわらず、BSP で KINT （INTKR）割り込みを有効にする必要があります。

I：割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、KEY > KEY INTKR の優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_KEY_INT に、選択した優先度が設定されます。

BSP_IRQ_CFG_KEY_INT 割り込みが BSP で有効になっている場合、対応する ISR が KINT ドライバで定義されています。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

このモジュールが動作するためには、
I：割り込みが有効になっている必要があります。

Key Matrix ドライバ パラメータの設定

e2 studio ISDE を使用して、Key Matrix ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

表：Key Matrix の設定

ISDE プロパティ	設定	設定値	説明
Keymatrix Channel Mask	channels	keymatrix_channels_t	各ビットがそのチャネルを有効にするかどうかを指定するビットマスク。たとえば、チャネル 0 を有効にするには、ビット 0 に 1 を設定します。
Trigger Type	trigger	keymatrix_trigger_t	有効化されたチャネルが、立ち上がりエッジと立ち下がりエッジのどちらを検出するか指定します。 I：立ち上がりエッジを検出するすべてのチャネル、または立ち下がりエッジを検出するすべてのチャネルのいずれか。
Interrupt enabled after initialization	autostart	true, false	モジュール割り込みを open 呼び出しの一部として有効にする必要があるかどうかを指定します。

ISDE プロパティ	設定	設定値	説明
Name	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、いずれかのチャネルで設定されているエッジが検出された場合に、割り込みサービス ルーチン （ISR）から呼び出されます。 I：コールバックなしでは、アプリケーションはイベントが発生したかどうかを判定することはできません。 I a: コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないでください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

4.2.20.3 Key Matrix ドライバ アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const keymatrix_instance_t  g_kint =

{

    .p_ctrl  = &g_kint_ctrl,

    .p_cfg = &g_kint_cfg,

    .p_api = &g_keymatrix_on_kint

};
```

KINT を使用した Key Matrix ドライバ アプリケーションを作成するには、以下の手順を実行します。

1) 前述のように、keymatrix_cfg_t の構造体を設定してモジュールを構成します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます。
2) KINT として実装された Key Matrix を使用して、Key Matrix インスタンスをオープンします。Key Matrix インタフェースを通じて KINT ドライバが呼び出されます。

```
g_keymatrix_on_kint.p_api->open(&g_kint.p_ctrl, g_kint.p_cfg_cfg)
```

g_kint.p_ctrl と g_kint_cfg は、keymatrix の設定手順の後に自動生成される制御および構成構造体です。

3) g_keymatrix_on_kint.p_api-> open 呼び出しを呼び出す前に autostart が true に設定されていると、モジュールの割り込みが有効になり、モジュールは設定されたエッジをチャネル上で検出できるようになります。
4) 初期化後にモジュールを有効にするには、以下を使用します。

```
g_keymatrix_on_kint.p_api->enable(&g_kint.p_ctrl)
```

g_kint.p_ctrl は、g_keymatrix_on_kint.p_api-> open 呼び出しで使用した制御構造体です。この呼び出しにより、モジュールの割り込みが有効になります。

5) 初期化後にモジュールを無効にするには、以下を使用します。

```
g_keymatrix_on_kint.p_api->disable(g_kint.p_ctrl)
```

g_kint.p_ctrl は、g_keymatrix_on_kint.p_api-> open 呼び出しで使用した制御構造体です。この呼び出しにより、モジュールの割り込みが無効になります。

6) 初期化後に、設定されたトリガー エッジを変更するには、次のインタフェースを使用します。

```
g_keymatrix_on_kint.p_api->triggerSet()
```

7) 次のインタフェースを呼び出して、keymatrix インスタンスをクローズします。

```
g_keymatrix_on_kint.p_api->close(&g_ keymatrix)
```

4.2.20.4 Key Matrix ドライバ使用上の注意

KINT を使用した DMAC/DTC のトリガー

トリガ エッジが検出されたときに、DMAC または DTC 周辺機器を使用したデータの転送をトリガするには、activation_source を ELC_EVENT_KEY_INT に設定して DMAC/DTC 転送を構成します。詳細については、転送インタフェースを参照してください。

KINT を使用した ELC イベントのトリガー

KINT モジュールは、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

4.2.20.5 Key Matrix ドライバの制限事項

このドライバは、ポーリング モードの動作をサポートしていません。Key Matrix ドライバとその実装のその他の制限事項については、SSP のリリース ノートを参照してください。

4.2.20.6 Key Matrix ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL Key Matrix Interface API	synergy/ssp/inc/driver/api/r_keymatrix_api.h
KINT Instance	synergy/ssp/inc/driver/instances/r_kint.h
KINT Driver	synergy/ssp/src/driver/r_kint

4.2.20.7 Key Matrix ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

Key Matrix ドライバは、API への変更なしに、次のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.21 ローパワー モードドライバ

ローパワー モード ドライバ （LPM Driver）は r_lpm 上に実装されており、ローパワー モード ハードウェア周辺機器へのアクセスおよび構成を提供する API を含みます。低電力モードインタフェースの LPM 実装は、SSP アーキテクチャの HAL ドライバ レイヤーに配置されています。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [パワー] > [r_lpm 上のローパワー モード ドライバ] を選択することで、ローパワー モード ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による LPM ドライバを使用するアプリケーションの作成

API リファレンスは、次のローパワー モード インタフェースの説明内に記載されています : 低電力モードインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

4.2.21.1 ローパワー モード ドライバの機能

LPM ドライバは、ローパワー モード ハードウェア周辺機器を使用する MCU 動作電力制御モードと MCU ローパワー モードの設定をサポートしています。

この LPM ドライバは、動作電力制御モードの以下の機能をサポートしています。

- 低電圧モード
- 低速モード
- 中速モード
- 高速モード
- サブ発振器速度モード

この LPM ドライバは、ローパワー モードの以下の機能をサポートしています。

- ディープ ソフトウェア スタンバイ モード
- ソフトウェア スタンバイ モード
- スリープ モード
- スヌーズ モード

LPM ドライバは、ディープ スタンバイ モード時に、内部の電力供給制御と IO ポートの状態の再設定を通じて消費電力の低減をサポートします。LPM ドライバは、MCU の他のハードウェア周辺機器の無効化と有効化をサポートしています。

I：すべての MCU で、すべての動作モードが利用可能とは限りません。すべての MCU で、すべてのローパワー モードが利用可能とは限りません。

4.2.21.2 e² studio ISDE による LPM ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

アプリケーションで実行時のチップの動作電力制御モードを変更する必要がある場合は、次のリソースが必要となります。CGC ドライバは、HAL/ 共通の一部として、デフォルトでプロジェクトに含まれています。

リソース	ISDE タブ	選択
Clock Generation Controller	Threads	r_cgc 上の g_cgc CGC ドライバ

LPM ドライバは、HAL/Common の一部またはスレッドの一部として、あるいはそれら両方の一部として、Synergy プロジェクトに追加できます。

LPM ドライバを HAL/Common に追加するには、次の手順を実行します。

- [Threads] タブを開きます。
- [Threads] ウィンドウの HAL/ 共通をハイライトします。
- [HAL/ 共通モジュール] ウィンドウの [新規] を左クリックします。
- [Driver] > [パワー] の順にカーソルを合わせます。
- [Driver] > [パワー] から、[r_lpm 上のローパワー モード] を選択します。

LPM ドライバを**新しいスレッド**に追加するには、次の手順を実行します。

- [Threads] タブを開きます。
- [Threads] ウィンドウのスレッドをハイライトします。
- [< スレッド名 > モジュール] ウィンドウで [新規] を左クリックします。
- [Driver] > [パワー] の順にカーソルを合わせます。
- [Driver] > [パワー] から、[r_lpm 上のローパワー モード] を選択します。

LPM ドライバ モジュールを、複数のスレッド、または 1 つのスレッドと HAL/ 共通に追加する場合は、LPM ドライバのインスタンスごとに固有の名前を付けてください。名前を変更するには、[Modules] ウィンドウでドライバのインスタンスをハイライトして、[Properties] ビューで名前のエントリを変更します。LPM ドライバの設定構造体のデフォルト設定も、このビューで変更できます。

4.2.21.3 MCU の動作モードの設定

動作状態およびローパワー モードの MCU の予想消費電力の詳細については、『MCU Synergy ハードウェア ユーザーズ マニュアル : マイクロコントローラ』の「電気的特性」セクションの「動作電流とスタンバイ電流」を参照してください。

デフォルトでは、MCU の動作モードは高速モードに設定されています。高速モードでは、すべてのクロック ソースを利用できます。各動作モードで使用可能な発振器の種類の詳細については、『MCU Synergy ユーザーズ マニュアル : マイクロコントローラ』内の「それぞれのモードで利用可能な発振器の表」を参照してください。MCU の動作モードは、このドライバを CGC ドライバと共に使用することで実行時に変更できます。動作電力制御モードを使用してシステム クロック ソースを使用できるように変更するために CGC ドライバを使用する必要があります。たとえば、LOCO （低速オンチップ発振器）は低速動作電力制御モードを使用したシステム クロックとして使用される必要があります。API がシステム クロックを変更する関数を提供する CGC ドライバです。

実行時の MCU の動作モードを変更するには、[Properties] ビューで LPM ドライバの構成構造体 : lpm_cfg_t のデフォルト設定を指定します。データ構造体は以下のフィールドを持っています。

```
typedef struct st_lpm_cfg

{

  lpm_operating_power_t  operating_power;

  lpm_subosc_t           sub_oscillator;

} lpm_cfg_t;
```

デフォルトでは次のように設定されますが、ユーザーが変更可能です。

ISDE プロパティ	設定	設定値	説明
Operating power control mode	operating_power	High speed operating mode, Middle speed operating mode, Low speed operating mode, Low voltage operating mode; LPM_OPERATING_POWER_HIGH_SPEED_MODE (Default)	動作電力を選択します。
Sub oscillator mode	sub_oscillator	Not enabled, enabled; LPM_SUBOSC_OTHER (Default)	サブ発振器を選択します。

このビューで選択可能な設定は、動作電力制御モードに関するもののみです。ここで選択された設定は、LPM API 関数 operatingPowerModeSet が呼び出されるまで適用されません。LPM API 関数の operatingPowerModeSet 関数は、LPM API 関数 init により呼び出されます。MCU の動作電力制御モードの設定と変更は、このように行うことが推奨されます。

動作電力制御モード間の遷移を実行時に複数回行う必要がある場合は、LPM API 構成構造体のインスタンスを作成する必要があります。以下は、複数の動作モードと複数のインスタンスを使用する場合の例です。

多くのモジュールのインスタンスを宣言する代わりに、このアプリケーションはモード間で頻繁に簡単に切り替え、すべての構成順列を使用することができる一つのインスタンスを使用します。構成間で切り替えが必要な唯一の変更は、一つの関数に渡されるパラメータを変更することだけです。このため、より容易に単一の LPM インスタンスを宣言して、それを lpm_cfg_t 構造体の複数の構成で使用することができるようになります。生成されるコードはより小さく効率的になります。

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */
const lpm_instance_t g_lpm =
{
    .p_ctrl = &g_lpm_ctrl,
    .p_cfg  = &g_lpm_cfg,
    .p_api = &g_lpm_on_lpm
};
```

In src/ hal_entry.c add,

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
  const lpm_cfg_t g_lpm_high_speed =
  {
    .operating_power = LPM_OPERATING_POWER_HIGH_SPEED_MODE,
    .sub_oscillator = LPM_SUBOSC_OTHER

  };
  const lpm_cfg_t g_lpm_low_speed =
  {
    .operating_power = LPM_OPERATING_POWER_LOW_SPEED_MODE,
    .sub_oscillator = LPM_SUBOSC_OTHER
```

```
};
const lpm_cfg_t g_lpm_middle_speed =
{
  .operating_power = LPM_OPERATING_POWER_MIDDLE_SPEED_MODE,
  .sub_oscillator = LPM_SUBOSC_OTHER
```

```
};
const lpm_cfg_t g_lpm_low_voltage =
{
  .operating_power = LPM_OPERATING_POWER_LOW_VOLTAGE_MODE,
  .sub_oscillator = LPM_SUBOSC_OTHER
```

```
    };
    /* Set operating mode to defaults set in Properties view of Synergy Configuration */

    g_lpm.p_api->init(g_lpm.p_cfg);
  /* Not shown: Set clocks for new operating mode before changing operating mode */
    /* Set operating mode to low speed */
    g_lpm.p_api->init(g_lpm_low_speed.p_cfg);
  /* Not shown: Set clocks for new operating mode before changing operating mode */
    /* Set operating mode to high speed */
    g_lpm.p_api->init(g_lpm_high_speed.p_cfg);
  /* Not shown: Set clocks for new operating mode before changing operating mode */
    /* Set operating mode to middle speed */
    g_lpm.p_api->init(g_lpm_middle_speed.p_cfg);
  /* Not shown: Set clocks for new operating mode before changing operating mode */
    /* Set operating mode to low voltage */
    g_lpm.p_api->init(g_lpm_low_voltage.p_cfg);
}
```

以下のコードは高速電力制御モードからサブ発信器動作電力制御モードへの移行プロセスの例を示します。まずクロック、そして次に動作電力制御モードを変更する必要があります。システム クロックの変更の詳細については、CGC の使用上の注意を参照してください。使用したい動作モードで使用可能な発振器の種類の詳細については、『MCU Synergy ユーザーズ マニュアル : マイクロコントローラ』内の表「各モードで使用可能な発振器」を参照してください。また、特定の動作モードに移行する前にオフにすべき、あるいは変更すべきシステム クロックや周辺機器の詳細については、『MCU Synergy ユーザーズ マニュアル : マイクロコントローラ』のレジスタアクセス セクション （2）「クロック関連のレジスタの有効な設定方法」を参照してください。

```
cgc_clock_t  clock_source = CGC_CLOCK_SUBCLOCK;

/* p_clock_cfg is used for PLL only, by clockStart() */

cgc_clock_cfg_t  p_clock_cfg;

g_cgc_on_cgc.systemClockSet( clock_source, &p_clock_cfg );

g_cgc_on_cgc.clockStop( CGC_CLOCK_MAIN_OSC  );

g_cgc_on_cgc.clockStop( CGC_CLOCK_PLL  );

g_cgc_on_cgc.clockStop( CGC_CLOCK_HOCO  );

g_cgc_on_cgc.clockStop( CGC_CLOCK_MOCO  );

g_cgc_on_cgc.clockStop( CGC_CLOCK_LOCO  );

g_lpm.p_api->operatingPowerModeSet(LPM_OPERATING_POWER_LOW_SPEED_MODE, LPM_SUBOSC_SE
LECT);
```

4.2.21.4 MCU のローパワー モードの設定

動作状態および低電力モードの MCU の予想消費電力の詳細については、MCU ユーザーズマニュアルの「電気的特性」セクションの「動作電流とスタンバイ電流」を参照してください。

スリープ モード

デフォルトでは、低電力モードとしてスリープが有効になっています。MCU をスリープから復帰させ通常のプログラム実行モードに戻す適切な割り込みまたはイベントを設定および有効化することを除き、特別な設定はスリープ モードに不要であるため、このモードは使用できる最も便利な低電力モードです。さらに、スリープ モード時には SRAM、プロセッサ レジスタ、およびハードウェア周辺機器の各状態がすべて維持されるので、スリープへの移行とスリープからの復帰に必要な時間も最小限で済みます。MCU をスリープ モードから復帰させることができるイベントと割り込みの完全な一覧については、MCU ユーザーズマニュアルの表「ノーマル モードに戻すための使用可能な割り込みソース」を参照してください。スリープを MCU のローパワー モードとして使用するように MCU を設定するには、LPM API 関数 lowPowerCfg を使用する必要があります。スリープ モードに直接移行するには、LPM API 関数 lowPowerModeEnter を使用する必要があります。

低電力モードとしてスリープを設定し、スリープ モードに切り替えるコード例を、以下に示します。

```
/* HAL-only entry function */

#include "hal_data.h"

void hal_entry(void)

{

    /* Not shown: Enable any interrupt to wake from sleep mode */

    /* Configure LPM peripheral for sleep mode */

    g_lpm.p_api->lowPowerCfg(LPM_LOW_POWER_MODE_SLEEP,

                             LPM_OUTPUT_PORT_ENABLE_RETAIN,

                             LPM_POWER_SUPPLY_DEEPCUT0,

                             LPM_IO_PORT_NO_CHANGE);

    /* Enter sleep mode */

    g_lpm.p_api->lowPowerModeEnter();

}
```

「ソフトウェア スタンバイ モード」

ソフトウェア スタンバイ モードでは、CPU、オンチップ周辺機能の大部分、およびすべての内部発振器が停止します。ただし、CPU 内蔵レジスタと SRAM データの内容、およびオンチップ周辺機能と I/O ポートの状態は保持されます。ソフトウェア スタンバイ モードでは大半の発振器が停止するため、このモードを使用すると、消費電力を大幅に削減できます。スリープ モードと同様、スタンバイ モードの場合にも、スタンバイ モードからの復帰のために割り込みまたはイベントを設定して有効化する必要があります。MCU をスリープ モードから復帰させることができるイベントと割り込みの完全な一覧については、MCU ユーザーズマニュアルの表「ノーマル モードに戻すための使用可能な割り込みソース」を参照してください。ソフトウェア スタンバイ モードに入る前に、Wake Up 割り込み有効化レジスタ （WUPEN）を変更しておく必要があります。詳細は、「Wake Up 割り込み有効化レジスタ （WUPEN）」および MCU ユーザーズマニュアルのソフトウェア スタンバイ モードのセクションをご覧ください。

```
/* HAL-only entry function */

#include "hal_data.h"

#define WUPEN_AGT1_UNDERFLOW_MASK (1 << 28)

void hal_entry(void)

{

    uint32_t wupen_value = 0;

    /* Not shown: Configuration of the interrupt or event to wake from standby */

    /* Set bit in WUPEN register to allow mcu to wake from standby through a specific

       interrupt or event. AGT1 underflow is used in this example.

    */

    /* Get current value of WUPEN register */

    g_lpm.p_api->wupenGet(&wupen_value);

  /* Set wake by AGT1 underflow bit in WUPEN register */

    g_lpm.p_api->wupenSet(wupen_value | WUPEN_AGT1_UNDERFLOW_MASK);

  /* Configure LPM peripheral for standby mode */

    g_lpm.p_api->lowPowerCfg(LPM_LOW_POWER_MODE_STANDBY,

                             LPM_OUTPUT_PORT_ENABLE_RETAIN,

                             LPM_POWER_SUPPLY_DEEPCUT0,

                             LPM_IO_PORT_NO_CHANGE);

    /* Enter standby mode */

    g_lpm.p_api->lowPowerModeEnter();

}
```

ソフトウェア スタンバイ モードを使用したスヌーズ モード

スヌーズ モードでは、MCU 周辺機器を使用して、MCU をローパワー ステータスで維持しながら、基本タスクを実行することができます。ADC や DTC、その他の周辺機器はスヌーズ モードで有効にできます。次

のコードはスヌーズ モードの有効化処理とスヌーズ中の DTC の実行処理を示します。タイマ AGT1 の設定処理は記されていません。MCU をソフトウェア スタンバイ モードにすると、AGT1 の比較一致 A が検出されたときにスヌーズに移行します。DTC の転送が完了すると、MCU はスヌーズ モードではなくなります。DTC の詳細については、DTC の使用上の注意を参照してください。

ISDE によって生成されたソース ファイルで transfer_on_dtc driver を格納しプロジェクトを作成すると、オーディオおよびメッセージ フレームワーク モジュール用に以下のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */
const transfer_instance_t g_transfer =
{
    .p_ctrl = &g_transfer_ctrl,
    .p_cfg = &g_transfer_cfg,
    .p_api = &g_transfer_on_dtc
};
```

```
#define WUPEN_AGT1_UNDERFLOW_MASK (1 << 28)
/* DTC settings */
g_transfer.p_cfg->p_info->p_src = &my_tx_data[0];
g_transfer.p_cfg->p_info->p_dest = &my_rx_data[0];
g_transfer.p_cfg->p_info->length = TRANSFER_SIZE;
g_transfer.p_api->open(g_transfer.p_ctrl, g_transfer.p_cfg));
/* snooze settings */
lpm_snooze_rxd0_t rdx0_mode = LPM_SNOOZE_RXD0_FALLING_EDGE_IGNORE;
lpm_snooze_dtc_t dtc_mode = LPM_SNOOZE_DTC_ENABLE;
lpm_snooze_request_t requests = LPM_SNOOZE_REQUEST_AGT1_COMPARE_A;
lpm_snooze_end_t triggers = LPM_SNOOZE_END_DTC_TRANS_COMPLETE;
g_lpm.p_api->snoozeEnable(rdx0_mode,
                dtc_mode,
                        requests,
```

```
                                triggers);

/* standby settings */

lpm_low_power_mode_t  power_mode = LPM_LOW_POWER_MODE_STANDBY;

lpm_output_port_enable_t  output_port_enable = LPM_OUTPUT_PORT_ENABLE_RETAIN;

lpm_power_supply_t  power_supply = LPM_POWER_SUPPLY_DEEPCUT0;

lpm_io_port_t  io_port_state = LPM_IO_PORT_RESET;

g_lpm.p_api->lowPowerCfg(power_mode,

                output_port_enable,

                                power_supply,

                                io_port_state);

/* Clear WUPEN */

g_lpm.p_api->wupenSet(0);

/* Wake from AGT1 interrupt underflow */

g_lpm.p_api->wupenSet(WUPEN_AGT1_UNDERFLOW_MASK);
```

ディープ ソフトウェア スタンバイ モード

ディープ ソフトウェア スタンバイ モードは S7G2 でのみ利用可能であり、ソフトウェア スタンバイよりもローパワー モードです。リセットピンのネゲート、または API typedef *lpm_deep_standby_t* で記述されるスリープ解除イベントのセットのいずれかによるリセットが発生した場合、MCU は毎回ディープ ソフトウェア スタンバイ モードから復帰します。以下のコードは、AGT1 アンダーフローを使用して MCU をディープ スタンバイ モードからスリープ解除し、ディープ ソフトウェア スタンバイ モードを有効にしてそのモードに移行する方法を示します。AGT1 が期限切れになり MCU のリセットを引き起こすと、MCU はリセットされます。

```
#define WUPEN_AGT1_UNDERFLOW_MASK (1 << 28)

/* Deep standby wake signal (wake will cause reset) */

g_lpm.p_api->deepStandbyCancelRequestEnable(LPM_DEEP_STANDBY_AGT1,

   LPM_CANCEL_REQUEST_EDGE_RISING);

/* Deep standby settings */

lpm_low_power_mode_t  power_mode = LPM_LOW_POWER_MODE_DEEP;

lpm_output_port_enable_t  output_port_enable = LPM_OUTPUT_PORT_ENABLE_RETAIN;

lpm_power_supply_t  power_supply = LPM_POWER_SUPPLY_DEEPCUT0;

lpm_io_port_t  io_port_state = LPM_IO_PORT_RESET;

g_lpm.p_api->lowPowerCfg(power_mode,

                output_port_enable,

                        power_supply,

                        io_port_state);

/* Clear WUPEN */

g_lpm.p_api->wupenSet(0);

/* Wake from AGT1 interrupt underflow */

g_lpm.p_api->wupenSet(WUPEN_AGT1_UNDERFLOW_MASK);

g_lpm.p_api->lowPowerModeEnter();
```

4.2.21.5 LPM ドライバ使用上の注意

- このドライバを使用して動作モードを変更するには、CGC モジュールを使用する必要があります。CGC ドライバに関する使用上の注意も確認してください。MCU の動作モードを適切に変更するために必要なイベントのシーケンスの詳細については、MCU ユーザーズマニュアルの「Low Power Modes」セクションを参照してください。
- このドライバを使用して割り込みに基づいて LPM 周辺機器を設定するには、BSP を使用して割り込みを設定および有効化する必要があります。
- LPM API 関数 g_lpm.p_api->init() の呼び出しによって設定されるのは、MCU の動作モードのみです。**init** 関数は、ローパワー モードの設定は行いません。

- ローパワー モードを設定するには、LPM API 関数 g_lpm.p_api->lowPowerCfg() を使用します。これを設定しておくと、LPM API 関数 g_lpm.p_api->lowPowerModeEnter() を使用して、いつでもローパワー モードに移行することができます。
- ディープ スタンバイまたはスヌーズを使用するには、LPM API 関数 g_lpm.p_api->deepStandbyCancelRequestEnable() および g_lpm.p_api->snoozeEnable を用いた LPM 周辺機器の追加設定が必要です。
- ソフトウェア スタンバイ モードに入る前に、Wake Up 割り込み有効化レジスタ （WUPEN）を変更しておく必要があります。詳細は、「Wake Up 割り込み有効化レジスタ （WUPEN）」および MCU のユーザーズマニュアルのソフトウェア スタンバイ モードのセクションをご覧ください。
- サブ発振器電力制御モードに移行するには、API 関数 g_lpm.p_api->operatingPowerModeSet を呼び出す前に、CGC モジュールを使用して MOCO クロックおよび HOCO クロックを止める必要があります。
- メイン発振器またはメイン発振器ソースを備えた PLL をシステム クロックに使用している場合、スタンバイ モードからの復帰時間は MOSCWTCR レジスタでのメイン発振器待ち時間により影響を受ける可能性があります。このレジスタ設定は、CGC ドライバ プロパティの [メイン発振器待ち時間設定] から変更可能です。詳細については、「電気的特性」の「ウェイクアップのタイミングおよび時間（Wakeup Timing and Duration）」表を参照してください。

4.2.21.6 LPM ドライバの割り込み

LPM ドライバは割り込みを直接使用しません。

4.2.21.7 LPM コールバック

LPM ドライバはコールバックを直接使用しません。

4.2.21.8 LPM ドライバの制限事項

- フラッシュの停止 （コード フラッシュの無効化）はサポートされていません。『S3A7 Synergy ハードウェア ユーザーズ マニュアル』の「フラッシュ操作制御レジスタ （FLSTOP）」セクションを参照してください。
- ソフトウェア スタンバイ モードでの SRAM 保持領域の縮小は、現時点ではサポートされていません。『S3A7 Synergy ハードウェア ユーザーズ マニュアル』の「省電力メモリ制御レジスタ （PSMCR）」セクションを参照してください。
- MCU はデバッガが付属した状態のソフトウェア スタンバイ モードやディープ ソフトウェア スタンバイ モードに移行したり、状態を維持したりすることはできません。MCU はデバッガにより、ソフトウェア スタンバイ モードやディープ ソフトウェア スタンバイ モードから起動することができます。ソフトウェア スタンバイ モードやディープ ソフトウェア スタンバイ モードを正確にテストし、確認するためにデバッガを付属させる必要があります。
- メイン発振器またはメイン発振器ソースを備えた PLL をシステム クロックに使用している場合、スタンバイ モードからの復帰時間は MOSCWTCR レジスタでのメイン発振器待ち時間により影響を受ける可能性があります。このレジスタ設定は、CGC ドライバ プロパティの [メイン発振器待ち時間設定

] から変更可能です。詳細については、「電気的特性」の「ウェイクアップのタイミングおよび時間（Wakeup Timing and Duration）」表を参照してください。

4.2.21.9 LPM ドライバでサポートされるデバイス

このドライバは、次のデバイスでテストされています。

- S7G2
- S3A7
- S124

4.2.21.10 LPM ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /synergy/ssp/ ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL LPM Interface API	synergy/ssp/inc/driver/api/r_lpm_api.h
LPM Instance	synergy/ssp/inc/driver/instances/r_lpm.h
LPM Driver Source Files	synergy/ssp/src/driver/r_lpm

4.2.22 低電圧検出ドライバ

低電圧検出ドライバ （LVD Driver）は r_lvd 上に実装されており、ローパワー モード ハードウェア周辺機器の構成へのアクセスを提供します。

このセクションでは、e^2 studio ISDE を使用して LVD ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [パワー] > [r_lvd 上の低電圧検出] を選択することで、LVD ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による LVD ドライバを使用するアプリケーションの作成

API リファレンスは、次の HAL LVD インタフェースの説明内に記載されています : 低電圧検出ドライバインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

LVD API および LVD インスタンスのユース ケースの概要については、以下をクリックしてください。

- LVD ドライバおよび API 低電圧検出ドライバインタフェース

- LVD API 低電圧検出ドライバインタフェース
- LVD HAL ドライバ LVD

4.2.22.1 LVD ドライバの機能

LVD ドライバは、MCU の構成可能な LVD 監視の構成をサポートします。LVD ドライバは、単一の LVD 監視の完全な構成に必要となるすべての情報を提供する、構成構造体を提供します。LVD 監視のインスタンスごとに、LVD ドライバのインスタンス 1 点が必要です。ただし LVD0 監視は例外で、これは実行時の構成が不可能であるため、OFS1 レジスタを介してコンパイル時に構成する必要があります。

LVD1 と LVD2 監視は、どちらも実行時に構成可能で、構成はこのドライバを用いて行います。open 関数を使うと、単一の関数呼び出しにより LVD 監視を構成し、有効化することができます。close 関数は、LVD 監視の無効化に使用します。statusGet 関数は、LVD 監視の現在のステータスを返します。ドライバがポーリング モードにある場合は、statusGet 関数を用いる必要があります （つまり、LVD 監視割り込みを有効化しない)。監視のステータスは 2 つのフラグで構成されます。最初のフラグは crossing_detected と呼ばれるラッチ フラグで、このフラグは監視対象となる電圧が電圧のしきい値を超過したかどうかを示します。ポーリング モードでは、このフラグは statusClear への呼び出しを通じてクリアする必要があります。LVD 割り込みが使用中である場合を除き、このフラグを明示的にクリアする必要はありません。LVD 割り込みでは、このフラグはユーザー コールバック関数の呼び出し後にドライバ コードによりクリアされます。もう一方のフラグは current_state で、これは監視対象となる電圧のしきい値に対する瞬間的なステータスです。このフラグはラッチされておらず、監視対象となる電圧の変化に伴い変化します。

4.2.22.2 e² studio ISDE による LVD ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照)。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照)。
2) プロジェクトを設定します （プロジェクトの設定を参照)。
3) ドライバを追加します （スレッドとドライバの追加を参照)。

LVD ドライバは、HAL/Common の一部または ThreadX スレッドの一部として、あるいはそれら両方の一部として、Synergy プロジェクトに追加できます。

LVD ドライバを HAL/Common に追加するには、次の手順を実行します。

- [Synergy 設定] タブを開きます
- [Threads] ウィンドウの HAL/ 共通をハイライトして、
- [HAL/ 共通モジュール] ウィンドウの [新規] を左クリックします
- [Driver] > [パワー] の順にマウスカーソルを合わせます
- [Driver] > [パワー] から [r_lvd 上の低電圧検出] を選択します

LVD ドライバを ThreadX スレッドに追加するには、次の手順を実行します。

- [Synergy 設定] タブを開きます

- [Threads] ウィンドウのスレッドをハイライトして、
- [< スレッド名 > モジュール] ウィンドウで [新規] を左クリックします。
- [Driver] > [パワー] の順にマウスカーソルを合わせます
- [Driver] > [パワー] から [r_lvd 上の低電圧検出] を選択します

LVD ドライバ モジュールを、複数のスレッド、または 1 つのスレッドと HAL/Common に追加する場合は、LPM ドライバのインスタンスごとに固有の名前を付けてください。名前を変更するには、[Modules] ウィンドウでドライバのインスタンスをハイライトして、[Properties] ビューで名前のエントリを変更します。LVD ドライバの設定構造体のデフォルト設定も、このビューで変更できます。

4.2.22.3 LVD ドライバの設定

次のオプションを実行することで、MCU で利用可能な構成可能 LVD 周辺機器を使用するよう、設定を構成できます。利用可能なオプションとそのデフォルト設定については、以下をご覧ください。

プロパティ	デフォルト
監視番号	1
デジタル フィルタ、有効なサンプル クロック レートを選択することで有効化 （S7G2 のみ）。	デジタル フィルタが無効化されています。
電圧のしきい値。	2.85V （Vdet1_13）（S7G2 のみ）。
検出レスポンス。リセット、割り込み、マスク不可能な割り込み、レスポンスなし （ポーリング モード）のいずれか。	電圧が検出のしきい値を超えた場合にトリガされるマスク可能な割り込み。
電圧スロープ。上昇または下降、あるいは両方。	電圧の低下に伴い検出されたしきい値の超過。
監視信号のネゲートは、リセット イベントまたは電圧の通常範囲内への復帰後に遅延可能です。	リセット信号のネゲートは、リセットからの遅延に基いています。
監視割り込みコールバック	NULL

これらの設定に対し適切な値を選択した後、ユーザーはプロジェクトに対し、LVD ドライバの open API 関数を呼び出すコードを追加する必要があります。この関数は、プログラムの寿命の初期に 1 回呼び出す必要があります。また、1 つまたは複数の LVD 周辺機器を変更する必要がある場合はいつでも関数を呼び出すことができます。LVD API 関数 open を呼び出すことで、有効化されたすべての LVD 監視に対し LVD ハードウェア周辺機器が有効になり、また無効化されたすべての LVD 監視に対し LVD ハードウェア周辺機器が無効になります。ユーザーは、open 関数により単一の LVD 周辺機器の構成を更新でき、また close 関数により LVD 周辺機器を無効化し、ドライバを閉じることができます。

次の構造体は、LVD の 1 つのインスタンスを記述したものです。

```
const lvd_instance_t g_lvd =

{

  .p_ctrl = &g_lvd_ctrl,

  .p_cfg = &g_lvd_cfg,

  .p_api = &g_lvd_on_lvd

};
```

プロジェクトに低電圧検出ドライバが追加された場合、e^2 studio ISDE は次のコードを生成します。

```
void hal_entry(void) {

  g_lvd.p_api->open(g_lvd.p_ctrl, g_lvd.p_cfg);

 ... (application code) ...

}
```

各 LVD 監視は、以下に示す API 関数 open を使用することで、いつでも個別に （再）構成および有効化することができます。

```
g_lvd.p_api->open(g_lvd.p_ctrl, g_lvd.p_cfg;
```

各 LVD 監視は、以下に示す API 関数 close を使用することで、いつでも個別に無効化することができます。

```
g_lvd.p_api->close(g_lvd.p_ctrl, g_lvd.p_cfg);
```

API 関数 s**tatusGet** を使うと、VCC が open 関数により設定した電圧しきい値を超過したかどうかを判断できます。このステータス フラグは電圧しきい値が超過した場合に設定およびラッチされ、API 関数 statusClear を使って明示的にクリアする必要があります。

```
lvd_status_t  monitor_status;

g_lvd.p_api->statusGet(g_lvd.p_ctrl, &monitor_status);

if(LVD_THRESHOLD_CROSSING_DETECTED  == monitor_status.crossing_detected)

{

  g_lvd.p_api->statusClear(g_lvd.p_ctrl);

}
```

電圧しきい値に対する VCC のアンラッチ（瞬間的な）ステータスは、API 関数 statusGet を使って取得する必要があります。このフラグは、の current_state メンバーです。このステータス フラグは、明示的にクリアする必要はありません。

```
lvd_status_t  monitor_status;

g_lvd.p_api->statusGet(g_lvd.p_ctrl, &monitor_status);

if(LVD_CURRENT_VOLTAGE_BELOW_THRESHOLD == monitor_status.current_state)

{

  . . .

}
```

close 関数を呼び出すことで、すべての LVD 周辺機器を無効化してドライバを閉じることができます。

```
g_lvd.p_api->close(g_lvd.p_ctrl);
```

4.2.22.4 LVD ドライバ使用上の注意

- このドライバを使用して LVD 周辺機器を構成し、割り込みを作成するには、Synergy 構成の [ICU] タブで対応する割り込みを有効化する必要があります。
- LVD 割り込みを使用する際、コールバック関数は必須ではありませんが、使用が推奨されます。
- 各 LVD 割り込みに対する固有のコールバック関数は必須ではありませんが、使用が推奨されます。
- クロック システムのクロックの初期化、構成、実行時変更は、このモジュール外で処理されます。このドライバは、ユーザーが選択したサンプル クロックの除算に基づくデジタル フィルタ サンプル クロックのみを変更します。このデジタル フィルタ サンプル クロックは、LOCO システム クロックから得られたものです。

- すべての MCU ですべての電圧しきい値が利用可能とは限りません。
- すべての MCU で LVD 監視に対する VCC 入力のデジタル フィルタリングが利用可能とは限りません。

LVD ドライバの依存関係

- LVD ドライバには、BSP が提供する機能が必要です。このドライバは、BSP がロックのレジスタ、ならびに割り込みの有効化およびクリアを目的として提供するハードウェア ロックを使用します。

4.2.22.5 LVD ドライバの割り込み

LVD ドライバは、1 つまたは複数の LVD 周辺機能割り込みを有効化するよう構成できます。割り込みを使用する場合、ユーザーはその割り込みに対するコールバック関数を提供する必要があります。LVD 割り込みに対するレスポンスが LVD 監視間で異なる場合は、各 LVD 割り込みに対し個別のコールバック ルーチンを提供する必要があります。つまり、監視 1 への割り込みに対するユーザーのレスポンスが周辺機器のシャットダウンで、監視 2 への割り込みに対するレスポンスが単に接続デバイスへのメッセージ送信である場合、監視 1 と 2 に対し個別の割り込みコールバック ルーチンが必要となります。

LVD コールバック

いずれかまたは両方の LVD 割り込みが有効化されている場合、LVD ドライバはコールバック ルーチンを使うことで、ユーザーがアクションを実行できるようにします。署名は以下のとおりで、r_lvd_api.h に定義されています。

```
void (* p_callback)(lvd_callback_args_t * p_args);
```

コールバック パラメータ構造体には、監視のステータス （crossing_detected、current_state）、監視番号、LVD ドライバ インスタンスに対するコンテキスト ポインタが含まれます。

4.2.22.6 LVD ドライバの制限事項

- 低電圧検出監視を構成 / 有効化するプロセスは、具体的な時間的制約とレジスタの書き込み順序に拘束されます。このような制約があるため、電圧監視を設定 / 有効化するプロセスは、全体を単一の関数で処理するのが最も効率的です。API 関数 open は、タイミングとレジスタの書き込み順序の制約を正しく強制するように、構成を実行し、監視を有効にします。
- MCU は、実行時には構成不可能である LVD ハードウェアを含む場合があります。このドライバは、これらの LVD 監視に対処しません。たとえば、S7G2 および S3A7 上の LVD 監視 0 は、OFS1 レジスタを介してのみ構成可能です。このレジスタは、実行時の変更はできませんが、[BSP プロパティ] タブの OFS1 レジスタ設定を通じて変更可能です。OFS1 レジスタの詳細については、MCU のユーザーマニュアルのセクション 7.2.2 を参照してください。ビット フィールドおよび設定の概要については以下をご覧ください。

4.2.22.7 LVD ドライバでサポートされるデバイス

このドライバは、次のデバイスでテストされています。

- S7G2

- S3A7
- S124

4.2.22.8 LVD ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /synergy/ssp/ ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL LVD Interface API	synergy/ssp/inc/driver/api/r_lvd_api.h
LVD Instance	synergy/ssp/inc/driver/instances/r_lvd.h
LVD Driver Source Files	synergy/ssp/src/driver/r_lvd

4.2.23 PDC ドライバ

パラレルデータキャプチャユニット（PDC）ドライバは、カメラ モジュールの画像をキャプチャするための汎用 API です。PDC ドライバは r_pdc 上に実装されており、MCU で利用可能な PDC 周辺機器をサポートします。このセクションでは、e^2 studio ISDE を使用して PDC ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [新規] > [Driver] > [グラフィックス] > [r_pdc 上の PDC ドライバ] を選択することで、PDC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による PDC ドライバを使用するアプリケーションの作成

API リファレンスは、次の HAL PDC インタフェースの説明内に記載されています : PDC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

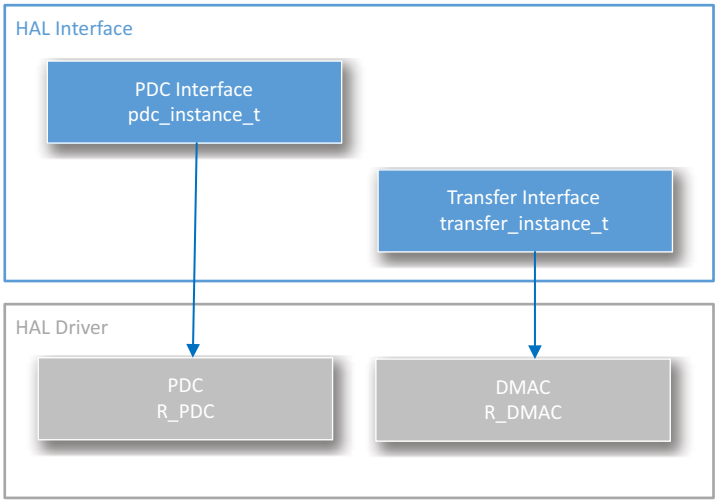

図 147: PDC ドライバ - ブロック図

4.2.23.1 PDC ドライバの機能

このドライバは、ユーザーの設定に従って MCU 上の PDC を制御します。接続された構成済みのカメラからキャプチャを開始します。キャプチャが完了し、コールバックを使用できる場合 （割り込みが有効）、ドライバはコールバックを呼び出します。その際、キャプチャ イメージが保存されたバッファへのポインタを示す引数 pdc_callback_args_t が渡されます。また、コールバックを生じされるイベント pdc_event_t も提供されます。

キャプチャ動作

キャプチャ動作には、MCU に接続された構成済みの外部カメラが必要です。キャプチャを実行するには、事前にカメラが構成されていること、かつマイクロコントローラに対し PIXCLK クロック入力が生成されていることが重要です。一部のインスタンスでは、カメラの構成には実行中のクロック入力が必要となる場合があります。このため、カメラを初期化する前には、PDC からカメラへの PCKO クロック出力を構成および開始する open を呼び出すようにしてください。カメラの構成が済んだら、captureStart を呼び出してイメージをキャプチャします。カメラ モジュールの構成には、IIC または SPI インタフェースの使用が必要となる場合があります。

ほとんどのインスタンスにおいて、カメラおよび PDF 周辺機器からのデータ レートは、ISR 内の CPU でのサービスに対して速すぎます。このため、PDC 周辺機器およびメモリからの高速転送を実行するには、このモジュールでは DMAC への転送ドライバの実装が必要となります。

4.2.23.2 e² studio ISDE による PDC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

参考資料

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

PDC インタフェースを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
PDC Driver	Threads	[Driver] > [グラフィックス] > [r_pdc 上の PDC ドライバ]
Interrupts	ICU	[PDC] > [PDC FRAME END および PDC INT]
PDC Clock	Clocks	アプリケーション例については、PDC クロックの設定
PDC Pins	Pins	アプリケーション例については、PDC ピンの設定
Transfer Driver	Threads	[Driver] > [転送] > [r_dmac 上の転送ドライバ]
Interrupts	ICU	[DMAC] > [DMACn] > [DMACn INT] （n=0 ～ 7）
Event Link Controller	Threads	IDSE には、デフォルトで ELC の 1 つのインスタンスが含まれます

PDC クロックの設定

e2 studio ISDE で、[Clocks] タブを使用して PDC クロックを設定します （クロックの設定を参照）。

PDC は PCLKB をそのクロック ソースとして使用します。このクロックの構成に関する唯一の制限事項は、PPCLKB 周波数がこれに応じて設定されるよう、PIXCLCK が 0.6 x PCLKB 未満でなければならない点です。

PCLKB の周波数を設定するには、e2 studio ISDE のクロック コンフィギュレータを使用します （クロックの設定を参照）。

実行時にクロック周波数を変更するには、CGC インタフェースを使用します。

PDC ピンの設定

e2 studio ISDE を使用して、[Pins] タブから PDC ピンを設定します （ピンの設定を参照）。

PDC を使用するには、ピン コンフィギュレータで PDC のポート ピンを PDC 周辺機器モードに設定する必要があります。ピン コンフィギュレータは、pin_cfg フィールドで関連するピンの PDC ピン設定を適切に構成します。PDC ピンが他のモジュールと機能を共有している場合、衝突はすべて解決されます。

PDC 割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから PDC 割り込みを設定します （割り込みの設定を参照）。

PDC フレーム終了割り込みと PDC エラー割り込みを有効化する必要があります。これにより、次の状況で割り込みが生成されます。

1) イメージがキャプチャされた場合の割り込み （フレーム終了）
2) エラーが発生した場合の割り込み （int）

I：割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、PDC > PDC FRAME END および PDC > INT の優先度を設定します （割り込みの設定を参照）。

これにより、synergy_cfg/ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_PDC_FRAME_END と BSP_IRQ_CFG_PDC_INT に、選択した優先度が設定されます。

これらの割り込みが BSP で有効になっている場合、対応する割り込みサービス ルーチン （ISR）が PDC ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、そのコールバック関数を呼び出します。

転送インタフェースもまた、完全なイメージがキャプチャされた時点で、割り込みが有効化されている必要があります。

Note：割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、DMAC > DMACn > DMACn INT （n は使用される DMAC チャネル）の優先度を設定します （割り込みの設定を参照）。

これにより、synergy_cfg/ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_DMACn_INT に、選択した優先度が設定されます。

この割り込みが発生すると、ISR は open においてコールバック関数が登録された場合に PDC コールバックを呼び出します。

PDC パラメータの設定

e2 studio ISDE を使用して、PDC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

Note：PDC ドライバでは、PDC の設定 （pdc_cfg_t）と転送インタフェースの設定 （transfer_cfg_t）の 2 つを行う必要があります。

ISDE プロパティ	設定	設定値	説明
Name of the data buffer to store image data	p_buffer	User-defined	作成するデータ バッファの名前を指定するか、ユーザーが PDC ドライバ外にデータ バッファを作成する場合は NULL に設定します。
Section where data buffer is allocated	-	User-defined	イメージ データ バッファの RAM セクションを指定します。通常は bss （内部 RAM）または sdram です。
Number of bytes per pixel	bytes_per_pixel	User-defined	キャプチャするイメージ データのピクセルごとのバイト数を指定します。
Number of image buffers	-	User-defined	作成するバッファの数を指定します。
Clock divider	clock_division	pdc_clock_division_t	PDC 周辺機器に対するクロック入力のクロック分周器を指定します。
Endian of image data	endian	pdc_endian_t	キャプチャするイメージ データのエンディアンを指定します。
HSYNC signal polarity	hsync_polarity	pdc_hsync_polarity_t	HSYNC 信号のアクティブ極性を指定します。
VSYNC signal polarity	vsync_polarity	pdc_vsync_polarity_t	VSYNC 信号のアクティブ極性を指定します。
Number of pixels to capture horizontally	x_capture_pixels	User-defined	キャプチャする水平ピクセル数。
Number of lines to capture vertically	y_capture_pixels	User-defined	キャプチャする垂直ライン数。
Horizontal pixel to start capture from	x_capture_start_pixel	User-defined	イメージ データのキャプチャを開始する水平位置。カメラのネイティブ解像度を下回るイメージのキャプチャが可能になります。

ISDE プロパティ	設定	設定値	説明
Line to start capture from	y_capture_start_pixel	User-defined	イメージ データのキャプチャを開始する垂直ライン。カメラのネイティブ解像度を下回るイメージのキャプチャが可能になります。
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、フレームがキャプチャされ処理可能な状態になるたびに、割り込みサービス ルーチン （ISR）から呼び出されます。 Attention : コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
Name	-	User-defined	-
Lower level transfer name	p_lower_lvl_transfer	transfer_instance_t	PDC ドライバで使用する転送インタフェースへのポインタ。

この表において、「プロパティ」は ISDE 内での [Properties] タブ名を表します。

転送インタフェース構成

ISDE プロパティ	設定	設定値	説明
Name	p_context	User-defined	転送インタフェースで使用するインタフェース名。

4.2.23.3 PDC ドライバ使用上の注意

データ バッファの設定

p_buffer が NULL 以外に設定されている場合、イメージ データの保存用に 1 つまたは複数のデータ バッファが作成されます。各バッファのサイズは、以下の式を使って計算されます。

バッファ サイズ （バイト）= x_capture_pixels x y_capture_pixels x bytes_per_pixel

解像度の高いカメラの場合、キャプチャしたイメージのデータが膨大な量になることがあります。このため、バッファを外部メモリ （SDRAM など）に作成することが必要となる場合があります。外部メモリを使用する場合は、バス帯域を考慮してください。たとえば、PDC を介して高フレームレート カメラで SDRAM へのキャプチャリングを行い、高リフレッシュ レートを備えた LCD ディスプレイ用にディスプレイ バッファを保持するのに SDRAM を使用すると、PDC からメモリへのデータ ボトルネックが発生し、これがオーバーラン エラー状態につながります。

p_buffer が NULL に設定されている場合、キャプチャするイメージ データの保存にはメモリが割り当てられません。このため、ユーザーが責任をもって、PDC に十分なサイズのメモリを確保する必要があります。たとえば、PDC は接続した LCD パネルのディスプレイ バッファに直接キャプチャを行うことができます。

4.2.23.4 PDC アプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const pdc_instance_t g_pdc =

{

    .p_ctrl = &g_pdc_ctrl,

    .p_cfg = &g_pdc_cfg,

    .p_api = &g_pdc_on_pdc

};
```

PDC HAL インタフェースを使用して PDC アプリケーションを作成するには、以下の手順を実行します。

1) r_pdc 上の g_pdc PDC ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが制御構造体 pdc_ctrl_t に加えて、src/synergy_gen フォルダ内に自動生成されます。

2) アプリケーション コードを hal_entry.c に追加します。

   Note：ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。

3) PDC インスタンスを開きます。PDC インタフェースを通じて PDC ドライバが呼び出されます。

```
g_pdc.p_api->open(g_pdc.p_ctrl, g_pdc.p_cfg)
```

   g_pdc.p_ctrl と g_pdc.p_cfg は、PDC の設定ステップの後で自動生成されます。

4) 必要に応じ、接続されたカメラを設定します。次の手順に進むには、カメラが PDC 周辺機器に対し PIXCLK 入力を生成している必要があります。

5) 以下を呼び出すことで、PDC が作成したバッファへのイメージのキャプチャを開始できます。

```
g_pdc.p_api->captureStart(g_pdc.p_ctrl, NULL)
```

   2 番目のパラメータへのポインタを渡すことで、PDC ドライバが open で指定したバッファを使用するようになります。異なるバッファを使用する場合は、captureStart

を呼び出す場合にこれを 2 番目のパラメータとして指定してください。

6) イメージをキャプチャすると、open で指定したコールバックが呼び出されます。pdc_callback_args_t パラメータでイベントのタイプとバッファのアドレスを確認します。コールバックは ISR から呼び出されるため、実行はできるだけ短くするようにしてください。

7) HSYNC と VSYNC ピンの状態が必要な場合は、次を呼び出すことで取得できます。

```
g_pdc.p_api->stateGet(g_pdc.p_ctrl,
 &state_data)
```

   変数 state_data は pdc_state_t 型で、ユーザーにより提供されます。

8) PDC インスタンスをクローズするには、以下を呼び出します：

```
g_pdc.p_api->close(g_pdc.p_ctrl))
```

シンプルな PDC アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src/ ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。ThreadX アプリケーションの場合、PDC コールバックからセマフォをポストすることで、完了したイメージ キャプチャを示すことができます。

4.2.23.5 PDC の制限事項

PDC ドライバとその実装の制限事項については、SSP のリリースノートを参照してください。

4.2.23.6 PDC ドライバでサポートされるデバイス

PDC インタフェースは、PDC 周辺機器ブロックを使用して、S7G2 でテストされています。

4.2.23.7 PDC ファイル

プロジェクト設定中に、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL PDC Interface API	synergy/ssp/inc/driver/api/r_pdc_api.h
HAL Transfer Interface API	synergy/ssp/inc/driver/api/r_transfer_api.h
PDC Instance	synergy/ssp/inc/driver/instances/r_pdc.h
DMAC Instance	synergy/ssp/inc/driver/instances/r_dmac.h
PDC Driver	synergy/ssp/src/driver/r_pdc
DMAC Driver	synergy/ssp/src/driver/r_dmac

4.2.24 QSPI ドライバ

QSPI ドライバは r_qspi に実装されており、クワッド SPI インタフェースを通じてマイクロコントローラに接続された QSPI フラッシュ デバイスの内容を消去およびプログラミングするための汎用 API を含みます。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [System**] > [QSPI**Driver on r_qspi] を選択することで、QSPI ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による QSPI ドライバを使用するアプリケーションの作成

API リファレンスは、次の HAL QSPI インタフェースの説明内に記載されています : クワッド SPI フラッシュインタフェース。

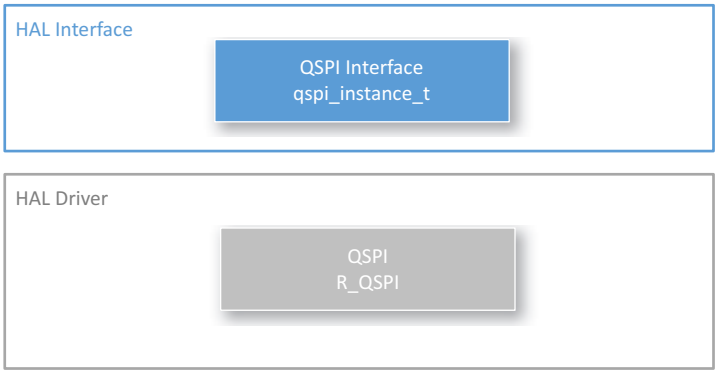

図 148: QSPI - ブロック図

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

4.2.24.1 QSPI ドライバの機能

QSPI ドライバは、以下の機能をサポートしています。

- 直接通信モードを使用したクワッド SPI フラッシュ デバイスへのアクセス。
- QSPI フラッシュ デバイスのデータの読み取り。
- QSPI フラッシュ デバイスのページのプログラミング。
- QSPI フラッシュ デバイスのセクターの消去。
- QSPI フラッシュ デバイスへのアクセスを制御するバンクの選択。

4.2.24.2 e² studio ISDE による QSPI ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

QSPI ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
QSPI Driver	Threads	[Driver] > [Storage] > [QSPI on r_qspi]

QSPI ドライバ用のクロックの設定

すべての QSPI デバイス設定は、BSP で行います。

QSPI ドライバ割り込みの設定

QSPI 割り込みは使用されないため、有効化されることはありません。

QSPI のコールバック ISR 関数の定義

コールバックはありません

QSPI ドライバ パラメータの設定

設定する QSPI パラメータはありません。

4.2.24.3 QSPI ドライバ アプリケーションの作成

一般的な QSPI アプリケーションは、QSPI フラッシュ デバイスでデータをプログラムまたは消去します。このドライバが開いていない場合、QSPI フラッシュ デバイスのコンテンツは 0x60000000 にマップされ、通常のメモリであるかのように読み取ることができます。

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

QSPI ドライバの例

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const qspi_instance_t  g_qspi =

{

    .p_ctrl  = &g_qspi_ctrl,

    .p_cfg = &g_qspi_cfg,

    .p_api = &g_qspi_on_qspi

};
```

QSPI ドライバを使用して QSPI アプリケーションを作成するには、次の手順を実行します。

1) QSPI の選択した QSPI モジュール g_qspi QSPI ドライバについて、ISDE の [Properties] タブを使用してモジュールを設定します。モジュールを設定すると、ISDE により、モジュール関連のヘッダーと設定ファイルが制御構造体 qspi_ctrl_t に加えて、src/ssp_gen フォルダ内に自動生成されます。
2) アプリケーション コードを hal_entry.c に追加します。

   Note：ThreadX を使用している場合は、選択したスレッド my_thread_entry.c にこのコードを追加します。
3) QSPI インスタンスをオープンします。QSPI ドライバは、次の QSPI インタフェースを通じて呼び出されます。

```
g_qspi.p_api->open(g_qspi.p_ctrl, g_qspi.p_cfg)
```

   g_qspi.p_ctrl と g_qspi.p_cfg は、QSPI の設定ステップの後で自動生成されます。
4) 以下を呼び出すことで、QSPI フラッシュ デバイス上の 1 つのセクターを消去します。

```
g_qspi.p_api->eraseSector(g_qspi.p_ctrl, p_device_address)
```

   パラメータ p_device_address は、消去を開始するフラッシュ デバイス空間アドレスの位置です。
5) 以下を呼び出して、QSPI フラッシュ デバイスにデータをプログラムします。

```
g_qspi.p_api->pageProgram (g_qspi.p_ctrl, p_device_address, p_memory_address, byte_count)
```

   この関数呼び出しでは、次のパラメータが使用されます。

   - p_device_address: フラッシュ デバイス内のデータを書き込むアドレス空間の位置
   - p_memory_address: フラッシュ デバイスに書き込むデータのメモリ アドレス
   - byte_count: 書き込むバイト数
6) 以下を呼び出して、QSPI インスタンスをクローズします。

```
g_qspi.p_api->close(g_qspi.p_ctrl))
```

シンプルな QSPI アプリケーションの場合、ThreadX アプリケーションを作成するときは、上記のコードをファイル hal_entry.c または my_thread_entry.c に追加します。これらのファイルは、プロジェクトの src/ ディレクトリ内にあり、アプリケーション コードを格納するために ISDE で生成されます。

4.2.24.4 QSPI ドライバの制限事項

QSPI ドライバのその他の制限事項については、SSP のリリース ノートを参照してください。

4.2.24.5 QSPI ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが synergy/ssp/ ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL QSPI Interface API	synergy/ssp/inc/driver/api/r_qspi_api.h
QSPI Instance	synergy/ssp/inc/driver/instances/r_qspi.h
QSPI Driver	synergy/ssp/src/driver/r_qspi

4.2.24.6 QSPI ドライバでサポートされるデバイス

このドライバは、QSPI 周辺機器ブロックおよび Micron 社製 N25Q256A QSPI フラッシュ デバイスを使用して、S7G2 および S3A7 でテストされています。

4.2.25 RTC ドライバ

RTC ドライバは、リアルタイム クロックのプログラミング用の汎用 API で、r_rtc 上に実装されています。このドライバは、MCU 上で利用可能な RTC 周辺機器をサポートしています。このセクションでは、e^2 studio ISDE を使用して RTC ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Timer] > [RTC Driver on r_rtc] を選択することで、RTC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による RTC ドライバを使用するアプリケーションの作成

API リファレンスは、次の HAL RTC インタフェースの説明内に記載されています : RTC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については SSP Architecture を参照してください。

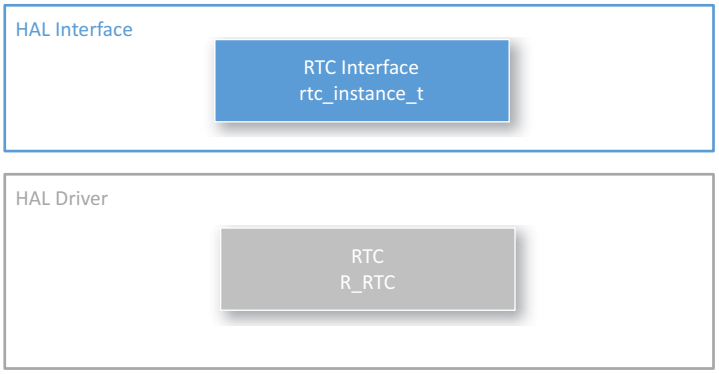

図 149: RTC ドライバ - ブロック図

4.2.25.1 RTC ドライバの機能

RTC ドライバは、リアルタイム クロックの以下の機能をサポートしています。

- RTC 周辺機器の設定
- クロックおよびカレンダー機能
- アラーム、周期、キャリー割り込み
- 時間キャプチャ機能
- 他の周辺機器へのイベント リンケージ

4.2.25.2 e² studio ISDE による RTC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

RTC ドライバを使用するアプリケーションでは、以下のリソースが必要です。

リソース	ISDE タブ	選択
RTC Driver	Threads	[Driver] > [RTC Driver on r_rtc]

リソース	ISDE タブ	選択
Interrupts	ICU	アプリケーション例については、RTC 割り込みの設定
RTC Clock	Clocks	アプリケーション例については、RTC ドライバ用のクロックの設定

RTC ドライバ用のクロックの設定

e^2 studio ISDE で、[Clocks] タブを使用してリアルタイム クロックを設定します （クロックの設定を参照）。

RTC モジュールは、次のクロック ソースを使用できます。

- LOCO （低速オンチップ オシレーター）
  - 低い電力消費
  - 低い精度
- サブクロック オシレーター
  - 高い電力消費
  - 高い精度
  - 高コスト （水晶が必要）

RTC 割り込みの設定

e^2 studio ISDE を使用して、[ICU] タブから RTC 割り込みを設定します （割り込みの設定を参照）。

RTC ドライバは、以下の 3 種類の割り込みをサポートしています。

- アラーム割り込み
- アラーム割り込みは、年、月、日、曜日、時、分、秒の任意の組み合わせに一致した場合に生成されます。
- 周期割り込み
- 周期割り込みは、2、1、½、¼、1/8、1/16、1/32、1/64、または 1/256 秒ごとに生成できます。
- キャリー割り込み
- キャリー割り込みが生成されるのは、第 2 のカウンタへのキャリーが発生した場合か、64-Hz カウンタへの読み取りアクセス中に R64CNT カウンタへのキャリーが発生した場合です。

RTC ドライバのコールバック ISR 関数の定義

ユーザー定義のコールバック関数を open で登録できます。このコールバック関数が指定されている場合、サポートされている 3 つの割り込みタイプのいずれかに対して割り込みサービス ルーチン （ISR）から呼び出されます。コールバック関数が呼び出されると、それによって構造体 （rtc_callback_args_t）へのポインタが渡されます。この構造体には、ユーザー定義のコンテキスト ポインタと、発生した割り込みの種類を示す情報が格納されています。

RTC ドライバ パラメータの設定

e^2 studio ISDE を使用して、RTC ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

RTC ドライバのビルド時構成

ISDE プロパティ	設定	設定値	説明
Parameter checking enable	#define RTC_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータ エラー チェックを有効または無効にします。

RTC の設定

ISDE プロパティ	設定	設定値	説明
Name	name	Arbitrary symbol (Default: "r_rtc")	RTC モジュール制御ブロック インスタンスに使用する名前。この名前は、他の可変インスタンスのプレフィックスにも使用されます。以下のコード例を参照してください。
Clock Source	clock_source	LOCO: Low power on-chip oscillator, SUB: sub-clock oscillator	RTC ブロックのクロック ソース。
Error Adjustment Value	error_adjustment_value	0	Attention : 非推奨の設定フィールド。0 を設定する必要があります。
Error Adjustment Type	error_adjustment_type	None	Attention: 非推奨の設定フィールド。None を設定する必要があります。
Callback	p_callback	User-defined	3 つの割り込みのうちいずれかが発生したときに呼び出される ISR の名前。この ISR に渡される引数には、呼び出しの原因となった割り込みが示されます。以下のコード例を参照してください。

ISDE プロパティ	設定	設定値	説明
-	p_context	User-defined	割り込みが発生したときに ISR に渡される、ユーザー定義コンテキストへのポインタ。

4.2.25.3 RTC ドライバ アプリケーションの作成

一般的な RTC アプリケーションは、ユーザーによって指示されたシステム設定に基づき、リアルタイム クロック コントローラを定期的に設定します。たとえば、時刻の設定、アラームの設定、周期割り込みの設定などです。

RTC アプリケーションは、RTC ドライバとオプションの ISR ハンドラーの呼び出しで構成されます。

他の API を呼び出す前にドライバをオープンする必要があります。open 呼び出しに渡す設定構造体では、クロック ソース、ISR ハンドラーの名前、ハンドラーのユーザー固有のコンテキストを指定します。設定構造体は、手動で定義するか、設定手順の中でのユーザー入力に基づいて ISDE によって生成します。

ドライバの関数にアクセスするには、HAL レイヤーを直接呼び出すか、RTC インタフェース構造体を使用します。このインタフェース構造体の名前は、モジュールの設定で入力した名前設定に基づきます。たとえば、名前が g_rtc の場合、インタフェース構造体の名前は g_rtc_api になります。

RTC ドライバの例

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const rtc_instance_t g_rtc =

{

    .p_ctrl = &g_rtc_ctrl,

    .p_cfg = &g_rtc_cfg,

    .p_api = &g_rtc_on_rtc

};
```

この例では、RTC ドライバを使用して GPIO を切り替えるための周期割り込みを使用して、設定および開始する方法を示しています。

Note：このコード例で使用されている変数 g_rtc と、関数名 periodic_callback は、ISDE の [Properties] ウィンドウで次の設定を使用してこのモジュールを設定した場合に生成されます。

設定	設定値
Name	g_rtc
Callback	periodic_callback

```
/* Set up the RTC periodic interrupt to fire once every second */

void periodic_interrupt_setup(void)

{

   /* Open the real time clock driver */

   g_rtc.p_api->open(g_rtc.p_ctrl, g_rtc.p_cfg);

   /* Set the periodic interrupt rate */

   g_rtc.p_api->periodicIrqRateSet(g_rtc.p_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECO
ND);

   /* Start the counter */

   g_rtc.p_api->calendarCounterStart(g_rtc.p_ctrl);

   /* Enable periodic interrupts */

   g_rtc.p_api->irqEnable(g_rtc.p_ctrl, RTC_EVENT_PERIODIC_IRQ);

   /* Close the real time clock driver */
```

```
   g_rtc.p_api->close(g_rtc.p_ctrl);
}
/* periodic ISR that toggles a GPIO pin */
void periodic_callback(rtc_callback_args_t * p_args)
{
   if (p_args->event == RTC_EVENT_PERIODIC_IRQ)
   {
      /* place GPIO toggle code here */
   }
}
```

4.2.25.4 RTC ドライバの制限事項

このモジュールは次のものをサポートしていません。

- バイナリ カウント モード
- バイナリ アラーム
- バイナリ キャプチャ
- カレンダー キャプチャ
- クロック エラー訂正
- 1-Hz/64-Hz クロック出力
- 周期イベント出力

RTC ドライバのその他の制限事項については、SSP のリリース ノートを参照してください。

4.2.25.5 RTC ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL RTC Interface API	synergy/ssp/inc/driver/api/r_rtc_api.h
RTC Instance	synergy/ssp/inc/driver/instances/r_rtc.h
RTC Driver	synergy/ssp/src/driver/r_rtc

4.2.25.6 RTC ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

4.2.26 SD/MMC ドライバおよび SDIO ドライバ

SD/MMC ドライバと SDIO ドライバは r_sdmmc 上に実装され、SD/MMC メディア デバイスならびに SDIO カードの読み取り / 書き込みと制御に使用されます。

SD/MMC モジュールはスタンドアロン SD カードとして使用できます。また、eMMC、メディア ドライバや SD/MMC モジュールは、FileX やその他の互換性のあるファイル システムと共に使用することもできます。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New (+)] > [Driver] > [Storage] > [SD/MMC Driver on r_sdmmc] を選択することで、SD/MMC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^{2}studio ISDE による SD/MMC ドライバを使用するアプリケーションの作成。

SD/MMC モジュールは、スタンドアロン SDIO カード ドライバとして使用することもできます。

Note：SD の仕様に準拠したホスト デバイスを開発する場合は、SD ホスト機器および周辺機器の使用許諾契約 （SD Host/Ancillary Product License Agreement）（SD HALA）に従う必要があります。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New (+)] > [Driver] > [Connectivity] > [SDIO Driver on r_sdmmc] を選択することで、SDIO ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^{2}studio ISDE による SD/MMC ドライバを使用するアプリケーションの作成。

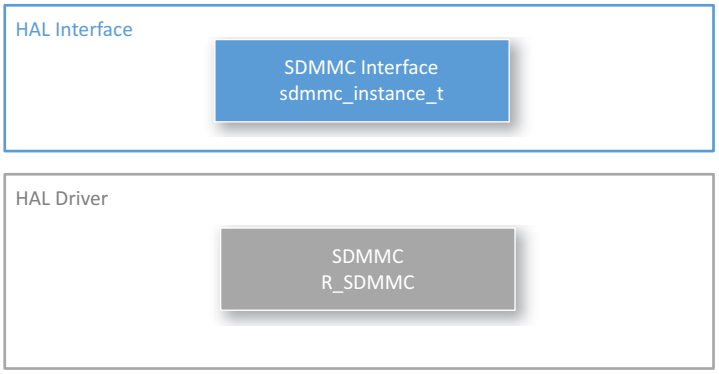

図 150: SD/MMC ドライバ - ブロック図

SD/MMC API 関数にアクセスするには、SDMMC インタフェース API を使用します。SD/MMC API リファレンスは、次の SDMMC インタフェースの説明内に記載されています : SDMMC インタフェース。

4.2.26.1 e²studio ISDE による SD/MMC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

SD カードと MMC メディアの SD/MMC モジュールの設定

SD/MMC メディア ドライバ モジュールは、e² studio ISDE の [Threads] タブの [HAL/Common Stacks]> [New (+)] > [Driver] > [Storage] > [SD/MMC Driver on r_sdmmc] で選択されています。

チャネル、メディア タイプ、バス幅パラメータを回路ボードに合うように選択する必要があります。転送パラメータと一部の割り込みが必要です。R_SDMMC_Read および R_SDMMC_Write 関数を使用し、かつ FileX を搭載したドライバを使用しない場合、転送終了の信号を送信するには SDMMC コールバックが必要です。

SDIO カードの SD/MMC ドライバの設定

SDIO ドライバ モジュールは、e² studio ISDE の [Threads] タブの [HAL/Common Modules]> [New (+)] > [Driver] > [Connectivity] > [SDIO Driver on r_sdmmc] で選択されています。

チャネル、メディア タイプ、バス幅パラメータを回路ボードに合うように選択する必要があります。転送パラメータと一部の割り込みが必要です。CMD53 関数 R_SDMMC_ReadIoExt と R_SDMMC_WriteIoExt を使用する場合、転送終了の信号を送信するには SDMMC コールバックが必要です。

SD/MMC クロックの設定

SDHI は PCLKA をそのクロック ソースとして使用します。データ レートを最適化する必要がない限り、SDMMC 周辺デバイスに対して固有のクロックを設定する必要はありません。SDMMC ドライバは、PCLKA 周波数と、SD、SDIO または eMMC デバイスで許可される最大クロック レート （これはメディア デバイスの初期化時に取得されます）に基づいて、適切な内蔵分周器を選択します。

SD/MMC ピンの設定

e^2 studio ISDE ピン コンフィギュレータを使用して、SDMMC 周辺機器の I/O ピンを設定します。ほとんどのボード、高速メモリ、SDIO デバイスにおいて、各品のドライブ能力は「中」または「高」に設定する必要があります。

SD/MMC ドライバ パラメータの設定

e^2 studio ISDE を使用して、SDMMC ドライバ パラメータを設定します （ドライバのスレッドへの追加とドライバの設定を参照）。

ISDE によって /synergy_cfg/ssp_cfg/driver ディレクトリに r_sdmmc_cfg.h ファイルが作成され、各共通構成パラメータに対して選択した設定がこのファイルに格納されます。

SD/MMC には、以下のコンポーネントの構成が必要です。

- SDMMC 共通構成
- SDMMC_on_SDMMC:sdmmc_hw_t

SD/MMC ドライバの共通構成パラメータ

ISDE プロパティ	設定	設定値	説明
Parameter Checking Enable	#define SDMMC_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータ エラー チェックを有効または無効にします。

SD/MMC のドライバ構成パラメータ

ISDE プロパティ	設定	設定値	説明
SDHIMMC ACCS	-	Disabled (default) priority level 0-15	SD および eMMC、SDIO への読み取り、書き込み、エラーに必要となる割り込み優先順位。有効にすることで、割り込みコールバックを呼び出します。

ISDE プロパティ	設定	設定値	説明
SDHIMMC CARD	-	Disabled (default) priority level 0-15	(オプション) カード取り外しの割り込み。カードが取り外された際、自動でデバイスを閉じます。有効にすることで、割り込みコールバックを呼び出します。
SDHIMMC SDIO	-	Disabled (default) priority level 0-15	SDIO の読み取り、書き込み、エラーに必要となる割り込みプライオリティ レベル。有効にすることで、割り込みコールバックを呼び出します。メモリ カードに使用されません。

SD/MMC ドライバの設定 sdmmc_hw_t

ISDE プロパティ	設定	設定値	説明
Name	-	Arbitrary symbol (Default: "g_sdmmc" or "g_sdio")	SDMMC モジュール制御ブロック インスタンスに使用する名前。この名前は、他の可変インスタンスのプレフィックスにも使用されます。
Channel	channel	0,1	SD/ MMC 周辺機器のチャネル (チャネル 0 または 1)
Media Type	media_type	Card, Embedded	メディアは、カードと組み込みデバイスのどちらかです。この設定により、カードの挿入 / 取り出しがあるのか、あるいは書き込み保護ピンが存在するのかをファームウェアに知らせます。
Bus Width	bus_width	1 Bit, 4 Bits, 8 Bits	ハードウェア インタフェースによって定義されたバス幅。(8 ビットは eMMC のみ)

ISDE プロパティ	設定	設定値	説明
Lower level Transfer name	transfer_instance_t	NULL (default) Name of transfer module	(必須) r_dmac または r_dtc モジュールで転送ドライバ名に設定します。r_dmac を使用する場合は r_dmac の割り込みを有効化します。他の転送パラメータ (モード、転送サイズ、転送先など) を設定する必要はありません。
Callback	-	NULL (default) Name of callback function	(FileX を使用しない場合は必須) ユーザー コールバック関数の名前に設定します。Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR

割り込みおよび DMA は、リリース 1.0.0 では SDMMC/SDIO のオプションでした。すべてのリードおよびライト関数はブロッキング関数であったため、処理が完了するまで関数が戻りませんでした。

リリース 1.1.0 ではメディアのリード / ライト関数 (R_SDMMC_Read および R_SDMMC_Write) および SDIO の拡張リード / ライト関数 (R_SDMMC_ReadIoExt および R_SDMMC_WriteIoExt) が非ブロッキング関数になったため、DMAC または DTC で割り込みと転送関数が必要です。リード / ライト関数は、初期動作が正常に開始したことを示す SSP_SUCCESS を返します。ただし、ユーザー アプリケーションはユーザー コールバックを待ち、読み取り / 書き込みの完了を示す SDMMC_EVENT_TRANSFER_COMPLETE または SDMMC_EVENT_TRANSFER_ERROR イベントを確認する必要があります。

FileX 、sf_el_fx、sf_block_media_sdmmc を使用する場合、コールバックは sf_block_media_sdmmc レイヤーで実装されます。SDMMC ドライバ レベルでの設定は不要です。ただし、割り込みおよび転送ドライバについては設定が必要です。

必要な割り込みは次のとおりです。

DTC で SD/MMC を使用 :

- SDHIMMCx ACCS
- SDHIMMCx DMA REQ

DMAC で SD/MMC を使用 :

- SDHIMMCx ACCS
- DMACx （DMAC 転送割り込み）

DTC で SDIO を使用 :

- SDHIMMCx ACCS
- SDHIMMCx SDIO
- SDHIMMCx DMA REQ

DMAC で SDIO を使用 :

- SDHIMMCx ACCS
- SDHIMMCx SDIO
- DMACx （DMAC 転送割り込み）

SDHIMMCx CARD はオプションです。

x= 使用するチャネル

4.2.27 セグメント LCD ドライバ

セグメント LCD ドライバは r_slcdc に実装され、セグメント LCD ディスプレイを制御するための汎用 API を含みます。このドライバは、MCU 上で利用可能な SLCD 周辺機器をサポートしています。このセクションでは、e^2 studio ISDE を使用して SLCD ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Graphics] > [SLCDC Driver on r_slcdc] を選択することで、SLCDC ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による SLCDC ドライバを使用するアプリケーションの作成

API リファレンスは、次の SLCDC インタフェースの説明内に記載されています : SLCDC インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

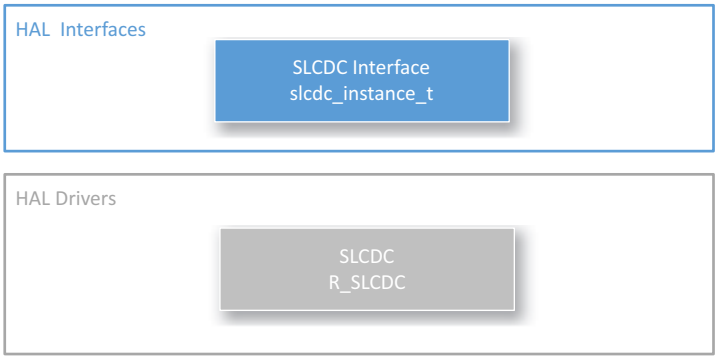

図 151: SLCD ドライバ - ブロック図

4.2.27.1 SLCDC ドライバの機能

このドライバはセグメント LCD コントローラ （SLCDC）を使用してデータをセグメント LCD に表示します。ドライバは LCD を初期化してデータを表示し、ドライブ電圧ジェネレーター、ディスプレイ波形、タイム スライス数、および LCD を稼働するバイアス メソッドを設定します。このモジュールでは、指定されたセグメントのセットにデータを表示するための関数、既存のセグメント データを更新するための関数、ディスプレイを有効化および無効化するための関数、ディスプレイ エリアを設定するための関数、およびコントラストを縫製するための関数を提供します。

このモジュールでは、次の機能の選択をサポートします。

- LCD ドライバ電圧ジェネレーターの内部電圧ブースト : 容量分割方式または外部抵抗分割方式を選択します。
- 表示バイアス : 1/2 バイアス法、1/3 バイアス法、または 1/4 バイアス法を選択します。
- 表示のタイム スライス : 静的、2 時分割、3 時分割、4 時分割、または 8 時分割を選択します。
- 表示波形 : 波形 A または波形 B を選択します。
- 表示データ領域。A パターン、B パターン、または点滅を選択します。表示データ領域は切り替えることができます。
- RTC 周期割り込み （PRD）を使用して、A パターンまたは B パターンの点滅表示を生成します。
- 電圧ブースト回路を稼働すると生成される参照電圧を 16 ステップ （コントラスト調整）で調整します。

4.2.27.2 e² studio ISDE による SLCDC ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

次のリソースは、SLCDC ドライバを使用するアプリケーションで必要となります。

リソース	ISDE タブ	選択
SLCDC Driver	Threads	新しいスレッドまたは HAL/Common をハイライトして、[New] > [Driver] > [Graphics] > [Segment LCD Driver on r_slcdc] の順に選択します。

SLCDC クロックの設定

SLCDC クロックは [Clocks] タブから設定できません。クロックは g_slcdc ドライバの [Properties] ウィンドウで設定します （SLCDC ドライバ パラメータの設定を参照）

セグメント LCDC ソース クロック LCDSRCCLK はセグメント LCDC の動作クロックとして使用されます。LCDSRCCLK は、Segment LCD Source Clock Control Register （SLCDSCKCR）の LCDSCKSEL[2:0] ビットで指定されます。

セグメント LCDC ソース クロックは、ISDE コンフィギュレータを使用してメイン、HOCO、LOCO、MOCO として設定できます。

SLCDC ピンの設定

e2 studio ISDE を使用して、[Pins] タブから SLCDC ピンを設定します （ピンの設定を参照）。

セグメントを使用するには、ピンがピン コンフィギュレータで SLCDC 周辺機器ピンとして設定されている必要があります。これにより、pin_cfg フィールドで関連するピンの IOPORT_CFG_PERIPHERAL_PIN と IOPORT_PERIPHERAL_LCD が設定されます。

SLCDC 割り込みの設定

中断構成は不要です。

SLCDC ドライバ パラメータの設定

e2 studio ISDE を使用して、SLCDC ドライバ g_slcdc パラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

<SLCDC 上の SLCDC ドライバ （SLCDC driver on SLCDC)> を追加すると、ファイル r_slcdc_cfg.h が ssp_cfg_driver ディレクトリに生成され、各構成パラメータの選択した設定が格納されます。

SLCDC の設定

設定	設定値	説明
slcdc_clock	slcdc_display_clock_t	SLCD クロック ソース（LCDSCKSEL）。
slcdc_clock_setting	slcdc_clk_div_t	LCD クロック設定 （LCDC0）、クロック除算
bias_method	slcdc_bias_method_t	LCD の表示バイアス法選択 （LBAS ビット）
time_slice	slcdc_time_slice_t	LCD 表示の時分割数選択 （LDTY ビット）
wave_form	slcdc_wave_form_t	LCD 表示波形選択 （LWAVE ビット）。
drive_volt_gen	slcdc_drive_volt_gen_t	LCD 駆動電圧生成回路選択（MDSTET ビット）。

SLCDC ドライバ スレッドの設定

このドライバは HAL ドライバであり、ThreadX RTOS に依存しません。しかし、HAL SLCDC ドライバを ThreadX RTOS のスレッドに追加することができます。

4.2.27.3 SLCDC ドライバ アプリケーションの作成

ISDE によって生成されたヘッダー ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const slcdc_instance_t g_slcdc =

{

  .p_ctrl = &g_slcdc_ctrl,

  .p_cfg = &g_slcdc_cfg,

  .p_api = &g_slcdc_on_slcdc

};
```

SLCDC を使用したセグメント LCD アプリケーションを作成するには、次の手順を実行します。

1) 前述のように、slcdc_cfg_t 構造体を設定して、モジュールを構成します。

2) ISDE でプロジェクト コンテンツを生成します。これにより、フレームワーク関連のヘッダー ファイルと設定ファイルが自動的に作成されます。設定構造体の例を以下に示します。

```
const slcdc_cfg_t  g_slcdc_cfg =

{

  .slcdc_clock  = SLCDC_CLOCK_LOCO,

 .slcdc_clock_setting = SLCDC_CLK_DIVISOR_LOCO_128,

  .bias_method = SLCDC_BIAS_3,

  .time_slice = SLCDC_SLICE_4,

  .wave_form = SLCDC_WAVE_A,

  .drive_volt_gen = SLCDC_VOLT_INTERNAL,
};
```

3) SLCDC インスタンスを開きます。SLCDC ドライバは、g_slcdc インスタンス構造体を通じて呼び出されます。

```
g_slcdc.p_api->open(g_slcdc.p_ctrl, g_slcdc.p_cfg)
```

p_ctrl と p_cfg は、SLCDC の設定ステップの後で自動生成されます。SLCDC をオープンすると、セグメント LCD のハンドルが返されます。上の例では p_ctrl です。このハンドルを使用して、各種の SLCDC 転送操作を実行します。以下に、このハンドルを使用して write 操作を行うコード例を示します。この例により、すべてのセグメントが有効化されます。

```
#define START_SEGMENT 3

#define END_SEGMENT 45

#define SEGMENT_COUNT (END_SEGMENT - START_SEGMENT)

for(int s=0; s<SEGMENT_COUNT; s++)

{

   segment_data[s] = 0x0F;

}

g_slcdc.p_api->write(g_slcdc.p_ctrl, START_SEGMENT, segment_data, SEGMENT_COUNT);
```

Note：セグメントのシーケンスを書き込むには、スタート セグメント番号と書き込み API に書き込むセグメントの数を渡します。

4) 次を呼び出して セグメント LCD を開始します。

```
g_slcdc.p_api->start(g_slcdc.p_ctrl)
```

5) 次を呼び出して表示領域または点滅表示を変更します。

```
g_slcdc.p_api->setdisplayArea(g_slcdc.p_ctrl, SLCDC_DISP_BLINK)
```

Note：表示点滅は、定期タイマ割り込みを設定した RTC モジュールによって可能です。定期タイマ割り込みの設定方法については、RTC ユーザードキュメントを参照してください。

6) 次を呼び出して、SLCDC を停止します。

```
g_slcdc.p_api->stop(g_slcdc.p_ctrl)
```

7) 以下を使用してコントラストを調整します。

```
g_slcdc.p_api->contrastIncrease(g_slcdc.p_ctrl)
```

と

```
g_slcdc.p_api->contrastDecrease(g_slcdc.p_ctrl)
```

1 単位ずつコントラスト値を変更する API。

8) 次を呼び出して、SLCDC インスタンスを閉じて、他の用途のために LCD リソースを解放します。

```
g_slcdc.p_api->close(g_slcdc.p_ctrl)
```

SLCDC ドライバのコード例

```
#define BLANK_DIGIT (0xFF)

#define DIGIT_ONES 0

#define DIGIT_TENS 2

#define DIGIT_HUNDREDS 3

#define START_SEGMENT 3

#define END_SEGMENT 45

#define MAIN_DIGIT_SEGMENT_01 16

#define MAIN_DIGIT_SEGMENT_00 17

#define MAIN_DIGIT_SEGMENT_11 12

#define MAIN_DIGIT_SEGMENT_10 15

#define MAIN_DIGIT_SEGMENT_21 10

#define MAIN_DIGIT_SEGMENT_20 11

#define SEGMENT_COUNT (END_SEGMENT - START_SEGMENT)

slcdc_size_t  segment_data[SEGMENT_COUNT];

typedef struct digitMap_s

{

   uint8_t data;

   uint8_t seg01;

   uint8_t seg00;

} digitMap_t;

const digitMap_t mainDigitTable[] =

{

   { 0, 0x05, 0x0F },

   { 1, 0x00, 0x06 },
```

```
    { 2, 0x06, 0x0B },
    { 3, 0x02, 0x0F },
    { 4, 0x03, 0x06 },
    { 5, 0x03, 0x0D },
    { 6, 0x07, 0x0D },
    { 7, 0x00, 0x07 },
    { 8, 0x07, 0x0F },
    { 9, 0x03, 0x0F },
    { BLANK_DIGIT, 0x00, 0x00 }
};
const digitMap_t timeDigitTable[] =
{
    { 0, 0x0A, 0x0F },
    { 1, 0x00, 0x06 },
    { 2, 0x06, 0x0D },
    { 3, 0x04, 0x0F },
    { 4, 0x0C, 0x06 },
    { 5, 0x0C, 0x0B },
    { 6, 0x0E, 0x0B },
    { 7, 0x00, 0x0E },
    { 8, 0x0E, 0x0F },
    { 9, 0x0C, 0x0F },
    { BLANK_DIGIT, 0x00, 0x00 }
};
```

```
void slcd_main_single_digit_set(int val, uint8_t digit)
{
  switch(digit)
  {
  case DIGIT_ONES:
    segment_data[MAIN_DIGIT_SEGMENT_00] = mainDigitTable[val].seg00;
    segment_data[MAIN_DIGIT_SEGMENT_01] = mainDigitTable[val].seg01;
    break;
  case DIGIT_TENS:
    segment_data[MAIN_DIGIT_SEGMENT_10] = mainDigitTable[val].seg00;
    segment_data[MAIN_DIGIT_SEGMENT_11] = mainDigitTable[val].seg01 | 0x08; // + renesas logo
    break;
  case DIGIT_HUNDREDS:
    segment_data[MAIN_DIGIT_SEGMENT_20] = mainDigitTable[val].seg00;
    segment_data[MAIN_DIGIT_SEGMENT_21] = mainDigitTable[val].seg01;
    break;
  }
  g_slcdc.p_api->write(g_slcdc.p_ctrl, START_SEGMENT, segment_data, SEGMENT_COUNT);
}
void slcd_display_all_off(void)
{
  for(int s=0; s<SEGMENT_COUNT; s++)
  {
    segment_data[s] = 0x00;
  }
```

```
  g_slcdc.p_api->write(g_slcdc.p_ctrl, START_SEGMENT, segment_data, SEGMENT_COUNT);
}
void slcd_main_decimalpoint_set(void)
{
  segment_data[MAIN_DIGIT_SEGMENT_01] |= 0x08;
  g_slcdc.p_api->write(g_slcdc.p_ctrl, START_SEGMENT, segment_data, SEGMENT_COUNT);
}
void slcd_show_version(uint8_t major, uint8_t minor)
{
  slcd_main_single_digit_set(minor, DIGIT_ONES);
  slcd_main_decimalpoint_set();
  slcd_main_single_digit_set(major, DIGIT_TENS);
}
void hal_entry (void)
{
  ssp_err_t  err = g_slcdc.p_api->open(g_slcdc.p_ctrl, g_slcdc.p_cfg);
 err = g_slcdc.p_api->start(g_slcdc.p_ctrl);
 err = g_slcdc.p_api->setdisplayArea(g_slcdc.p_ctrl, SLCDC_DISP_A);
 slcd_display_all_off();
  slcd_show_version(1, 0); //Display version
  while(1)
  {}
}
```

4.2.27.4 SLCDC ドライバの制限事項

SLCDC モジュールには既知の制限事項はありません。

4.2.27.5 SLCDC ドライバでサポートされるデバイス

SLCDC ドライバ モジュールは、MCU の SLCDC 周辺機器にアクセスします。このドライバは、S3A7 でテストされています。

SLCDC ドライバは、API への変更なしに、次のファミリをサポートするように設計されています。

- S3A7

4.2.28 SPI ドライバ

SPI ドライバは、SPI プロトコルを使用した通信用の汎用 API です。このドライバは、MCU で利用可能な SPI および SCI 周辺機器をサポートするため、r_rspi と r_sci_spi に実装されています。このセクションでは、e^2 studio ISDE を使用して SPI ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Connectivity] > [SPI Driver on r_rspi] または [New] > [Driver] > [Connectivity] > [SPI Driver on r_sci_spi] を選択することで、SPI ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による SPI ドライバを使用するアプリケーションの作成。

API リファレンスは、次の SPI インタフェースの説明内に記載されています : SPI インタフェース。

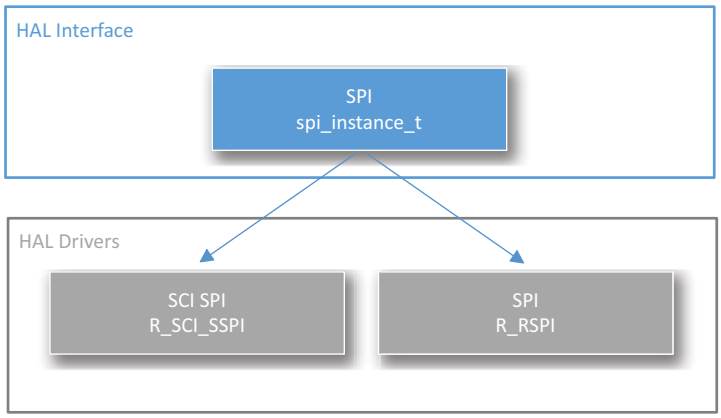

図 152: SPI ドライバ - ブロック図

4.2.28.1 SPI ドライバの機能

SPI ドライバは、マスター モードの SPI 通信を設定します。このドライバでは、以下のことが可能です。

- ドライバの初期化。
- SPI オペレーションによるシリアル通信です。

ドライバは、コールバックへのサポートも提供します。コールバック関数は、次のイベント spi_event_t で呼び出されます。

- 転送中断
- 転送完了
- モード障害
- エラー イベント

4.2.28.2 SPI ドライバ用モジュールの選択

SPI モジュール （SPI および SCI）のいずれも 8 ビットのデータ移行をサポートしています。加えて、SPI モジュールは 8、16、および 32 ビットのデータ移行もサポートしています。どちらの SPI モジュールも、チップ セレクトとして設定された GPIO ピンをサポートします。さらに、SPI 周辺機器は、専用チップ選択信号 SSLn0 から SSLn3 までをサポートします。SPI 周辺機器では、すべてのチップ選択処理はハードウェアによって行われます。

4.2.28.3 e^2 studio ISDE による SPI ドライバを使用するアプリケーションの作成

ドライバは、e^2 studio ISDE の SSP に組み込まれています （e^2 studio ISDE ユーザーガイドを参照）。

e^2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

SPI と SCI モジュールを使用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
SCI Driver for SPI on SCI	Threads	新しいスレッドまたは HAL/ 共通をハイライトして、[New] > [Driver] > [Connectivity] > [SPI Driver r_sci_spi] の順に選択します。
SCI Common driver	Threads	SCI 共通がスレッドに自動的に追加されます。このドライバの [Properties] ウィンドウで、簡易 SPI モードを有効化します。

リソース	ISDE タブ	選択
DTC Transfer driver	Threads	送信および受信 DTC 転送ドライバがデフォルトで追加されます。デフォルトの転送モードは、DTC を介した転送です。SPI 転送モード（非 DTC）では DTC ドライバを削除します。
Interrupts	Threads または ICU	モジュールのプロパティからチャネルを選択して、選択したチャネルの SCIn RXI、TXI、TEI、および ERI 割り込みを有効化します。

SPI と SPI モジュールを使用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
SPI Driver	Threads	新しいスレッドまたは HAL/ 共通をハイライトして、[New] > [Driver] > [Connectivity] > [SPI Driver r_rspi）] の順に選択します。
DTC Transfer driver	Threads	送信および受信 DTC 転送ドライバがデフォルトで追加されます。デフォルトの転送モードは、DTC を介した転送です。SPI 転送モード（非 DTC）では DTC ドライバを削除します。DTC 転送モードでは、RSPI は 32 ビットのデータ転送のみをサポートします。8 ビットおよび 16 ビットのデータ転送では、DTC モジュールを削除します。
SPI Interrupts	Threads または ICU	モジュールのプロパティからチャネルを選択して、選択したチャネルの RSPIn SPRI、SPTI、SPII、および SPEI 割り込みを有効化します。

SPI ドライバ用の SPI クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して SPI クロックを設定します （クロックの設定を参照）。

SPI は PCLKB をそのクロック ソースとして使用します。PCLKB 周波数を設定するには、実行時に e^2 studio ISDE または CGC インタフェースのクロック コンフィギュレータを使用します。

SPI ドライバ用の SPI ピンの設定

e^2 studio ISDE を使用して、[Pins] タブから SPI ピンを設定します （ピンの設定を参照）。

SPI を使用するには、出力ピンとして使用する I/O ポート ピンが、ピン コンフィギュレータで SPI 周辺機器ピンとして設定されている必要があります。外部チップ選択を使用するには、チップを構成して、ピンを GPIO 出力として選択します。

これにより、pin_cfg フィールドで関連するピンのピン構成が適切に設定されます。

SPI ドライバの SPI 割り込みの設定

e^2 studio ISDE を使用して、[Threads] タブから SPI 割り込みを設定します （割り込みの設定を参照）。

SCI SPI 割り込み

SCI SPI の割り込みを有効にするには、ドライバ モジュールをハイライトし、e^2 studio ISDE のプロジェクト コンフィギュレータの [Threads] タブで、SCIn RXI、TXI、TEI、および ERI 割り込み （n は SCI チャネル番号）の優先度を設定します （割り込みの設定を参照）。

これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

SPI 割り込み

Note：SPI の割り込みを有効にするには、ドライバ モジュールをハイライトし、プロジェクト コンフィギュレータの [ICU] タブで、RSPIn SPRI、SPTI、SPII、および SPEI 割り込み （n は SPI チャネル番号）の優先度を設定します （割り込みの設定を参照）。

これにより、ssp_cfg/bsp/bsp_irq_cfg.h の対応する割り込みに、選択した優先度が設定されます。

Attention：割り込みを異なる優先度に設定すると、適切に動作しなくなる可能性があります。

SPI ドライバ用のパラメータの設定

e^2 studio ISDE を使用して、SPI ドライバパラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定。

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

Note：すべてのパラメータは、SPI ドライバ構成構造体 spi_cfg_t に設定されます。

SPI ドライバ モジュールの設定

ISDE プロパティ	設定	設定値	説明
Channel	channel	Integer value between 0 and 9	デバイスが接続されている SCI または SPI チャネル番号。
Operating mode	operating_mode	Master, Slave	マスター デバイスまたはスレーブ デバイスとして設定します。 Note：SSP の現在のバージョンは、SPI マスター モードのみをサポートしています。
Clock Phase	clk_phase	Master, SlaveData sampling on even edge and data variation on odd edge. Data sampling on odd edge and data variation on even edge	奇数または偶数クロック エッジのいずれかのデータ サンプリングを選択します。
Clock Polarity	clk_polarity	High when idle, Low when idle	アイドル時のクロック レベル。
Mode Fault Error	mode_fault	Mode fault error Disable, Mode fault error Enable	モード障害エラー （マスターまたはスレーブ）のフラグを示します
Bit Order	bit_order	MSB First , LSB First	MSB ファーストと LSB ファーストのいずれかの送信順序を選択します
Bitrate	bitrate	Arbitrary integer value	送信または受信レート。ビット / 秒。
Callback	p_callback	Optional If NULL, driver will perform a blocking transfer	オプションのコールバック関数ポインタ。
Extend	p_extend	Optional	SPI ハードウェアに依存する拡張設定。SPI で有効。

SPI ドライバの拡張設定

SPI ドライバには、ハードウェアに固有の拡張設定が数多く存在します。

Note：すべてのパラメータは、SPI 拡張ドライバ構成構造体 spi_on_rspi_cfg_t に設定されます。

SPI の SPI ドライバ拡張設定

ISDE プロパティ	設定	設定値	説明
SPI Mode	rspi_clksyn	rspi_operation_t	動作モードを選択します（SPI またはクロック同期）
SPI Communication Mode	rspi_comm	rspi_communication_t	通信方式を選択します（全二重または送信のみ）
SSL Polarity	ssl_polarity	rspi_ssl_polarity_t	SSLn の信号極性を選択します
Loopback	loopback	rspi_loopback_t	loopback1 または loopback2 を選択します
MOSI Idle	mosi_idle	rspi_mosi_idle_t	MOSI アイドル固定値および選択肢を選択します
Parity	parity	rspi_parity_t	パリティ値を選択し、パリティ値の有効 / 無効を設定します
SSL Select	ssl_select	rspi_ssl_select_t	使用するスレーブを選択します （0-SSL0、1-SSL1、2-SSL2、3-SSL3）
SSL Level Keep	ssl_level_keep	rspi_ssl_level_keep_t	転送完了後の SSL レベルを選択します （0- ネゲート、1- 維持）
Clock Delay	clock_delay	rspi_clock_delay_t	クロック遅延を 0 ～ 7 の範囲で選択します
Negation Delay	ssl_neg_delay	rspi_ssl_negation_delay_t	スレーブ起動ネゲート遅延を 0 ～ 7 の範囲で選択します
Next Access Delay	access_delay	rspi_access_delay_t	次アクセス遅延を 0 ～ 7 の範囲で選択します

4.2.28.4 SPI 周辺機器の SPI ドライバ アプリケーションの作成

ISDE によって生成されたヘッダーファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const spi_instance_t  g_spi =

{

  .p_ctrl  = &g_spi_ctrl,

  .p_cfg = &g_spi_cfg,

  .p_api = &g_spi_on_rspi

};
```

SPI を使用して SPI アプリケーションを作成するには、次の手順を実行します。g_spi.p_api->open() 関数をまず呼び出す必要があります。残りの呼び出しは、アプリケーションの要件に応じて任意の順序で使用できます。

1) プロジェクトに SPI ドライバを追加してモジュールを設定し、e^2 studio ISDE の [Properties] ウィンドウでプロパティを設定します。モジュールを設定すると、モジュール関連のヘッダーと設定ファイルが自動的に生成されます。
2) SPI によって実装された SPI を使用して、SPI インスタンスをオープンします。SPI インタフェースを通じて SPI ドライバが呼び出されます。

```
g_spi.p_api->open  (g_spi.p_ctrl, g_spi.p_cfg)
```

p_ctrl と p_cfg は、SPI の設定手順の後に自動生成される制御および構成構造体のインスタンスです。

3) 以下を呼び出して、スレーブ デバイスへの書き込みを初期化します

```
g_spi.p_api->write  (g_spi.p_ctrl, source, length,
SPI_BIT_WIDTH_8_BITS);
```

g_spi.p_ctrl は、open 呼び出しで使用した同じ制御インスタンスです。

4) 以下を呼び出して、スレーブ デバイスからの読み取りを初期化します

```
g_spi.p_api->read(g_spi.p_ctrl, dst16, length,
SPI_BIT_WIDTH_16_BITS);
```

g_spi.p_ctrl は、open 呼び出しで使用した同じ制御インスタンスです。

5) SPI チャネルをクローズするには、次を呼び出します。

```
g_spi.p_api->close(g_spi,p_ctrl)
```

g_spi.p_ctrl は、open 呼び出しで使用した同じ制御構造体です。

4.2.28.5 DTC （データトランスファコントローラ）を使用した SPI データ転送

データ転送サポートでは、MCU のデータトランスファコントローラ モジュールを取り込むことで、SPI HAL ドライバ モジュールが有効化されます。これで、CPU への介入なしで DTC を通じて SPI 転送を実行できます。DTC 転送には SCI と SPI モジュールの両方を使用できます。

構成に SPI ドライバを追加すると、デフォルトでスレッドに送信および受信 DTC 転送ドライバが追加されます。デフォルトの転送モードは、DTC を介した転送です。

非 DTC 転送を実行するには、構成から送信および受信 DTC ドライバを削除します。

SCI SPI は、DTC と SPI のいずれの転送モードでも、8 ビットデータ転送のみを実行します。RSPI は、DTC 転送モードにおいて 32 ビットのデータ転送のみをサポートし (S124 では DTC では 16 ビットのデータ転送のみをサポートします)、SPI 転送モードでは 8 および 16、32 ビットのデータ転送をサポートします。

4.2.28.6 SPI ドライバの制限事項

SPI インタフェースは、SCI 上の SPI と SPI 周辺機器の両方について、マスター モードのみを実装しています。スレーブ モードはサポートされていません。

4.2.28.7 SPI ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
HAL SPI Interface API	synergy/ssp/inc/driver/api/r_spi_api.h
SPI on SCI Instance	synergy/ssp/inc/driver/instances/r_sci_spi.h
SPI on SCI Driver	synergy/ssp/src/driver/r_sci_spi
SPI Instance	synergy/ssp/inc/driver/instances/r_rspi.h
SPI Driver	synergy/ssp/src/driver/r_rspi

4.2.28.8 SPI ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

SPI ドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.29 タイマ ドライバ

タイマ ドライバは、タイマ アプリケーション用の汎用 API で、MCU 上で使用できる 2 つのタイマ周辺機器（AGT と GPT）をサポートしています。このため、タイマ ドライバは r_agt または r_gpt 上に実装することができます。このセクションでは、e^2 studio ISDE を使用してタイマ ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Timer] > [Timer Driver on r_agt] または [New] > [Driver] > [Timer] > [Timer Driver on r_gpt] を選択することで、タイマ ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE によるタイマ ドライバを使用するアプリケーションの作成

API リファレンスは、次のタイマ インタフェースの説明内に記載されています : タイマインタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

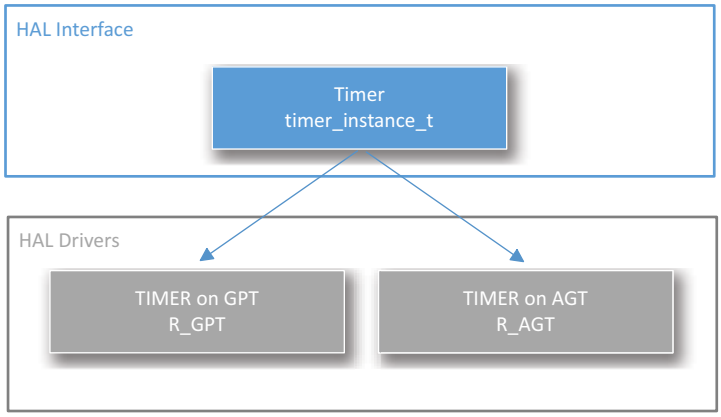

図 153: タイマ ドライバ - ブロック図

4.2.29.1 タイマ ドライバの機能

タイマ ドライバは、ユーザーが指定した期間にタイマを設定します。この期間が経過すると、以下のいずれかのイベントが発生します。

- CPU に割り込み、ユーザー コールバック関数が指定されていればそれを呼び出します。
- ポートピンをトグルします。
- DMAC/DTC を使用してデータを転送します （転送インタフェースで設定されている場合）。
- 別の周辺機器を起動します （ELC インタフェースで設定されている場合）。

下図は、指定した期間後にポート ピンをトグルするか、CPU 割り込を生成するためのフローチャートを示します。

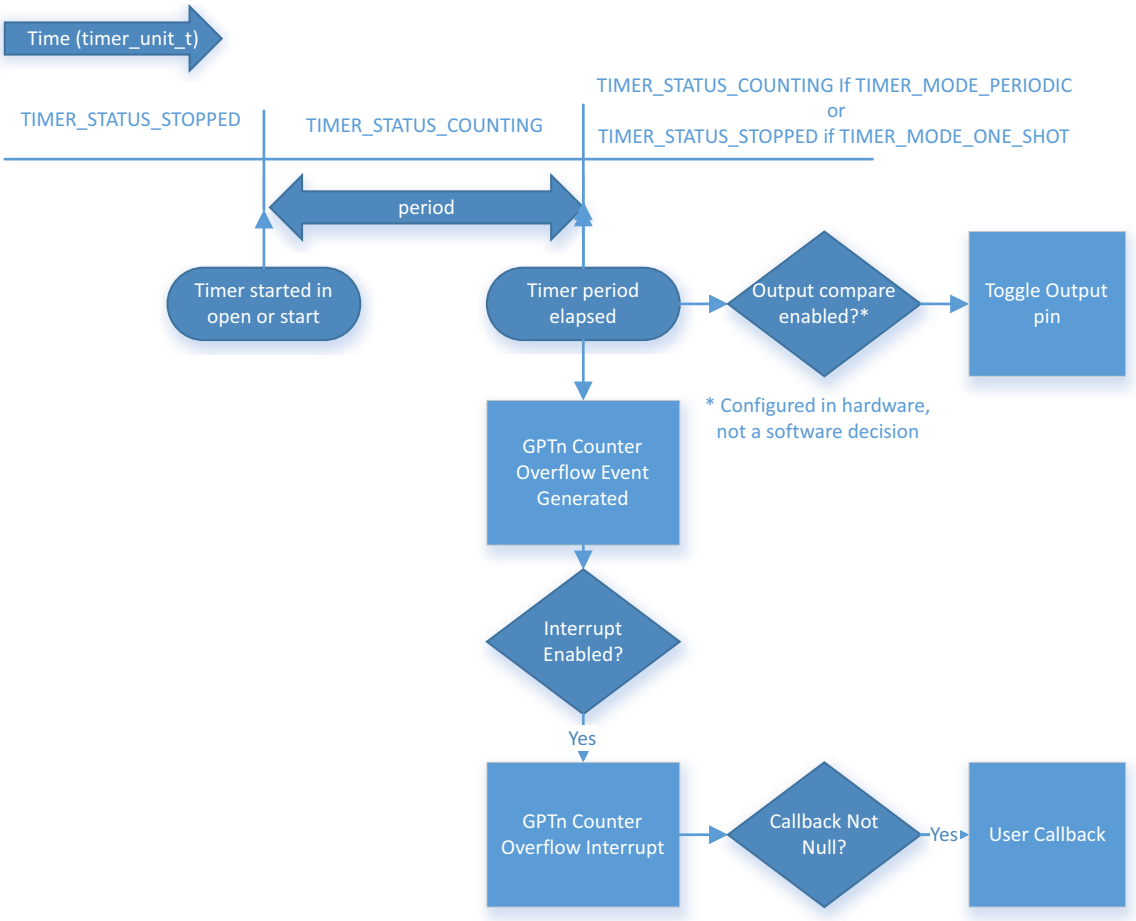

図 154: GPT タイマ - 周期またはワンショット モード

4.2.29.2 タイマ ドライバ用モジュールの選択

GPT モジュールは、ほとんどの汎用タイマ アプリケーションに推奨されますが、基本的なタイマ機能にはどちらのモジュールでも使用できます。一方のタイマ モジュールが他方よりも推奨されるユース ケースを以下で説明します。

GPT タイマ モジュールの選択

GPT モジュールは、PCLKA のみでクロック供給可能な、高解像度の 32 ビット カウンタを使用します。GPT には、MCU 上で SGT よりも多くのチャネルがあるため、GPT を使用することで、リソースが衝突する可能性が低くなります。

AGT タイマ モジュールの選択

AGT モジュールは、PCLKB、LOCO、または Fsub でクロック供給可能な 16 ビット カウンタを使用します。LOCO または Fsub でクロック供給された場合、AGT 割り込みを使用して MCU をスリープ モードから起動できます。2 つのチャネルがあり、チャネル 1 はチャネル 0 のオーバーフローによってクロック供給できるため、事実上 32 ビットのカスケードされたタイマが作成されます。

4.2.29.3 e² studio ISDE によるタイマ ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

GPT でタイマ ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
GPT Driver	Threads	[Driver] > [Timer] > [Timer Driver on r_gpt]。1 つ以上のタイマ ドライバを追加する場合、プロパティ エディターの [Name] フィールドで各モジュールに固有の名前を作成するようにします。
Interrupts	Threads	[GPT] > [GPTn] > [GPTn COUNTER OVERFLOW] オーバーフロー割り込み。
GPT Clock	Clocks	アプリケーション例については、GPT クロックの設定
GPT Pins	Pins	アプリケーション例については、GPT ピンの設定

AGT でタイマ ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
AGT Driver	Threads	[Driver] > [Timer] > [Timer Driver on r_gpt]。1 つ以上のタイマ ドライバを追加する場合、プロパティ エディターの [Name] フィールドで各モジュールに固有の名前を作成するようにします。
Interrupts	Threads	[AGT] > [AGTn] > [AGTI] アンダーフロー割り込み。
AGT Clock	Clocks	アプリケーション例については、AGT クロックの設定
AGT Pins	Pins	アプリケーション例については、AGT ピンの設定

アプリケーションの DAC 割り込みでデータ転送またはイベント トリガが必要な場合に、次のリソースはオプションです。

リソース	ISDE タブ	選択
Data Transfer Controller Driver	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dtc]
DMA Controller	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dmac]
Event Link Controller	Threads	ISDE には、デフォルトで ELC の 1 つのインスタンスが含まれます。

GPT クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して GPT クロックを設定します （クロックの設定を参照）。

GPT タイマは、PCLKA 周波数に基づいてクロック供給されます。PCLKA 周波数を設定するには、実行時に e^2 studio ISDE クロックの設定または CGC インタフェースのクロック コンフィギュレータを使用します。

最大期間は、タイマの種類 （32 ビットまたは 16 ビット）に依存します。

120 MHz で動作する PCLKA を使用した 32 ビット解像度の GPT では、最大期間は約 36650 秒であり、10 時間を少し超えます。

32 MHz で動作する PCLKA を使用した 16 ビット解像度の GPT では、最大期間は約 2.09 秒です。

GPT ピンの設定

e2 studio ISDE を使用して、[Pins] タブから GPT ピンを設定します （ピンの設定を参照）。

GPT との出力比較を使用するには、出力ピンとして使用する I/O ポート ピンが、ピン コンフィギュレータで GPT 周辺機器 （GTIOC）ピンとして設定されている必要があります （ピンの設定を参照）。

ピン コンフィギュレータでは、関連するピンのために IOPORT_CFG_PERIPHERAL_PIN と IOPORT_PERIPHERAL_GPT1 が pin_cfg フィールドで設定されます。

GPT 割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから GPT 割り込みを設定します （割り込みの設定を参照）。

以下の状況では、選択した使用チャネルに対する GPT カウンタ オーバーフロー割り込みを BSP で有効にする必要があります。

1) タイマ期間が経過したらソフトウェア割り込みを取得する。
2) ワンショット モードを使用する。

割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、GPT > GPTn COUNTER OVERFLOW 割り込み （n は GPT チャネル番号）の優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_GPTn_COUNTER_OVERFLOW に、選択した優先度が設定されます。

BSP で GPTn COUNTER OVERFLOW 割り込みが有効になっている場合、対応する ISR がタイマ ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

Attention : irq が TRANSFER_IRQ_END に設定された DTC 周辺機器と共に使用した場合、割り込みはスキップされることがあります。

タイマ ドライバ用の GPT パラメータの設定

e2 studio ISDE を使用して、r_gpt に実装されたタイマ ドライバのパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

Note : ほとんどのパラメータは、すべてのタイマに共通であり、タイマ インタフェースの構成構造体 timer_cfg_t に設定されます。アプリケーションで出力ピンを使用する場合は、タイマが一致する場合にピン出力を HIGH と LOW のどちらにするかなど、GPT ピン固有のパラメータも timer_on_gpt_cfg_t で設定する必要があります。これらの構造体は、どちらも設定入力に基づいて ISDE により生成されます。

この表において、「プロパティ」は ISDE 内での [Properties] タブ名を表します。

GPT の設定

ISDE プロパティ	設定	設定値	説明
Mode	mode	One Shot, Periodic Default: Periodictimer_mode_t	Attention：ワンショット機能は、GPT ハードウェアでは使用できません。そのため、期限が切れたときに呼び出される ISR 内でタイマを停止することにより、ソフトウェアで実装されています。そのため、ワンショット モードでは、コールバックを使用しない場合でも ISR を有効にする必要があります。
Period Value	period	timer_period_t	次のタイマ期間の計算を参照してください。
Period Unit	unit	timer_unit_t	次のタイマ期間の計算を参照してください。
Channel	channel	0-13 for S7G2, 0-9 for S3A7, 0-7 for S124	物理ハードウェア チャネル。
Auto Start	autostart	True, false Default: true	設定後にタイマを起動するには、true に設定します。start を呼び出すまで測定を無効状態にするには false に設定します。
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、タイマの期間が経過するたびに割り込みサービス ルーチン （ISR）から呼び出されます。 Attention：コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

ISDE プロパティ	設定	設定値	説明
GTIOCA/B Output Enabled	timer_on_gpt_cfg_t	True, False Default: False	GPT 用に設定されたポート ピン上でタイマ信号を出力するには、true を設定します。タイマ信号を出力しない場合は false を設定します。
GTIOCA/B Stop Level	timer_on_gpt_cfg_t	Pin Level Low, Pin Level High, Pin Level Retained	タイマが停止したときの出力ピン レベルを制御します。

AGT クロックの設定

AGT タイマは、PCLKB、LOCO、または Fsub 周波数に基づいてクロック供給されます。AGT クロックは、e^2 studio ISDE の [Properties] ウィンドウで選択できます （「configuring-the-agt-parameters」を参照）。クロック周波数を設定するには、実行時に e^2 studio ISDE クロックの設定または CGC インタフェースのクロック コンフィギュレータを使用します。

最大周期は、入力クロック周波数に依存します。

60 MHz で動作する PCLKB を使用した 16 ビット解像度の AGT では、最大周期は約 8.7 ms です。

32 kHz で動作する Fsub を使用した 16 ビット解像度の AGT では、最大周期は約 2.0 秒です。

AGT ピンの設定

AGT との出力比較を使用するには、出力ピンとして使用する I/O ポート ピンが、ピン コンフィギュレータで AGT 周辺機器 （AGTO または AGTIO）ピンとして設定されている必要があります （ピンの設定を参照）。

ピン コンフィギュレータでは、関連するピンのために IOPORT_CFG_PERIPHERAL_PIN と IOPORT_PERIPHERAL_AGT が pin_cfg フィールドで設定されます。

AGT 割り込みの設定

以下の状況では、選択した使用チャネルに対する AGT カウンタ オーバーフロー割り込みを BSP で有効にする必要があります。

1) タイマ期間が経過したらソフトウェア割り込みを取得する。
2) ワンショット モードを使用する。

割り込みを有効にするには、e^2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、AGT > AGTn > AGTn AGTI 割り込み （n は AGT チャネル番号）の優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_AGTn_AGTI に、選択した優先度が設定されます。

BSP で AGTn AGTI 割り込みが有効になっている場合、対応する ISR がタイマ ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

Attention : irq が TRANSFER_IRQ_END に設定された DTC 周辺機器と共に使用した場合、割り込みはスキップされることがあります。

タイマ ドライバ用の AGT パラメータの設定

e^2 studio ISDE を使用して、r_agt に実装されたタイマ ドライバのパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

Note : ほとんどのパラメータは、すべてのタイマに共通であり、タイマ インタフェースの構成構造体 timer_cfg_t に設定されます。アプリケーションが出力ピンを使用する場合は、未定。

AGT の設定

ISDE プロパティ	設定	設定値	説明
Mode	mode	ワンショット、周期。デフォルトは周期 timer_mode_t	Attention : ワンショット機能は、GPT ハードウェアでは使用できません。そのため、期限が切れたときに呼び出される ISR 内でタイマを停止することにより、ソフトウェアで実装されています。そのため、ワンショット モードでは、コールバックを使用しない場合でも ISR を有効にする必要があります。
Period	period	timer_period_t	次のタイマ期間の計算を参照してください。
Unit	unit	timer_unit_t	次のタイマ期間の計算を参照してください。
Channel	channel	0-13 for S7G2, 0-9 for S3A7, 0-7 for S124	物理ハードウェア チャネル。
Autostart	autostart	True, false Default: True	設定後にタイマを起動するには、true に設定します。start を呼び出すまで測定を無効状態にするには false に設定します。

ISDE プロパティ	設定	設定値	説明
Callback	p_callback	User-defined	ユーザー コールバック関数を open で登録できます。このコールバック関数が指定されている場合、タイマの期間が経過するたびに割り込みサービス ルーチン （ISR）から呼び出されます。 Attention：コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。
Output Inverted	output_inverted	true, false	出力信号 low を開始するには false を設定します。出力信号 high を開始するには true を設定します。
Count Source	count_source	Clock PCLCB, Clock PCLKB/8, Clock PCLKB/2, Clock LOGO, Clock AGT0 Underflow, Cock AGT0 Fsub	AGT カウンタのクロック ソース。
AGTO Output enabled	agto_output_enabled	true, false	AGT 用に設定されたポート ピン （AGTO ピン）上でタイマ信号を出力するには、true を設定します。タイマ信号を出力しない場合は false を設定します。
AGTIO Output enabled	agtio_output_enabled	true, false	AGT 用に設定されたポート ピン （AGTIO ピン）上でタイマ信号を出力するには、true を設定します。タイマ信号を出力しない場合は false を設定します。

4.2.29.4 タイマ ドライバ使用上の注意

GPT 出力タイマ信号

タイマ出力が設定されている場合 （GTIOCA/B 出力有効に True が設定されている場合）、出力ピンは GTIOCA/B 停止レベルで開始され、期間が経過するたびに切り替わります。最初の切り替えは、タイマ起動後に初めて期間が経過したときに起こります。

ワンショット モードでは、タイマのカウントが開始されるときにも切り替わるように出力が設定されます。これによりパルスが生成されます。タイマは、カウントが始まるときに停止レベルから切り替わり、カウントが終了するときに停止レベルに戻ります。

タイマ期間の計算

タイマ期間は、タイマが期限切れになるまでの時間として定義されます。出力比較を使用する場合、出力ピンが期間あたり 1 回切り替わるため、従来の期間 （立ち上がりエッジから立ち上がりエッジ）は、ソフトウェアで指定された期間の 2 倍になります。

現在のクロック設定に基づく実行時期間計算は、open および periodSet から使用できます。

タイマ期間が raw カウント以外に指定された場合、現在のタイマ クロック周波数を使用して期間が計算されます （GPT クロックの設定または AGT クロックの設定を参照してください）。現在のクロック周波数は、クロック発生回路 （CGC）のレジスタに基づいて決定され、PLL がメイン クロックの場合は、BSP マクロ #define BSP_CFG_XTAL_HZ で指定された水晶周波数も使用されます。この周波数は、次の表の適切な式で、clk_freq_hz として使用されます。

表 : タイマ期間の計算

タイマ単位 unit	式
TIMER_UNIT_PERIOD_NSEC	counts = (period * clk_freq_hz) / 1000000000
TIMER_UNIT_PERIOD_USEC	counts = (period * clk_freq_hz) / 1000000
TIMER_UNIT_PERIOD_MSEC	counts = (period * clk_freq_hz) / 1000
TIMER_UNIT_PERIOD_SEC	counts = (period * clk_freq_hz)
TIMER_UNIT_FREQUENCY_HZ	counts = (clk_freq_hz) /period
TIMER_UNIT_FREQUENCY_KHZ	counts = (clk_freq_hz) / (1000 * period)

必要な期間がカウンタ サイズ （32 ビットまたは 1 ビット）よりも長い場合、ドライバは、結果が現在のサイズに収まる最も小さい除数を選択します。カウンタ値が、最大の除数 （1024）を持つカウンタ サイズよりも大きい場合は、エラー コード （SSP_ERR_INVALID_ARGUMENT）が返されます。

GPT を使用した DMAC/DTC のトリガー

タイマ期間が経過したときに、DMAC または DTC 周辺機器を使用してデータの転送をトリガするには、activation_source を ELC_EVENT_GPTn_COUNTER_OVERFLOW （n は GPT チャネル番号）に設定して DMAC/DTC 転送を設定します。詳細については、転送インタフェースを参照してください。

I a:DTC を使用したワンショット モードでタイマを使用する場合は、転送全体が完了してから割り込みによりタイマが停止します（irq が TRANSFER_IRQ_END に設定されている場合）。タイマ期間が経過した後に 1 回の転送のみを生成するには、irq を TRANSFER_IRQ_EACH に設定するか、DMAC を使用して転送します。

GPT を使用した ELC イベントのトリガー

GPT タイマは、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

AGT を使用した DMAC/DTC のトリガー

タイマ期間が経過したときに、DMAC または DTC 周辺機器を使用してデータの転送をトリガするには、activation_source を ELC_EVENT_AGTn_AGTI （n は AGT チャネル番号）に設定して DMAC/DTC 転送を設定します。詳細については、転送インタフェースを参照してください。

Attention：DTC を使用したワンショット モードでタイマを使用する場合は、転送全体が完了してから割り込みによりタイマが停止します （irq が TRANSFER_IRQ_END に設定されている場合）。タイマ期間が経過した後に 1 回の転送のみを生成するには、irq を TRANSFER_IRQ_EACH に設定するか、DMAC を使用して転送します。

AGT を使用した ELC イベントのトリガー

AGT タイマは、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

4.2.29.5 タイマ ドライバ アプリケーションの作成

```
/* Instance structure to use this module. */

const timer_instance_t g_timer =

{

    .p_ctrl = &g_timer_ctrl,

    .p_cfg = &g_timer_cfg,

    .p_api = &g_timer_on_gpt

};
```

GPT を使用したタイマ アプリケーションを作成するには、以下の手順を実行します。

1) モジュールを上の説明のように設定します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます （以下の構造体が含まれます : timer_cfg_t および timer_on_gpt_cfg_t）。

2) 設定時にタイマに付けた名前を使用して、タイマ インスタンスをオープンします。タイマ インタフェースを通じて適切なドライバが呼び出されます。g_timer というタイマの場合は以下のようになります。

```
g_timer.p_api->open(g_timer.p_ctrl, g_timer.p_cfg)
```

タイマの設定手順後に、g_timer.p_ctrl と g_timer.p_cfg が自動生成されます。

3) 自動起動が False に設定された場合は、次のインタフェースを呼び出してタイマを起動します。

```
g_timer.p_api->start(g_timer.p_ctrl)
```

4) 必要に応じて、timer_api_t から他の API を使用します。

タイマの制限事項

GPT の電源をオフにする

GPT モジュールは、close() API で GPT のモジュール ストップ ビットを設定しません。これは意図的なもので、GPT モジュール ストップ ビットは複数の GPT チャネルを制御し、GPT モジュールには他の GPT モジュールがアプリケーションで使用されているかどうかを判断することはできないためです。GPT にモジュール ストップ ビットを設定し、使用中の GPT チャネルがない場合に電力消費量を削減するには、この手順に従ってください。

1) MCU のユーザーズマニュアルを開きます （たとえば、r01um0001eu0085_synergy_s7g2.pdf は S7G2 マニュアルのバージョン 0.85 です）。

2) 「ローパワー モード （Low Power Modes）」の章で、「モジュール ストップ制御レジスタ D （Module Stop Control Register D）」の「レジスタの説明 （Register Descriptions）」の章を確認します。

3) MSTPD5 および MSTPD6 を確認し、アプリケーションで使用するチャネルがどちらのビットに含まれているかを調べます。

4) Synergy 構成ツール （configuration.xml）の [Threads] タブで次の順に選択し、プロジェクトに LPM ドライバを追加します : [New] > [Driver] > [Power] > [Low Power Modes Driver on r_lpm]。

5) アプリケーション コードで、手順 3 で特定した MSTP ビット内の他の GPT チャネルが、アプリケーションにより現在使用されていないことを確認します。使用中のチャネルがなければ、LPM API を使用して GPT チャネル セットの電源をオフにします。a. g_lpm0 という名前の LPM モジュールを使用した MSTPD5 により制御された GPT チャネルの電源をオフにするには、以下を呼び出します。

```
g_lpm0->p_api. moduleStop(LPM_MSTP_GPT_CH7_0);  // Ignore the channel numbers in the enum value – this is for MSTPD5
```

b. MSTPD6 により制御された GPT チャネルの電源をオフにするには、以下を呼び出します。

```
g_lpm0->p_api. moduleStop(LPM_MSTP_GPT_CH13_8); // Ignore the channel numbers in the enum value – this is for MSTPD6
```

4.2.29.6 タイマ ドライバでサポートされるデバイス

このドライバは、次の MCU でテストされています。

- S7G2
- S3A7
- S124

4.2.29.7 タイマ ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL Timer Interface API	synergy/ssp/inc/driver/api/r_timer_api.h
GPT Instance	synergy/ssp/inc/driver/instances/r_gpt.h
GPT Driver	synergy/ssp/src/driver/r_gpt/r_gpt.c
AGT Instance	synergy/ssp/inc/driver/instances/r_agt.h
AGT Driver	synergy/ssp/src/driver/r_agt/r_agt.c

4.2.30 転送ドライバ

転送ドライバは、転送アプリケーション用の汎用ドライバであり、MCU 上で使用できる 2 つの転送周辺機器 （DMAC と DTC）をサポートしています。このため、ドライバは r_dmac および r_dtc 上に実装することができます。このセクションでは、e^2 studio ISDE を使用して転送ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Transfer] > [Transfer Driver on r_dmac] または [New] > [Driver] > [Transfer] > [Transfer Driver on r_dtc] を選択することで、転送ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による転送ドライバを使用するアプリケーションの作成

API リファレンスは、次の転送インタフェースの説明内に記載されています : 転送インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

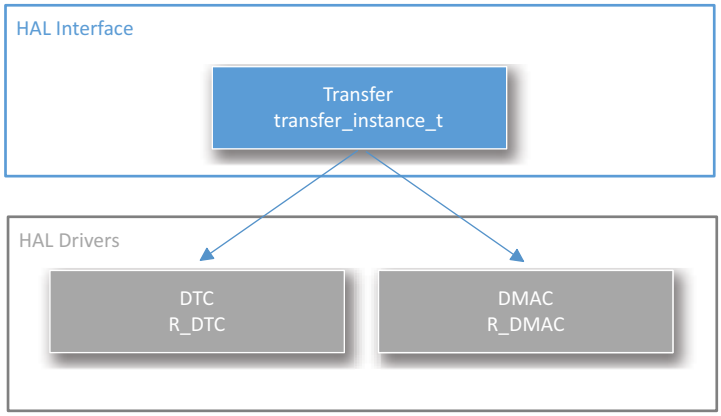

図 155: 転送ドライバ - ブロック図

4.2.30.1 転送ドライバの機能

転送ドライバは、割り込みまたはイベントの発生時に、ユーザー指定のソースからユーザー指定の宛先にデータを移動します。

4.2.30.2 転送ドライバ用モジュールの選択

ほとんどの汎用転送アプリケーションでは DTC モジュールが推奨されますが、基本的な転送機能を実行するにはどちらのモジュールでも使用できます。一方の転送モジュールが他方より推奨されるユース ケースを以下で説明します。

DTC 転送モジュールの選択

DTC モジュールでは、システム内のすべての割り込み用のスロットを持つ、RAM ベースのベクトル テーブルが使用されます。DTC 転送が完了すると、アクティベーション ソース割り込みが呼び出されます。DTC を使用するには、アクティベーション ソース割り込みを有効にする必要があります。一般に、アクティベーション ソース割り込みは、TRANSFER_IRQ_EACH が構成で指定されていないかぎり、転送が完了するまで DTC によってミュートされます。たとえば、長さが 16 である通常モード転送がタイマによってトリガーされた場合、転送が有効である間、最初の 15 回のタイマ割り込みは発生しません。タイマ割り込みは 16 番目の転送の後で発生します。DTC ではチェーンされた転送も許可されます。つまり、単一のアクティベーション ソース割り込みの後で複数の転送が発生できます。この機能はドライバによってサポートされていますが、e^2 studio ISDE の外部で構成する必要があります。

DMAC 転送モジュールの選択

DMAC モジュールは DMAC 周辺レジスタを使用するため、システム内の転送数はデバイス上の DMAC チャネル数に制限されます。DMAC を使用するためにアクティベーション ソースを有効にする必要はありません。DMAC 転送が完了すると、DMAC 割り込みが呼び出されます。アクティベーション ソース割り込みが有効になっている場合は、転送がトリガーされるのと同時に発生します。DMAC 割り込みが有効にされている場合は、すべての転送が完了した後で発生します。たとえば、長さが 16 であるノーマル モード転送がタ

イマによってトリガされた場合、タイマ割り込みは各割り込みが発生するのと同時に発生し、DMAC 割り込みは 16 番目の割り込みが完了した後で発生します。DMAC ではチェーンされた転送はサポートされません。

4.2.30.3 e2 studio ISDE による転送ドライバを使用するアプリケーションの作成

ドライバは、e2 studio ISDE に組み込まれています （e2 studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

DTC で転送ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
DTC Driver	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dtc]。1 つ以上の転送ドライバを追加する場合、プロパティ エディターの [Name] フィールドで各ドライバに固有の名前を作成するようにします。
Interrupts	Threads	アクティベーションソースに対応する割り込みを有効にする必要があります。
DTC Clock	Clocks	アプリケーション例については、DTC クロックの設定
DTC Pins	Pins	アプリケーション例については、DTC ピンの設定

DMAC で転送ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
DMAC Driver	Threads	[Driver] > [Transfer] > [Transfer Driver on r_dmac]。1 つ以上の転送ドライバを追加する場合、プロパティ エディターの [Name] フィールドで各ドライバに固有の名前を作成するようにします。

リソース	ISDE タブ	選択
Interrupts	Threads	[DMAC] > [DMACn] > [DMACn INT] アンダーフロー割り込み。
DMAC Clock	Clocks	アプリケーション例については、DMAC クロックの設定
DMAC Pins	Pins	アプリケーション例については、DMAC ピンの設定

DTC クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して DTC クロックを設定します （クロックの設定を参照）。

DTC は、ICLK 周波数に基づいてクロック供給されます。ICLK 周波数を設定するには、実行時に e^2 studio ISDE クロックの設定または CGC インタフェースのクロック コンフィギュレータを使用します。

DTC ピンの設定

DTC はいずれのピンとも関連付けられていません。

DTC 割り込みの設定

e^2 studio ISDE を使用して、[ICU] タブから DTC 割り込みを設定します （割り込みの設定を参照）。

DTC を使用するには、アクティベーション ソース割り込みを有効にする必要があります。

l a:irq が TRANSFER_IRQ_END に設定された DTC 周辺機器と共に使用した場合、割り込みはスキップされることがあります。

DMAC クロックの設定

e^2 studio ISDE で、[Clocks] タブを使用して DMAC クロックを設定します （クロックの設定を参照）。

DMAC は、ICLK 周波数に基づいてクロック供給されます。ICLK 周波数を設定するには、実行時に e^2 studio ISDE クロックの設定または CGC インタフェースのクロック コンフィギュレータを使用します。

DMAC ピンの設定

DMAC はいずれのピンとも関連付けられていません。

DMAC 割り込みの設定

DMAC を使用するのに割り込みは必要ありません。転送の完了時に割り込みを受信するには、e^2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、DMAC > DMACn > DMACn INT 割り込み（n は DMAC チャネル番号）を有効にします。

DMACn INT 割り込みが BSP で有効になっている場合、対応する ISR が DMAC ドライバで定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

4.2.30.4 転送ドライバ使用上の注意

ノーマル モード

通常モードでは、アクティベーション ソース イベントが発生するたびに単一の転送がトリガーされます。単一の転送は、size で選択された設定に応じて、1 バイト、2 バイト、または 4 バイトです。転送が発生するたびに、length が 1 デクリメントされます。length が 0 に達すると、転送は完了です。

リピート モード

リピート モードでは、アクティベーション ソース イベントが発生するたびに単一の転送がトリガーされます。単一の転送は、size で選択された設定に応じて、1 バイト、2 バイト、または 4 バイトです。転送が発生するたびに、length が 1 デクリメントされます。length が 0 に達すると、length に初期値がリロードされ、転送が再開始されます。リピート エリアがソースに設定されている場合、転送の再開始時にソース レジスタにも初期値がリロードされます。また、リピート エリアが宛先に設定されている場合、転送の再開始時に宛先レジスタに初期値がリロードされます。

ブロック モード

ブロック モードでは、アクティベーション ソース イベントが発生するたびに転送長全体が転送されます。たとえば、アクティベーション ソースがタイマ、バイト サイズが 2、バイト長が 12 である転送がブロック モードで構成されている場合、アクティベーション イベントが発生するたびに 24 バイトが転送されます。転送が発生するたびに、length が 1 デクリメントされます。length が 0 に達すると、length に初期値がリロードされ、転送が再開始されます。リピート エリアがソースに設定されている場合、転送の再開始時にソース レジスタにも初期値がリロードされます。また、リピート エリアが宛先に設定されている場合、転送の再開始時に宛先レジスタに初期値がリロードされます。

アドレス モード

size （1 バイト、2 バイト、または 4 バイト）が転送されるたびに、ソース ポインタと宛先ポインタはそれぞれ src_addr_mode と dest_addr_mode によって調整されます。たとえば、src_addr_mode が TRANSFER_ADDR_MODE_INCREMENTED に設定され、size が TRANSFER_SIZE_4_BYTES に設定されている場合、p_dest ポインタは転送のたびに 4 （転送サイズ）インクリメントされます。TRANSFER_ADDR_MODE_FIXED に設定されている場合、ポインタは変化しません。

チェーンされた転送

チェーンされた転送は DTC でのみサポートされています。チェーンされた転送を使用するには、transfer_info_t 構造体の配列を作成します。最後の転送を除くすべての転送について、chain_mode を TRANSFER_CHAIN_MODE_ENABLED に設定します。p_info を、transfer_info_t 構造体用の配列内にある最初の構造体のベースに設定します。

4.2.30.5 転送ドライバ アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const transfer_instance_t  g_transfer =

{

    .p_ctrl  = &g_transfer_ctrl,

    .p_cfg = &g_transfer_cfg,

    .p_api = &g_transfer_on_dtc

};
```

DTC を使用する転送アプリケーションを作成するには、次の手順を実行します。

1) モジュールを上の説明のように設定します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます （以下の構造体が含まれます : transfer_cfg_t。）

2) 設定時に転送に付けた名前を使用して、転送インスタンスをオープンします。転送インタフェースを通じて適切なドライバが呼び出されます。g_transfer という転送の場合は、以下のようになります。

```
g_transfer.p_api->open(g_transfer.p_ctrl, g_transfer.p_cfg)
```

転送の設定手順後に、g_transfer.p_ctrl と g_transfer.p_cfg が自動生成されます。

3) Auto Enable が False に構成された場合、呼び出しまたは次のものによって転送を有効にします。

```
g_transfer.p_api->enable(g_transfer.p_ctrl)
```

または

```
g_transfer.p_api->reset(g_transfer.p_ctrl, p_src, p_dest, length)
```

4) 必要に応じて、transfer_api_t から他の API を使用します。

4.2.30.6 転送ドライバでサポートされるデバイス

このドライバは、S7G2 と S3A7 でテストされています。

転送ドライバは、API への変更なしに、次のファミリをサポートするように設計されています。

- S124

4.2.30.7 転送ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが /ssp ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL Transfer Interface API	synergy/ssp/inc/driver/api/r_transfer_api.h
DTC Instance	synergy/ssp/inc/driver/instances/r_dtc.h
DTC Driver	synergy/ssp/src/driver/r_dtc/r_dtc.c
DMAC Instance	synergy/ssp/inc/driver/instances/r_dmac.h
DMAC Driver	synergy/ssp/src/driver/r_dmac/r_dmac.c

4.2.31 UART ドライバ

UART ドライバは r_sci_uart に実装されており、UART プロトコルを使用したシリアル通信用の汎用 API を含みます。このドライバは、MCU で利用可能な SCI 周辺機器および UART モードをサポートします。このセクションでは、e^2 studio ISDE を使用して UART ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Connectivity] > [UART Driver on r_sci_uart] を選択することで、UART ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE による UART ドライバを使用するアプリケーションの作成。

API リファレンスは、次の UART インタフェースの説明内に記載されています : UART インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、SSP Architecture を参照してください。

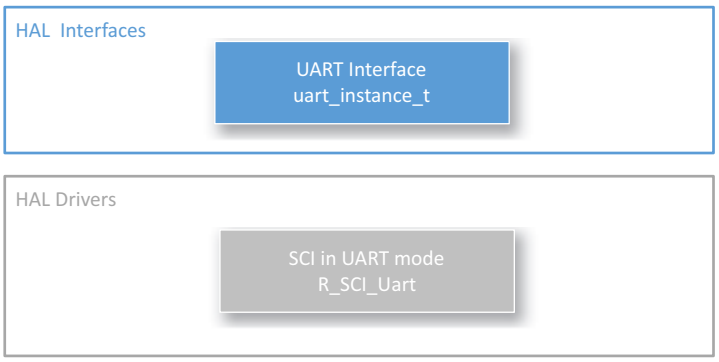

図 156: UART ドライバ - ブロック図

4.2.31.1 UART ドライバの機能

UART ドライバは、汎用 UART プロトコルをサポートしています。

UART モード（UARTon SCI）の SCI 周辺機器で使用される UART ドライバは、標準的な UART プロトコルに加えて、次の機能をサポートしています。

- 全二重 UART 通信
- 複数のチャネルを使用した同時通信
- 割り込み駆動型のデータ送信と受信
- イベント コードを引数として渡す、ユーザー コールバック関数の呼び出し
- 実行時のボーレートの変更
- UART トランザクション中のハードウェア リソースのロック
- CTS/RTS ハードウェア フロー制御 （関連付けられた IOPORT ピンを使用し、ユーザー定義コールバック関数でサポート）
- DTC 転送モジュールとの統合

4.2.31.2 e² studio ISDE による UART ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e² studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

UART と SCI モジュールを使用する場合は、次のリソースを追加します。

リソース	ISDE タブ	選択
UART Driver for UART on SCI	Threads	[Driver] > [Communications] > [UART Driver on r_sci_uart]
SCI Common driver	Threads	SCI 共通がスレッドに自動的に追加されます。このドライバの [Properties] ウィンドウで、調歩同期式モードを有効化します。
Interrupts	ICU	[SCI] > [SCIn SCIn RXI] および [SCIn TXI]、[SCIn TEI]、[SCIn ERI] 割り込みを、選択されたチャネルに対して設定します。

UARTonSCI を使用した UART ドライバ用のクロックの設定

e2 studio ISDE で、[Clocks] タブを使用して UARTonSCI クロックを設定します （クロックの設定を参照）。

SCI モジュールは、次のクロックを使用できます。

- PCLKA （S124 の場合は PCLKB）
- 選択したチャネル n に対する SCKn ピンからの外部クロック

UARTonSCI を使用した UART ドライバ用のピンの設定

e2 studio ISDE を使用して、[Pins] タブから UARTonSCI ピンを設定します （ピンの設定を参照）。

選択した SCI チャネルの UARTonSCI RX および TX ピンを設定します。

I :S7G2 PE-HMI-WVGA1 ボード上の UARTonSCI モジュールは、PORT5 ピン 8 （ピン P508）および PORT11 ピン 6 （ピン PB06）を使用して、RS-232C/RS-485 トランシーバー上で RS232C ポートをアクティブ化します。ピン P508 と PB06 を IOPORT ピンとして設定し、その出力レベルを HIGH に設定します。

4.2.31.3 UARTonSCI を使用した UART ドライバ用の割り込みの設定

UARTonSCI RXI 割り込み

RXI 割り込みは、UART ポートから受信するデータのフローを制御するために使用されます。受信データ数が期待される読み取り長に達した場合、ISR はユーザー定義のコールバック関数を呼び出して、受信データが完了したことを通知します。外部 RTS 操作オプションが有効になっている場合、ISR は RTS 外部ピン制御のための UART コールバック関数を 2 回呼び出します （ISR の先頭と ISR の最後）。コールバック関数を使用して、RTS 関数をエミュレートできます （「UARTonSCI ハードウェア フロー制御」を参照）。この割り込みは、open で、構成パラメータ SCI_UART_CFG_RX_ENABLE で受信が有効になっている限りアクティブ化されます。

Note : 割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、SCI > SCIn RXI 割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_SCIn_RXI に、選択した優先度が設定されます。

UARTonSCI TXI 割り込み

TXI 割り込みは、write による要求に従い、UART ポートへの連続するデータ転送を処理します。送信循環バッファにデータがなくなると、ISR は TXI 割り込みを非アクティブ化し、TEI 割り込みをアクティブにして、データ送信の最後のシーケンスを処理します。この割り込みは、write で、構成パラメータ SCI_UART_CFG_TX_ENABLE によって送信が有効になっている限りアクティブ化されます。

Note : 割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、SCI > SCIn TXI 割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_SCIn_TXI に、選択した優先度が設定されます。

UARTonSCI TEI 割り込み

TEI 割り込みは、write によって要求された UART ポートへの最後のデータ転送を処理するために使用されます。この割り込みは、TXI ISR によりアクティブ化され、自動的に非アクティブ化されます。

Note：割り込みを有効にするには、e^2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、SCI > SCIn TEI 割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_SCIn_TEI に、選択した優先度が設定されます。

UARTonSCI ERI 割り込み

ERI 割り込みは、UART 受信で発生したエラーを処理するために使用されます。この割り込みは、open で、構成パラメータ SCI_UART_CFG_RX_ENABLE によって受信が有効になっている限りアクティブ化されます。

Note：割り込みを有効にするには、e^2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、SCI > SCIn ERI 割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_SCIn_ERI に、選択した優先度が設定されます。

4.2.31.4 UARTonSCI 用の UART ドライバ パラメータの設定

e^2 studio ISDE を使用して、UART HAL ドライバ パラメータを設定します。

RTOS を使用しないアプリケーションの場合：HAL ドライバの追加と設定。

ThreadX アプリケーションの場合：ドライバのスレッドへの追加とドライバの設定。

表：SCI の共通設定

ISDE プロパティ	設定	設定値	説明
Asynchronous Mode (r_sci_uart)	mode	Enabled, Disabled (Default)	非同期モード有効。UART ポートとして使用する任意の SCI で有効にする必要があります。

表：SCI_UART 上の UART ドライバの設定

ISDE プロパティ	設定	設定値	説明
Parameter checking	#define SCI_UART_CFG_PARAM_CHECKING_ENABLE	Use system setting (Default), Enabled, Disabled	パラメータ エラー チェックを有効または無効にします。

ISDE プロパティ	設定	設定値	説明
Name	name	Arbitrary symbol (Default: "g_uart")	UARTonSCI モジュール制御ブロック インスタンスに使用する名前。この名前は、他の可変インスタンスのプレフィックスにも使用されます。
Name of UART callback function to be defined by user	p_callback	Arbitrary symbol (Default: "user_uart_callback")	名前は有効な C シンボルである必要があります。
External RTS operation	#define SCI_UART_CFG_EXTERNAL_RTS_OPERATION	Enable, Disable (Default)	RTS 信号として使用する IOPORT ピンを有効にします。RTS 機能では、この設定パラメータを「有効」にし、設定「RTS 外部ピン制御のための UART コールバック関数名」を指定します。
Reception	#define SCI_UART_CFG_RX_ENABLE	Enable (Default), Disable	SCI 上のすべての UART チャネルについて UART の受信を有効または無効にします。この設定パラメータに「無効」を設定すると、コード サイズが小さくなります。UART 受信のためのコード部分がコンパイルされないためです。このパラメータは、個々の UARTT チャネルに対しては設定できません。
Transmission	#define SCI_UART_CFG_TX_ENABLE	Enable (Default), Disable	SCI 上のすべてのチャネルについて UART 送信を有効または無効にします。この設定に「無効」を設定すると、UART 送信のためのコード部分がコンパイルから除外されるため、コード サイズが小さくなりますが、この設定に「無効」を設定できるのは、UART ポートとして機能する他の SCI チャネルが送信を行わない場合だけです。

ISDE プロパティ	設定	設定値	説明
Channel	channel	Integer between 0 and 9	SCI チャネル番号
Data Bits	data_bits	7bit, 8bit (Default), 9 bit	UART データ ビット
Parity	parity	None (Default), Odd, Even	UART パリティ ビット
Stop Bits	stop_bits	1 stop bit (Default), 2 stop bits	UART ストップ ビット
CTS/RTS Selection	ctsrts_en	CTS (Default), RTS	SCI チャネル n の CTSn/RTSn ピンに対して CTS または RTS を選択します。SCI ハードウェアは、このピン上で CTS と RTS のいずれかの制御信号をサポートしますが、両方はサポートしません。CTS と RTS の両方を使用するアプリケーションでは、この設定パラメータに「CTS」を選択し、設定「外部 RTS 操作」を有効にし、設定「RTS 外部ピン制御のための UART コールバック関数名」を指定します。
DTC Transfer Module for Transmission	p_transfer_tx	Name of DTC transfer module or NULL if unused	送信のハードウェア加速で使用される DTC 転送モジュール。
DTC Transfer Module for Reception	p_transfer_rx	Name of DTC transfer module or NULL if unused	受信のハードウェア加速で使用される DTC 転送モジュール。
Clock Source	clk_src	Internal Clock(Default), External Clock 8x baudrate, External Clock 16x baudrate	ボーレート クロック発信器ブロックで使用するクロック ソースを選択します。
Baudrate Clock Output from SCK pin	baudclk_out	Enable, Disable (Default)	選択したチャネル n に対し SCKn ピン上でボーレート クロックを出力するためのオプション設定。

ISDE プロパティ	設定	設定値	説明
Start bit detection	rx_edge_start	Falling Edge (Default), Low Level	受信におけるスタートビット検出モード。この設定には、通常は「立ち下がりエッジ」を設定します。
Noise Cancel	noisecancel_en	Enable, Disable (Default)	RXDn ピン上でデジタル ノイズ キャンセルを有効にします。SCI のデジタル ノイズ フィルタ ブロックは、2 ステージのフリップフロップ回路で構成されます。詳細については、MCU ユーザーズマニュアルのノイズ キャンセルのセクションを参照してください。
Name of UART callback function for the RTS external pin control to be defined by user	p_extpin_ctrl	Arbitrary symbol(Default: "NULL")	名前は有効な C シンボルである必要があります。

4.2.31.5 UART ドライバ使用上の注意

UARTonSCI ハードウェア フロー制御

SCI ハードウェア モジュールは、同時に RTS 信号または CTS 信号のいずれかについてハードウェア フロー制御をサポートします。CTS と RTS は、CTSn/RTSn ピンで多重化され、ユーザーが、ユースケースに応じて、いずれかのハードウェア フロー制御信号を排他的に使用できるようになっています。UARTonSCI ドライバ モジュールは、この仕様を拡張し、RTS 信号用の追加のピンを有効にすることで、CTS 信号と RTS 信号の両方を制御できます。このモードを有効にするには、以下のように UARTonSCI を設定します。

- #define SCI_UART_CFG_EXTERNAL_RTS_OPERATION を有効に設定します。
- ctsrts_en を CTS に設定します。
- p_extpin_ctrl で、ユーザー コールバック関数の名前を、「RTS 外部ピン制御のための UART コールバック関数名」に指定します。

UARTonSCI ドライバ モジュールは、以下のように、RXI ISR の処理の先頭と終わりで、定義されたユーザー コールバック関数を呼び出します。

p_extpin_ctrl コールバック関数の引数 "level" は、選択した SCI チャネルの RTS ピンの信号レベルを表します。

Note：ドライバ モジュールは、GPIO ピンを初期化したり、それを制御することはしません。ユーザーは、UART 受信を開始する前に GPIO ピンを初期化する必要があります。

下図は、RTS/CTS ハードウェア フロー制御と、RTS 信号として使用される外部 GPIO ピンのタイミング図を示しています。

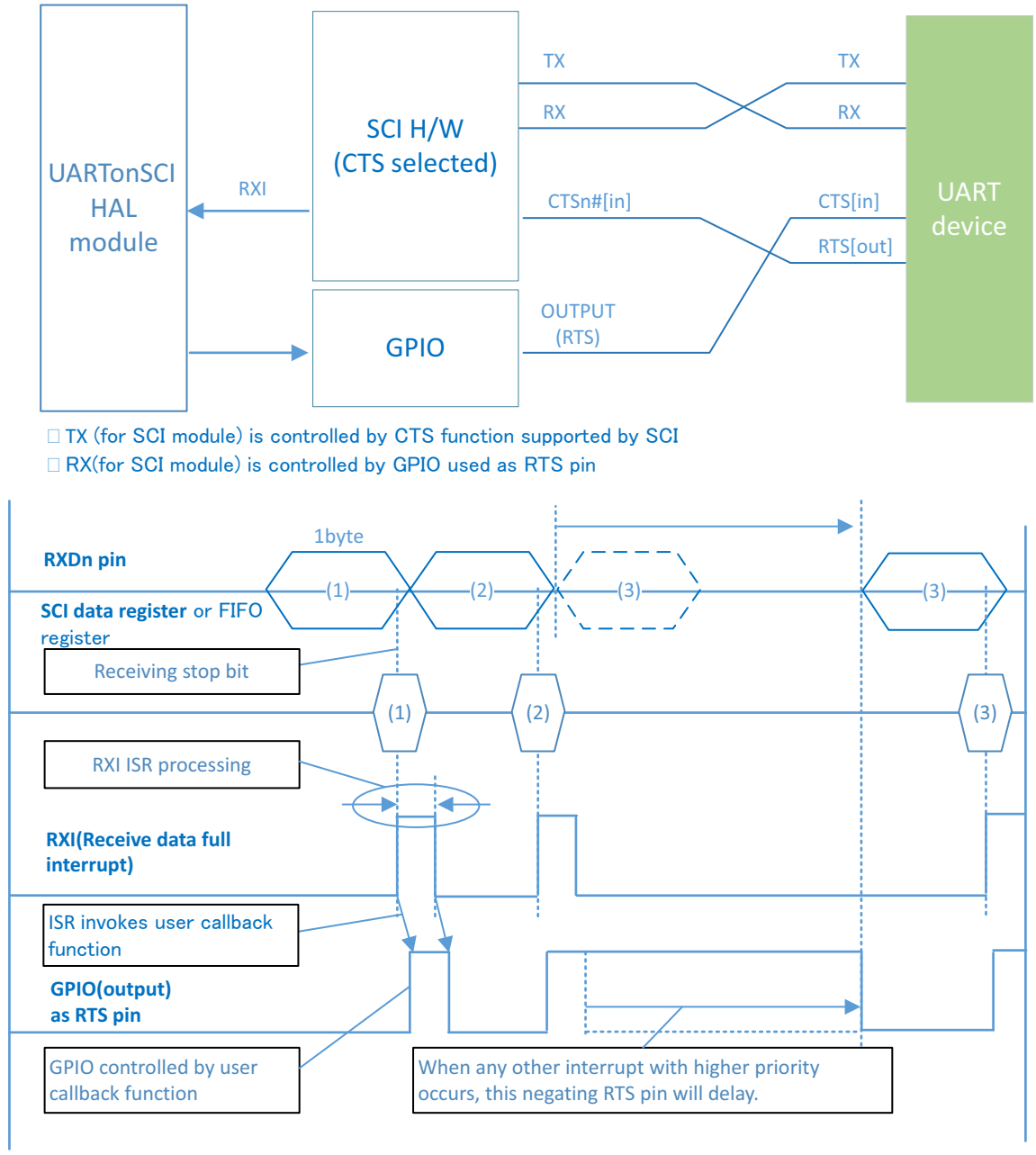

図 157: 外部 GPIO を使用した CTS/RTS ハードウェア制御

4.2.31.6 UARTonSCI を使った UART ドライバ アプリケーションの作成

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const uart_instance_t  g_uart =

{

.p_ctrl  = &g_uart_ctrl,

.p_cfg = &g_uart_cfg,

.p_api = &g_uart_on_sci

};
```

```
// Example code

// Creates a variable for the error

ssp_err_t  err;

// Creates a UART control block (You can use the project generator on
ISDE to generate this configuration)

uart_ctrl_t  g_uart5;

// Creates an extented configuration for UART interface

// (You can use the project generator on ISDE to generate this
configuration)

uart_on_sci_cfg_t  uart_sci5_cfg_t =

{

.clk_src  = SCI_CLK_SRC_INT,

.baudclk_out = false,

.rx_edge_start = true,

.noisecancel_en = false,

.noisecancel_lvl = NOISE_CANCEL_LVL1
```

```
};

// Creates a configuration structure (You can use the project generator
on ISDE to generate this configuration)

uart_cfg_t g_uart5_cfg =

{

.channel = 5, // SCI channel5

.baud_rate = 115200, // 115200bps

.data_bits = UART_DATA_BITS_8, // Data bits 8bits

.parity = UART_PARITY_OFF, // Parity none

.stop_bits = UART_STOP_BITS_1, // Stop bit 1bit

.ctsrts_en = false,

.p_callback = (void *) user_uart_callback, // User callback function

.p_transfer_tx = NULL, // DTC for transmission

.p_transfer_rx = NULL, // DTC for reception

.p_extend = (void *) &uart_sci5_cfg_t // Pointer to the UART
extended configuration

};

// Initializes the SCI channel5 as an UART port

err = g_uart_on_sci.open (&g_uart5, &g_uart5_cfg);
```

4.2.31.7 UART ドライバの制限事項

このドライバは、割り込みベースの操作をサポートしていますが、ポーリング UART モードはサポートしていません。

このドライバは、非バッファリング UART モードをサポートしていません。

このドライバは、イベント リンク機能をサポートしていません。

4.2.31.8 UART ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが ssp/ ディレクトリに抽出されます。

SSP パックのディレクトリとファイル :

モジュール	ディレクトリ
UART Interface API	synergy/ssp/inc/driver/api/r_uart_api.h
UARTonSCI Instance	synergy/ssp/inc/driver/instances/r_sci_uart.h
UARTonSCI Driver	synergy/ssp/src/driver/r_sci_uart

UART ドライバでサポートされるデバイス

このドライバは、S7G2 でテストされています。

UART ドライバは、API への変更なしに、以下のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.32 ウォッチドッグドライバ

ウォッチドッグ ドライバ （WDT ドライバ）は、ウォッチドッグ タイマ アプリケーション用の汎用 API です。このドライバは、MCU で利用可能な WDT および独立 WDT （IWDT）周辺機器をサポートするため、r_wdt および r_iwdt に実装されています。このセクションでは、e^2 studio ISDE を使用してウォッチドッグ ドライバを構成する方法と、API 関数をアプリケーションに組み込む方法について説明します。

e^2 studio ISDE のプロジェクト コンフィギュレータでは、[Threads] タブの [Modules] ペインで [New] > [Driver] > [Monitoring] > [Watchdog Driver on r_wdt] または [New] > [Driver] > [Monitoring] > [Watchdog Driver on r_iwdt] を選択することで、ウォッチドッグ ドライバ モジュールを追加および構成できます。詳細については、以下を参照してください : e^2 studio ISDE によるウォッチドッグ ドライバを使用するアプリケーションの作成

API リファレンスは、次の WDT インタフェースの説明内に記載されています : WDT インタフェース。

SSP でインタフェースを利用してプログラムを作成する方法については、を参照してください。SSP Architecture

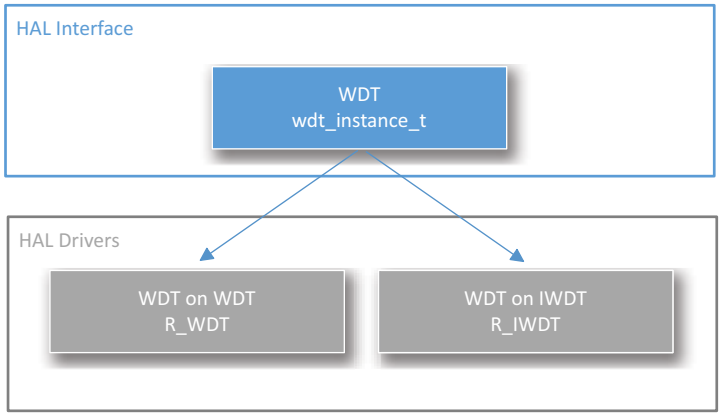

図 158: WDT ドライバ - ブロック図

WDT アプリケーション例については、以下を参照してください : チュートリアル : Using HAL Drivers - Programming the WDT

4.2.32.1 ウォッチドッグ ドライバの機能

このドライバはウォッチドッグ タイマ （WDT）インタフェースを設定します。WDT が許可されたリフレッシュ ウィンドウの外でアンダーフローまたはリフレッシュされた場合、次のいずれかのイベントが生じる可能性があります。

- デバイスのリセット
- NMI の生成

下の図は、WDT の動作の例を示しています。カウンタの有効なリフレッシュ周期でリフレッシュされた場合、タイマ カウントの値はリセットされます。カウントがアンダーフローするか、有効なリフレッシュ周期外にリフレッシュされた場合、WDT はデバイスをリセットするか、NMI を生成します。

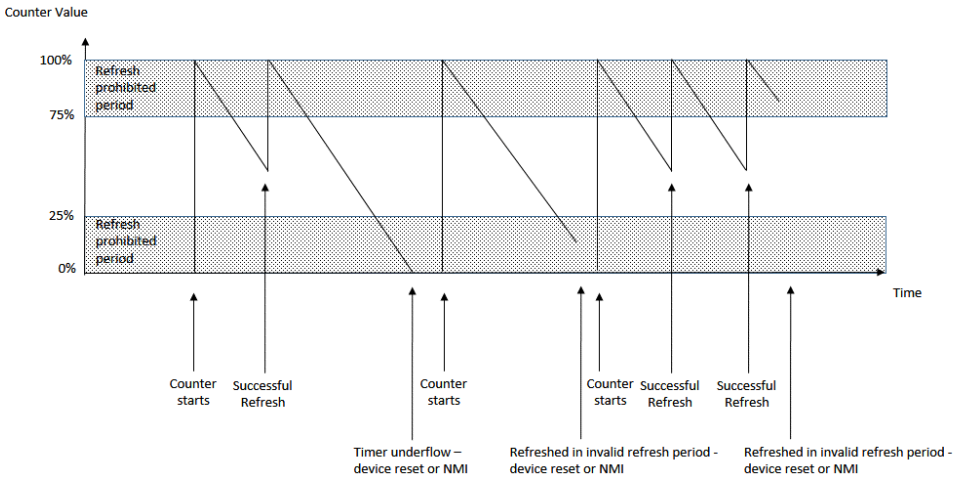

図 159: WDT - 操作

ウォッチドッグ ドライバ用モジュールの選択

ウォッチドッグ ドライバは、WDT または IWDT 周辺機器およびその HAL モジュールをサポートします。各周辺機器のユース ケースは、次のセクションで説明されています。

どのウォッチドッグ タイマを実装するかを決定する際には、次の点を考慮してください。

- WDT はアプリケーションから開始することができます。
- IWDT には、安全性を向上させる独自のクロック ソースがあります。

WDT の使用

WDT は、WDT レジスタからレジスタ スタート モードで設定できます。また WDT は、オプション関数選択レジスタ 0 （OFS0）に格納されているパラメータを使用して、リセット後に自動的にハードウェアで設定することもできます。

IWDT の使用

IWDT は、オプション関数選択レジスタ 0 （OFS0）に格納されているパラメータを使用して、リセット後に自動的にハードウェアで設定することができます。

4.2.32.2 e² studio ISDE によるウォッチドッグ ドライバを使用するアプリケーションの作成

ドライバは、e² studio ISDE の SSP に組み込まれています （e² studio ISDE ユーザーガイドを参照）。

e2 studio ISDE でプロジェクトの作成と設定を行い、ドライバを追加します。

1) プロジェクトを作成します （プロジェクトの作成を参照）。
2) プロジェクトを設定します （プロジェクトの設定を参照）。
3) ドライバを追加します （スレッドとドライバの追加を参照）。

WDT モジュールでウォッチドッグ ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
WDT Driver	Threads	[Driver] > [Monitoring] > [Watchdog Driver on r_wdt]
Interrupts	Threads	アプリケーション例については、WDT の割り込みの設定
Clock	Clocks	アプリケーション例については、WDT 用のクロックの設定

IWDT モジュールでウォッチドッグ ドライバを使用するアプリケーションでは、次のリソースが必要です。

リソース	ISDE タブ	選択
WDT Driver	Threads	[Driver] > [Monitoring] > [Watchdog Driver on r_iwdt]
Interrupts	Threads	アプリケーション例については、IWDT の割り込みの設定
Clock	Clocks	アプリケーション例については、IWDT 用のクロックの設定

WDT 用のクロックの設定

e2 studio ISDE で、[Clocks] タブを使用して WDT クロックを設定します （クロックの設定を参照）。

初期設定では、WDT クロックは PCLKB 周波数に基づいて動作します。PCLKB 周波数を設定するには、ランタイムで e2 studio ISDE のクロック コンフィギュレータまたは CGC インタフェースを使用します。

PCLKB が 60 MHz で動作している状態での最大タイムアウト周期は約 2.2 秒です。

IWDT 用のクロックの設定

IWDT の専用クロックは設定された周波数で動作しており、変更することはできません。

WDT および IWDT ピンの設定

WDT と IWDT のいずれも動作にピンは必要ありません。

WDT の割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから WDT 割り込みを設定します （割り込みの設定を参照）。

アンダーフローまたは無効なリフレッシュで NMI 割り込みを生成するよう WDT が構成されている場合、割り込みは BSP で有効になっている必要があります。

I : 割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、CWDT > CWDT NMIUNDF N 割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_WDT_UNDERFLOW に、選択した優先度が設定されます。

CWDT NMIUNDF N 割り込みが BSP で有効になっている場合、対応する ISR が定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

IWDT の割り込みの設定

e2 studio ISDE を使用して、[ICU] タブから WDT 割り込みを設定します （割り込みの設定を参照）。

アンダーフローまたは無効なリフレッシュで NMI 割り込みを生成するよう IWDT が構成されている場合、割り込みは BSP で有効になっている必要があります。

I : 割り込みを有効にするには、e2 studio ISDE のプロジェクト コンフィギュレータの [ICU] タブで、IWDT > IWDT NMIUNDF N 割り込みの優先度を設定します。これにより、ssp_cfg/bsp/bsp_irq_cfg.h の BSP_IRQ_CFG_IWDT_UNDERFLOW に、選択した優先度が設定されます。

IWDT NMIUNDF N 割り込みが BSP で有効になっている場合、対応する ISR が定義されます。ISR は、ユーザー コールバック関数が open で登録されていれば、それを呼び出します。

ウォッチドッグ ドライバ パラメータの設定

e2 studio ISDE を使用して、WDT モジュールのウォッチドッグ ドライバのパラメータを設定します。

RTOS を使用しないアプリケーションの場合 : HAL ドライバの追加と設定

ThreadX アプリケーションの場合 : ドライバのスレッドへの追加とドライバの設定。

表 : WDT の設定

参考資料

ISDE プロパティ	設定	設定値	説明
Start Mode	start_mode	wdt_start_mode_t	スタート モードをレジスタ スタートまたはオートスタートに設定します。
Start Watchdog After Configuration	autostart	true, false	WDT が初期化の際に開始するかどうかを制御します。
Timeout	timeout	wdt_timeout_t	WDT タイムアウト期間。
Clock division Ratio	clock_division	wdt_clock_division_t	WDT クロック分周器。
Window Start Position	window_start	wdt_window_start_t	許可されたリフレッシュ周期の開始位置。
Window End Position	window_end	wdt_window_end_t	許可されたリフレッシュ周期の終了位置。
Reset Control	reset_control	wdt_reset_control_t	WDT が MCU をリセットするか、NMI を生成するかを選択します。
Stop Control	stop_control	wdt_stop_control_t	WDT が低電力モードでカウントを停止するかどうかを選択します。
NMI Callback	p_context	User-defined	-
-	p_extend	User-defined	-

ISDE プロパティ	設定	設定値	説明
-	p_callback	User-defined	コールバック。ユーザーコールバック関数を open で登録できます。このコールバック関数が指定されている場合、IRQn がトリガされるたびに割り込みサービス ルーチン （ISR）から呼び出されます。 Attention : コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

IWDT 用のウォッチドッグ ドライバの設定

Note : オプション関数選択レジスタ 0 （OFS0）の設定で IWDT が構成および開始されると、IWDT は下記に示した wdt_cfg_t オプションのサブセットをサポートするようになります。

表 : WDT の設定

ISDE プロパティ	設定	設定値	説明
-	p_context	User-defined	-
-	p_extend	User-defined	-

ISDE プロパティ	設定	設定値	説明
NMI callback	p_callback	User-defined	コールバック。ユーザーコールバック関数を open で登録できます。このコールバック関数が指定されている場合、IRQn がトリガされるたびに割り込みサービス ルーチン （ISR）から呼び出されます。 Attention : コールバックは ISR から呼び出されるため、ブロッキング呼び出しを使用したり、長時間処理することはしないように注意してください。ISR の中で長時間費やすと、システムの応答性に影響を与えかねません。

表 : オプション関数選択レジスタ 0 （OFS0）設定

設定	設定値	説明
IWDTSTRT	0 or 1	0 - IWDT はリセット後に開始 ; 1 - IWDT は無効
IWDTTOPS	0, 1, 2 or 3	タイムアウト期間の選択
IWDTCKS	0, 2, 3, 4, 5 or 15	クロック分割比
IWDTRPES	0, 1, 2 or 3	ウィンドウのリフレッシュの終了位置
IWDTRPSS	0, 1, 2 or 3	ウィンドウのリフレッシュの開始位置
IWDTRSTIRQS	0 or 1	0 - アンダーフローまたは無効なリフレッシュで NMI が生成されます ; 1 - アンダーフローまたは無効なリフレッシュでデバイスがリセットされます
IWDTSTPCTL	0 or 1	0 - ロー モードでカウントは継続されます ; 1 - 低電力モードでカウントは停止します

OFS0 レジスタの内容の詳細については、MCU ユーザーズマニュアルを参照してください。

4.2.32.3 ウォッチドッグ ドライバ使用上の注意

WDT 期間の計算

WDT は PCLKB から動作します。WDT で最大のパラメータがあり、PCLKB が 60 MHz であると仮定した場合、最後のリフレッシュからデバイスのリセットまたは NMI の生成までの時間は、下記のように 2.2 秒をわずかに上回ります。

PLCKB = 60 MHz

クロック分割比 = PCLKB/8192

タイムアウト周期 = 16384 サイクル

WDT クロック周波数 = 60 MHz / 8192 = 7.324 kHz

サイクル時間 = 1 / 7.324 kHz = 136.53 マイクロ秒

タイムアウト = 136.53 マイクロ秒 x 16384 サイクル = 2.23 秒

IWDT 期間の計算

IWDT は IWDTCLK から動作します。WDT で最大のパラメータがあり、IWDTCLK 周波数が 15 kHz であると仮定した場合、最後のリフレッシュからデバイスのリセットまたは NMI の生成までの時間は、下記のように 35 秒をわずかに下回ります。

IWDTLCK = 15 kHz

クロック分割比 = IWTCLK/256

タイムアウト周期 = 2048 サイクル

IWDT クロック周波数 = 15 kHz / 256 = 58.59 Hz

サイクル時間 = 1 / 58.59 Hz = 17.07 マイクロ秒

タイムアウト = 17.07 マイクロ秒 x 2048 サイクル = 34.95 秒

WDT を使用した DMAC/DTC のトリガー

WDT カウンタがアンダーフローした場合、または有効なリフレッシュ周期外にリフレッシュが試みられた場合、DMAC または DTC 周辺機器を使用してデータの転送をトリガするには、NMI を生成するように WDT を設定し、activation_source を ELC_EVENT_WDT_UNDERFLOW に設定して DMAC/DTC 転送を構成します。詳細については、転送インタフェースを参照してください。

IWDT を使用した DMAC/DTC のトリガー

ウォッチドッグ カウンタがアンダーフローした場合、または有効なリフレッシュ周期外にリフレッシュが試みられた場合、DMAC または DTC 周辺機器を使用してデータの転送をトリガするには、activation_source を ELC_EVENT_IWDT_UNDERFLOW に設定して DMAC/DTC 転送を構成します。詳細については、転送インタフェースを参照してください。

WDT を使用した ELC イベントのトリガー

WDT は、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

IWDT を使用した ELC イベントのトリガー

IWDT は、elc_peripheral_t にリストされている他の周辺機器の起動をトリガできます。詳細については、ELC インタフェースを参照してください。

4.2.32.4 ウォッチドッグ ドライバによる WDT を使用するアプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const wdt_instance_t  g_wdt =

{

    .p_ctrl  = &g_wdt_ctrl,

    .p_cfg = &g_wdt_cfg,

    .p_api = &g_wdt_on_wdt

};
```

r_wdt 上の g_wdt ウォッチドッグ ドライバの [Properties] ウィンドウでの設定に応じ、ISDE は ISDE が生成したデータ ファイル内に wdt_cfg_t 構造体を作成します。

例 :

```
wdt_cfg_t g_wdt_cfg = {
    .start_mode = WDT_START_MODE_REGISTER,
   .start_wdt_after_cfg = true,
    .timeout = WDT_TIMEOUT_16384,
    .clock_division = WDT_CLOCK_DIVISION_8192,
    .window_start = WDT_WINDOW_START_100,
    .window_end = WDT_WINDOW_END_0,
    .reset_control = WDT_RESET_CONTROL_RESET,
    .stop_control = WDT_STOP_CONTROL_DISABLE,
    .p_callback = NULL,
    .p_extend = NULL,
    .p_context = NULL,
};
```

WDT を使用したウォッチドッグ アプリケーションを作成するには、次の手順を実行します。

1) ISDE で WDT を構成し、下記の WDT API 呼び出しを ISDE で生成されたファイル ssp/src/ssp_gen/hal_entry.c （RTOS 以外のアプリケーション用）に追加します。
2) WDT として実装されたウォッチドッグを使用して、ウォッチドッグ インスタンスをオープンします。WDT ドライバは、ウォッチドッグ インタフェースを通じて呼び出されます。

```
g_wdt_on_wdt.p_api->open(g_wdt.p_ctrl, g_wdt.p_cfg)
```

ウォッチドッグの設定手順後に、g_wdt.p_ctrl と g_wdt.p_cfg が自動生成されます。

3) WDT がカウントしている間に実行するコードを追加します。
4) 次を呼び出して、WDT を開始 （初期化の際に開始されていない場合）およびリフレッシュします :

```
g_wdt_on_wdt.p_api->refresh(g_wdt.p_ctrl)
```

4.2.32.5 ウォッチドッグ ドライバによる IWDT を使用するアプリケーションの作成

ISDE で生成されたファイルを使ってアプリケーション コードを追加します。

RTOS を使用しないアプリケーションの場合 : RTOS を使用しないアプリケーション

ThreadX アプリケーションの場合 : ThreadX アプリケーション

ISDE によって生成されたソース ファイルでプロジェクトを作成すると、次のインスタンス構造体が設定されます。

```
/* Instance structure to use this module. */

const wdt_instance_t  g_iwdt =

{

  .p_ctrl  = &g_wdt_ctrl,

  .p_cfg = &g_wdt_cfg,

  .p_api = &g_wdt_on_iwdt

};
```

IWDT 上の g_wdt ウォッチドッグ ドライバの [Properties] ウィンドウでの設定に応じ、ISDE は ISDE が生成したデータ ファイル内に wdt_cfg_t 構造体を作成します。

例 :

```
wdt_cfg_t  g_iwdt_cfg = {

    .p_context  = NULL,

    .p_extend = NULL,

    .p_callback = NULL,
};
```

IWDT を使用したウォッチドッグ アプリケーションを作成するには、次の手順を実行します。

1) ISDE の [BSP] タブで、BSP の OFS0 レジスタを変更します。
2) 前述のように、wdt_cfg_t の構造体を設定してモジュールを構成します。モジュールを構成すると、モジュール関連のヘッダーと構成ファイルが自動的に生成されます。
3) IWDT として実装されたウォッチドッグを使用して、ウォッチドッグ インスタンスをオープンします。IWDT ドライバは、ウォッチドッグ インタフェースを通じて呼び出されます。

```
g_wdt_on_iwdt.p_api->open(g_iwdt.p_ctrl, g_iwdt.p_cfg)
```

ウォッチドッグの設定手順後に、g_iwdt.p_ctrl と g_iwdt.p_cfg が自動生成されます。

4) IWDT がカウントしている間に実行するコードを追加します。
5) 次を呼び出して、IWDT をリフレッシュします。

```
g_wdt_on_iwdt.p_api->refresh(g_iwdt.p_ctrl)
```

4.2.32.6 ウォッチドッグ ドライバ使用上の制限事項

JLink デバッガを使用する場合、WDT カウンタはカウントしないため、デバイスのリセットや NMI の生成は行われません。

実行できるアクティブなタスクがない場合、ThreadX は MCU をスリープ モードにします。WDT または IWDT がローパワー モードでカウンタを停止するよう設定されている場合、ThreadX RTOS と使用する場合にはアプリケーションはタイマを再度開始する必要があります。

4.2.32.7 ウォッチドッグ ドライバでサポートされるデバイス

このドライバは、次のファミリでテストされています。

- S7G2

ウォッチドッグ タイマ ドライバは、API への変更なしに、次のファミリをサポートするように設計されています。

- S3A7
- S124

4.2.32.8 ウォッチドッグ ドライバ ファイル

プロジェクト設定中、ISDE により、次の表に記載されているファイルが synergy/ssp/ ディレクトリに抽出されます。

モジュール	ディレクトリ
HAL WDT Interface API	synergy/ssp/inc/driver/api/r_wdt_api.h
WDT Instance	synergy/ssp/inc/driver/instances/r_wdt.h
WDT Driver	synergy/ssp/src/driver/r_wdt
IWDT Instance	synergy/ssp/inc/driver/instances/r_iwdt.h
IWDT Driver	synergy/ssp/src/driver/r_iwdt

第 5 章 API リファレンス：フレームワーク インタフェース

フレームワーク インタフェースは、機能的フレームワーク レイヤー アプリケーションに対する共通 API を提供します。フレームワーク インタフェースは、1 つまたは複数のフレームワーク レイヤー ドライバによって実装できます。

- ADC 周期フレームワークインタフェース
- オーディオフレームワークインタフェース
- オーディオ再生フレームワークインタフェース
- ブロックメディアフレームワークインタフェース
- 通信フレームワークインタフェース
- コンソールフレームワークインタフェース
- GUIXTM インタフェース
- 外部 IRQ フレームワークインタフェース
- I2C フレームワーク
- JPEG デコードフレームワークインタフェース
- メッセージングフレームワークインタフェース
- パワー プロファイルフレームワークインタフェース
- SPI フレームワークインタフェース
- スレッド監視フレームワークインタフェース
- CTSU フレームワークインタフェース
- CTSU ボタンフレームワークインタフェース
- CTSU スライダフレームワークインタフェース
- タッチパネルフレームワークインタフェース

5.1 ADC 周期フレームワークインタフェース

RTOS 統合された ADC 周期フレームワーク インタフェース。

5.1.1 概要

ThreadX 対応の汎用的な周期 ADC サンプリング フレームワークです。このフレームワークでは、一定インターバルで ADC をサンプリングし、指定された数のサンプルをバッファに書き込んだ後、アプリケーションに通知します。ドライバによってハードウェア トリガが使用され、時間同期のサンプリングが行われます。初期構成とスキャン プロセスの開始後、フレームワークはハードウェア タイマを使用してワンショット モードで ADC スキャンをトリガします。各スキャンは 1 つ以上のチャネルで構成できます。スキャンが完了するたびに DTC が ADC 割り込みに割り込んで、スキャン結果をユーザー バッファに移動します。各スキャンはサンプリング反復として定義され、チャネルがシーケンスの場合 （例 : チャネル 1、2、3、4）は、各スキャンで生成されるサンプル数がチャネル数と等しくなります。チャネルがシーケンスでない場合（例 : チャネル 1、3、4、5）は、各スキャンで生成されるサンプルに、間の未使用チャネルのデータも含まれます。したがって 2 番目の例では、使用チャネルとして構成されているのは 4 チャネルですが、スキャンごとに 5 つのサンプルがユーザー バッファに格納されることになります。ユーザーは、その回数のサンプリングが完了した後に通知を受ける、サンプリング繰り返し回数の総数を指定します。指定された回数のサンプリング反復が完了し、各反復で生成されたデータがユーザー バッファに格納されると、コールバック関数経由でユーザーに通知され、バッファ内の有効なデータのインデックスと、指定された回数のサンプリング反復が完了したことを示すイベントが提供されます。ユーザーが停止 API 呼び出しを使用してスキャン プロセスを停止しない限り、タイマによってスキャンが継続的にトリガされ、フレームワークが循環バッファとして扱うユーザー バッファにデータが書き込まれます。このため、バッファ長は、すべての反復が完了した後に生成される合計サンプル数の少なくとも 2 倍の長さが必要です。2 番目の例では、1 回の反復で 5 つのサンプルが生成されるため、たとえばサンプル カウントを 3 に設定すると、コールバックが呼び出されるまでに 15 個のサンプルがバッファに格納されることになります。したがってこの例では、バッファ長を 30 以上に設定する必要があります。バッファの名前と長さは、フレームワーク構成構造体を介して指定します。

以下によって実装されます。ADC インタフェース

関連する SSP アーキテクチャのトピック :

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

ADC 周期フレームワーク インタフェースの説明 : ADC 周期フレームワーク

5.1.2 インタフェース API

sf_adc_periodic_api_t

関数名	説明
.open	ミューテックスを取得した後、HAL レイヤーでドライバを初期化します
.start	タイマ イベントを介した ADC のトリガが、ドライバにより有効化されます。
.stop	ハードウェア トリガ （タイマ）が、それ以上の ADC スキャンをトリガしないように停止します。
.close	チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.1.3 データ構造体

- sf_adc_periodic_callback_args_t
- sf_adc_periodic_ctrl_t
- sf_adc_periodic_cfg_t
- sf_adc_periodic_instance_t

5.1.4 列挙

- sf_adc_periodic_event_t

5.1.5 定義

- #define SF_ADC_PERIODIC_API_VERSION_MAJOR

  初期値 : (1)

  このファイルで定義された API のバージョン
- #define SF_ADC_PERIODIC_API_VERSION_MINOR

  初期値 : (1)

5.1.6 API データ

5.1.6.1 sf_adc_periodic_event_t

sf_adc_periodic_event_t

詳細説明

コールバック イベントのオプション。

列挙値

名前	説明
SF_ADC_PERIODIC_EVENT_NEW_DATA	新しいデータは、バッファに格納されます。

5.1.7 API 構造

5.1.7.1 sf_adc_periodic_callback_args_t

sf_adc_periodic_callback_args_t

詳細説明

ADC コールバック引数の定義

変数

- sf_adc_periodic_event_t event

  周期 ADC コールバック イベント。

- uint32_t buffer_index

  新しいデータが格納されているバッファへのインデックス。

- void const * p_context

  ユーザー データのプレースホルダー。

5.1.7.2 sf_adc_periodic_ctrl_t

sf_adc_periodic_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、SF_ADC_PERIODIC_Open の呼び出し時に実行されます

変数

- uint32_t open

  ドライバによって使用され、制御ブロックへのポインタが有効かどうかを確認します。

- TX_MUTEX mutex

  ローレベル ドライバ ハードウェア レジスタへのアクセスを保護するためのミューテックス。

- adc_api_t const * p_api_adc

  ローレベル ADC ドライバ関数ポインタへのポインタ。

- timer_api_t const * p_api_timer

  ローレベル タイマ ドライバ関数ポインタへのポインタ。

- transfer_api_t const * p_api_transfer

  ローレベル転送ドライバ関数ポインタへのポインタ。

- adc_ctrl_t ctrl_adc

  ローレベル ADC ドライバ制御ブロック。

- timer_ctrl_t ctrl_timer

  ローレベル タイマ ドライバ制御ブロック。

- transfer_ctrl_t ctrl_transfer

  ローレベル転送ドライバ制御ブロック。

- void const *volatile p_src_transfer

  ローレベル転送メソッドのソース ポインタ。

- adc_data_size_t * p_data_buffer

  サンプルを格納するバッファへのポインタ。

- uint32_t data_buffer_length

  サンプルを格納するデータ バッファの長さ。

- uint32_t data_buffer_index

  データが次に書き込まれるデータ バッファのインデックス。

- uint32_t sample_count

  アプリケーションを通知する前にバッファされるチャネルごとのサンプル。

- uint32_t dtc_transfer_length

  要求された数のサンプルに対する DTC 転送の合計長。

- void(* p_callback)(sf_adc_periodic_callback_args_t *p_args)

  コールバック関数。

- void const * p_context

  ユーザー データのプレースホルダー。

5.1.7.3 sf_adc_periodic_cfg_t

sf_adc_periodic_cfg_t

詳細説明

RTOS 統合された ADC ドライバの設定

変数

- adc_instance_t const *const p_lower_lvl_adc

  ADC インスタンスへのポインタ。

- timer_instance_t const *const p_lower_lvl_timer

  タイマ インスタンスへのポインタ。

- transfer_instance_t const *const p_lower_lvl_transfer

  転送インスタンスへのポインタ。

- adc_data_size_t * p_data_buffer

  サンプルを格納するバッファへのポインタ。

- uint32_t data_buffer_length

  サンプルを格納するデータ バッファの長さ。

- uint32_t sample_count

  アプリケーションを通知する前にバッファされるチャネルごとのサンプル。

- elc_event_t scan_trigger

  ADC スキャンを開始するハードウェア トリガ。

- void(* p_callback)(sf_adc_periodic_callback_args_t *p_args)

  コールバック関数。

- void const * p_context

  ユーザー データのプレースホルダー。

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

5.1.7.4 sf_adc_periodic_api_t

sf_adc_periodic_api_t

詳細説明

フレームワーク周期 ADC API 構造体。実装には、以下の API が使用されます。

5.1.7.5 open

ssp_err_t(* sf_adc_periodic_api_t::open)(sf_adc_periodic_ctrl_t *const p_ctrl, sf_adc_periodic_cfg_t const *const p_cfg)

詳細説明

ミューテックスを取得した後、HAL レイヤーでドライバを初期化します

表 1: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。要素はここで初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

bps

定義 : sf_adc_periodic_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_ADC_PERIODIC_Open の呼び出し時に実行されます

定義 :

定義 : sf_adc_periodic_cfg_t const *const p_cfg

RTOS 統合された ADC ドライバの設定

- sf_adc_periodic_cfg_t::adc_instance_t

  ADC インスタンスへのポインタ。
- sf_adc_periodic_cfg_t::timer_instance_t

  タイマ インスタンスへのポインタ。
- sf_adc_periodic_cfg_t::transfer_instance_t

  転送インスタンスへのポインタ。
- sf_adc_periodic_cfg_t::adc_data_size_t

  サンプルを格納するバッファへのポインタ。

- sf_adc_periodic_cfg_t::data_buffer_length

  サンプルを格納するデータ バッファの長さ。

- sf_adc_periodic_cfg_t::sample_count

  アプリケーションを通知する前にバッファされるチャネルごとのサンプル。

- sf_adc_periodic_cfg_t::elc_event_t

  ADC スキャンを開始するハードウェア トリガ。

- sf_adc_periodic_cfg_t::p_callback

  コールバック関数。

- sf_adc_periodic_cfg_t::p_context

  ユーザー データのプレースホルダー。

- sf_adc_periodic_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

5.1.7.6 start

ssp_err_t(* sf_adc_periodic_api_t::start)(sf_adc_periodic_ctrl_t *const p_ctrl)

詳細説明

タイマ イベントを介した ADC のトリガが、ドライバにより有効化されます。

> I a: この関数が呼び出された時間から、ハードウェア タイマカウントが期限切れになり、スキャンがトリガされる時間までは遅延があります。

表 2: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_ADC_PERIODIC_Open で設定された制御ブロックへのポインタ。

定義 : sf_adc_periodic_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_ADC_PERIODIC_Open の呼び出し時に実行されます

5.1.7.7 stop

ssp_err_t(* sf_adc_periodic_api_t::stop)(sf_adc_periodic_ctrl_t *const p_ctrl)

詳細説明

ハードウェア トリガ （タイマ）が、それ以上の ADC スキャンをトリガしないように停止します。

表 3: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_ADC_PERIODIC_Open で設定された制御ブロックへのポインタ。

定義 : sf_adc_periodic_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_ADC_PERIODIC_Open の呼び出し時に実行されます

5.1.7.8 close

ssp_err_t(* sf_adc_periodic_api_t::close)(sf_adc_periodic_ctrl_t *const p_ctrl)

詳細説明

チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。

表 4: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_ADC_PERIODIC_Open で設定された制御ブロックへのポインタ。

定義 : sf_adc_periodic_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_ADC_PERIODIC_Open の呼び出し時に実行されます

5.1.7.9 versionGet

ssp_err_t(* sf_adc_periodic_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 5: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

5.1.7.10 sf_adc_periodic_instance_t

sf_adc_periodic_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_adc_periodic_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_adc_periodic_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_adc_periodic_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.2 オーディオフレームワークインタフェース

RTOS 統合されたオーディオ フレームワーク インタフェース。

5.2.1 概要

このオーディオ インタフェースは、ThreadX 対応のオーディオ フレームワーク インタフェースです。このインタフェースは、オーディオフレームワーク により、タイマ ドライバ、転送ドライバ、および再生用のドライバ （DAC、PWM （未実装)、または I2S （未実装)）を使用して実装されます。

使用されるインタフェース：

- 転送インタフェース
- DAC インタフェース
- タイマインタフェース

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

オーディオ フレームワーク インタフェースの説明：オーディオ再生フレームワーク

5.2.2 インタフェース API

sf_audio_playback_api_t

関数名	説明
.open	オーディオ再生用のスレッドを作成し、使用する HAL レイヤー ドライバを設定してオーディオ フレームワークを設定します。この関数は、他のオーディオ関数の前に呼び出す必要があります。
.close	クローズ API は、内部ドライバ データのクリーンアップを処理します。
.start	オーディオを再生します。現在、16 ビット モノラル PCM バッファのみがサポートされています。

関数名	説明
.pause	オーディオ再生を一時停止します。DMA/DTC 転送をトリガーするペリフェラルが停止し、SF_AUDIO_PLAYBACK_Start() に対して進行中のオーディオ再生を一時停止するように通知するフラグがポストされます。
.stop	オーディオ再生を停止します。SF_AUDIO_PLAYBACK_Start() が再生を停止して復帰します。
.resume	オーディオ再生を再開します。SF_AUDIO_PLAYBACK_Start() に対し、DMA/DTC 転送をトリガーするペリフェラルを再開始するように通知するフラグがポストされます。
.volumeSet	ソフトウェア音量制御を設定します。ソフトウェア ボリューム制御は、ハードウェア上のすべてのストリームにグローバルに適用されます。
.versionGet	指定されたポインタにバージョン情報を格納します。

5.2.3 データ構造体

- sf_audio_playback_data_t
- sf_audio_playback_common_ctrl_t
- st_sf_audio_playback_ctrl
- sf_audio_playback_common_cfg_t
- sf_audio_playback_cfg_t
- sf_audio_playback_instance_t

5.2.4 列挙

- sf_audio_playback_status_t

5.2.5 定義

- #define SF_AUDIO_PLAYBACK_API_VERSION_MAJOR

  初期値 : (1)

  このファイルで定義された API のバージョン

- #define SF_AUDIO_PLAYBACK_API_VERSION_MINOR

  初期値 : (1)

- #define SF_AUDIO_PLAYBACK_STACK_SIZE

  初期値 : (2048)

  実証分析から得られたオーディオ スレッドの最適スタック サイズ。

- #define SF_AUDIO_PLAYBACK_MESSAGE_WORDS

  初期値 :((sizeof(sf_message_payload_audio_t) + 3) / 4)

  オーディオ再生のメッセージ サイズ （4 バイト ワード単位、端数は切り上げ)。

- #define SF_AUDIO_PLAYBACK_MAX_MESSAGES

  初期値 : (4)

  オーディオ再生メッセージ キュー内のメッセージの最大数。

- #define SF_AUDIO_PLAYBACK_PER_WORD

  初期値 : (4)

  ワードあたりのバイト数を定義するマクロ。

- #define SF_AUDIO_PLAYBACK_MAX_VOLUME

  初期値 : (255)

  最大音量を定義するマクロ。

- #define SF_AUDIO_PLAYBACK_MESSAGE_MEM_BYTES

  初期値 :(#define SF_TOUCH_PANEL_MESSAGE_WORDS * #define SF_TOUCH_BYTES_PER_WORD * #define SF_TOUCH_PANEL_MAX_MESSAGES)

  オーディオ再生キューのメモリ サイズ。

5.2.6 API データ

5.2.6.1 sf_audio_playback_status_t

sf_audio_playback_status_t

詳細説明

オーディオ再生のステータス。

列挙値

名前	説明
SF_AUDIO_PLAYBACK_STATUS_STOPPED	ストリームを使用できます。

名前	説明
SF_AUDIO_PLAYBACK_STATUS_PAUSED	ストリームが一時停止されています。
SF_AUDIO_PLAYBACK_STATUS_PLAYING	ストリームがデータを再生中です。
SF_AUDIO_PLAYBACK_STATUS_WAITING	ストリームがパケット間で、次のデータ メッセージを待っています。

5.2.7 API 構造

5.2.7.1 sf_audio_playback_data_t

sf_audio_playback_data_t

詳細説明

再生のためのオーディオ データ。

変数

- sf_message_header_t header

  メッセージング フレームワーク ペイロードの必須の共通メンバー。

- sf_audio_playback_data_type_t type

  データ型。非圧縮である必要があります。

- uint32_t size_bytes

  データ サイズ （バイト単位）。

- void const * p_data

  データへのポインタ。データの開始アドレスは、4 バイトでアラインする必要があります。

- UINT loop_timeout

  タイムアウト値を ThreadX ティック カウント（0x00000001 ～ 0xFFFFFFFE）で指定すると、ティック カウントが期限切れになるまでループ再生されます。

- bool stream_end

  これにより、他のスレッドがデータをポストできるようになります。この論理ビットストリームで送信するデータが他に存在しない場合は、true に設定します。他のパケットが準備されている場合は、false に設定します。

5.2.7.2 sf_audio_playback_common_ctrl_t

sf_audio_playback_common_ctrl_t

詳細説明

オーディオ共通制御ブロック。初期化しないでください。初期化は、SF_AUDIO_PLAYBACK_Open の最初の呼び出し時に実行されます。すべてのストリームで共有されます。

変数

- uint32_t open

  ドライバが初期化済みかどうかの判定に使用されます。

- void const * p_next_buffer

  次のバッファ （現在のバッファが完了したときに再生されるバッファ）へのポインタ。

- uint32_t next_length

  次のバッファ （現在のバッファが完了したときに再生されるバッファ）の長さ。

- sf_message_instance_t const * p_message

  メッセージ制御ブロックへのポインタ。

- TX_QUEUE * p_queue

  SF_MESSAGE_EVENT_CLASS_AUDIO イベントにサブスクライブされるキュー。

- sf_audio_playback_hw_instance_t const * p_lower_lvl_hw

  ハードウェア API が使用されています。

- sf_audio_playback_ctrl_t * p_stream[SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS]

  特定のデータをストリーミングします。

- TX_THREAD thread

  メインのオーディオ スレッド。

- TX_EVENT_FLAGS_GROUP flags

  オーディオ スレッドでの待機を解除するためのイベント フラグ。

- sf_audio_playback_data_type_t data_type

  ハードウェアで要求されるサンプル フォーマット。

- uint8_t volume

  範囲は 0 （消音）～ 255 （最大、開いた時点でのデフォルト）です。

- uint8_t buffer_index

  使用されるピンポン バッファ。

- uint8_t stack[SF_AUDIO_PLAYBACK_STACK_SIZE]

  オーディオ スレッド用のスタック。

- int16_t samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]

  転送時に変換されたデータの格納に使用するピンポン バッファ。

- bool playing

  オーディオ インスタンスの状態 （true の場合は、現在再生中）

5.2.7.3 st_sf_audio_playback_ctrl

st_sf_audio_playback_ctrl

詳細説明

オーディオ ストリーム制御ブロック。初期化しないでください。初期化は、SF_AUDIO_PLAYBACK_Open の呼び出し時に実行されます。

変数

- uint32_t open

  ドライバが初期化済みかどうかの判定に使用されます。

- TX_THREAD * p_owner

  このインデックスでストリームを開始したスレッドへのポインタ。複数のスレッドが同じストリームのデータをインターリーブすることを避けるために使用されます。

- void(* p_callback)(sf_message_callback_args_t *p_args)

  sf_audio_playback_api_t::start に渡されたバッファ再生の完了時に呼び出されるコールバック。

- uint8_t class_instance

  メッセージング フレームワークへのストリーム指定に使用されるクラス インスタンス。

- uint32_t samples_remaining

  このストリームの残りのデータ サンプルの内部状態。

- uint32_t index

  このストリームの現在のデータ インデックスの内部状態。

- uint32_t end

  ループ再生の完了を追跡するために使用されます。

- sf_audio_playback_data_t * p_data[2]

  キューから読み取られたオーディオ データ。

- sf_audio_playback_status_t status

  現在のストリームのステータス。

- sf_audio_playback_common_ctrl_t * p_common_ctrl

  このストリームが使用するハードウェア制御ブロックへのポインタ。

5.2.7.4 sf_audio_playback_common_cfg_t

sf_audio_playback_common_cfg_t

詳細説明

RTOS 統合されたオーディオ フレームワークの共通の構成。すべてのストリームで共有されます。

変数

- UINT priority

  オーディオ再生スレッドの優先順位。

- sf_audio_playback_hw_instance_t const * p_lower_lvl_hw

  ハードウェア インスタンス。

- sf_message_instance_t const * p_message

  オーディオ メッセージをポストするために使用するメッセージング フレームワーク インスタンスへのポインタ。

- TX_QUEUE * p_queue

  このオーディオ ストリームに対して指定されたメッセージング フレームワーク キューへのポインタ。SF_MESSAGE_EVENT_CLASS_AUDIO イベント クラスにサブスクライブする必要があります。

- void const * p_extend

  実装固有の拡張設定。

5.2.7.5 sf_audio_playback_cfg_t

sf_audio_playback_cfg_t

詳細説明

RTOS 統合されたオーディオ フレームワークのストリームごとの構成。

変数

- void(* p_callback)(sf_message_callback_args_t *p_args)

  sf_audio_playback_api_t::start に渡されたバッファ再生の完了時に呼び出されるコールバック。コールバックを行わない場合は、NULL に設定します。

- sf_audio_playback_common_ctrl_t * p_common_ctrl

  このストリームが使用するハードウェア制御ブロックへのポインタ。

- sf_audio_playback_common_cfg_t const * p_common_cfg

  同じハードウェアを使用するすべてのストリームによって共有される共通の設定へのポインタ。

- uint8_t class_instance

  メッセージング フレームワークへのストリーム指定に使用されるクラス インスタンス。

5.2.7.6 sf_audio_playback_api_t

sf_audio_playback_api_t

詳細説明

オーディオ再生 API 構造体。オーディオ再生の実装には、以下の API が使用されます。

5.2.7.7 open

ssp_err_t(* sf_audio_playback_api_t::open)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_cfg_t const *const p_cfg)

概要説明

オーディオ再生用のスレッドを作成し、使用する HAL レイヤー ドライバを設定してオーディオ フレームワークを設定します。この関数は、他のオーディオ関数の前に呼び出す必要があります。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_Open

表 6: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられたデバイス構造体へのポインタ。デバイス制御構造体は、この関数で初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 :

定義 : sf_audio_playback_cfg_t const *const p_cfg

RTOS 統合されたオーディオ フレームワークのストリームごとの構成。

- sf_audio_playback_cfg_t::p_callback

  sf_audio_playback_api_t::start に渡されたバッファ再生の完了時に呼び出されるコールバック。コールバックを行わない場合は、NULL に設定します。

- sf_audio_playback_cfg_t::sf_audio_playback_common_ctrl_t

  このストリームが使用するハードウェア制御ブロックへのポインタ。

- sf_audio_playback_cfg_t::sf_audio_playback_common_cfg_t

  同じハードウェアを使用するすべてのストリームによって共有される共通の設定へのポインタ。
- sf_audio_playback_cfg_t::class_instance

  メッセージング フレームワークへのストリーム指定に使用されるクラス インスタンス。

5.2.7.8 close

ssp_err_t(* sf_audio_playback_api_t::close)(sf_audio_playback_ctrl_t *const p_ctrl)

概要説明

クローズ API は、内部ドライバ データのクリーンアップを処理します。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_Close

表 7: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

5.2.7.9 start

ssp_err_t(* sf_audio_playback_api_t::start)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)

概要説明

オーディオを再生します。現在、16 ビット モノラル PCM バッファのみがサポートされています。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_Start

  l :sf_audio_playback_cfg_t::p_message と sf_audio_playback_cfg_t::p_queue に指定されたパラメータを使用して SF_MESSAGE_Open を呼び出し、メッセージング フレームワーク制御ブロックとキューを構成します。

表 8: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_data	複数のビットを書き換えることもできます。	データ、説明、タイムアウトの各値、および同期オプションへのポインタ。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウトは、オーディオ キューを登録するために必要な最大時間を表します。指定できる値は、TX_NO_WAIT （0x00000000）、TX_WAIT_FOREVER （0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

パラメータ p_data

定義 : sf_audio_playback_data_t*const p_data

再生のためのオーディオ データ。

- sf_audio_playback_data_t::header

  メッセージング フレームワーク ペイロードの必須の共通メンバー。

- sf_audio_playback_data_t::type

  データ型。非圧縮である必要があります。

- sf_audio_playback_data_t::loop_timeout

  データ サイズ （バイト単位）。

- sf_audio_playback_data_t::p_data

  データへのポインタ。データの開始アドレスは、4 バイトでアラインする必要があります。

- sf_audio_playback_data_t::stream_end

  タイムアウト値を ThreadX ティック カウント （0x00000001 ～ 0xFFFFFFFE）で指定すると、ティック カウントが期限切れになるまでループ再生されます。

- sf_audio_playback_data_t これにより、他のスレッドがデータをポストできるようになります。

  これにより、他のスレッドがデータをポストできるようになります。この論理ビットストリームで送信するデータが他に存在しない場合は、true に設定します。他のパケットが準備されている場合は、false に設定します。

パラメータ timeout

const

5.2.7.10 pause

ssp_err_t(* sf_audio_playback_api_t::pause)(sf_audio_playback_ctrl_t *const p_ctrl)

概要説明

オーディオ再生を一時停止します。DMA/DTC 転送をトリガする周辺機能が停止し、SF_AUDIO_PLAYBACK_Start に対して進行中のすべての再生を一時停止するように通知するフラグがポストされます。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_Pause

  I：この関数を使用する前に、SF_AUDIO_PLAYBACK_Start を呼び出してください。SF_AUDIO_PLAYBACK_Start の前に SF_AUDIO_PLAYBACK_Pause を呼び出しても効果はなく、エラー コードも返されません。

表 9: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

5.2.7.11 stop

ssp_err_t(* sf_audio_playback_api_t::stop)(sf_audio_playback_ctrl_t *const p_ctrl)

概要説明

オーディオ再生を停止します。SF_AUDIO_PLAYBACK_Start によって再生が停止されて復帰します。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_Stop

  I：この関数を使用する前に、SF_AUDIO_PLAYBACK_Start を呼び出してください。
  SF_AUDIO_PLAYBACK_Start の前に SF_AUDIO_PLAYBACK_Stop を呼び出しても効果はなく、エラー コードも返されません。

表 10:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

5.2.7.12 resume

ssp_err_t(* sf_audio_playback_api_t::resume)(sf_audio_playback_ctrl_t *const p_ctrl)

概要説明

オーディオ再生を再開します。SF_AUDIO_PLAYBACK_Start に対し、DMA/DTC 転送をトリガする周辺機能を再開始するように通知するフラグがポストされます。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_Resume

  I：この関数を使用する前に、SF_AUDIO_PLAYBACK_Pause を呼び出してください。
  SF_AUDIO_PLAYBACK_Pause の前に SF_AUDIO_PLAYBACK_Resume を呼び出しても効果はなく、エラー コードも返されません。

表 11: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

5.2.7.13 volumeSet

ssp_err_t(* sf_audio_playback_api_t::volumeSet)(sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const volume)

概要説明

ソフトウェア音量制御を設定します。ソフトウェア ボリューム制御は、ハードウェア上のすべてのストリームにグローバルに適用されます。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_VolumeSet

  I a: ソフトウェア音量制御を使用すると、解像度が低下します。また余分のメモリや処理帯域幅が必要になる可能性があります。

表 12: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
volume	複数のビットを書き換えることもできます。	要求された音量レベル。有効な範囲は、0 （消音、再生を停止）～ 255 （最大音量、開いたときのデフォルト値）。

パラメータ volume

uint8_t

5.2.7.14 versionGet

ssp_err_t(* sf_audio_playback_api_t::versionGet)(ssp_version_t *const p_version)

概要説明

指定されたポインタにバージョン情報を格納します。

詳細説明

また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_VersionGet

表 13: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	UART ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

パラメータ p_version

5.2.7.15 sf_audio_playback_instance_t

sf_audio_playback_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_audio_playback_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_audio_playback_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_audio_playback_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.3 オーディオ再生フレームワークインタフェース

RTOS 統合されたオーディオ再生フレームワークインタフェース。

5.3.1 概要

オーディオ データのバッファを再生するオーディオ再生ドライバ。

以下によって実装されます。DAC オーディオ再生フレームワーク

オーディオ フレームワーク インタフェースの説明 : オーディオ再生フレームワーク

5.3.2 インタフェース API

sf_audio_playback_hw_api_t

関数名	説明
.open	読み書きおよび制御のためにデバイス チャネルを開きます。
.start	オーディオ再生ハードウェアを開始します。
.stop	オーディオ再生ハードウェアを停止します。
.play	オーディオ バッファを再生します。
.dataTypeGet	指定されたポインタ p_data_type に、指定されたデータ型を格納します。
.close	オーディオ ドライバを閉じます。
.versionGet	ドライバのバージョンを返します。

5.3.3 データ構造体

- sf_audio_playback_data_type_t
- sf_audio_playback_hw_callback_args_t
- sf_audio_playback_hw_cfg_t
- sf_audio_playback_hw_ctrl_t
- sf_audio_playback_hw_instance_t

5.3.4 定義

- #define SF_AUDIO_PLAYBACK_HW_API_VERSION_MAJOR

  初期値 : (1)

- #define SF_AUDIO_PLAYBACK_HW_API_VERSION_MINOR

  初期値 : (1)

5.3.5 API 構造

5.3.5.1 sf_audio_playback_data_type_t

sf_audio_playback_data_type_t

詳細説明

オーディオのデータ型。

変数

- uint8_t scale_bits_max

  ビット単位の最大分解能。

- bool is_signed

  符号付きサンプルの場合は 1、符号なしサンプルの場合は 0 に設定します。

5.3.5.2 sf_audio_playback_hw_callback_args_t

sf_audio_playback_hw_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- void * p_context

  ユーザー データのプレースホルダー。sf_audio_playback_hw_cfg_t の sf_audio_playback_hw_api_t::open に設定されます。

5.3.5.3 sf_audio_playback_hw_cfg_t

sf_audio_playback_hw_cfg_t

詳細説明

オーディオ再生ドライバ構成。

変数

- void(* p_callback)(sf_audio_playback_hw_callback_args_t *p_args)

  再生完了時に呼び出されるコールバック。コールバックを行わない場合は、NULL に設定します。

- void * p_context

  ユーザー データのプレースホルダー。sf_audio_playback_hw_callback_args_t 内のユーザー コールバックに渡されます。

- void const * p_extend

  ハードウェアに依存する設定。

5.3.5.4 sf_audio_playback_hw_ctrl_t

sf_audio_playback_hw_ctrl_t

詳細説明

インタフェース制御ブロック

変数

- void(* p_callback)(sf_audio_playback_hw_callback_args_t *p_args)

  再生完了時に呼び出されるコールバック。

- void * p_context

  ユーザー データのプレースホルダー。sf_audio_playback_hw_callback_args_t 内のユーザー コールバックに渡されます。

- void * p_extend

  ハードウェアに依存する制御ブロック。

5.3.5.5 sf_audio_playback_hw_api_t

sf_audio_playback_hw_api_t

詳細説明

オーディオ再生 API 定義。

5.3.5.6 open

(* sf_audio_playback_hw_api_t::open)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)

詳細説明

読み書きおよび制御のためにデバイス チャネルを開きます。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_Open

表 14: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御ブロックに割り当てられたメモリへのポインタ。
p_cfg	複数のビットを書き換えることもできます。	ハードウェア構成へのポインタ。

定義 : sf_audio_playback_hw_ctrl_t

インタフェース制御ブロック

定義 :

定義 : sf_audio_playback_hw_cfg_t const *const p_cfg

オーディオ再生ドライバ構成。

- sf_audio_playback_hw_cfg_t::p_callback

  再生完了時に呼び出されるコールバック。コールバックを行わない場合は、NULL に設定します。

- sf_audio_playback_hw_cfg_t::p_context

  ユーザー データのプレースホルダー。sf_audio_playback_hw_callback_args_t 内のユーザー コールバックに渡されます。

- sf_audio_playback_hw_cfg_t::p_extend

  ハードウェアに依存する設定。

5.3.5.7 start

(* sf_audio_playback_hw_api_t::start)(sf_audio_playback_hw_ctrl_t *const p_ctrl)

詳細説明

オーディオ再生ハードウェアを開始します。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_Start

表 15: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open で初期化された制御ブロックへのポインタ。

定義 : sf_audio_playback_hw_ctrl_t

インタフェース制御ブロック

5.3.5.8 stop

(* sf_audio_playback_hw_api_t::stop)(sf_audio_playback_hw_ctrl_t *const p_ctrl)

詳細説明

オーディオ再生ハードウェアを停止します。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_Stop

表 16: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open で初期化された制御ブロックへのポインタ。

定義 : sf_audio_playback_hw_ctrl_t

インタフェース制御ブロック

5.3.5.9 play

(* sf_audio_playback_hw_api_t::play)(sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const *const p_buffer, uint32_t length)

詳細説明

オーディオ バッファを再生します。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_Play

表 17: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open で初期化された制御ブロックへのポインタ。
p_buffer	複数のビットを書き換えることもできます。	再生する PCM サンプルが格納されたバッファへのポインタ。データは、オーディオ再生ハードウェア用にスケーリングされている必要があります。

表 17: パラメータ (続き)

名前	方向	説明
length	複数のビットを書き換えることもできます。	p_buffer 単位のデータ長。

定義 : sf_audio_playback_hw_ctrl_t

インタフェース制御ブロック

パラメータ **p_buffer**

パラメータ **length**

uint32_t

5.3.5.10 dataTypeGet

(* sf_audio_playback_hw_api_t::dataTypeGet)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)

詳細説明

指定されたポインタ p_data_type に、指定されたデータ型を格納します。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet

表 18: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open で初期化された制御ブロックへのポインタ。
p_data_type	out	ハードウェアが要求するオーディオサンプル データ型へのポインタ。

定義 : sf_audio_playback_hw_ctrl_t

インタフェース制御ブロック

パラメータ **p_data_type**

定義 : sf_audio_playback_data_type_t*const p_data_type

オーディオのデータ型。

- sf_audio_playback_data_type_t::scale_bits_max

  ビット単位の最大分解能。

- sf_audio_playback_data_type_t::is_signed

  符号付きサンプルの場合は 1、符号なしサンプルの場合は 0 に設定します。

5.3.5.11 close

(* sf_audio_playback_hw_api_t::close)(sf_audio_playback_hw_ctrl_t *const p_ctrl)

詳細説明

オーディオ ドライバを閉じます。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_Close

表 19: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open で初期化された制御ブロックへのポインタ。

定義 : sf_audio_playback_hw_ctrl_t

インタフェース制御ブロック

5.3.5.12 versionGet

(* sf_audio_playback_hw_api_t::versionGet)(*const p_version)

詳細説明

ドライバのバージョンを返します。また電力消費を低減できます。

- SF_AUDIO_PLAYBACK_HW_DAC_VersionGet

表 20: パラメータ

名前	方向	説明
p_version	out	バージョン情報が格納される変数へのポインタ。

パラメータ p_version

5.3.5.13 sf_audio_playback_hw_instance_t

sf_audio_playback_hw_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_audio_playback_hw_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_audio_playback_hw_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_audio_playback_hw_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.4 ブロックメディアフレームワークインタフェース

Synergy ブロック メディア デバイスにアクセスするための RTOS 統合されたファイル システム インタフェース。

このインタフェースは FileX I/O からブロック メディア デバイスへの適応レイヤーを提供します。

5.4.1 概要

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

FileX インタフェースの説明も参照してください。FileX 適応フレームワーク

5.4.2 インタフェース API

sf_block_media_api_t

関数名	説明
.open	読み書きおよび制御のためにデバイス チャネルを開きます。
.read	メディア チャネルからデータを読み取ります。
.write	メディア チャネルにデータを書き込みます。
.ioctl	制御コマンドを送信して、メディア ポートからステータスを受信します。
.close	開いているメディア チャネルを閉じます。
.versionGet	ドライバのバージョンを返します。

5.4.3 データ構造体

- sf_block_media_cfg_t
- sf_block_media_ctrl_t
- sf_block_media_instance_t

5.4.4 定義

- #define BLOCK_MEDIA_API_VERSION_MAJOR

  初期値 : (1)
- #define BLOCK_MEDIA_API_VERSION_MINOR

  初期値 : (1)

5.4.5 API 構造

5.4.5.1 sf_block_media_cfg_t

sf_block_media_cfg_t

詳細説明

インタフェース設定

変数

- uint32_t block_size

  バイト単位のブロック サイズ。
- void const * p_extend

  インスタンスに依存する設定。

5.4.5.2 sf_block_media_ctrl_t

sf_block_media_ctrl_t

詳細説明

インタフェース制御ブロック

変数

- uint32_t block_size

  バイト単位のブロック サイズ。
- void const * p_extend

  インスタンスのチャネル情報。

5.4.5.3 sf_block_media_api_t

sf_block_media_api_t

詳細説明

Block Media の共有インタフェース定義

5.4.5.4 open

(* sf_block_media_api_t::open)(sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const *const p_cfg)

詳細説明

読み書きおよび制御のためにデバイス チャネルを開きます。また電力消費を低減できます。

- SF_Block_Media_SDMMC_Open

表 21: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	チャネルのメディア設定構造体へのポインタ。

定義 :

定義 : sf_block_media_cfg_t const *const p_cfg

インタフェース設定

- sf_block_media_cfg_t::block_size

  バイト単位のブロック サイズ。

- sf_block_media_cfg_t::p_extend

  インスタンスに依存する設定。

5.4.5.5 read

(* sf_block_media_api_t::read)(sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

詳細説明

メディア チャネルからデータを読み取ります。また電力消費を低減できます。

- SF_Block_Media_SDMMC_Read

表 22: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	チャネルのメディア設定構造体へのポインタ。
p_dest	複数のビットを書き換えることもできます。	データを読み出す宛先アドレス。
start_sector	複数のビットを書き換えることもできます。	読み取る開始セクターのアドレス。
sector_count	複数のビットを書き換えることもできます。	読み取るセクター数。

定義 :

パラメータ **p_dest**

uint8_t

パラメータ **start_sector**

uint32_t

パラメータ **sector_count**

uint32_t

5.4.5.6 write

(* sf_block_media_api_t::write)(sf_block_media_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count)

詳細説明

メディア チャネルにデータを書き込みます。また電力消費を低減できます。

- SF_Block_Media_SDMMC_Write

表 23: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	チャネルのメディア設定構造体へのポインタ。

表 23: パラメータ (続き)

名前	方向	説明
p_src	複数のビットを書き換えることもできます。	書き込むデータのソース アドレス。
start_sector	複数のビットを書き換えることもできます。	書き込み先の開始セクターのアドレス。
sector_count	複数のビットを書き換えることもできます。	書き込むセクター数。

定義 :

パラメータ p_src

uint8_t

パラメータ start_sector

uint32_t

パラメータ sector_count

uint32_t

5.4.5.7 ioctl

(* `sf_block_media_api_t`::ioctl)(`sf_block_media_ctrl_t` *p_ctrl, const command, void *p_data)

詳細説明

制御コマンドを送信して、メディア ポートからステータスを受信します。また電力消費を低減できます。

- SF_Block_Media_SDMMC_Control

表 24: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	チャネルのメディア設定構造体へのポインタ。
command	複数のビットを書き換えることもできます。	実行するコマンド。
p_data	入力 / 出力	データ入出力のための void ポインタ。

定義 :

パラメータ **command**

パラメータ **p_data**

const

5.4.5.8 close

(* sf_block_media_api_t::close)(sf_block_media_ctrl_t *p_ctrl)

詳細説明

開いているメディア チャネルを閉じます。また電力消費を低減できます。

- SF_Block_Media_SDMMC_Close

表 25: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	チャネルのメディア設定構造体へのポインタ。

定義 :

5.4.5.9 versionGet

(* sf_block_media_api_t::versionGet)(*const p_version)

詳細説明

ドライバのバージョンを返します。また電力消費を低減できます。

- SF_Block_Media_SDMMC_VersionGet

表 26: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	チャネルのメディア設定構造体へのポインタ。
p_version	out	バージョン情報を返す先のメモリ アドレス。

定義 :

パラメータ p_version

5.4.5.10 sf_block_media_instance_t

sf_block_media_instance_t

詳細説明

インタフェース インスタンス

変数

- sf_block_media_ctrl_t * p_ctrl

  デバイス ドライバ制御構造体へのブロック メディア ポインタ。

- sf_block_media_cfg_t const * p_cfg

  デバイス ドライバ設定構造体への Block Media ポインタ。

- sf_block_media_api_t const * p_api

  デバイス ドライバ API 構造体へのブロック メディア ポインタ。

5.5 通信フレームワークインタフェース

RTOS 統合された通信フレームワーク インタフェース。

以下によって実装されます。

- UART フレームワークインスタンス - UART の実装
- USB 通信フレームワーク - USBX™ CDC ACM デバイス実装
- Telnet 通信フレームワーク - NetX™ telnet サーバーの実装

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

フレームワーク通信インタフェースの説明も参照してください：通信フレームワーク

5.5.1 インタフェース API

sf_comms_api_t

関数名	説明
.open	通信ドライバを初期化します。
.close	通信ドライバをクリーンアップします。
.read	通信ドライバからデータを読み取ります。この呼び出しは、要求された数のバイトを読み取った後、またはドライバへのアクセスを待っている間にタイムアウトが発生した場合に復帰します。
.write	通信ドライバにデータを書き込みます。この呼び出しは、すべてのバイトが書き込まれた後、またはドライバへのアクセスを待っている間にタイムアウトが発生した場合に復帰します。
.lock	通信ドライバをロックします。通信ドライバへの排他的アクセスを予約します。
.unlock	通信ドライバをロック解除します。通信ドライバへの排他的アクセスを解放します。
.versionGet	ドライバ バージョンを指定された p_version に格納します。

5.5.2 データ構造体

- sf_comms_ctrl_t
- sf_comms_cfg_t
- sf_comms_instance_t

5.5.3 列挙

- sf_comms_lock_t

5.5.4 定義

- #define SF_COMMS_API_VERSION_MAJOR

  初期値 : (1)

  このファイルで定義された API のバージョン
- #define SF_COMMS_API_VERSION_MINOR

  初期値 : (1)

5.5.5 API データ

5.5.5.1 sf_comms_lock_t

sf_comms_lock_t

詳細説明

通信ロック

列挙値

名前	説明
SF_COMMS_LOCK_TX	送信をロックします。
SF_COMMS_LOCK_RX	受信をロックします。
SF_COMMS_LOCK_ALL	送信および受信をロックします。

5.5.6 API 構造

5.5.6.1 sf_comms_ctrl_t

sf_comms_ctrl_t

詳細説明

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます

変数

- void const * p_extend

  ローレベル通信制御構造体へのポインタ。

5.5.6.2 sf_comms_cfg_t

sf_comms_cfg_t

詳細説明

RTOS 統合された通信ドライバの構成

変数

- void const * p_extend

  ローレベル通信制御構造体へのポインタ。

5.5.6.3 sf_comms_api_t

sf_comms_api_t

詳細説明

フレームワーク通信 API 構造体。実装には、以下の API が使用されます。

5.5.6.4 open

ssp_err_t(* `sf_comms_api_t`::open)(`sf_comms_ctrl_t` *const p_ctrl, `sf_comms_cfg_t` const *const p_cfg)

詳細説明

通信ドライバを初期化します。

表 27: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた制御構造体へのポインタ。制御構造体は、この関数で初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : sf_comms_ctrl_t

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます

定義 :

定義 : sf_comms_cfg_t const *const p_cfg

RTOS 統合された通信ドライバの構成

- sf_comms_cfg_t::p_extend

  ローレベル通信制御構造体へのポインタ。

5.5.6.5 close

ssp_err_t(* sf_comms_api_t::close)(sf_comms_ctrl_t *const p_ctrl)

詳細説明

通信ドライバをクリーンアップします。

表 28: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	通信ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

定義 : sf_comms_ctrl_t

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます。

5.5.6.6 read

ssp_err_t(* sf_comms_api_t::read)(sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout)

詳細説明

通信ドライバからデータを読み取ります。この呼び出しは、要求された数のバイトを読み取った後、またはドライバへのアクセスを待っている間にタイムアウトが発生した場合に復帰します。

表 29:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	通信ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_dest	複数のビットを書き換えることもできます。	データを読み出す宛先アドレス
バイト	複数のビットを書き換えることもできます。	読み取りデータ長
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT（0x00000000）、TX_WAIT_FOREVER（0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

定義 : sf_comms_ctrl_t

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます

パラメータ p_dest

uint8_t

パラメータ bytes

uint32_t

パラメータ timeout

const

5.5.6.7 write

ssp_err_t(* sf_comms_api_t::write)(sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)

詳細説明

通信ドライバにデータを書き込みます。この呼び出しは、すべてのバイトが書き込まれた後、またはドライバへのアクセスを待っている間にタイムアウトが発生した場合に復帰します。

表 30:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	通信ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	データの読み取り元のソース アドレス
バイト	複数のビットを書き換えることもできます。	書き込みデータ長
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT（0x00000000）、TX_WAIT_FOREVER（0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

定義 : sf_comms_ctrl_t

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます

パラメータ p_src

uint8_t

パラメータ bytes

uint32_t

パラメータ timeout

const

5.5.6.8 lock

ssp_err_t(* sf_comms_api_t::lock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type, UINT timeout)

詳細説明

通信ドライバをロックします。通信ドライバへの排他的アクセスを予約します。

表 31: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	通信ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
lock_type	複数のビットを書き換えることもできます。	ロックのタイプ、送信チャネルまたは受信チャネル
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT（0x00000000）、TX_WAIT_FOREVER（0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

定義 : sf_comms_ctrl_t

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます

パラメータ lock_type

定義 : sf_comms_lock_tlock_type

通信ロック

パラメータ timeout

const

5.5.6.9 unlock

ssp_err_t(* sf_comms_api_t::unlock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type)

詳細説明

通信ドライバをロック解除します。通信ドライバへの排他的アクセスを解放します。

表 32: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	通信ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
lock_type	複数のビットを書き換えることもできます。	ロックのタイプ、送信チャネルまたは受信チャネル

定義 : sf_comms_ctrl_t

通信制御構造体。初期化しないでください。初期化は、open の呼び出し時に実行されます

パラメータ lock_type

定義 : sf_comms_lock_tlock_type

通信ロック

5.5.6.10 versionGet

ssp_err_t(* sf_comms_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

ドライバ バージョンを指定された p_version に格納します。

表 33: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	通信ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_version	複数のビットを書き換えることもできます。	格納されるメモリ バージョンへのポインタ。

パラメータ p_version

5.5.6.11 sf_comms_instance_t

sf_comms_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_comms_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_comms_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_comms_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.6 コンソールフレームワークインタフェース

RTOS に統合されたコンソール フレームワーク インタフェース。

5.6.1 概要

このモジュールは、ThreadX 対応のコンソール フレームワークです。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

コンソール インタフェースの説明も参照してくださいコンソールフレームワーク

5.6.2 インタフェース API

sf_console_api_t

関数名	説明
.open	この関数は、コンソールを設定します。この関数は、他のコンソール関数の前に呼び出す必要があります。
.close	クローズ API は、内部ドライバ データのクリーンアップを処理します。
.prompt	メニューのプロンプト文字列を出力して入力を待ちます。メニューに基づいて入力を解析し、コマンドが識別された場合は、コールバック関数を呼び出します。
.parse	メニューで入力文字列を探し、見つかった場合はコールバック関数を呼び出します。
.read	指定した宛先にデータをバイト単位で読み出し、入力をコンソールにエコーします。バックスペース キー、デリート キー、左右の矢印キーがサポートされています。改行を示す CR または CR+LF、CR+NULL が受信されるか、入力したデータが指定の入力バイト数を超えると、読み取りが完了します。バッファが SF_CONSOLE_MAX_INPUT_LENGTH をオーバーフローした場合は、エラー コードが返されます。

関数名	説明
.write	この書き込み API は、ミューテックス オブジェクトを取得し、UART HAL レイヤーで UART データ送信を処理します。
.argumentFind	入力文字列内でコマンド ライン引数を検索し、引数の直後の文字のインデックスおよび整数に変換された文字列番号を返します。
.versionGet	指定されたポインタにバージョン情報を格納します。

5.6.3 データ構造体

- sf_console_ctrl_t
- sf_console_callback_args_t
- sf_console_command_t
- st_sf_console_menu
- sf_console_cfg_t
- sf_console_instance_t

5.6.4 型定義

- sf_console_cb_args_t

5.6.5 定義

- #define SF_CONSOLE_API_VERSION_MAJOR

  初期値 : (1)

  このファイルで定義された API のバージョン
- #define SF_CONSOLE_API_VERSION_MINOR

  初期値 : (2)
- #define SF_CONSOLE_HELP_COMMAND

  初期値 :((uint8_t *) "?")

  メニュー内の各コマンドとヘルプを出力するコマンド

- #define SF_CONSOLE_MENU_PREVIOUS_COMMAND

  初期値 :((uint8_t *) "^")

  前のコマンド
- #define SF_CONSOLE_ROOT_MENU_COMMAND

  初期値 :((uint8_t *) "~")

  ルート メニュー コマンド
- #define SF_CONSOLE_MAX_INPUT_LENGTH

  初期値 :(128)

  入力長
- #define SF_CONSOLE_CALLBACK_NEXT_FUNCTION

  初期値 :(void(*)(sf_console_callback_args_t * p_args)) 0x70000000

  このコマンドから次のメニュー レイヤーにアクセスするためのマクロです。

5.6.6 API データ

5.6.6.1 sf_console_cb_args_t

typedef sf_console_callback_args_t sf_console_cb_args_t

詳細説明

非推奨の定義、代わりに sf_console_callback_args_t を使用してください。

5.6.7 API 構造

5.6.7.1 sf_console_ctrl_t

sf_console_ctrl_t

詳細説明

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

変数

- sf_console_menu_t const * p_current_menu

  現在のメニューがここに格納されます。
- sf_comms_instance_t const * p_comms

  通信ドライバ インスタンスへのポインタ。

- uint8_t new_line

  入力コマンドをトランスミッターにエコーするかどうかの指定。

- bool echo

  入力コマンドをトランスミッターにエコーするかどうかの指定。

- uint8_t input[SF_CONSOLE_MAX_INPUT_LENGTH]

  ユーザー入力の格納に使用される入力バッファ。

5.6.7.2 sf_console_callback_args_t

sf_console_callback_args_t

詳細説明

コンソール コールバック構造体

変数

- sf_console_ctrl_t * p_ctrl

  このコールバックを発生させたコマンドを受け取ったコンソールへのポインタ。

- uint8_t const * p_remaining_string

  解析コマンドの後に残る文字列。

- uint8_t const * context

  ユーザー定義のデータへのポインタ。

- uint32_t bytes

  入力文字列の残りのバイト数。

5.6.7.3 sf_console_command_t

sf_console_command_t

詳細説明

関連付けられたコールバックを使用してコンソールメニューを作成するためのコンソール コマンド構造体。

変数

- uint8_t * command

  コマンド文字列。

- uint8_t * help

  コマンドの説明。

- void(* callback)(sf_console_callback_args_t *p_args)
  コマンドを選択したときに呼び出されるコールバック。
- void const * context
  コールバックに渡されるユーザー定義のコンテキスト。

5.6.7.4 st_sf_console_menu

st_sf_console_menu

詳細説明

コンソール メニュー構造体。

変数

- struct const * menu_prev
  前のメニュー。
- uint8_t const * menu_name
  プロンプトとして使用されるメニュー名。
- uint32_t num_commands
  このメニューに含まれるコマンドの数。
- sf_console_command_t const * command_list
  長さが num_commands のコマンド配列へのポインタ。

5.6.7.5 sf_console_cfg_t

sf_console_cfg_t

詳細説明

RTOS 統合されたコンソール フレームワークの設定。

変数

- sf_comms_instance_t const * p_comms
  通信ドライバ インスタンスへのポインタ。
- sf_console_menu_t const * p_initial_menu
  Open を実行すると表示される最初のメニュー。
- bool echo
  入力コマンドをトランスミッターにエコーするかどうかの指定。

- bool autostart

  true の場合は、初期化の後に p_initial_menu を使用したプロンプトが発生します。

5.6.7.6 sf_console_api_t

sf_console_api_t

詳細説明

コンソール フレームワーク API 構造体。コンソール実装には、以下の API が使用されます。

5.6.7.7 open

(* sf_console_api_t::open)(sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const *const p_cfg)

概要説明

この関数は、コンソールを設定します。この関数は、他のコンソール関数の前に呼び出す必要があります。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_Open

表 34: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられたデバイス構造体へのポインタ。デバイス制御構造体は、この関数で初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : sf_console_ctrl_t

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

定義 :

定義 : sf_console_cfg_t const *const p_cfg

RTOS 統合されたコンソール フレームワークの設定。

- sf_console_cfg_t::sf_comms_instance_t

  通信ドライバ インスタンスへのポインタ。

- sf_console_cfg_t::p_initial_menu

  Open を実行すると表示される最初のメニュー。

- sf_console_cfg_t::echo

  入力コマンドをトランスミッターにエコーするかどうかの指定。

- sf_console_cfg_t::autostart

  true の場合は、初期化の後に p_initial_menu を使用したプロンプトが発生します。

5.6.7.8 close

(* sf_console_api_t::close)(sf_console_ctrl_t *const p_ctrl)

概要説明

クローズ API は、内部ドライバ データのクリーンアップを処理します。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_Close

表 35: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

定義 : sf_console_ctrl_t

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

5.6.7.9 prompt

(* sf_console_api_t::prompt)(sf_console_ctrl_t *const p_ctrl, const *const p_menu, UINT const timeout)

概要説明

メニューのプロンプト文字列を出力して入力を待ちます。メニューに基づいて入力を解析し、コマンドが識別された場合は、コールバック関数を呼び出します。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_Prompt

表 36:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_menu	複数のビットを書き換えることもできます。	このプロンプトに対する有効な入力コマンドのメニューへのポインタ
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT（0x00000000）、TX_WAIT_FOREVER（0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

定義 : `sf_console_ctrl_t`

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

パラメータ p_menu

パラメータ timeout

const

5.6.7.10 parse

(* `sf_console_api_t`::parse)(`sf_console_ctrl_t` *const p_ctrl, const *const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes)

概要説明

メニューで入力文字列を探し、見つかった場合はコールバック関数を呼び出します。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_Parse

表 37: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_cmd_list	複数のビットを書き換えることもできます。	このプロンプトに対する有効な入力コマンドのメニューへのポインタ
p_input	複数のビットを書き換えることもできます。	コマンド リストで検索する NULL 終端文字列へのポインタ
バイト	複数のビットを書き換えることもできます。	入力文字列の長さ。

定義 : sf_console_ctrl_t

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

パラメータ p_cmd_list

パラメータ p_input

uint8_t

パラメータ bytes

uint32_t

5.6.7.11 read

(* sf_console_api_t::read)(sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout)

概要説明

指定した宛先にデータをバイト単位で読み出し、入力をコンソールにエコーします。バックスペース キー、デリート キー、左右の矢印キーがサポートされています。改行を示す CR または CR+LF、CR+NULL が受信されるか、入力したデータが指定の入力バイト数を超えると、読み取りが完了します。バッファが SF_CONSOLE_MAX_INPUT_LENGTH をオーバーフローした場合は、エラー コードが返されます。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_Read

表 38: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_dest	複数のビットを書き換えることもできます。	データを読み出す宛先アドレス
バイト	複数のビットを書き換えることもできます。	読み取りデータ長
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT (0x00000000)、TX_WAIT_FOREVER (0xFFFFFFFF)、または ThreadX ティック カウントで表したタイムアウト値 (0x00000001 ～ 0xFFFFFFFE) です。

定義 : sf_console_ctrl_t

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

パラメータ p_dest

uint8_t

パラメータ bytes

uint32_t

パラメータ timeout

uint32_t

5.6.7.12 write

(* sf_console_api_t::write)(sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout)

概要説明

この書き込み API は、ミューテックス オブジェクトを取得し、UART HAL レイヤーで UART データ送信を処理します。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_Write

表 39:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	NULL 終端された文字列へのポインタ。長さは SF_CONSOLE_MAX_WRITE_LENGTH 未満でなければなりません。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT（0x00000000）、TX_WAIT_FOREVER（0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

定義 : sf_console_ctrl_t

コンソール ハンドル。初期化しないでください。初期化は、SF_CONSOLE_Open の呼び出し時に実行されます

パラメータ p_src

uint8_t

パラメータ timeout

uint32_t

5.6.7.13 argumentFind

(* sf_console_api_t::argumentFind)(uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data)

概要説明

入力文字列内でコマンド ライン引数を検索し、引数の直後の文字のインデックスおよび整数に変換された文字列番号を返します。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_ArgumentFind

表 40: パラメータ

名前	方向	説明
p_arg	複数のビットを書き換えることもできます。	検索する引数へのポインタ。
p_src	複数のビットを書き換えることもできます。	その中から引数を検索するソース文字列へのポインタ。
p_index	out	インデックスの格納場所へのポインタ。入力文字列内で引数が見つからなかった場合は、-1 に設定されます。インデックスが要求されていない場合は、NULL が渡されます。
p_data	out	引数の直後のデータを格納する場所へのポインタ。入力文字列内で引数が見つからなかった場合は、-1 に設定されます。データが要求されていない場合は、NULL が渡されます。

パラメータ p_arg

uint8_t

パラメータ p_src

パラメータ p_index

パラメータ p_data

5.6.7.14 versionGet

(* sf_console_api_t::versionGet)(*const p_version)

概要説明

指定されたポインタにバージョン情報を格納します。

詳細説明

また電力消費を低減できます。

- SF_CONSOLE_VersionGet

表 41: パラメータ

名前	方向	説明
p_version	out	ここに格納された、使用されているコードおよび API のバージョン。

パラメータ p_version

5.6.7.15 sf_console_instance_t

sf_console_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_console_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_console_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_console_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.7 GUIX™ インタフェース

Express Logic GUIX™ を Synergy グラフィックス ドライバに適応させるためのインタフェース定義。

5.7.1 概要

Synergy グラフィックス デバイス ドライバと GUIX™ を結合する SF_EL_GX フレームワーク モジュールのインタフェースです。このインタフェースは、GUIX™ を次のドライバに適応させることができます。

- LCD に画像を表示するための表示ドライバ （GLCDC など）
- 2DG エンジンによって画像をレンダリングするための Dave/2d ドライバ
- JPEG エンジンによって画像をレンダリングするための JPEG ドライバ
- ハードウェア アクセラレーションを使用しないソフトウェア画像レンダリング

以下によって実装されます。GUIXTM フレームワーク

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

GUIX™ インタフェースの説明 : GUIXTM Synergy ポートモジュール

5.7.2 インタフェース API

sf_el_gx_api_t

関数名	説明
.open	SSP GUIX™ 適応フレームワークを開きます。
.close	SSP GUIX™ 適応フレームワークを閉じます。
.versionGet	バージョンを取得します。
.setup	Express Logic GUIX™ ドライバ セットアップ エントリ。
.canvasInit	キャンバスの初期化。初期キャンバスのメモリアドレスを設定します。

5.7.3 データ構造体

- sf_el_gx_callback_args_t
- sf_el_gx_ctrl_t
- sf_el_gx_cfg_t
- sf_el_gx_instance_t

5.7.4 列挙

- sf_el_gx_state_t
- sf_el_gx_device_t
- sf_el_gx_event_t

5.7.5 定義

- #define SF_EL_GX_API_VERSION_MAJOR

  初期値 :(1)

  GUIX™ 統合ドライバ フレームワークの API バージョン
- #define SF_EL_GX_API_VERSION_MINOR

  初期値 :(1)

5.7.6 API データ

5.7.6.1 sf_el_gx_state_t

sf_el_gx_state_t

詳細説明

SSP GUIX™ 適応フレームワークの状態コード

列挙値

名前	説明
SF_EL_GX_CLOSED	
SF_EL_GX_OPENED	

名前	説明
SF_EL_GX_CONFIGURED	

5.7.6.2 sf_el_gx_device_t

sf_el_gx_device_t

詳細説明

GUIX のローレベル デバイス コード

列挙値

名前	説明
SF_EL_GX_DEVICE_NONE	ハードウェアではない。
SF_EL_GX_DEVICE_DISPLAY	表示デバイス。
SF_EL_GX_DEVICE_DRW	2D グラフィック エンジン
SF_EL_GX_DEVICE_JPEG	JPEG デコーダ。

5.7.6.3 sf_el_gx_event_t

sf_el_gx_event_t

詳細説明

表示イベント コード

列挙値

名前	説明
SF_EL_GX_EVENT_ERROR	ローレベル ドライバ エラーが発生しました。
SF_EL_GX_EVENT_DISPLAY_VSYNC	表示インタフェース VSYNC。
SF_EL_GX_EVENT_UNDERFLOW	表示インタフェース アンダーフロー。

5.7.7 API 構造

5.7.7.1 sf_el_gx_callback_args_t

sf_el_gx_callback_args_t

詳細説明

SSP GUIX 適応フレームワークのコールバック引数

変数

- sf_el_gx_device_t device
  デバイス コード。
- sf_el_gx_event_t event
  ローレベル ハードウェアのイベント コード。
- uint32_t error
  SF_EL_GX_EVENT_ERROR の場合のエラー コード。

5.7.7.2 sf_el_gx_ctrl_t

sf_el_gx_ctrl_t

詳細説明

SSP GUIX 適応フレームワークの制御ブロック

変数

- GX_DISPLAY * p_display
  GUIX 表示コンテキストへのポインタ。
- display_instance_t * p_display_instance
  表示インスタンスへのポインタ。
- display_runtime_cfg_t * p_display_runtime_cfg
  実行時表示構成へのポインタ。
- void * p_canvas
  キャンバスへのポインタ （予約済み）
- void * p_framebuffer_read
  フレーム バッファ （表示用）へのポインタ
- void * p_framebuffer_write
  フレーム バッファ （レンダリング用）へのポインタ

- void(* p_callback)(sf_el_gx_callback_args_t *p_args)

  コールバック関数へのポインタ。
- void * p_context

  コンテキストへのポインタ。
- TX_SEMAPHORE semaphore

  フレーム バッファのフリップ同期のセマフォ。
- bool rendering_enable

  レンダリングと表示の間の同期フラグ。
- bool display_list_flushed

  表示リストがフラッシュされたことを示すフラグ。
- sf_el_gx_state_t state

  このモジュールの状態。
- void * p_jpegbuffer

  JPEG 作業バッファへのポインタ。
- uint32_t jpegbuffer_size

  JPEG 作業バッファのサイズ。
- uint16_t rotation_angle

  画面のローテーション角度 （0/90/270）

5.7.7.3 sf_el_gx_cfg_t

sf_el_gx_cfg_t

詳細説明

SSP GUIX 適応フレームワークの構成構造体

変数

- display_instance_t * p_display_instance

  表示インスタンスへのポインタ。
- display_runtime_cfg_t * p_display_runtime_cfg

  実行時表示構成へのポインタ。
- void * p_canvas

  キャンバスへのポインタ （予約済み）

- void * p_framebuffer_a

  フレーム バッファ （A）へのポインタ

- void * p_framebuffer_b

  フレーム バッファ （B）へのポインタ

- void(* p_callback)(sf_el_gx_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void * p_context

  コンテキストへのポインタ。

- void * p_jpegbuffer

  JPEG 作業バッファへのポインタ。

- uint32_t jpegbuffer_size

  JPEG 作業バッファのサイズ。

- uint16_t rotation_angle

  画面のローテーション角度 （0/90/270）

5.7.7.4 sf_el_gx_api_t

sf_el_gx_api_t

詳細説明

SSP GUIX 適応フレームワークの共有インタフェース定義

5.7.7.5 open

ssp_err_t(* sf_el_gx_api_t::open)(sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const p_cfg)

詳細説明

SSP GUIX 適応フレームワークを開きます。また電力消費を低減できます。

- SF_EL_GX_Open

表 42: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_EL_GX 制御ブロック構造体へのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	SF_EL_GX 構成構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : sf_el_gx_ctrl_t

SSP GUIX 適応フレームワークの制御ブロック

定義 :

定義 : sf_el_gx_cfg_t const *const p_cfg

SSP GUIX 適応フレームワークの構成構造体

- sf_el_gx_cfg_t::display_instance_t
  表示インスタンスへのポインタ。
- sf_el_gx_cfg_t::display_runtime_cfg_t
  実行時表示構成へのポインタ。
- sf_el_gx_cfg_t::p_canvas
  キャンバスへのポインタ （予約済み）
- sf_el_gx_cfg_t::p_framebuffer_a
  フレーム バッファ （A）へのポインタ
- sf_el_gx_cfg_t::p_framebuffer_b
  フレーム バッファ （B）へのポインタ
- sf_el_gx_cfg_t::p_callback
  コールバック関数へのポインタ。
- sf_el_gx_cfg_t::p_context
  コンテキストへのポインタ。
- sf_el_gx_cfg_t::p_jpegbuffer
  JPEG 作業バッファへのポインタ。

- sf_el_gx_cfg_t::jpegbuffer_size
  JPEG 作業バッファのサイズ。
- sf_el_gx_cfg_t::rotation_angle
  画面のローテーション角度 （0/90/270）

5.7.7.6 close

ssp_err_t(* sf_el_gx_api_t::close)(sf_el_gx_ctrl_t *const p_ctrl)

詳細説明

SSP GUIX 適応フレームワークを閉じます。また電力消費を低減できます。

- SF_EL_GX_Close

表 43:　パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_EL_GX 制御ブロック構造体へのポインタ。

定義 : sf_el_gx_ctrl_t

SSP GUIX 適応フレームワークの制御ブロック

5.7.7.7 versionGet

ssp_err_t(* sf_el_gx_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

バージョンを取得します。また電力消費を低減できます。

- SF_EL_GX_VersionGet

表 44:　パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン情報を格納するメモリへのポインタ。

パラメータ **p_version**

5.7.7.8 setup

UINT(* sf_el_gx_api_t::setup)(GX_DISPLAY *p_display)

詳細説明

Express Logic GUIX ドライバ セットアップ エントリ。また電力消費を低減できます。

- SF_EL_GX_Setup

表 45:　パラメータ

名前	方向	説明
p_display	複数のビットを書き換えることもできます。	GUIX ディスプレイ ドライバ セットアップ関数へのポインタ。

Parameter p_display

const

5.7.7.9 canvasInit

ssp_err_t(* sf_el_gx_api_t::canvasInit)(sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT *p_window_root)

詳細説明

キャンバスの初期化。初期キャンバスのメモリアドレスを設定します。また電力消費を低減できます。

- SF_EL_GX_CanvasInit

表 46:　パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_EL_GX 制御ブロック構造体へのポインタ。
p_window_root	複数のビットを書き換えることもできます。	GUIX ルート ウィンドウ コンテキストへのポインタ。

定義 : sf_el_gx_ctrl_t

SSP GUIX 適応フレームワークの制御ブロック

Parameter p_window_root

const

5.7.7.10 sf_el_gx_instance_t

sf_el_gx_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_el_gx_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_el_gx_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_el_gx_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.8 外部 IRQ フレームワークインタフェース

RTOS 統合された外部 IRQ フレームワーク インタフェース。

5.8.1 概要

このモジュールは、スイッチなどのバイナリ信号を含む外部入力のための、ThreadX 対応の外部 IRQ フレームワーク インタフェースです。このインタフェースは、外部 IRQ フレームワーク によって実装されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

外部 IRQ フレームワーク インタフェースの説明 : 外部 IRQ フレームワーク

5.8.2 インタフェース API

sf_external_irq_api_t

関数名	説明
.open	ミューテックスを取得した後、HAL レイヤーでドライバの初期化を処理します。
.wait	外部割り込みの次回の期限切れを待って復帰します。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。
.close	チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。

5.8.3 データ構造体

- sf_external_irq_ctrl_t
- sf_external_irq_cfg_t
- sf_external_irq_instance_t

5.8.4 列挙

- sf_external_irq_event_t

5.8.5 定義

- #define SF_EXTERNAL_IRQ_API_VERSION_MAJOR

  初期値 :(1)
- #define SF_EXTERNAL_IRQ_API_VERSION_MINOR

  初期値 :(1)

5.8.6 API データ

5.8.6.1 sf_external_irq_event_t

sf_external_irq_event_t

詳細説明

外部割り込みの期限切れ時点での処理のオプション。

列挙値

名前	説明
SF_EXTERNAL_IRQ_EVENT_NONE	期限切れの際に、特に処理を行いません。データ転送に使用できます。
SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT	内部セマフォにポストします。SF_EXTERNAL_IRQ_Wait を使用している場合は、これを使用します。

5.8.7 API 構造

5.8.7.1 sf_external_irq_ctrl_t

sf_external_irq_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、SF_EXTERNAL_IRQ_Open の呼び出し時に実行されます

変数

- uint32_t open

  ドライバによって使用され、制御ブロックが有効かどうかを確認します。

- TX_MUTEX mutex

  ローレベル ドライバ ハードウェア レジスタへのアクセスを保護するためのミューテックス。

- TX_SEMAPHORE semaphore

  SF_EXTERNAL_IRQ_Wait に対して使用されるセマフォ。

- external_irq_api_t const * p_api

  ローレベル ドライバ関数ポインタへのポインタ。

- external_irq_ctrl_t ctrl

  ローレベル ドライバ制御ブロック。

- bool callback_used

  ドライバによって使用され、待機が使用可能かどうかを確認します。

5.8.7.2 sf_external_irq_cfg_t

sf_external_irq_cfg_t

詳細説明

RTOS 統合された外部ドライバの設定

変数

- external_irq_instance_t const * p_lower_lvl_irq

  下位レイヤーとの連携に必要なすべての情報

- sf_external_irq_event_t event

  外部 IRQ の発生時に行われる処理を選択します。

5.8.7.3 sf_external_irq_api_t

sf_external_irq_api_t

詳細説明

外部 IRQ フレームワーク API 構造体。外部 IRQ の実装には、以下の API が使用されます。

5.8.7.4 open

ssp_err_t(* sf_external_irq_api_t::open)(sf_external_irq_ctrl_t *const p_ctrl, sf_external_irq_cfg_t const *const p_cfg)

詳細説明

ミューテックスを取得した後、HAL レイヤーでドライバの初期化を処理します。また電力消費を低減できます。

- SF_EXTERNAL_IRQ_Open

表 47: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。デバイス制御構造体は、この関数で初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : sf_external_irq_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_EXTERNAL_IRQ_Open の呼び出し時に実行されます

定義 :

定義 : sf_external_irq_cfg_t const *const p_cfg

RTOS 統合された外部ドライバの設定

- sf_external_irq_cfg_t::external_irq_instance_t

  下位レイヤーとの連携に必要なすべての情報

- sf_external_irq_cfg_t::sf_external_irq_event_t

  外部 IRQ の発生時に行われる処理を選択します。

  列挙値 :

  – SF_EXTERNAL_IRQ_EVENT_NONE

  – SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT

5.8.7.5 wait

ssp_err_t(* sf_external_irq_api_t::wait)(sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)

詳細説明

外部割り込みの次回の期限切れを待って復帰します。

I：この関数を使用する前に、SF_EXTERNAL_IRQ_Open を呼び出して外部 IRQ を構成してください。SF_EXTERNAL_IRQ_Open で、sf_external_irq_cfg_t::sf_external_irq_event_t を SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT に設定します。

また電力消費を低減できます。

- SF_EXTERNAL_IRQ_Wait

表 48:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_EXTERNAL_IRQ_Open で設定されたハンドル。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。選択できる値は、TX_NO_WAIT、システム クロック カウントで表した値 （1 ～ 0xFFFFFFFF）、または TX_WAIT_FOREVER のいずれかです。

定義 : `sf_external_irq_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、SF_EXTERNAL_IRQ_Open の呼び出し時に実行されます

パラメータ timeout

const

5.8.7.6 versionGet

ssp_err_t(* `sf_external_irq_api_t`::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- SF_EXTERNAL_IRQ_VersionGet

表 49: パラメータ

名前	方向	説明
p_version	out	ここに格納された、使用されているコードおよび API のバージョン。

パラメータ p_version

5.8.7.7 close

ssp_err_t(* sf_external_irq_api_t::close)(sf_external_irq_ctrl_t *const p_ctrl)

詳細説明

チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。また電力消費を低減できます。

- SF_EXTERNAL_IRQ_Close

表 50: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。

定義 : sf_external_irq_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_EXTERNAL_IRQ_Open の呼び出し時に実行されます

5.8.7.8 sf_external_irq_instance_t

sf_external_irq_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_external_irq_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_external_irq_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。
- sf_external_irq_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.9 I²C フレームワーク

RTOS 統合された I²C フレームワーク インタフェース。

5.9.1 概要

ThreadX 対応の I²C インタフェースです。複数のハードウェア周辺機器より、I²C インタフェース I2C インタフェースを介した HAL レイヤーで実装できます。

HAL レイヤーへの接続は、SF_I2C_Open でドライバ構造体を渡すことで確立されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

SPI フレームワーク インタフェースの説明 : I2C フレームワーク

5.9.2 インタフェース API

sf_i2c_api_t

関数名	説明
.open	指定された I²C デバイスをバス上で開きます。
.read	I²C デバイスからデータを受信します。
.write	I²C デバイスにデータを送信します。
.reset	進行中の転送を中止し、I²C ペリフェラルを強制的にレディ状態にします。この関数は、デバイスで進行中の I²C 転送を安全に終了します。転送が中止した場合、中止イベントによるコールバックを介してユーザーに通知されます。コールバックがオプションのため、この関数は、この状況で特定のエラー コードも返します。
.close	制御ハンドルで指定された I²C デバイスを無効にします。バスにデバイスが接続されていない場合は、バスが使用する RTOS サービスを終了します。

関数名	説明
.lock	バスをデバイスにロックします。ロックすると、一定時間 （ロックとロック解除の間)、デバイスがバスを予約できるようになります。これにより、一部のニーズに対応して、デバイスが複数の読み書き操作を中断することなく完了できるようになります。
.unlock	バスを特定のデバイスからロック解除し、他のデバイスによって使用できるようにします。これにより、他のデバイスがそのバスを使用して、バス上で読み書きを実行できます。
.version	I^2C フレームワークのバージョンを取得します。

5.9.3 データ構造体

- sf_i2c_bus_t
- sf_i2c_cfg_t
- st_sf_i2c_ctrl
- sf_i2c_instance_t

5.9.4 列挙

- sf_i2c_dev_state_t

5.9.5 定義

- #define SF_I2C_API_VERSION_MAJOR

  初期値 :(1)

  ドライバ インタフェースを含めます。
- #define SF_I2C_API_VERSION_MINOR

  初期値 :(1)

5.9.6 API データ

5.9.6.1 sf_i2c_dev_state_t

sf_i2c_dev_state_t

詳細説明

SF I²C デバイスのステータス

列挙値

名前	説明
SF_I2C_DEV_STATE_CLOSED	I²C デバイスは閉じています。
SF_I2C_DEV_STATE_OPENED	I²C デバイスは開かれています。

5.9.7 API 構造

5.9.7.1 sf_i2c_bus_t

sf_i2c_bus_t

詳細説明

I²C バスを定義するデータ構造体。

変数

- uint8_t channel
  チャネル。
- TX_MUTEX * p_lock_mutex
  このチャネルのロック ミューテックス ハンドル。
- TX_EVENT_FLAGS_GROUP * p_sync_eventflag
  I²C データ転送用のイベント フラグ オブジェクトへのポインタ。
- sf_i2c_ctrl_t ** pp_curr_ctrl
  バスを使用している現在のデバイス。
- uint8_t * p_bus_name
  バスを識別するためのユーザー定義の名前。デバッグに役立ちます。
- i2c_api_master_t * p_lower_lvl_api
  フレームワークで使用される I²C HAL インタフェースへのポインタ。
- uint8_t device_count
  初期値は 0 です。

5.9.7.2 sf_i2c_cfg_t

sf_i2c_cfg_t

詳細説明

フレームワーク I2C ドライバの設定

変数

- sf_i2c_bus_t * p_bus

  デバイスが使用するバス。

- i2c_cfg_t const * p_lower_lvl_cfg

  I2C HAL 設定へのポインタ。

5.9.7.3 st_sf_i2c_ctrl

st_sf_i2c_ctrl

詳細説明

I2C デバイス コンテキスト。初期化しないでください。初期化は、SF_I2C_Open の呼び出し時に実行されます。

変数

- sf_i2c_bus_t * p_bus

  このデバイスを使用しているバス。設定構造体からのコピー。

- i2c_cfg_t lower_lvl_cfg

  I2C ペリフェラルの設定。バス設定のために使用します。

- i2c_ctrl_t lower_lvl_ctrl

  I2C ペリフェラル制御ブロック。

- sf_i2c_dev_state_t dev_state

  デバイスのステータス。

- bool locked

  バスをデバイスに対してロックおよびロック解除します。

- bool restarted

  デバイスがリスタートを発行したかどうかを示します。

5.9.7.4 sf_i2c_api_t

sf_i2c_api_t

詳細説明

I2C フレームワークの共有インタフェース定義

5.9.7.5 open

ssp_err_t(* sf_i2c_api_t::open)(sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)

概要説明

指定された I2C デバイスをバス上で開きます。

詳細説明

また電力消費を低減できます。

- SF_I2C_Open

表 51: パラメータ

名前	方向	説明
p_ctrl	out	デバイスの I2C フレームワーク ドライバ コンテキストの制御ハンドル(値はこの関数から返されます)。この値は、ユーザーがクリアする必要があります。
p_cfg	複数のビットを書き換えることもできます。	I2C の設定には、I2C バスとローレベル設定が含まれています

定義 :

定義 : sf_i2c_cfg_t const *const p_cfg

フレームワーク I2C ドライバの設定

- sf_i2c_cfg_t::sf_i2c_bus_t

  デバイスが使用するバス。

- sf_i2c_cfg_t::i2c_cfg_t

  I2C HAL 設定へのポインタ。

5.9.7.6 read

ssp_err_t(* sf_i2c_api_t::read)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout)

概要説明

I2C デバイスからデータを受信します。

詳細説明

また電力消費を低減できます。

- SF_I2C_Read

表 52: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	すでに開かれている I2C SF 制御構造体へのポインタ。
p_dest	複数のビットを書き換えることもできます。	読み取ったデータの格納場所へのポインタ。
バイト	複数のビットを書き換えることもできます。	読み取るバイト数。
restart	複数のビットを書き換えることもできます。	読み取り後にリスタート条件を発行するかどうかを指定します。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT (0x00000000)、TX_WAIT_FOREVER (0xFFFFFFFF)、または ThreadX ティック カウントで表したタイムアウト値 (0x00000001 ～ 0xFFFFFFFE) です。

パラメータ p_dest

uint8_t

パラメータ bytes

uint32_t

パラメータ restart

const

パラメータ timeout

uint32_t

5.9.7.7 write

ssp_err_t(* sf_i2c_api_t::write)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout)

概要説明

I²C デバイスにデータを送信します。

詳細説明

また電力消費を低減できます。

- SF_I2C_Write

表 53: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	すでに開かれている I²C 制御構造体へのポインタ。
p_src	複数のビットを書き換えることもできます。	書き込みデータを取得する場所へのポインタ。
バイト	複数のビットを書き換えることもできます。	書き込むバイト数。
restart	複数のビットを書き換えることもできます。	書き込み後にリスタート条件を発行するかどうかを指定します。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT (0x00000000)、TX_WAIT_FOREVER (0xFFFFFFFF)、または ThreadX ティック カウントで表したタイムアウト値 (0x00000001 ～ 0xFFFFFFFE) です。

パラメータ p_src

uint8_t

パラメータ bytes

uint32_t

パラメータ restart

const

パラメータ **timeout**

uint32_t

5.9.7.8 reset

ssp_err_t(* sf_i2c_api_t::reset)(sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)

概要説明

進行中の転送を中止し、I²C ペリフェラルを強制的にレディ状態にします。

詳細説明

また電力消費を低減できます。

- SF_I2C_Reset

この関数は、デバイスで進行中の I²C 転送を安全に終了します。転送が中止した場合、中止イベントによるコールバックを介してユーザーに通知されます。コールバックがオプションのため、この関数は、この状況で特定のエラー コードも返します。

表 54:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	I²C ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。指定できる値は、TX_NO_WAIT（0x00000000）、TX_WAIT_FOREVER（0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

パラメータ **timeout**

uint32_t

5.9.7.9 close

ssp_err_t(* sf_i2c_api_t::close)(sf_i2c_ctrl_t *const p_ctrl)

概要説明

制御ハンドルで指定された I²C デバイスを無効にします。バスにデバイスが接続されていない場合は、バスが使用する RTOS サービスを終了します。

詳細説明

また電力消費を低減できます。

- SF_I2C_Close

表 55: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの I²C フレームワーク ドライバ コンテキストの制御ハンドル

5.9.7.10 lock

ssp_err_t(* sf_i2c_api_t::lock)(sf_i2c_ctrl_t *const p_ctrl)

概要説明

バスをデバイスにロックします。ロックすると、一定時間 （ロックとロック解除の間)、デバイスがバスを予約できるようになります。これにより、一部のニーズに対応して、デバイスが複数の読み書き操作を中断することなく完了できるようになります。

詳細説明

また電力消費を低減できます。

- SF_I2C_Lock

表 56: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの I²C フレームワーク ドライバ コンテキストの制御ハンドル

5.9.7.11 unlock

ssp_err_t(* sf_i2c_api_t::unlock)(sf_i2c_ctrl_t *const p_ctrl)

概要説明

バスを特定のデバイスからロック解除し、他のデバイスによって使用できるようにします。これにより、他のデバイスがそのバスを使用して、バス上で読み書きを実行できます。

詳細説明

また電力消費を低減できます。

- SF_I2C_Unlock

表 57: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの I²C フレームワーク ドライバ コンテキストの制御ハンドル

5.9.7.12 version

ssp_err_t(* sf_i2c_api_t::version)(ssp_version_t *const p_version)

概要説明

I²C フレームワークのバージョンを取得します。

詳細説明

また電力消費を低減できます。

- SF_I2C_VersionGet

表 58: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの I²C フレームワーク制御ブロック用のハンドル

5.9.7.13 sf_i2c_instance_t

sf_i2c_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_i2c_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_i2c_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_i2c_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.10 JPEG デコードフレームワークインタフェース

RTOS 統合された JPEG デコード フレームワーク インタフェース。

5.10.1 概要

ランタイムでの JPEG デコードに使用するための ThreadX 対応の汎用 JPEG デコード フレームワークです。これは、ハードウェアまたはソフトウェアによって実装できます。Synergy に関して、このインタフェースはオンチップ JPEG デコード エンジンによって実装されます。HAL レイヤーへの接続は、SF_JPEG_Decode_Open 内でドライバ構造体を渡して確立されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

フレームワーク JPEG デコード インタフェースの説明 : JPEG デコード フレームワーク

5.10.2 インタフェース API

sf_jpeg_decode_api_t

関数名	説明
.open	ミューテックスを取得した後、HAL レイヤーでドライバの初期化を処理します。この関数は呼び出しスレッドに復帰する前に、ミューテックスを解放します。
.inputBufferSet	JPEG コーデック モジュールにデータを供給します。
.outputBufferSet	JPEG コーデック モジュールから処理済みのデータを読み取ります。
.linesDecodedGet	JPEG コーデックによってデコードされたライン数を取得します。
.horizontalStrideSet	水平ストライド値を設定します。
.imageSubsampleSet	これにより、デコードされた画像サイズをアプリケーションによってランタイムに縮小できます。これにより、デコードされた画像サイズをアプリケーションによって縮小できます。
.wait	現在の JPEG コーデック操作が完了するまで待機します

関数名	説明
.statusGet	JPEG コーデックのステータスを取得します
.imageSizeGet	この関数は、JPEG 画像をデコードする場合のみ機能します。詳細説明
.pixelFormatGet	画像のピクセル フォーマットを取得します。詳細説明
.close	JPEG コーデック デバイスを閉じます。未完了のコーデック操作が中断され、出力データが破棄されます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.10.3 データ構造体

- sf_jpeg_decode_ctrl_t
- sf_jpeg_decode_cfg_t
- sf_jpeg_decode_instance_t

5.10.4 定義

- #define SF_JPEG_DECODE_API_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義された API のバージョン
- #define SF_JPEG_DECODE_API_VERSION_MINOR

  初期値 :(1)

5.10.5 API 構造

5.10.5.1 sf_jpeg_decode_ctrl_t

sf_jpeg_decode_ctrl_t

詳細説明

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

変数

- uint32_t open

  ドライバが開いているかどうかを示します。

- uint32_t state

  ドライバによって使用され、制御ブロックへのポインタが有効かどうかを確認します。

- TX_MUTEX mutex

  ローレベル ドライバ ハードウェアへのアクセスを保護するためのミューテックス。

- TX_EVENT_FLAGS_GROUP events

  HAL ドライバがフレームワーク ドライバへの通知に使用するイベント フラグ。

- jpeg_decode_api_t const * p_api

  ローレベル ドライバ関数ポインタへのポインタ。

- jpeg_decode_ctrl_t ctrl

  ローレベル ドライバ制御ブロック。

5.10.5.2 sf_jpeg_decode_cfg_t

sf_jpeg_decode_cfg_t

詳細説明

RTOS 統合された JPEG ドライバの設定

変数

- jpeg_decode_instance_t const * p_lower_lvl_jpeg_decode

  このインタフェースを実装するドライバ構造体へのポインタ。r_jpeg_decode.c に設定済みのドライバ構造体が用意されており、r_jpeg_decode.h に extern されています。

5.10.5.3 sf_jpeg_decode_api_t

sf_jpeg_decode_api_t

詳細説明

JPEG デコード API 構造体。実装には、以下の API が使用されます。

5.10.5.4 open

(* sf_jpeg_decode_api_t::open)(sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg)

詳細説明

ミューテックスを取得した後、HAL レイヤーでドライバの初期化を処理します。この関数は呼び出しスレッドに復帰する前に、ミューテックスを解放します。

表 59: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。要素はここで初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : `sf_jpeg_decode_ctrl_t`

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

定義 :

定義 : `sf_jpeg_decode_cfg_t` const *const p_cfg

RTOS 統合された JPEG ドライバの設定

- `sf_jpeg_decode_cfg_t`::`jpeg_decode_instance_t`

  このインタフェースを実装するドライバ構造体へのポインタ。r_jpeg_decode.c に設定済みのドライバ構造体が用意されており、r_jpeg_decode.h に extern されています。

5.10.5.5 inputBufferSet

(* `sf_jpeg_decode_api_t`::inputBufferSet)(`sf_jpeg_decode_ctrl_t` *const p_ctrl, void *const p_buffer, uint32_t const buffer_size)

詳細説明

JPEG コーデック モジュールにデータを供給します。

表 60: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。

表 60: パラメータ (続き)

名前	方向	説明
p_buffer	複数のビットを書き換えることもできます。	JPEG コーデック モジュールで処理するデータが含まれているバッファ。バッファの開始アドレスは、8 バイトでアラインする必要があります
buffer_size	複数のビットを書き換えることもできます。	データ バッファのサイズ。8 バイトの倍数である必要があります

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **p_buffer**

const

パラメータ **buffer_size**

uint32_t

5.10.5.6 outputBufferSet

(* sf_jpeg_decode_api_t::outputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

詳細説明

JPEG コーデック モジュールから処理済みのデータを読み取ります。

表 61: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
p_buffer	複数のビットを書き換えることもできます。	JPEG コーデック モジュールからの出力を保持するユーザー定義のバッファ スペース。バッファの開始アドレスは、8 バイトでアラインする必要があります
buffer_size	複数のビットを書き換えることもできます。	出力データ バッファのサイズ

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ p_buffer

const

パラメータ buffer_size

uint32_t

5.10.5.7 linesDecodedGet

(* sf_jpeg_decode_api_t::linesDecodedGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)

詳細説明

JPEG コーデックによってデコードされたライン数を取得します。

表 62: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
p_lines	out	出力バッファにデコードされたライン数。

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ p_lines

uint32_t

5.10.5.8 horizontalStrideSet

(* sf_jpeg_decode_api_t::horizontalStrideSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

詳細説明

水平ストライド値を設定します。

表 63: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
horizontal_stride	out	水平ストライド値をピクセル単位で設定します

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **horizontal_stride**

uint32_t

5.10.5.9 imageSubsampleSet

(* sf_jpeg_decode_api_t::imageSubsampleSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, horizontal_subsample, vertical_subsample)

詳細説明

これにより、デコードされた画像サイズをアプリケーションによってランタイムに縮小できます。これにより、デコードされた画像サイズをアプリケーションによって縮小できます。

表 64: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
horizontal_subsample	複数のビットを書き換えることもできます。	水平サブサンプル値を設定します
vertical_subsample	複数のビットを書き換えることもできます。	垂直サブサンプル値を設定します

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **horizontal_subsample**

パラメータ **vertical_subsample**

5.10.5.10 wait

(* sf_jpeg_decode_api_t::wait)(sf_jpeg_decode_ctrl_t *const p_ctrl, *const p_status, uint32_t timeout)

詳細説明

現在の JPEG コーデック操作が完了するまで待機します

表 65:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
p_status	out	JPEG コーデック モジュールの現在のステータス
timeout	out	待機する時間の長さ （ThreadX ティック単位）

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **p_status**

パラメータ **timeout**

uint32_t

5.10.5.11 statusGet

(* sf_jpeg_decode_api_t::statusGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, *const p_status)

詳細説明

JPEG コーデックのステータスを取得します

表 66: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
p_status	out	JPEG コーデック モジュールの現在のステータス

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **p_status**

5.10.5.12 imageSizeGet

(* sf_jpeg_decode_api_t::imageSizeGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

詳細説明

この関数は、JPEG 画像をデコードする場合のみ機能します。詳細説明

表 67: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
p_horizontal_size	out	画像の幅 （ピクセル単位）
p_vertical_size	out	画像の高さ （ピクセル単位）

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **p_horizontal_size**

uint16_t

パラメータ **p_vertical_size**

uint16_t

5.10.5.13 pixelFormatGet

(* sf_jpeg_decode_api_t::pixelFormatGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, *const p_color_space)

詳細説明

画像のピクセル フォーマットを取得します。詳細説明

表 68: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で初期化された制御ブロックへのポインタ。
p_color_space	out	画像の色空間

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

パラメータ **p_color_space**

5.10.5.14 close

(* sf_jpeg_decode_api_t::close)(sf_jpeg_decode_ctrl_t *const p_ctrl)

詳細説明

JPEG コーデック デバイスを閉じます。未完了のコーデック操作が中断され、出力データが破棄されます。

表 69: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_JPEG_Decode_Open で設定された制御ブロックへのポインタ。

定義 : sf_jpeg_decode_ctrl_t

JPEG フレームワーク制御ブロック。初期化しないでください。初期化は、SF_JPEG_Decode_Open の呼び出し時に実行されます。

5.10.5.15 versionGet

(* sf_jpeg_decode_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 70: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

5.10.5.16 sf_jpeg_decode_instance_t

sf_jpeg_decode_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_jpeg_decode_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_jpeg_decode_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_jpeg_decode_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.11 メッセージングフレームワークインタフェース

RTOS に統合されたメッセージング フレームワーク インタフェース。

5.11.1 概要

このモジュールは、ThreadX 対応のメッセージング フレームワークです。このインタフェースは、メッセージングフレームワーク によって実装されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

メッセージング インタフェースの説明：メッセージングフレームワーク

5.11.2 インタフェース API

sf_message_api_t

関数名	説明
.open	メッセージ フレームワークを初期化します。メッセージング フレームワーク制御ブロックを開始し、構成パラメータに対応する作業メモリを構成します。
.close	メッセージ フレームワークを終了処理します。
.bufferAcquire	ブロックから渡されるメッセージ用に、バッファを取得します。
.bufferRelease	SF_MESSAGE_BufferAcquire で取得されたバッファを解放します。
.post	サブスクライバーに対してメッセージをポストします。
.pend	メッセージを保留します。
.versionGet	メッセージング フレームワークのバージョンを取得します。

5.11.3 データ構造体

- sf_message_header_t
- sf_message_instance_range_t
- sf_message_subscriber_t
- sf_message_subscriber_list_t
- sf_message_callback_args_t
- sf_message_post_err_t
- sf_message_buffer_ctrl_t
- sf_message_buffer_ctrl_t
- sf_message_cfg_t
- sf_message_acquire_cfg_t
- sf_message_post_cfg_t
- sf_message_instance_t

5.11.4 列挙

- sf_message_state_t
- sf_message_callback_event_t
- sf_message_priority_t
- sf_message_release_option_t

5.11.5 定義

- #define SF_MESSAGE_API_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義された API のバージョン
- #define SF_MESSAGE_API_VERSION_MINOR

  初期値 :(1)

5.11.6 API データ

5.11.6.1 sf_message_state_t

sf_message_state_t

詳細説明

メッセージング フレームワークの状態

列挙値

名前	説明
SF_MESSAGE_STATE_CLOSED	メッセージング フレームワークが閉じています。
SF_MESSAGE_STATE_OPENED	メッセージング フレームワークが開いています。

5.11.6.2 sf_message_callback_event_t

sf_message_callback_event_t

詳細説明

メッセージング コールバック レスポンス

列挙値

名前	説明
SF_MESSAGE_CALLBACK_EVENT_ACK	ACK 応答。
SF_MESSAGE_CALLBACK_EVENT_NAK	NAK 応答。

5.11.6.3 sf_message_priority_t

sf_message_priority_t

詳細説明

メッセージング フレームワークの状態

列挙値

名前	説明
SF_MESSAGE_PRIORITY_NORMAL	送信するメッセージに通常の優先順位を付けます。
SF_MESSAGE_PRIORITY_HIGH	送信するメッセージに 1 つ前のメッセージより高い優先順位を付けます。

5.11.6.4 sf_message_release_option_t

sf_message_release_option_t

詳細説明

メッセージング オプション

列挙値

名前	説明
SF_MESSAGE_RELEASE_OPTION_NONE	オプションなし。
SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE	バッファ強制解放オプション。
SF_MESSAGE_RELEASE_OPTION_ACK	ACK 応答 （ACK と NAK の両方が同時に設定された場合は NAK になることに注意してください）。
SF_MESSAGE_RELEASE_OPTION_NAK	NAK 応答。

5.11.7 API 構造

5.11.7.1 sf_message_header_t

sf_message_header_t

詳細説明

メッセージ ヘッダー定義

変数

- uint32_t event

- uint32_t class_instance

  < イベント クラス コード

  イベント クラス インスタンス番号

- uint32_t code

  イベント コード。

- struct{} event_b

  このメンバーの定義については、ソース コードを参照してください。

- union{} union{}

  このメンバーの定義については、ソース コードを参照してください。

5.11.7.2 sf_message_instance_range_t

sf_message_instance_range_t

詳細説明

サブスクライバー リストの定義

変数

- uint8_t start

  イベント クラスのインスタンス範囲の始点。

- uint8_t end

  イベント クラスのインスタンス範囲の終点。

5.11.7.3 sf_message_subscriber_t

sf_message_subscriber_t

詳細説明

メッセージ サブスクライバー

変数

- TX_QUEUE * p_queue

  サブスクライバー スレッド用のメッセージ キューへのポインタ。

- sf_message_instance_range_t instance_range

  メッセージを受信するイベント クラス インスタンスの範囲。

5.11.7.4 sf_message_subscriber_list_t

sf_message_subscriber_list_t

詳細説明

メッセージ サブスクライバー リスト

変数

- sf_message_event_class_t event_class
  イベント クラス コード。
- uint16_t number_of_nodes
  サブスクライバー グループ内のノード数。
- sf_message_subscriber_t ** pp_subscriber_group
  イベント クラスのサブスクライバー グループ。

5.11.7.5 sf_message_callback_args_t

sf_message_callback_args_t

詳細説明

メッセージ フレームワークのコールバック パラメータ

変数

- sf_message_callback_event_t event
  イベント コード。
- void const * p_context
  コールバック時にユーザーに提供されるコンテキスト。

5.11.7.6 sf_message_post_err_t

sf_message_post_err_t

詳細説明

エラー情報構造体をポストします

変数

- TX_QUEUE * p_queue
  キュー。

5.11.7.7 sf_message_buffer_ctrl_t

sf_message_buffer_ctrl_t

詳細説明

バッファ制御ブロック構造体

変数

- void(* p_callback)(sf_message_callback_args_t *)
  オプションのユーザー コールバック関数。
- void const * p_context
  コールバック時にユーザーに提供されるコンテキスト。
- sf_message_buffer_ctrl_t::st_buffer_ctrl_flagstruct flag_b

5.11.7.8 sf_message_buffer_ctrl_t::st_buffer_ctrl_flag

sf_message_buffer_ctrl_t::st_buffer_ctrl_flag

概要説明

< フラグ

詳細説明

変数

- uint32_t semaphore
  バッファの解放を防止するカウンティング セマフォ。
- uint32_t buffer_keep
  バッファ キープ要求。
- uint32_t nak_response
  NAK （サブスクライバーが複数存在する場合の論理和 （OR）ロジック）
- uint32_t reserved
  予備ビット。
- uint32_t in_use
  使用中のバッファ。

5.11.7.9 sf_message_buffer_ctrl_t

sf_message_ctrl_t

詳細説明

メッセージング フレームワーク制御ブロック構造体

変数

- TX_BLOCK_POOL block_pool

  メモリ ブロック プール制御へのポインタ。

- sf_message_subscriber_list_t ** pp_subscriber_lists

  サブスクライバー リストへのポインタ配列。

- uint32_t buffer_size

  メッセージ バッファのバイト数。

- uint32_t number_of_buffers

  割り当てられたバッファの数。

- uint16_t number_of_subscriber_groups

  サブスクライバー グループの数。

- sf_message_state_t state

  メッセージ フレームワークのステータス。

5.11.7.10 sf_message_cfg_t

sf_message_cfg_t

詳細説明

メッセージング フレームワーク設定構造体定義

変数

- void * p_work_memory_start

  メモリ領域の開始アドレス。

- uint32_t work_memory_size_bytes

  作業メモリ領域のサイズ （バイト単位)。

- uint32_t buffer_size

  メッセージ ブロックのバイト数。

- sf_message_subscriber_list_t ** pp_subscriber_lists

  サブスクライバー リストへのポインタ配列。

- uint8_t * p_block_pool_name

  ブロック プール名へのポインタ。

5.11.7.11 sf_message_acquire_cfg_t

sf_message_acquire_cfg_t

詳細説明

メッセージング フレームワーク ポスト API 関数設定構造体定義

変数

- bool buffer_keep

  バッファ キープ オプション。

5.11.7.12 sf_message_post_cfg_t

sf_message_post_cfg_t

詳細説明

メッセージング フレームワーク取得 API 関数設定構造体定義

変数

- sf_message_priority_t priority

  メッセージの優先度。

- void(* p_callback)(sf_message_callback_args_t *)

  ユーザー コールバック関数。

- void const * p_context

  コールバック時にユーザーに提供されるコンテキスト。

5.11.7.13 sf_message_api_t

sf_message_api_t

詳細説明

メッセージング フレームワーク API 構造体。実装には、以下の API が使用されます。

5.11.7.14 open

ssp_err_t(* sf_message_api_t::open)(sf_message_buffer_ctrl_t *const p_ctrl, sf_message_cfg_t const *const p_cfg)

詳細説明

メッセージ フレームワークを初期化します。メッセージング フレームワーク制御ブロックを開始し、構成パラメータに対応する作業メモリを構成します。また電力消費を低減できます。

- SF_MESSAGE_Open

表 71: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	メッセージング制御ブロックへのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : sf_message_buffer_ctrl_t

メッセージング フレームワーク制御ブロック構造体

定義 :

定義 : sf_message_cfg_t const *const p_cfg

メッセージング フレームワーク設定構造体定義

- sf_message_cfg_t::p_work_memory_start

  メモリ領域の開始アドレス。

- sf_message_cfg_t::work_memory_size_bytes

  作業メモリ領域のサイズ （バイト単位)。

- sf_message_cfg_t::buffer_size

  メッセージ ブロックのバイト数。

- sf_message_cfg_t::sf_message_subscriber_list_t

  サブスクライバー リストへのポインタ配列。

- sf_message_cfg_t::p_block_pool_name

  ブロック プール名へのポインタ。

5.11.7.15 close

ssp_err_t(* sf_message_api_t::close)(sf_message_buffer_ctrl_t *const p_ctrl)

詳細説明

メッセージ フレームワークを終了処理します。また電力消費を低減できます。

- SF_MESSAGE_Close

表 72: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	メッセージング制御ブロックへのポインタ

定義 : sf_message_buffer_ctrl_t

メッセージング フレームワーク制御ブロック構造体

5.11.7.16 bufferAcquire

ssp_err_t(* sf_message_api_t::bufferAcquire)(sf_message_buffer_ctrl_t const *const p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const p_acquire_cfg, uint32_t const wait_option)

詳細説明

ブロックから渡されるメッセージ用に、バッファを取得します。また電力消費を低減できます。

- SF_MESSAGE_BufferAcquire

表 73: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	メッセージング制御ブロックへのポインタ
pp_buffer	入力 / 出力	割り当てられたバッファ メモリへのポインタへのポインタ
p_acquire_cfg	複数のビットを書き換えることもできます。	バッファ取得設定へのポインタ
wait_option	複数のビットを書き換えることもできます。	待機オプション （TX_NO_WAIT、TX_WAIT_FOREVER、または数値）

定義 : sf_message_buffer_ctrl_t

メッセージング フレームワーク制御ブロック構造体

パラメータ pp_buffer

定義 : sf_message_header_t**pp_buffer

メッセージ ヘッダー定義

- sf_message_header_t::event
- sf_message_header_t::class_instance

  < イベント クラス コード
- sf_message_header_t::code

  イベント コード。
- sf_message_header_t::event_b
- sf_message_header_t::struct{}

パラメータ p_acquire_cfg

定義 : sf_message_acquire_cfg_t const *const p_acquire_cfg

メッセージング フレームワーク ポスト API 関数設定構造体定義

- sf_message_acquire_cfg_t::buffer_keep

  バッファ キープ オプション。

パラメータ wait_option

uint32_t

5.11.7.17 bufferRelease

ssp_err_t(* sf_message_api_t::bufferRelease)(sf_message_buffer_ctrl_t *const p_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t const option)

詳細説明

SF_MESSAGE_BufferAcquire で取得されたバッファを解放します。また電力消費を低減できます。

- SF_MESSAGE_BufferRelease

表 74: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	メッセージング制御ブロックへのポインタ
p_buffer	複数のビットを書き換えることもできます。	以下によって割り当てられたバッファへのポインタ： SF_MESSAGE_BufferAcquire

表 74: パラメータ (続き)

名前	方向	説明
option	複数のビットを書き換えることもできます。	バッファの解放に関するオプション (SF_MESSAGE_RELEASE_OPTION_NONE、SF_MESSAGE_RELEASE_OPTION_ACK、SF_MESSAGE_RELEASE_OPTION_NAK、SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE)

定義 : sf_message_buffer_ctrl_t

メッセージング フレームワーク制御ブロック構造体

パラメータ p_buffer

定義 : sf_message_header_t*const p_buffer

メッセージ ヘッダー定義

- sf_message_header_t::event
- sf_message_header_t::class_instance

  < イベント クラス コード
- sf_message_header_t::code

  イベント コード。
- sf_message_header_t::event_b
- sf_message_header_t::struct{}

パラメータ option

5.11.7.18 post

ssp_err_t(* sf_message_api_t::post)(sf_message_buffer_ctrl_t *const p_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)

詳細説明

サブスクライバーに対してメッセージをポストします。また電力消費を低減できます。

- SF_MESSAGE_Post

表 75: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	メッセージング制御ブロックへのポインタ
p_buffer	複数のビットを書き換えることもできます。	以下によって割り当てられたバッファへのポインタ：SF_MESSAGE_BufferAcquire
p_post_cfg	複数のビットを書き換えることもできます。	メッセージ ポスト設定へのポインタ
wait_option	複数のビットを書き換えることもできます。	待機オプション （TX_NO_WAIT、TX_WAIT_FOREVER、または数値）

定義 : sf_message_buffer_ctrl_t

メッセージング フレームワーク制御ブロック構造体

パラメータ p_buffer

定義 : sf_message_header_tconst *const p_buffer

メッセージ ヘッダー定義

- sf_message_header_t::event
- sf_message_header_t::class_instance

  < イベント クラス コード
- sf_message_header_t::code

  イベント コード。
- sf_message_header_t::event_b
- sf_message_header_t::struct{}

パラメータ p_post_cfg

定義 : sf_message_post_cfg_t const *const p_post_cfg

メッセージング フレームワーク取得 API 関数設定構造体定義

- sf_message_post_cfg_t::sf_message_priority_t

  メッセージの優先度。

  列挙値 :

  – SF_MESSAGE_PRIORITY_NORMAL

– SF_MESSAGE_PRIORITY_HIGH

- sf_message_post_cfg_t::p_callback

  ユーザー コールバック関数。

- sf_message_post_cfg_t::p_context

  コールバック時にユーザーに提供されるコンテキスト。

パラメータ wait_option

uint32_t

5.11.7.19 pend

ssp_err_t(* sf_message_api_t::pend)(sf_message_buffer_ctrl_t const *const p_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option)

詳細説明

メッセージを保留します。また電力消費を低減できます。

- SF_MESSAGE_Pend

表 76: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	メッセージング制御ブロックへのポインタ
p_queue	複数のビットを書き換えることもできます。	threadX メッセージ キュー オブジェクトへのポインタ
pp_buffer	入力 / 出力	メッセージが格納されているバッファへのポインタへのポインタ。
wait_option	複数のビットを書き換えることもできます。	待機オプション （TX_NO_WAIT、TX_WAIT_FOREVER、または数値）

定義 : sf_message_buffer_ctrl_t

メッセージング フレームワーク制御ブロック構造体

パラメータ p_queue

const

パラメータ pp_buffer

定義 : sf_message_header_t**pp_buffer

メッセージ ヘッダー定義

- sf_message_header_t::event
- sf_message_header_t::class_instance

  < イベント クラス コード
- sf_message_header_t::code

  イベント コード。
- sf_message_header_t::event_b
- sf_message_header_t::struct{}

パラメータ wait_option

uint32_t

5.11.7.20 versionGet

ssp_err_t(* sf_message_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

メッセージング フレームワークのバージョンを取得します。また電力消費を低減できます。

- SF_MESSAGE_VersionGet

表 77: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン番号を格納するメモリへのポインタ

パラメータ p_version

5.11.7.21 sf_message_instance_t

sf_message_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_message_buffer_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_message_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_message_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.12 パワー プロファイルフレームワークインタフェース

パワー プロファイル フレームワーク インタフェース。

5.12.1 概要

アプリケーションを所定のローパワー構成のいずれかの状態に移行するためのフレームワークです。アプリケーションは、API 呼び出しを行ってローパワー スリープモードに移行し、外部割り込みまたは周期 RTC 割り込みによってスリープを解除できます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

フレームワーク パワー プロファイル インタフェースの説明 : パワープロファイルフレームワーク

5.12.2 インタフェース API

sf_power_profiles_api_t

関数名	説明
.open	ミューテックスを取得した後、HAL レイヤーで、下位レイヤー ドライバを初期化します
.sleep	MCU をソフトウェア スタンバイ モードにします。
.close	チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.12.3 データ構造体

- sf_power_profiles_callback_args_t
- sf_power_profiles_ctrl_t
- sf_power_profiles_cfg_t
- sf_power_profiles_instance_t

5.12.4 列挙

- sf_power_profiles_mode_t
- sf_power_profiles_event_t

5.12.5 定義

- #define SF_POWER_PROFILES_API_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義された API のバージョン
- #define SF_POWER_PROFILES_API_VERSION_MINOR

  初期値 :(1)

5.12.6 API データ

5.12.6.1 sf_power_profiles_mode_t

sf_power_profiles_mode_t

詳細説明

コールバック イベントのオプション。

列挙値

名前	説明
SF_POWER_PROFILES_MODE_RUN	標準開始モード （スリープなし）
SF_POWER_PROFILES_MODE_RUN	RTC を使用したウェイクアップ。
SF_POWER_PROFILES_MODE_RTC	外部ソースからのウェイクアップ。

5.12.6.2 sf_power_profiles_event_t

sf_power_profiles_event_t

詳細説明

コールバック イベントのオプション。

列挙値

名前	説明
SF_POWER_PROFILES_EVENT_PRE_SLEEP	スリープモードに移行する直前にコールバックを実行します。
SF_POWER_PROFILES_EVENT_POST_SLEEP	ウェイクアップの直後にコールバックを実行します。

5.12.7 API 構造

5.12.7.1 sf_power_profiles_callback_args_t

sf_power_profiles_callback_args_t

詳細説明

電力プロファイル コールバック引数定義

変数

- sf_power_profiles_event_t event

  電力プロファイル コールバック イベント。

- void * p_context

  ユーザー データのプレースホルダー。

5.12.7.2 sf_power_profiles_ctrl_t

sf_power_profiles_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、SF_POWER_PROFILES_Open の呼び出し時に実行されます

変数

- uint32_t open

  ドライバによって使用され、制御ブロックへのポインタが有効かどうかを確認します。

- ioport_cfg_t const * p_wake_ioport_pin_tbl

  ウェイクアップの I/O ポート設定へのポインタ。

- ioport_cfg_t const * p_sleep_ioport_pin_tbl

  スリープの I/O ポート設定へのポインタ。

- sf_power_profiles_mode_t operating_mode

  使用する電力プロファイル モード。

- lpm_api_t const * p_api_lpm

  ローレベル ローパワー ドライバ関数ポインタへのポインタ。

- rtc_api_t const * p_api_rtc

  ローレベル RTC ドライバ関数ポインタへのポインタ。

- rtc_ctrl_t * p_ctrl_rtc

  ローレベル RTC ドライバ制御ブロックへのポインタ。

- void(* p_callback)(sf_power_profiles_callback_args_t *p_args)

  コールバック関数。

- void * p_context

  ユーザー データのプレースホルダー。

5.12.7.3 sf_power_profiles_cfg_t

sf_power_profiles_cfg_t

詳細説明

RTOS 統合された電力プロファイル ドライバの構成

変数

- ioport_cfg_t const *const p_wake_ioport_pin_tbl

  ウェイクアップの I/O ポート設定へのポインタ。

- ioport_cfg_t const *const p_sleep_ioport_pin_tbl

  スリープの I/O ポート設定へのポインタ。

- sf_power_profiles_mode_t operating_mode

  使用する電力プロファイル モード。

- bool retain_output_signals

  (実装予定 :) True (デフォルト) ==> アドレス バスとバス コントロール信号が、出力状態を保持します False ==> 信号が高インピーダンス状態に設定されます

- lpm_instance_t const *const p_lower_lvl_lpm

  LPM インスタンスへのポインタ。

- rtc_instance_t const *const p_lower_lvl_rtc

  RTC インスタンスへのポインタ (存在する場合)。

- void(* p_callback)(sf_power_profiles_callback_args_t *p_args)

  コールバック関数。

- void * p_context

  ユーザー データのプレースホルダー。

5.12.7.4 sf_power_profiles_api_t

sf_power_profiles_api_t

詳細説明

フレームワーク電力プロファイル API 構造体。実装には、以下の API が使用されます。

5.12.7.5 open

ssp_err_t(* sf_power_profiles_api_t::open)(sf_power_profiles_ctrl_t *const p_ctrl, sf_power_profiles_cfg_t const *const p_cfg)

詳細説明

ミューテックスを取得した後、HAL レイヤーで、下位レイヤー ドライバを初期化します。以下として実装されます。

- SF_POWER_PROFILES_Open

表 78: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。要素はここで初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : sf_power_profiles_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_POWER_PROFILES_Open の呼び出し時に実行されます

定義 :

定義 : sf_power_profiles_cfg_t const *const p_cfg

RTOS 統合された電力プロファイル ドライバの構成

- sf_power_profiles_cfg_t::ioport_cfg_t

  ウェイクアップの I/O ポート設定へのポインタ。

- sf_power_profiles_cfg_t::ioport_cfg_t

  スリープの I/O ポート設定へのポインタ。

- sf_power_profiles_cfg_t::sf_power_profiles_mode_t

  使用する電力プロファイル モード。

  列挙値 :

  – SF_POWER_PROFILES_MODE_RUN

  – SF_POWER_PROFILES_MODE_RUN

  – SF_POWER_PROFILES_MODE_RTC

- sf_power_profiles_cfg_t::retain_output_signals

  (実装予定 :) True （デフォルト) ==> アドレス バスとバス コントロール信号が、出力状態を保持します False ==> 信号が高インピーダンス状態に設定されます

- sf_power_profiles_cfg_t::lpm_instance_t

  LPM インスタンスへのポインタ。

- sf_power_profiles_cfg_t::rtc_instance_t

  RTC インスタンスへのポインタ （存在する場合)。

- sf_power_profiles_cfg_t::p_callback

  コールバック関数。

- sf_power_profiles_cfg_t::p_context

  ユーザー データのプレースホルダー。

5.12.7.6 sleep

ssp_err_t(* sf_power_profiles_api_t::sleep)(sf_power_profiles_ctrl_t *const p_ctrl)

詳細説明

MCU をソフトウェア スタンバイ モードにします。また電力消費を低減できます。

- SF_POWER_PROFILES_Sleep

  I : 電力プロファイル動作モードが SF_POWER_PROFILES_MODE_RTC に設定されている場合、MCU は RTC とそれに関連するクロックも実行できるローパワー モードに移行するように構成されます。RTC が指定されたインターバルで割り込むように RTC を構成するのは、アプリケーションの役目です。

表 79: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_POWER_PROFILES_Open で設定された制御ブロックへのポインタ。

定義 : sf_power_profiles_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_POWER_PROFILES_Open の呼び出し時に実行されます

5.12.7.7 close

ssp_err_t(* sf_power_profiles_api_t::close)(sf_power_profiles_ctrl_t *const p_ctrl)

詳細説明

チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。また電力消費を低減できます。

- SF_POWER_PROFILES_Close

表 80: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SF_POWER_PROFILES_Open で設定された制御ブロックへのポインタ。

定義 : sf_power_profiles_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_POWER_PROFILES_Open の呼び出し時に実行されます

5.12.7.8 versionGet

ssp_err_t(* sf_power_profiles_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 81: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

5.12.7.9 sf_power_profiles_instance_t

sf_power_profiles_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_power_profiles_ctrl_t * p_ctrl
  このインスタンスの制御構造体へのポインタ。
- sf_power_profiles_cfg_t const * p_cfg
  イベント クラスのインスタンス範囲の始点。
- sf_power_profiles_api_t const * p_api
  イベント クラスのインスタンス範囲の終点。

5.13 SPI フレームワークインタフェース

RTOS 統合された SPI フレームワーク インタフェース。

5.13.1 概要

この SSP インタフェースは、ThreadX 対応の SPI フレームワークへのアクセスを提供します。このインタフェースは、SPI フレームワーク によって実装されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

SPI フレームワーク インタフェースの説明 : SPI フレームワーク

5.13.2 インタフェース API

sf_spi_api_t

関数名	説明
.open	指定された SPI デバイスをバス上で開きます。
.read	SPI デバイスからデータを受信します。
.write	SPI デバイスにデータを送信します。
.writeRead	SPI デバイスとの間で、データの送信と受信を同時に実行します （全二重通信)。この読み書き API は、ミューテックス オブジェクトを取得し、SPI HAL レイヤーで SPI データ送信を処理するとともに、SPI HAL レイヤーからデータを受信します。またこの API は、イベント フラグ待ちを使用して、データ転送の完了と同期します。
.close	制御ハンドルで指定された SPI デバイスを無効にします。またバスにデバイスが接続されていない場合は、RTOS サービスを終了します。この関数は、ハンドルで指定された SPI チャネルへの電源を遮断し、関連付けられた割り込みを無効にします。

関数名	説明
.lock	バスをデバイスにロックします。ロックすると、一定時間 （ロックとロック解除の間)、デバイスがバスを予約できるようになります。これにより、デバイスが複数の読み書き操作を中断なく完了することができます。
.unlock	特定のデバイスのバスをロック解除し、他のデバイスがそのバスを使用できるようにします。
.version	基礎となるドライバのバージョン情報を取得します。

5.13.3 データ構造体

- sf_spi_bus_t
- sf_spi_cfg_t
- st_sf_spi_ctrl
- sf_spi_instance_t

5.13.4 列挙

- sf_spi_dev_state_t

5.13.5 定義

- #define SF_SPI_API_VERSION_MAJOR

  初期値 :(1)
- #define SF_SPI_API_VERSION_MINOR

  初期値 :(1)

5.13.6 API データ

5.13.6.1 sf_spi_dev_state_t

sf_spi_dev_state_t

詳細説明

SF SPI デバイスのステータス

列挙値

名前	説明
SF_SPI_DEV_STATE_CLOSED	SPI デバイスは閉じています。
SF_SPI_DEV_STATE_OPENED	SPI デバイスは開かれています。

5.13.7 API 構造

5.13.7.1 sf_spi_bus_t

sf_spi_bus_t

詳細説明

SPI バスを定義するデータ構造体。

変数

- uint8_t channel

  チャネル。
- uint32_t freq_hz_max

  サポートされている最大バス周波数。
- uint32_t freq_hz_min

  サポートされている最小バス周波数。
- TX_MUTEX * p_lock_mutex

  このチャネルのロック ミューテックス ハンドル。
- TX_EVENT_FLAGS_GROUP * p_sync_eventflag

  SPI データ転送用のイベント フラグ オブジェクトへのポインタ。
- sf_spi_ctrl_t ** pp_curr_ctrl

  バスを使用している現在のデバイス。
- uint8_t * p_bus_name

  ペリフェラル名 SSPI/RSPI
- spi_api_t * p_lower_lvl_api

  フレームワークで使用される SPI HAL インタフェースへのポインタ。
- uint8_t device_count

  初期値は 0 です。

5.13.7.2 sf_spi_cfg_t

sf_spi_cfg_t

詳細説明

フレームワーク SPI ドライバの設定。

変数

- sf_spi_bus_t * p_bus

  デバイスが使用するバス。

- ioport_port_pin_t chip_select

  このデバイスに対するチップの選択。

- ioport_level_t chip_select_level_active

  CS 極性、アクティブ High または Low。

- spi_cfg_t const * p_lower_lvl_cfg

  SPI HAL 設定へのポインタ。

5.13.7.3 st_sf_spi_ctrl

st_sf_spi_ctrl

詳細説明

SPI デバイス コンテキスト。初期化しないでください。初期化は、SF_SPI_Open の呼び出し時に実行されます。

変数

- sf_spi_bus_t * p_bus

  このデバイスを使用しているバス （cfg からコピー)

- ioport_port_pin_t chip_select

  このデバイスに対するチップの選択 （cfg からコピー)

- ioport_level_t chip_select_level_active

  CS 極性、アクティブ High または Low （cfg からコピー)

- spi_cfg_t lower_lvl_cfg

  SPI ペリフェラルの設定、バス設定に使用されます。

- spi_ctrl_t lower_lvl_ctrl

  SPI ペリフェラル制御ブロック。

- sf_spi_dev_state_t dev_state

  デバイスのステータス。

- bool locked

  バスをデバイスに対してロックおよびロック解除します。

5.13.7.4 sf_spi_api_t

sf_spi_api_t

詳細説明

SPI フレームワーク インタフェースの定義。

5.13.7.5 open

ssp_err_t(* sf_spi_api_t::open)(sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)

概要説明

指定された SPI デバイスをバス上で開きます。

詳細説明

表 82: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御ブロック用のユーザー定義のストレージへのポインタ。
p_cfg	複数のビットを書き換えることもできます。	SPI フレームワーク設定構造体へのポインタ。

定義 :

定義 : sf_spi_cfg_t const *const p_cfg

フレームワーク SPI ドライバの設定。

- sf_spi_cfg_t::sf_spi_bus_t

  デバイスが使用するバス。

- sf_spi_cfg_t::chip_select

  このデバイスに対するチップの選択。

- sf_spi_cfg_t::ioport_level_t

  CS 極性、アクティブ High または Low。

- sf_spi_cfg_t::spi_cfg_t

  SPI HAL 設定へのポインタ。

5.13.7.6 read

ssp_err_t(* sf_spi_api_t::read)(sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

概要説明

SPI デバイスからデータを受信します。

詳細説明

I：この関数を使用する前に、open を呼び出して SPI デバイスを構成してください。

表 83:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの制御ブロックへのポインタ。
p_dest	out	SPI デバイスから受け取ったデータをコピーする宛先バッファへのポインタ。要求したデータ量を保持するための容量を確保することは、呼び出し側の責任です。
length	複数のビットを書き換えることもできます。	転送するデータの単位数を指定します （単位サイズは bit_width によって指定)。
bit_width	複数のビットを書き換えることもできます。	転送データ ビット幅を指定します。
timeout	複数のビットを書き換えることもできます。	タイムアウト。指定できる値は、TX_NO_WAIT （0x00000000）、TX_WAIT_FOREVER （0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

パラメータ **p_dest**

const

パラメータ **length**

uint32_t

パラメータ **bit_width**

パラメータ **timeout**

uint32_t

5.13.7.7 write

ssp_err_t(* sf_spi_api_t::write)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

概要説明

SPI デバイスにデータを送信します。

詳細説明

I：この関数を使用する前に、open を呼び出して SPI デバイスを構成してください。

表 84:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	SPI デバイス宛てにデータを送信するソース データ バッファへのポインタ。 引数に NULL を指定することはできません。
length	複数のビットを書き換えることもできます。	転送するデータの単位数を指定します （単位サイズは bit_width によって指定)。

表 84: パラメータ (続き)

名前	方向	説明
bit_width	複数のビットを書き換えることもできます。	転送データ ビット幅を指定します。
timeout	複数のビットを書き換えることもできます。	タイムアウト。指定できる値は、TX_NO_WAIT （0x00000000）、TX_WAIT_FOREVER （0xFFFFFFFF）、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

パラメータ **p_src**

const

パラメータ **length**

uint32_t

パラメータ **bit_width**

パラメータ **timeout**

uint32_t

5.13.7.8 writeRead

ssp_err_t(* sf_spi_api_t::writeRead)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

概要説明

SPI デバイスとの間で、データの送信と受信を同時に実行します （全二重通信）。

詳細説明

この読み書き API は、ミューテックス オブジェクトを取得し、SPI HAL レイヤーで SPI データ送信を処理するとともに、SPI HAL レイヤーからデータを受信します。またこの API は、イベント フラグ待ちを使用して、データ転送の完了と同期します。

I：この関数を使用する前に、open を呼び出して SPI を構成してください。

表 85: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	SPI デバイス宛てにデータを送信するソース データ バッファへのポインタ。 引数に NULL を指定することはできません。
p_dest	out	SPI デバイスから受け取ったデータをコピーする宛先バッファへのポインタ。要求したデータ量を保持するための容量を確保することは、呼び出し側の責任です。
length	複数のビットを書き換えることもできます。	転送するデータの単位数を指定します （単位サイズは bit_width によって指定)。
bit_width	複数のビットを書き換えることもできます。	転送データ ビット幅を指定します。
timeout	複数のビットを書き換えることもできます。	タイムアウト。指定できる値は、TX_NO_WAIT （0x00000000)、TX_WAIT_FOREVER （0xFFFFFFFF)、または ThreadX ティック カウントで表したタイムアウト値 （0x00000001 ～ 0xFFFFFFFE）です。

パラメータ **p_src**

const

パラメータ **p_dest**

const

パラメータ **length**

uint32_t

パラメータ **bit_width**

パラメータ **timeout**

uint32_t

5.13.7.9 lock

ssp_err_t(* sf_spi_api_t::lock)(sf_spi_ctrl_t *const p_ctrl)

概要説明

バスをデバイスにロックします。ロックすると、一定時間 （ロックとロック解除の間)、デバイスがバスを予約できるようになります。これにより、デバイスが複数の読み書き操作を中断なく完了することができます。

詳細説明

表 86: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの制御ブロックへのポインタ。

5.13.7.10 unlock

ssp_err_t(* sf_spi_api_t::unlock)(sf_spi_ctrl_t *const p_ctrl)

概要説明

特定のデバイスのバスをロック解除し、他のデバイスがそのバスを使用できるようにします。

詳細説明

表 87: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	デバイスの制御ブロックへのポインタ。

5.13.7.11 version

ssp_err_t(* sf_spi_api_t::version)(ssp_version_t *const p_version)

概要説明

基礎となるドライバのバージョン情報を取得します。

詳細説明

表 88: パラメータ

名前	方向	説明
p_version	out	バージョン番号を返すメモリ位置へのポインタ

パラメータ p_version

5.13.7.12 sf_spi_instance_t

sf_spi_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_spi_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_spi_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_spi_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.14 スレッド監視フレームワークインタフェース

RTOS 統合されたスレッド監視用フレームワーク インタフェース。

スレッドが不正な動作を行った場合に、デバイスをリセットします。このフレームワーク モジュールは、WDT と IWDT HAL モジュールの両方をサポートしています。

5.14.1 概要

指定した周期でスレッドを実行するアプリケーションでスレッドを監視するための、ThreadX 対応のウォッチドッグ タイマ スレッド監視フレームワークです。監視対象のスレッドは、SF_THREAD_MONITOR_ThreadRegister によってそれ自体を登録し、実行のたびに SF_THREAD_MONITOR_CountIncrement を呼び出してカウントをインクリメントします。監視対象の各スレッドはまた、通常の実行で想定される最大 / 最小カウント値を提供します。

スレッド監視は周期的に実行され、監視対象の各スレッドのカウント値をチェックします。カウント値が想定された値の範囲から外れている場合、ウォッチドッグ タイマはデバイスのリセットを許可します。すべてのスレッド カウントが想定の範囲内である場合は、ウォッチドッグ タイマがリフレッシュされます。

スレッド監視では、WDT モジュールと IWDT モジュールがサポートされています。

フレームワーク レイヤーを使用すると、ソフトウェア プロジェクト全体を保護することができます。これには、選択したウォッチドッグ タイマ （IWDT は、独自のクロック ソースがあり、リセット後に自動的に開始するので最も安全です）のリフレッシュが許可されるウィンドウ （時間枠）を指定し、その中でプライオリティの高いスレッド （フレームワーク レイヤー）を周期的に実行します。このスレッドは、システム内の他のすべてのスレッドの状態を監視します。監視対象のスレッドのいずれかが想定どおりに動作しない場合、ウォッチドッグ タイマがリフレッシュされず、システムのリセットが許可されます。スレッドが想定どおり動作する場合は、ウォッチドッグ タイマがリフレッシュされます。

それ以外のスレッドの監視は、監視対象の各スレッドが実行されるたびに、そのタスクのグローバル カウント変数をインクリメントして行われます。スレッド監視スレッドは、各スレッドのカウント変数を監視して、その変数が想定される範囲内であることを確認します。変数のいずれかが範囲外の場合はデバイスのリセットが許可され、範囲内であればすべての変数が 0 にクリアされて、ウォッチドッグがリフレッシュされます。想定範囲は、プロファイリング モードを使用して決定します。

このアプローチについては、以下の記事を参照してください。

Jack Ganssle, "Great Watchdog Timers for Embedded Systems,"http://www.ganssle.com/watchdogs.htm

この方法では、各スレッドでカウント変数をインクリメントする必要がありますが、このためのオーバーヘッドは保護がもたらす大きなメリットに比べればごくわずかです。

使用されるインタフェース : WDT インタフェース

関連する SSP アーキテクチャのトピック :

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

スレッド監視インタフェースの説明 : スレッド監視フレームワーク

5.14.2 インタフェース API

sf_thread_monitor_api_t

関数名	説明
.open	WDT または IWDT モジュールを構成します。構成データを使用して WDT/IWDT のタイムアウト期間が決定され、登録されたスレッドを監視するスレッドが作成されます。
.close	スレッド監視スレッドをサスペンドします。これにより、ウォッチドッグ ペリフェラルがリフレッシュされなくなります。
.threadRegister	監視対象のスレッドを登録します。
.threadUnregister	スレッドを監視対象から除外します。
.countIncrement	監視対象のスレッドのカウント値を安全にインクリメントします。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.14.3 データ構造体

- sf_thread_monitor_cfg_t
- sf_thread_monitor_thread_counter_t
- sf_thread_monitor_counter_min_max_t
- sf_thread_monitor_ctrl_t
- sf_thread_monitor_instance_t

5.14.4 定義

- #define THREAD_MONITOR_THREAD_STACK_SIZE

  初期値 :400u
- #define SF_THREAD_MONITOR_API_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義された API のバージョン

- #define SF_THREAD_MONITOR_API_VERSION_MINOR

  初期値 :(1)

5.14.5 API 構造

5.14.5.1 sf_thread_monitor_cfg_t

sf_thread_monitor_cfg_t

詳細説明

RTOS スレッド監視ドライバの構成

変数

- wdt_instance_t const * p_lower_lvl_wdt

  ローレベル ウォッチドッグ インスタンスへのポインタ。

- bool profiling_mode_enabled

  プロファイリング モードを有効化または無効化します。

- UINT priority

  スレッド監視スレッドのプライオリティ。

5.14.5.2 sf_thread_monitor_thread_counter_t

sf_thread_monitor_thread_counter_t

詳細説明

監視対象の各スレッド用のカウンタ ブロック。

変数

- uint32_t current_count

  スレッドの現在のカウント値。

- uint32_t minimum_count

  最小期待カウント値。現在のカウントがこの値よりも小さい場合、ウォッチドッグがリセットされます。

- uint32_t maximum_count

  最大期待カウント値。現在のカウントがこの値よりも大きい場合、ウォッチドッグがリセットされます

- bool active

  監視スレッドに対し、このカウント データが現在アクティブかどうかを示します。スレッドを登録した時点では、カウントが途中からになり、監視対象とすべきでないため、この値が false に設定されます。スレッド監視スレッドによってすべてのカウントがゼロにクリアされると、この値が true に変わります。

- TX_THREAD * p_thread

  このカウンタ データのスレッドへのポインタ。

5.14.5.3 sf_thread_monitor_counter_min_max_t

sf_thread_monitor_counter_min_max_t

詳細説明

監視対象の各スレッド用のカウンタ ブロック。

変数

- uint32_t minimum_count

  最小期待カウント値。現在のカウントがこの値よりも小さい場合、ウォッチドッグがリセットされます。

- uint32_t maximum_count

  最大期待カウント値。現在のカウントがこの値よりも大きい場合、ウォッチドッグがリセットされます

5.14.5.4 sf_thread_monitor_ctrl_t

sf_thread_monitor_ctrl_t

詳細説明

スレッド監視制御ブロック。初期化しないでください。初期化は、SF_THREAD_MONITOR_Open の呼び出し時に実行されます

変数

- uint32_t open

  ドライバによって使用され、制御構造体が有効かどうかを確認します

- wdt_instance_t const * p_lower_lvl_wdt

  ウォッチドッグ周辺機能のインタフェース構造体へのポインタ

- uint32_t timeout_period_msec

  ウォッチドッグ タイムアウト期間の時間 （ミリ秒単位)。スレッドを監視する期間を計算するために使用します。

- uint32_t timeout_period_watchdog_clocks

  ウォッチドッグの最大カウント値。カウントへの同期に使用されます。

- bool profiling_mode_enabled

  ドライバによって使用され、プロファイリング モードが有効かどうかを確認します。

- TX_MUTEX mutex

  スレッド カウンタへのアクセスを保護するためのミューテックス。

- uint32_t profiling_mode_check

  prfiling_mode_enabled == true の場合に、プロファイリング モードが有効であることを確認するために使用される値。

- sf_thread_monitor_thread_counter_t thread_counters[THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS]

  スレッド カウンタ情報用のデータ ストレージ。

- TX_THREAD thread

  スレッド監視スレッド。

- void const * p_extend

  拡張構成データ。

- uint8_t stack[THREAD_MONITOR_THREAD_STACK_SIZE]

  スレッド監視スレッドのスタック。

5.14.5.5 sf_thread_monitor_api_t

sf_thread_monitor_api_t

詳細説明

スレッド監視 API 構造体。スレッド監視の実装には、以下の API が使用されます。

5.14.5.6 open

(* sf_thread_monitor_api_t::open)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_cfg_t const *const p_cfg)

概要説明

WDT または IWDT モジュールを構成します。構成データを使用して WDT/IWDT のタイムアウト期間が決定され、登録されたスレッドを監視するスレッドが作成されます。

詳細説明

また電力消費を低減できます。

- SF_THREAD_MONITOR_Open

表 89: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。要素はここで初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : sf_thread_monitor_ctrl_t

スレッド監視制御ブロック。初期化しないでください。初期化は、SF_THREAD_MONITOR_Open の呼び出し時に実行されます

定義 :

定義 : sf_thread_monitor_cfg_t const *const p_cfg

RTOS スレッド監視ドライバの構成

- sf_thread_monitor_cfg_t::wdt_instance_t

  ローレベル ウォッチドッグ インスタンスへのポインタ。

- sf_thread_monitor_cfg_t::profiling_mode_enabled

  プロファイリング モードを有効化または無効化します。

- sf_thread_monitor_cfg_t::priority

  スレッド監視スレッドのプライオリティ。

5.14.5.7 close

(* sf_thread_monitor_api_t::close)(sf_thread_monitor_ctrl_t *const p_ctrl)

概要説明

スレッド監視スレッドをサスペンドします。これにより、ウォッチドッグ ペリフェラルがリフレッシュされなくなります。

詳細説明

また電力消費を低減できます。

- SF_THREAD_MONITOR_Close

表 90: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_THREAD_MONITOR_Open で設定された制御構造体。

定義 : sf_thread_monitor_ctrl_t

スレッド監視制御ブロック。初期化しないでください。初期化は、SF_THREAD_MONITOR_Open の呼び出し時に実行されます

5.14.5.8 threadRegister

(* sf_thread_monitor_api_t::threadRegister)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max)

概要説明

監視対象のスレッドを登録します。

詳細説明

また電力消費を低減できます。

- SF_THREAD_MONITOR_ThreadRegister

表 91: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_THREAD_MONITOR_Open で設定された制御構造体。
p_counter_min_max	複数のビットを書き換えることもできます。	登録されるスレッドの値の最小値および最大値が含まれている構造体へのポインタ。

定義 : sf_thread_monitor_ctrl_t

スレッド監視制御ブロック。初期化しないでください。初期化は、SF_THREAD_MONITOR_Open の呼び出し時に実行されます

Parameter p_counter_min_max

定義 : sf_thread_monitor_counter_min_max_tconst *p_counter_min_max

監視対象の各スレッド用のカウンタ ブロック。

- sf_thread_monitor_counter_min_max_t::minimum_count

  最小期待カウント値。現在のカウントがこの値よりも小さい場合、ウォッチドッグがリセットされます。

- sf_thread_monitor_counter_min_max_t::maximum_count

  最大期待カウント値。現在のカウントがこの値よりも大きい場合、ウォッチドッグがリセットされます

5.14.5.9 threadUnregister

(* sf_thread_monitor_api_t::threadUnregister)(sf_thread_monitor_ctrl_t *const p_ctrl)

概要説明

スレッドを監視対象から除外します。

詳細説明

また電力消費を低減できます。

- SF_THREAD_MONITOR_ThreadUnregister

表 92: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_THREAD_MONITOR_Open で設定された制御構造体。

定義 : sf_thread_monitor_ctrl_t

スレッド監視制御ブロック。初期化しないでください。初期化は、SF_THREAD_MONITOR_Open の呼び出し時に実行されます

5.14.5.10 countIncrement

(* sf_thread_monitor_api_t::countIncrement)(sf_thread_monitor_ctrl_t *const p_ctrl)

概要説明

監視対象のスレッドのカウント値を安全にインクリメントします。

詳細説明

また電力消費を低減できます。

- SF_THREAD_MONITOR_CountIncrement

表 93: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	SF_THREAD_MONITOR_Open で設定された制御構造体。

定義 : sf_thread_monitor_ctrl_t

スレッド監視制御ブロック。初期化しないでください。初期化は、SF_THREAD_MONITOR_Open の呼び出し時に実行されます

5.14.5.11 versionGet

(* sf_thread_monitor_api_t::versionGet)(*const p_version)

概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

詳細説明

また電力消費を低減できます。

- SF_THREAD_MONITOR_VersionGet

表 94: パラメータ

名前	方向	説明
p_version	入力 / 出力	API およびコード バージョンを格納する構造体へのポインタ。

パラメータ p_version

5.14.5.12 sf_thread_monitor_instance_t

sf_thread_monitor_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_thread_monitor_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_thread_monitor_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_thread_monitor_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.15 CTSU フレームワークインタフェース

RTOS 統合された CTSU フレームワークインタフェース。

5.15.1 概要

CTSU HAL ドライバを制御するための ThreadX 対応の CTSU インタフェースです。CTSU ハードウェアの実行と、スキャン結果の読み戻しに使用することができます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

CTSU フレームワーク インタフェースの説明：静電容量タッチフレームワーク

5.15.2 インタフェース API

sf_touch_ctsu_api_t

関数名	説明
.open	タッチ CTSU フレームワークを初期化します。またローレベル ハードウェアを構成し、構成された更新周波数に従ってスキャンを実行します。
.read	選択したオプションに従って、未加工データ、バイナリデータ、その他の派生データなどの結果を CTSU から読み取ります。
.close	進行中のスキャンを停止し、内部スレッドを破棄してフレームワークを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.15.3 データ構造体

- sf_touch_ctsu_callback_args_t
- sf_touch_ctsu_ctrl_t
- sf_touch_ctsu_cfg_t

- sf_touch_ctsu_button_instance_t

5.15.4 定義

- #define SF_TOUCH_CTSU_VERSION_MAJOR

  初期値 :(1)
- #define SF_TOUCH_CTSU_VERSION_MINOR

  初期値 :(0)

5.15.5 API 構造

5.15.5.1 sf_touch_ctsu_callback_args_t

sf_touch_ctsu_button_callback_args_t

詳細説明

コールバック引数構造体

変数

- void * p_context

5.15.5.2 sf_touch_ctsu_ctrl_t

sf_touch_ctsu_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_CTSU_Open の呼び出し時に実行されます

変数

- uint32_t open

  ドライバによって使用され、制御ブロックが有効かどうかを確認します。
- uint16_t update_hz

  スキャンの実行周波数。
- ctsu_instance_t * p_lower_lvl_instance

  CTSU インスタンスへのポインタ。
- uint32_t cb_index

  コールバック レジストリ テーブルのインデックスを示します。

5.15.5.3 sf_touch_ctsu_cfg_t

sf_touch_ctsu_cfg_t

詳細説明

RTOS 統合された CTSU タッチ フレームワークの構成。

変数

- UINT priority

  タッチ パネル スレッドの優先順位。

- uint16_t update_hz

  スキャンの実行周波数。これは最新の open() 呼び出しにより設定されます。

- void(* p_callback)(sf_touch_ctsu_callback_args_t *p_args)

  コールバック関数。

- void * p_context

  コールバックの引数。

- ctsu_instance_t * p_ctsu_instance

  CTSU インスタンス。

5.15.5.4 sf_touch_ctsu_api_t

sf_touch_ctsu_api_t

詳細説明

CTSU フレームワーク API 構造体。実装には、以下の API が使用されます。

5.15.5.5 open

(* sf_touch_ctsu_api_t::open)(sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)

詳細説明

タッチ CTSU フレームワークを初期化します。またローレベル ハードウェアを構成し、構成された更新周波数に従ってスキャンを実行します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Open

表 95: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : sf_touch_ctsu_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_CTSU_Open の呼び出し時に実行されます

定義 :

定義 : sf_touch_ctsu_cfg_t const *const p_cfg

RTOS 統合された CTSU タッチ フレームワークの構成。

- sf_touch_ctsu_cfg_t::priority

  タッチ パネル スレッドの優先順位。

- sf_touch_ctsu_cfg_t::update_hz

  スキャンの実行周波数。これは最新の open() 呼び出しにより設定されます。

- sf_touch_ctsu_cfg_t::p_callback

  コールバック関数。

- sf_touch_ctsu_cfg_t::p_context

  コールバックの引数。

- sf_touch_ctsu_cfg_t::ctsu_instance_t

  CTSU インスタンス。

5.15.5.6 read

(* sf_touch_ctsu_api_t::read)(sf_touch_ctsu_ctrl_t *const p_ctrl, void *p_dest, opts, const *channels, const uint16_t count)

詳細説明

選択したオプションに従って、未加工データ、バイナリ データ、その他の派生データなどの結果を CTSU から読み取ります。また電力消費を低減できます。

- SF_TOUCH_CTSU_Read

表 96: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_dest	out	読み取ったデータの書き込み先の場所へのポインタ
opts	複数のビットを書き換えることもできます。	読み取りオプション
channels	複数のビットを書き換えることもできます。	データを読み取るチャネル / チャネル ペアを指定します
channels	複数のビットを書き換えることもできます。	データを読み取るチャネル / チャネル ペアの数を指定します

定義 : `sf_touch_ctsu_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_CTSU_Open の呼び出し時に実行されます

パラメータ p_dest

const

パラメータ opts

パラメータ channels

パラメータ channels

5.15.5.7 close

(* `sf_touch_ctsu_api_t`::close)(`sf_touch_ctsu_ctrl_t` *const p_ctrl)

詳細説明

進行中のスキャンを停止し、内部スレッドを破棄してフレームワークを閉じます。また電力消費を低減できます。

- SF_TOUCH_CTSU_Close

表 97: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : sf_touch_ctsu_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_CTSU_Open の呼び出し時に実行されます

5.15.5.8 versionGet

(* sf_touch_ctsu_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- SF_TOUCH_CTSU_VersionGet

表 98: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

5.15.5.9 sf_touch_ctsu_button_instance_t

sf_touch_ctsu_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_touch_ctsu_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_touch_ctsu_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_touch_ctsu_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.16 CTSU ボタンフレームワークインタフェース

RTOS 統合された CTSU ボタン フレームワーク インタフェース。

このフレームワーク レイヤーは、CTSU フレームワーク レイヤーを使用して、ボタン インタフェースを実装します。ユーザーはこのボタン フレームワークを使用することで、Workbench から生成された構成構造体を使用する複数のボタンを構成して使用できます。各ボタンのアクション （押す、離すなど）によって、ボタン ID とイベント タイプを示す引数を使用して、そのボタンのコールバックが実行されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

CTSU ボタン フレームワーク インタフェースの説明 : 静電容量タッチボタンフレームワーク

5.16.1 インタフェース API

sf_touch_ctsu_button_api_t

関数名	説明
.open	タッチ CTSU ボタン フレームワークを初期化します。またローレベル ハードウェアを構成し、すべてのボタンに対してコールバック関数を登録します。
.enable	構成されたボタンのコールバック通知を有効化します。
.disable	構成されたボタンのコールバック通知を無効化します。
.close	タッチ CTSU ボタン フレームワークを閉じます。他のモジュールによって使用されていない場合は、ボタン フレームワークと下位レイヤーを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.16.2 データ構造体

- sf_touch_ctsu_button_callback_args_t
- sf_touch_ctsu_button_individual_t
- sf_touch_ctsu_button_cfg_t

- sf_touch_ctsu_button_hdl
- sf_touch_ctsu_button_ctrl_t
- sf_touch_ctsu_button_instance_t

5.16.3 列挙

- sf_touch_button_state_t

5.16.4 型定義

- sf_touch_ctsu_button_id

5.16.5 定義

- #define SF_TOUCH_CTSU_BUTTON_VERSION_MAJOR

  初期値 :(01)

  API のバージョン番号。
- #define SF_TOUCH_CTSU_BUTTON_VERSION_MINOR

  初期値 :(00)

5.16.6 API データ

5.16.6.1 sf_touch_button_state_t

sf_touch_button_state_t

詳細説明

ボタンの状態。

列挙値

名前	説明
TOUCH_BUTTON_STATE_RELEASED	ボタンは離された状態です。
TOUCH_BUTTON_STATE_PRESSED	ボタンは押された状態です。
TOUCH_BUTTON_STATE_LONG_HOLD	ボタンは長押しされた状態です (sf_touch_ctsu_button_config.h で指定された時間)

名前	説明
TOUCH_BUTTON_STATE_SHORT_HOLD	ボタンは短押しされた状態です (sf_touch_ctsu_button_config.h で指定された時間)。
TOUCH_BUTTON_STATE_STUCK	ボタンはスタックしています。
TOUCH_BUTTON_STATE_INITIAL	ボタンが正常に初期化されました。
TOUCH_BUTTON_STATE_CLOSING	ボタンは無効化されており、イベントを生成しません。
TOUCH_BUTTON_STATE_MULTI_TOUCH	複数のタッチ要素がタッチされています。
TOUCH_BUTTON_STATE_DISABLED	ボタンの更新が無効化されています。

5.16.6.2 sf_touch_ctsu_button_id

typedef uint32_t sf_touch_ctsu_button_id

詳細説明

各ボタンに対して使用される一意の識別子

5.16.7 API 構造

5.16.7.1 sf_touch_ctsu_button_callback_args_t

sf_touch_ctsu_button_callback_args_t

詳細説明

ボタン コールバック引数の定義

変数

- sf_touch_ctsu_button_id id
  ボタンの一意の識別子。
- sf_touch_button_state_t event
  ボタン コールバック イベント。
- void const * p_context
  ユーザー データのプレースホルダー。

5.16.7.2 sf_touch_ctsu_button_individual_t

sf_touch_ctsu_button_individual_t

詳細説明

ソフトウェアにおけるボタンとその動作の定義。

変数

- ctsu_channel_pair_t button_channel

  ボタンを構成するチャネル / チャネル ペアを定義します。

- uint8_t release_enable

  ボタン離しイベントを有効化します

- uint8_t press_enable

  ボタン押しイベントを有効化します

- uint8_t repeat_enable

  繰り返しイベントを有効化します

- uint8_t shorthold_enable

  短押しイベントを有効化します

- uint8_t longhold_enable

  長押しイベントを有効化します

- uint8_t byte

  イベントのバイト表現を有効化します。

- union{} event_enable

  このメンバーの定義については、ソース コードを参照してください。

- uint16_t debounce_threshold

  ボタンがタッチされたと判定されて状態が変化するまでの連続した時間の長さ。

- sf_touch_ctsu_button_id id

  ボタンの一意の識別子。

5.16.7.3 sf_touch_ctsu_button_cfg_t

sf_touch_ctsu_button_cfg_t

詳細説明

ボタン構成構造体

変数

- sf_touch_ctsu_button_instance_t const *const p_lower_lvl_touch_framework

  タッチ フレームワーク インスタンスへのポインタ。

- uint32_t button_count

  構成内のボタンの数。

- sf_touch_ctsu_button_individual_t ** pp_button_cfgs

  ボタン構成へのポインタ。

- void(* p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)

  コールバック関数。

- void const * p_context

  ユーザー データのプレースホルダー。

- void const * p_extend

  インスタンス固有の設定値に対応するための拡張パラメータ。

5.16.7.4 sf_touch_ctsu_button_hdl

sf_touch_ctsu_button_hdl

詳細説明

各ボタンのハンドル

変数

- sf_touch_ctsu_button_individual_t button_cfg

  個別のボタン構成。

- sf_touch_button_state_t state

  ボタンの現在の状態を表します。

- int16_t offset

  結果配列でのオフセットおよびバイナリでのビット オフセット。

- sf_touch_button_state_t prev_state

  ボタンの以前の状態を保持します。

- uint32_t debounce_counter

  カウンタが復帰しないようにします。

- uint32_t active_event_counter

  ボタンが 2 つの状態の間にある時間の長さ。

- uint32_t press_down_counter

  ボタンが押されたままになる時間の長さ。

- uint32_t open

  ボタンが開いていることを示します。

5.16.7.5 sf_touch_ctsu_button_ctrl_t

sf_touch_ctsu_button_ctrl_t

詳細説明

このインタフェースの制御構造体

変数

- uint32_t opened

  初期化状態を保存します。

- sf_touch_ctsu_button_hdl * p_button_hdl

  ボタン ハンドルへのポインタ。

- uint32_t button_count

  ボタン カウント。

- sf_touch_ctsu_api_t const * p_lower_lvl_api

  ローレベル インスタンスの API 構造体へのポインタ。

- sf_touch_ctsu_ctrl_t lower_lvl_ctrl

  ローレベル インスタンスの制御構造体。

- void const * p_context

  ユーザー データのプレースホルダー。

- void(* p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)

  コールバック関数。

5.16.7.6 sf_touch_ctsu_button_api_t

sf_touch_ctsu_button_api_t

詳細説明

タッチ CTSU ボタン フレームワーク API 構造体。実装には、以下の API が使用されます。

5.16.7.7 open

ssp_err_t(* sf_touch_ctsu_button_api_t::open)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_cfg_t const *const p_cfg)

詳細説明

タッチ CTSU ボタン フレームワークを初期化します。またローレベル ハードウェアを構成し、すべてのボタンに対してコールバック関数を登録します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Button_Open

表 99: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : `sf_touch_ctsu_button_ctrl_t`

このインタフェースの制御構造体

定義 :

定義 : `sf_touch_ctsu_button_cfg_t` const *const p_cfg

ボタン構成構造体

- `sf_touch_ctsu_button_cfg_t`::`sf_touch_ctsu_button_instance_t`
  タッチ フレームワーク インスタンスへのポインタ。
- `sf_touch_ctsu_button_cfg_t`::button_count
  構成内のボタンの数。
- `sf_touch_ctsu_button_cfg_t`::`sf_touch_ctsu_button_individual_t`
  ボタン構成へのポインタ。
- `sf_touch_ctsu_button_cfg_t`::p_callback
  コールバック関数。
- `sf_touch_ctsu_button_cfg_t`::p_context
  ユーザー データのプレースホルダー。
- `sf_touch_ctsu_button_cfg_t`::p_extend
  インスタンス固有の設定値に対応するための拡張パラメータ。

5.16.7.8 enable

ssp_err_t(* `sf_touch_ctsu_button_api_t`::enable)(`sf_touch_ctsu_button_ctrl_t` *const p_ctrl, sf_touch_ctsu_button_id const button_id)

詳細説明

構成されたボタンのコールバック通知を有効化します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Button_Enable

表 100: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
button_id	複数のビットを書き換えることもできます。	ボタンの一意の識別子

定義 : `sf_touch_ctsu_button_ctrl_t`

このインタフェースの制御構造体

パラメータ button_id

const

5.16.7.9 disable

ssp_err_t(* `sf_touch_ctsu_button_api_t`::disable)(`sf_touch_ctsu_button_ctrl_t` *const p_ctrl, sf_touch_ctsu_button_id const button_id)

詳細説明

構成されたボタンのコールバック通知を無効化します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Button_Disable

表 101: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
button_id	複数のビットを書き換えることもできます。	ボタンの一意の識別子

定義 : `sf_touch_ctsu_button_ctrl_t`

このインタフェースの制御構造体

パラメータ **button_id**

const

5.16.7.10 close

ssp_err_t(* sf_touch_ctsu_button_api_t::close)(sf_touch_ctsu_button_ctrl_t *const p_ctrl)

詳細説明

タッチ CTSU ボタン フレームワークを閉じます。他のモジュールによって使用されていない場合は、ボタン フレームワークと下位レイヤーを閉じます。また電力消費を低減できます。

- SF_TOUCH_CTSU_Button_Close

表 102: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : sf_touch_ctsu_button_ctrl_t

このインタフェースの制御構造体

5.16.7.11 versionGet

ssp_err_t(* sf_touch_ctsu_button_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Button_VersionGet

表 103: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ **p_version**

5.16.7.12 sf_touch_ctsu_button_instance_t

sf_touch_ctsu_button_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_touch_ctsu_button_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_touch_ctsu_button_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_touch_ctsu_button_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

5.17 CTSU スライダフレームワークインタフェース

RTOS 統合された CTSU スライダフレームワークインタフェース。

このフレームワーク レイヤーは、CTSU フレームワーク レイヤーを使用して、スライダまたはホイール インタフェースを実装します。ユーザーはこのスライダ フレームワークを使用することで、Workbench から生成された構成構造体を使用する複数のスライダ / ホイールを構成して使用できます。各スライダ / ホイールのアクション （押す、離すなど）によって、イベント タイプに応じた引数を使用して、そのスライダのコールバックが実行されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェースとは SSP インタフェース
- SSP レイヤーとは SSP 定義レイヤー
- SSP インタフェースおよびモジュールの使用方法 SSP モジュールの使用

CTSU スライダ フレームワーク インタフェースの説明 : 静電容量タッチスライダフレームワーク

5.17.1 インタフェース API

sf_touch_ctsu_slider_api_t

関数名	説明
.open	タッチ CTSU スライダ フレームワークを初期化します。またローレベル ハードウェアを構成し、すべてのスライダに対してコールバック関数を登録します。
.enable	構成されたスライダのコールバック通知を有効化します。
.disable	構成されたスライダのコールバック通知を無効化します。
.close	タッチ CTSU スライダ フレームワークを閉じます。他のモジュールによって使用されていない場合は、スライダ フレームワークと低レイヤーを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.17.2 データ構造体

- sf_touch_ctsu_slider_callback_args_t
- sf_touch_ctsu_slider_cfg_t

- sf_touch_ctsu_slider_ctrl_t
- sf_touch_ctsu_slider_instance_t

5.17.3 列挙

- sf_touch_ctsu_slider_state_t
- sf_slider_type_t

5.17.4 型定義

- sf_touch_ctsu_slider_id_t

5.17.5 定義

- #define SF_TOUCH_CTSU_SLIDER_VERSION_MAJOR

  初期値 :(01)

  API のバージョン番号。
- #define SF_TOUCH_CTSU_SLIDER_VERSION_MINOR

  初期値 :(00)

5.17.6 API データ

5.17.6.1 sf_touch_ctsu_slider_state_t

sf_touch_ctsu_slider_state_t

詳細説明

スライダの状態

列挙値

名前	説明
SF_TOUCH_CTSU_SLIDER_STATE_NO_CHANGE	スライダはリリースされた状態です
SF_TOUCH_CTSU_SLIDER_STATE_INITIALIZED	スライダは押された状態です
SF_TOUCH_CTSU_SLIDER_STATE_TOUCHED	スライダが押されています
SF_TOUCH_CTSU_SLIDER_STATE_RELEASED	スライダがリリースされています

名前	説明
SF_TOUCH_CTSU_SLIDER_STATE_CLOSED	スライダは無効化されており、イベントを生成しません。
SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH	複数のタッチ要素がタッチされています
SF_TOUCH_CTSU_SLIDER_STATE_DISABLED	スライダの更新が無効化されています。
SF_TOUCH_CTSU_SLIDER_STATE_HELD	スライダがホールドされています。(長押し)

5.17.6.2 sf_slider_type_t

sf_slider_type_t

詳細説明

スライダのタイプの定義

列挙値

名前	説明
SF_SLIDER_TYPE_LINEAR	スライダは線形タイプです
SF_SLIDER_TYPE_CIRCULAR	スライダは円形タイプです (ホイール)

5.17.6.3 sf_touch_ctsu_slider_id_t

typedef uint32_t sf_touch_ctsu_slider_id_t

詳細説明

各スライダに対して使用される一意の識別子

5.17.7 API 構造

5.17.7.1 sf_touch_ctsu_slider_callback_args_t

sf_touch_ctsu_slider_callback_args_t

詳細説明

スライダ コールバック引数の定義

変数

- sf_touch_ctsu_slider_id_t id
  スライダの一意の識別子。
- uint32_t event
  スライダ コールバック イベント。
- uint32_t current_position
  現在のスライダ位置。
- void const * p_context
  ユーザー データのプレースホルダー。

5.17.7.2 sf_touch_ctsu_slider_cfg_t

sf_touch_ctsu_slider_cfg_t

詳細説明

スライダの構成構造体

変数

- sf_touch_ctsu_button_instance_t * p_lower_lvl_touch_framework
  タッチ フレームワーク インスタンスへのポインタ
- uint32_t slider_count
  構成内のスライダの数
- void(* p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)
  コールバック関数
- void const * p_context
  ユーザー データのプレースホルダー
- void const * p_extend
  インスタンス固有の設定値に対応するための拡張パラメータ。

5.17.7.3 sf_touch_ctsu_slider_ctrl_t

sf_touch_ctsu_slider_ctrl_t

詳細説明

このインタフェースの制御構造体

変数

- uint32_t opened

  フレームワークの初期化状態を保存します。

- uint32_t slider_count

  スライダ カウント。

- sf_touch_ctsu_api_t const * p_lower_lvl_api

  ローレベル インスタンスの API 構造体へのポインタ。

- sf_touch_ctsu_ctrl_t lower_lvl_ctrl

  ローレベル インスタンスの制御構造体。

- void const * p_context

  ユーザー データのプレースホルダー。

- void(* p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)

  イベントの発生時に呼び出す関数

5.17.7.4 sf_touch_ctsu_slider_api_t

sf_touch_ctsu_slider_api_t

詳細説明

タッチ CTSU スライダ フレームワーク API 構造体。実装には、以下の API が使用されます。

5.17.7.5 open

ssp_err_t(* sf_touch_ctsu_slider_api_t::open)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_cfg_t const *const p_cfg)

詳細説明

タッチ CTSU スライダ フレームワークを初期化します。またローレベル ハードウェアを構成し、すべてのスライダに対してコールバック関数を登録します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Slider_Open

表 104: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

表 104: パラメータ (続き)

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : sf_touch_ctsu_slider_ctrl_t

このインタフェースの制御構造体

定義 :

定義 : sf_touch_ctsu_slider_cfg_t const *const p_cfg

スライダの構成構造体

- sf_touch_ctsu_slider_cfg_t::sf_touch_ctsu_button_instance_t
  タッチ フレームワーク インスタンスへのポインタ
- sf_touch_ctsu_slider_cfg_t::slider_count
  構成内のスライダの数
- sf_touch_ctsu_slider_cfg_t::p_callback
  コールバック関数
- sf_touch_ctsu_slider_cfg_t::p_context
  ユーザー データのプレースホルダー
- sf_touch_ctsu_slider_cfg_t::p_extend
  インスタンス固有の設定値に対応するための拡張パラメータ。

5.17.7.6 enable

ssp_err_t(* sf_touch_ctsu_slider_api_t::enable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)

詳細説明

構成されたスライダのコールバック通知を有効化します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Slider_Enable

表 105: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

表 105: パラメータ (続き)

名前	方向	説明
slider_id	複数のビットを書き換えることもできます。	スライダの一意の識別子

定義 : sf_touch_ctsu_slider_ctrl_t

このインタフェースの制御構造体

パラメータ slider_id

5.17.7.7 disable

ssp_err_t(* sf_touch_ctsu_slider_api_t::disable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)

詳細説明

構成されたスライダのコールバック通知を無効化します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Slider_Disable

表 106: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
slider_id	複数のビットを書き換えることもできます。	スライダの一意の識別子

定義 : sf_touch_ctsu_slider_ctrl_t

このインタフェースの制御構造体

パラメータ slider_id

5.17.7.8 close

ssp_err_t(* sf_touch_ctsu_slider_api_t::close)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl)

詳細説明

タッチ CTSU スライダ フレームワークを閉じます。他のモジュールによって使用されていない場合は、スライダ フレームワークと低レイヤーを閉じます。また電力消費を低減できます。

- SF_TOUCH_CTSU_Slider_Close

表 107: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : sf_touch_ctsu_slider_ctrl_t

このインタフェースの制御構造体

5.17.7.9 versionGet

ssp_err_t(* sf_touch_ctsu_slider_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- SF_TOUCH_CTSU_Slider_VersionGet

表 108: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

5.17.7.10 sf_touch_ctsu_slider_instance_t

sf_touch_ctsu_slider_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_touch_ctsu_slider_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_touch_ctsu_slider_cfg_t const * p_cfg
  イベント クラスのインスタンス範囲の始点。
- sf_touch_ctsu_slider_api_t const * p_api
  イベント クラスのインスタンス範囲の終点。

5.18 タッチパネルフレームワークインタフェース

RTOS 統合されたタッチ パネル フレームワーク インタフェース。

5.18.1 概要

このモジュールは、タッチ イベントをスキャンし、それをタッチ イベント サブスクライバーに配布するためにメッセージング フレームワークにポストする ThreadX 対応のコンソール フレームワークです。このインタフェースは、I2C タッチパネルフレームワーク によって実装されます。

使用されるインタフェース：

- 外部 IRQ フレームワークインタフェース
- I2C インタフェース
- メッセージングフレームワークインタフェース

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

タッチ パネル フレームワーク インタフェースの説明：タッチ パネル フレームワーク

5.18.2 インタフェース API

sf_touch_panel_api_t

関数名	説明
.open	必要な RTOS オブジェクトを作成し、ハードウェア固有の初期化のためのローレベル モジュールを呼び出し、メッセージ キューにタッチ データをポストするスレッドを作成します。
.calibrate	指定された期待座標に基づいて、較正ルーチンを開始します。許容誤差が期待タッチ ポイントと実測タッチポイントの差よりも大きい場合は、SSP_SUCCESS が返されます （以下の計算式を使用）： p_calibrate->tolerance_pixels ^ 2 > (p_calibrate->x - x_measured) ^ 2 + (p_calibrate->y - y_measured) ^ 2
.start	タッチ イベントのスキャンを開始します。

関数名	説明
.stop	タッチ イベントのスキャンを停止します。
.reset	リセット ピンが提供されている場合は、タッチ コントローラをリセットし、I2C バスをリセットします。
.close	タッチ スレッドを終了し、HAL レイヤーでチャネルを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

5.18.3 データ構造体

- sf_touch_panel_payload_t
- sf_touch_panel_ctrl_t
- sf_touch_panel_cfg_t
- sf_touch_panel_calibrate_t
- sf_touch_panel_instance_t

5.18.4 列挙

- sf_touch_panel_event_t

5.18.5 定義

- #define SF_TOUCH_PANEL_API_VERSION_MAJOR

  初期値 :(1)
- #define SF_TOUCH_PANEL_API_VERSION_MINOR

  初期値 :(1)
- #define SF_TOUCH_PANEL_MESSAGE_WORDS

  初期値 :((sizeof(`sf_touch_panel_payload_t`) + 3) / 4)

  タッチ パネルのメッセージ サイズ （4 バイト ワード単位、端数は切り上げ）。
- #define SF_TOUCH_PANEL_MAX_MESSAGES

  初期値 :(4)

  タッチ パネル メッセージ キュー内のメッセージの最大数。

- #define SF_TOUCH_BYTES_PER_WORD

  初期値 :(4)

  ワードあたりのバイト数を定義するマクロ。

- #define SF_TOUCH_PANEL_MESSAGE_MEM_BYTES

  初期値 :(#define SF_TOUCH_PANEL_MESSAGE_WORDS * #define SF_TOUCH_BYTES_PER_WORD * #define SF_TOUCH_PANEL_MAX_MESSAGES)

  タッチ メッセージ キューのメモリ サイズ。

5.18.6 API データ

5.18.6.1 sf_touch_panel_event_t

sf_touch_panel_event_t

詳細説明

タッチ イベント リスト。

列挙値

名前	説明
SF_TOUCH_PANEL_EVENT_INVALID	タッチ データが無効です。
SF_TOUCH_PANEL_EVENT_HOLD	タッチは、最後のタッチ イベントの後、移動していません。
SF_TOUCH_PANEL_EVENT_MOVE	タッチは、最後のタッチ イベントの後、移動しました。
SF_TOUCH_PANEL_EVENT_DOWN	新しいタッチ イベントがレポートされました。
SF_TOUCH_PANEL_EVENT_UP	タッチが解放されました。
SF_TOUCH_PANEL_EVENT_NONE	有効なタッチ イベントが発生しませんでした。

5.18.7 API 構造

5.18.7.1 sf_touch_panel_payload_t

sf_touch_panel_payload_t

詳細説明

メッセージ キューにポストされたタッチ データ ペイロード。

変数

- sf_message_header_t header

  メッセージング フレームワークの必須のヘッダー。

- int16_t x

  x 座標。

- int16_t y

  Y 座標。

- sf_touch_panel_event_t event_type

  タッチ イベント タイプ。

5.18.7.2 sf_touch_panel_ctrl_t

sf_touch_panel_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

変数

- uint32_t open

  ドライバによって使用され、制御ブロックが有効かどうかを確認します。

- uint16_t hsize_pixels

  画面の幅 （ピクセル単位）

- uint16_t vsize_pixels

  画面の高さ （ピクセル単位)

- sf_message_instance_t const * p_message

  メッセージング フレームワーク制御ブロックへのポインタ。

- uint8_t event_class_instance

  タッチ イベント クラス メッセージ ポスト用のイベント クラス インスタンス番号。

- sf_touch_panel_payload_t * p_payload

  ペイロードの格納に使用されたバッファへのポインタ。

- sf_touch_panel_payload_t last_payload

  比較のために、キューに課された最後のペイロードを格納します。

- TX_MUTEX mutex

  共有リソースへのアクセスを保護するためのミューテックス。

- TX_EVENT_FLAGS_GROUP flags

  内部通信用のイベント フラグ。

- TX_THREAD thread

  メインのタッチ パネル スレッド。

- void * p_lower_lvl_ctrl

  ローレベル制御ブロックへのポインタ。

- uint16_t update_hz

  反復的な （SF_TOUCH_PANEL_EVENT_DOWN または SF_TOUCH_PANEL_EVENT_HOLD）タッチ イベントをレポートする周期 （ヘルツ単位）。この値は、ドライバで RTOS ティックに変換され、近似整数値の RTOS ティックに丸められます。

5.18.7.3 sf_touch_panel_cfg_t

sf_touch_panel_cfg_t

詳細説明

RTOS 統合されたタッチ パネル フレームワークの設定。

変数

- uint16_t hsize_pixels

  画面の幅 （ピクセル単位）

- uint16_t vsize_pixels

  画面の高さ （ピクセル単位）

- UINT priority

  タッチ パネル スレッドの優先順位。

- sf_message_instance_t const * p_message

  メッセージング フレームワーク制御ブロックへのポインタ。

- uint8_t event_class_instance

  タッチ イベント クラス メッセージ ポスト用のイベント クラス インスタンス番号。

- uint16_t update_hz

  反復的な （SF_TOUCH_PANEL_EVENT_DOWN または SF_TOUCH_PANEL_EVENT_HOLD）タッチ イベントをレポートする周期 （ヘルツ単位）。この値は、ドライバで RTOS ティックに変換され、近似整数値の RTOS ティックに丸められます。

- void const * p_extend

  ハードウェアに固有の拡張機能へのポインタ。sf_touch_panel_<instance>.h を参照してください

5.18.7.4 sf_touch_panel_calibrate_t

sf_touch_panel_calibrate_t

詳細説明

SF_TOUCH_PANEL_Calibrate に渡された較正データ。

変数

- uint16_t x

  期待 x 座標。

- uint16_t y

  期待 y 座標。

- uint16_t tolerance_pixels

  期待座標からの許容される直線偏差 （ピクセル単位)。

- void const * p_extend

  ハードウェアに固有の拡張機能へのポインタ。sf_touch_panel_<instance>.h の sf_touch_panel_<instance>_cfg_t を参照してください

5.18.7.5 sf_touch_panel_api_t

sf_touch_panel_api_t

詳細説明

タッチ パネル API 構造体。タッチ パネルの実装には、以下の API が使用されます。

5.18.7.6 open

ssp_err_t(* sf_touch_panel_api_t::open)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_cfg_t const *const p_cfg)

詳細説明

必要な RTOS オブジェクトを作成し、ハードウェア固有の初期化のためのローレベル モジュールを呼び出し、メッセージ キューにタッチ データをポストするスレッドを作成します。また電力消費を低減できます。

- SF_TOUCH_PANEL_I2C_Open

表 109:　パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。この制御構造体は、この関数で初期化されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : `sf_touch_panel_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

定義 :

定義 : `sf_touch_panel_cfg_t` const *const p_cfg

RTOS 統合されたタッチ パネル フレームワークの設定。

- `sf_touch_panel_cfg_t`::hsize_pixels

  画面の幅 （ピクセル単位）

- `sf_touch_panel_cfg_t`::vsize_pixels

  画面の高さ （ピクセル単位）

- `sf_touch_panel_cfg_t`::priority

  タッチ パネル スレッドの優先順位。

- `sf_touch_panel_cfg_t`::`sf_message_instance_t`

  メッセージング フレームワーク制御ブロックへのポインタ。

- `sf_touch_panel_cfg_t`::event_class_instance

  タッチ イベント クラス メッセージ ポスト用のイベント クラス インスタンス番号。

- `sf_touch_panel_cfg_t`::update_hz

  反復的な （SF_TOUCH_PANEL_EVENT_DOWN または SF_TOUCH_PANEL_EVENT_HOLD）タッチ イベントをレポートする周期 （ヘルツ単位）。この値は、ドライバで RTOS ティックに変換され、近似整数値の RTOS ティックに丸められます。

- `sf_touch_panel_cfg_t`::p_extend

  ハードウェアに固有の拡張機能へのポインタ。sf_touch_panel_<instance>.h を参照してください

5.18.7.7 calibrate

ssp_err_t(* sf_touch_panel_api_t::calibrate)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_calibrate_t const *const p_expected, sf_touch_panel_payload_t const *const p_actual, ULONG timeout)

詳細説明

指定された期待座標に基づいて、較正ルーチンを開始します。許容誤差が期待タッチ ポイントと実測タッチポイントの差よりも大きい場合は、SSP_SUCCESS が返されます （以下の計算式を使用）：p_calibrate->tolerance_pixels ^ 2 > (p_calibrate->x - x_measured) ^ 2 + (p_calibrate->y - y_measured) ^ 2 以下として実装されます。

- SF_TOUCH_PANEL_I2C_Calibrate

表 110: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	touch_panel_api_t::open で設定されたハンドル。
p_expected	複数のビットを書き換えることもできます。	期待座標および許容される誤差。
p_actual	複数のビットを書き換えることもできます。	SF_MESSAGE_EVENT_CLASS_TOUCH イベント クラスから受け取ったメッセージ ペイロードへのポインタ。
timeout	複数のビットを書き換えることもできます。	ThreadX タイムアウト。選択できる値は、TX_NO_WAIT、システム クロック カウントで表した値 （1 ～ 0xFFFFFFFF）、または TX_WAIT_FOREVER のいずれかです。

定義 : sf_touch_panel_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

パラメータ p_expected

定義 : sf_touch_panel_calibrate_tconst *const p_expected

SF_TOUCH_PANEL_Calibrate に渡された較正データ。

- sf_touch_panel_calibrate_t::x

  期待 x 座標。

- sf_touch_panel_calibrate_t::y

  期待 y 座標。

- sf_touch_panel_calibrate_t::tolerance_pixels

  期待座標からの許容される直線偏差 （ピクセル単位)。

- sf_touch_panel_calibrate_t::p_extend

  ハードウェアに固有の拡張機能へのポインタ。sf_touch_panel_<instance>.h の sf_touch_panel_<instance>_cfg_t を参照してください

パラメータ p_actual

定義 : sf_touch_panel_payload_tconst *const p_actual

メッセージ キューにポストされたタッチ データ ペイロード。

- sf_touch_panel_payload_t::header

  メッセージング フレームワークの必須のヘッダー。

- sf_touch_panel_payload_t::x

  x 座標。

- sf_touch_panel_payload_t::y

  Y 座標。

- sf_touch_panel_payload_t::event_type

  タッチ イベント タイプ。

パラメータ timeout

const

5.18.7.8 start

ssp_err_t(* sf_touch_panel_api_t::start)(sf_touch_panel_ctrl_t *const p_ctrl)

概要説明

タッチ イベントのスキャンを開始します。

詳細説明

また電力消費を低減できます。

- SF_TOUCH_PANEL_I2C_Start

表 111: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	touch_panel_api_t::open で設定されたハンドル。

定義 : sf_touch_panel_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

5.18.7.9 stop

ssp_err_t(* sf_touch_panel_api_t::stop)(sf_touch_panel_ctrl_t *const p_ctrl)

概要説明

タッチ イベントのスキャンを停止します。

詳細説明

また電力消費を低減できます。

- SF_TOUCH_PANEL_I2C_Stop

表 112: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	touch_panel_api_t::open で設定されたハンドル。

定義 : sf_touch_panel_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

5.18.7.10 reset

ssp_err_t(* sf_touch_panel_api_t::reset)(sf_touch_panel_ctrl_t *const p_ctrl)

概要説明

リセット ピンが提供されている場合は、タッチ コントローラをリセットし、I^2C バスをリセットします。

詳細説明

また電力消費を低減できます。

- SF_TOUCH_PANEL_I2C_Reset

I：較正は含まれていません。リセット後に較正が必要な場合は、この関数の後、アプリケーションから calibrate を使用してください。

表 113: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	touch_panel_api_t::open で設定されたハンドル。

定義 : `sf_touch_panel_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

5.18.7.11 close

ssp_err_t(* `sf_touch_panel_api_t`::close)(`sf_touch_panel_ctrl_t` *const p_ctrl)

概要説明

タッチ スレッドを終了し、HAL レイヤーでチャネルを閉じます。

詳細説明

また電力消費を低減できます。

- SF_TOUCH_PANEL_I2C_Close

表 114: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	touch_panel_api_t::open で設定されたハンドル。

定義 : `sf_touch_panel_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_Open の呼び出し時に実行されます

5.18.7.12 versionGet

ssp_err_t(* sf_touch_panel_api_t::versionGet)(ssp_version_t *const p_version)

概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

詳細説明

また電力消費を低減できます。

- SF_TOUCH_PANEL_I2C_VersionGet

表 115: パラメータ

名前	方向	説明
p_version	out	ここに格納された、使用されているコードおよび API のバージョン。

パラメータ p_version

5.18.7.13 sf_touch_panel_instance_t

sf_touch_panel_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sf_touch_panel_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sf_touch_panel_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sf_touch_panel_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

第 6 章 API リファレンス：フレームワーク レイヤー

フレームワーク レイヤーは、機能のユースケースのための RTOS 対応ドライバを提供します。

- ADC 周期フレームワーク
- オーディオフレームワーク
- DAC オーディオ再生フレームワーク
- I2S オーディオ再生フレームワーク
- BLOCK_MEDIA_SDMMC
- コンソールフレームワーク
- FX_IO フレームワーク
- GUIXTM フレームワーク
- Telnet 通信フレームワーク
- USB 通信フレームワーク
- 外部 IRQ フレームワーク
- I2C フレームワーク
- JPEG フレームワーク
- メッセージングフレームワーク
- パワープロファイルフレームワーク
- SPI フレームワーク
- スレッド監視フレームワーク
- CTSU フレームワーク
- CTSU ボタンフレームワーク
- CTSU スライダフレームワーク
- I2C タッチパネルフレームワーク
- UART フレームワークインスタンス

6.1 ADC 周期フレームワーク

RTOS 統合された ADC フレームワーク。

6.1.1 Functions

- SF_ADC_PERIODIC_Open
- SF_ADC_PERIODIC_Start
- SF_ADC_PERIODIC_Stop
- SF_ADC_PERIODIC_Close
- SF_ADC_PERIODIC_VersionGet

6.1.2 定義

- #define SF_ADC_PERIODIC_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_ADC_PERIODIC_CODE_VERSION_MINOR

  初期値 :(1)

6.1.3 SF_ADC_PERIODIC_Open

```
ssp_err_t  SF_ADC_PERIODIC_Open ( sf_adc_periodic_ctrl_t  *const p_ctrl ,
 sf_adc_periodic_cfg_t  const *const p_cfg )
```

6.1.3.1 概要説明

周期 ADC フレームワークを構成し、オプションとしてタイマを開始します。

6.1.3.2 詳細説明

SF_ADC_PERIODIC_Open 関数は、使用する ADC ユニット用のミューテックスを取得した後、p_api パラメータ内でドライバの .open 関数を呼び出します。ミューテックスは、ドライバ レイヤーのオープン関数の後に解放されます。

表 116: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、HAL ドライバを参照してください。
チャネルは現在動作中です。	ミューテックスが、要求されたユニットで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。
内部エラーが発生しました。	内部 ThreadX エラーが発生しました。これは通常、ミューテックスの作成 / 使用または内部スレッドの作成の失敗を意味します。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数はあらゆるユニットで再入可能です。

6.1.3.3 関数のステップ

- 他のフレームワーク レイヤー関数で使用するためのドライバ構造体ポインタを保存します
- ミューテックスを作成します
- それを変更するために、タイマ構成構造体のローカル コピーを作成します
- コールバックを NULL に設定します
- ADC HAL 構成構造体のローカル インスタンスを宣言します。
- それを変更するために、ADC 構成構造体のローカル コピーを作成します
- HAL コールバック関数に割り込みます
- ISR で使用するフレームワーク モジュールのハンドルに対し、HAL ADC ドライブ コンテキストを設定します
- HW レジスタにアクセスする下位レイヤーを呼び出す前に、ミューテックスを取得します。
- ADC HAL ドライバを初期化します
- ADC HAL チャネル構成を初期化します

- タイマ HAL ドライバを初期化します
- スキャンするすべてのチャネルに ADC スキャン データを転送するように DTC を構成します
- DTC が正常に構成されなかった場合、タイマと ADC インスタンスを閉じます。DTC open() のエラーがユーザーに返されるため、戻り値はチェックされません
- timer open() に失敗した場合、オープンを求める後続の呼び出しが失敗しないよう、ADC HAL ドライバを閉じます。timer open() のエラーがユーザーに返されるため、戻り値はチェックされません
- scanCfg() に失敗した場合、オープンを求める後続の呼び出しが失敗しないよう、ADC HAL ドライバを閉じます。scanCfg() のエラーがユーザーに返されるため、戻り値はチェックされません
- 低レイヤーの初期化のいずれかに失敗した場合は、ミューテックスを返し、これを削除してエラーコードとともに関数を終了します
- ミューテックスをポストします
- ミューテックスを削除します
- エラーを返します
- ミューテックス エラーがある場合はこれを返します
- 制御ブロックをオープンとしてマークし、それが有効であることを他のタスクに示します

6.1.4 SF_ADC_PERIODIC_Start

```
ssp_err_t  SF_ADC_PERIODIC_Start ( sf_adc_periodic_ctrl_t  *const p_ctrl )
```

6.1.4.1 概要説明

ミューテックスを取得し、周期 ADC スキャンを開始した後、ミューテックスを解放します。

6.1.4.2 詳細説明

I a: この関数が呼び出された時間から、ハードウェア タイマカウントが期限切れになり、スキャンがトリガされる時間までは遅延があります。

表 117: 戻り値

名前	説明
SSP_SUCCESS	ADC の周期的なスキャンは正常に開始しました。

表 117: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_ADC_PERIODIC_Open を呼び出して構成します。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません (p_ctrl->p_api->start が NULL)。
内部エラーが発生しました。	内部 ThreadX エラーが発生しました。これは通常、ミューテックスの作成 / 使用または内部スレッドの作成の失敗を意味します。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.1.4.3 関数のステップ

- ミューテックスを取得し、カウンタを開始した後、ミューテックスを解放します
- ADC によるハードウェア トリガの受信を有効化します
- ADC HAL でスキャンが正常に有効化されている場合は、
- タイマを開始して、ADC トリガ イベントを生成します
- ミューテックスを返します。
- ミューテックス エラーがある場合はこれを返します

6.1.5 SF_ADC_PERIODIC_Stop

```
ssp_err_t  SF_ADC_PERIODIC_Stop ( sf_adc_periodic_ctrl_t  *const p_ctrl )
```

6.1.5.1 概要説明

ミューテックスを取得し、周期 ADC スキャンを停止した後、ミューテックスを解放します。

6.1.5.2 詳細説明

表 118: 戻り値

名前	説明
SSP_SUCCESS	周期的な ADC スキャンは正常に停止しました。
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_ADC_PERIODIC_Open を呼び出して構成します。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません (p_ctrl->p_api->stop が NULL)。
内部エラーが発生しました。	内部 ThreadX エラーが発生しました。これは通常、ミューテックスの作成 / 使用または内部スレッドの作成の失敗を意味します。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.1.5.3 関数のステップ

- ミューテックスを取得し、カウンタを停止した後、ミューテックスを解放します
- ミューテックス エラーがある場合はこれを返します

6.1.6 SF_ADC_PERIODIC_Close

```
ssp_err_t SF_ADC_PERIODIC_Close ( sf_adc_periodic_ctrl_t *const p_ctrl )
```

6.1.6.1 概要説明

このクローズ関数は、ユニットのミューテックスを取得し、ドライバのクローズ関数を呼び出した後、ミューテックスを解放します。

6.1.6.2 詳細説明

表 119: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_ADC_PERIODIC_Open を呼び出して構成します。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません(p_ctrl->p_api->close が NULL)。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.1.6.3 関数のステップ

- これは後にハードウェア レジスタにアクセスするので、ミューテックスを取得します
- HAL レイヤー ドライバを閉じます
- 制御ブロックの情報を閉じることによって、このブロックが閉じていることを他の関数に示します
- 使用された RTOS サービスを削除します
- ミューテックス エラーがある場合はこれを返します

6.1.7 SF_ADC_PERIODIC_VersionGet

```
ssp_err_t  SF_ADC_PERIODIC_VersionGet ( ssp_version_t  *const p_version )
```

6.1.7.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.1.7.2 詳細説明

表 120: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.1.8 モジュール

- ビルドタイム構成

6.1.8.1 ビルドタイム構成

定義

- #define SF_ADC_PERIODIC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.2 オーディオフレームワーク

RTOS 統合されたオーディオ フレームワーク。

6.2.1 概要

このモジュールは、以下を実装します : このモジュールは、オーディオフレームワークインタフェース を実装します。

6.2.2 Functions

- SF_AUDIO_PLAYBACK_Open
- SF_AUDIO_PLAYBACK_Close
- SF_AUDIO_PLAYBACK_Start
- SF_AUDIO_PLAYBACK_Pause
- SF_AUDIO_PLAYBACK_Stop
- SF_AUDIO_PLAYBACK_Resume
- SF_AUDIO_PLAYBACK_VolumeSet
- SF_AUDIO_PLAYBACK_VersionGet

6.2.3 定義

- #define SF_AUDIO_PLAYBACK_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_AUDIO_PLAYBACK_CODE_VERSION_MINOR

  初期値 :(1)

6.2.4 SF_AUDIO_PLAYBACK_Open

```
ssp_err_t  SF_AUDIO_PLAYBACK_Open (  sf_audio_playback_ctrl_t *const p_ctrl ,
 sf_audio_playback_cfg_t   const *const p_cfg )
```

6.2.4.1 詳細説明

表 121: 戻り値

名前	説明
SSP_SUCCESS	オーディオ ハードウェアが正常に設定されました。
SSP_ERR_OUT_OF_MEMORY	一度に開くストリームの数は、SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS に制限されています。この数値を超えると、メモリ不足のエラーが発生します。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数は再入可能ではありません。DTC ディスクリプタ ブロックが内部で格納されるようになるまで、オーディオ フレームワークの 1 つのインスタンスのみが許可されます。

6.2.4.2 関数のステップ

- ハードウェアをまだ開いていない場合は開きます。
- バッファ再生の完了時に、再生スレッドを通知するイベント フラグを作成します。
- ストリーム ポインタを共通制御ブロックに格納します。
- オーディオ データのキュー ポインタを格納し、ストリーム オーナー ポインタをクリアします。
- 開いたストリームを他の API が使用できるようにマークします。

6.2.5 SF_AUDIO_PLAYBACK_Close

```
ssp_err_t  SF_AUDIO_PLAYBACK_Close (  sf_audio_playback_ctrl_t *const p_ctrl )
```

6.2.5.1 詳細説明

表 122: 戻り値

名前	説明
SSP_SUCCESS	オーディオ インスタンスが正常に閉じられました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数は再入可能です。

6.2.5.2 関数のステップ

- ハードウェア制御ブロックでストリームを不使用としてマークし、すべてのストリームが閉じられているかどうかを判断します。
- ハードウェア制御ブロックを再構成できるように、不使用としてマークします。
- ローレベル ドライバを閉じます
- 使用された RTOS サービスを削除します
- 制御ブロックを再構成できるように、不使用としてマークします。

6.2.6 SF_AUDIO_PLAYBACK_Start

```
ssp_err_t  SF_AUDIO_PLAYBACK_Start (  sf_audio_playback_ctrl_t *const p_ctrl ,
 sf_audio_playback_data_t  *const p_data ,  UINT const timeout )
```

6.2.6.1 詳細説明

表 123: 戻り値

名前	説明
SSP_SUCCESS	バッファが正常に再生されました
SSP_ERR_TIMEOUT	再生が終了する前にタイムアウトが発生しました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.2.6.2 関数のステップ

- 現在のストリーム オーナーからのみオーディオ データがポストされることを確認します。
- ストリームに対してオーナーが指定されていない場合は、新しいストリーム オーナーを格納します。
- メッセージ ヘッダーをオーディオ開始イベントに設定します。インスタンスをストリーム インスタンスに設定します。
- オーディオ データ付きのメッセージをオーディオ スレッドに送信します。

6.2.7 SF_AUDIO_PLAYBACK_Pause

```
ssp_err_t  SF_AUDIO_PLAYBACK_Pause (  sf_audio_playback_ctrl_t *const p_ctrl )
```

6.2.7.1 詳細説明

表 124:　戻り値

名前	説明
SSP_SUCCESS	オーディオ再生が一時停止されました。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数は再入可能です。

6.2.8 SF_AUDIO_PLAYBACK_Stop

```
ssp_err_t  SF_AUDIO_PLAYBACK_Stop (  sf_audio_playback_ctrl_t *const p_ctrl )
```

6.2.8.1 詳細説明

表 125: 戻り値

名前	説明
SSP_SUCCESS	オーディオ再生が停止しました。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数は再入可能です。

6.2.9 SF_AUDIO_PLAYBACK_Resume

ssp_err_t SF_AUDIO_PLAYBACK_Resume (sf_audio_playback_ctrl_t *const p_ctrl)

6.2.9.1 詳細説明

表 126: 戻り値

名前	説明
SSP_SUCCESS	オーディオ再生が再開されました。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数は再入可能です。

6.2.10 SF_AUDIO_PLAYBACK_VolumeSet

ssp_err_t SF_AUDIO_PLAYBACK_VolumeSet (sf_audio_playback_ctrl_t *const p_ctrl ,
uint8_t volume)

6.2.10.1 概要説明

ソフトウェア音量制御を設定します。

6.2.10.2 詳細説明

I a: ソフトウェア音量制御を使用すると、解像度が低下します。また余分のメモリや処理帯域幅が必要になる可能性があります。

表 127: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	オーディオ ドライバの Open 呼び出しで初期化されたデバイス制御ブロックへのポインタ。
volume	複数のビットを書き換えることもできます。	要求された音量レベル。有効な範囲は、0 （消音、再生を停止）～ 255 （最大音量、開いたときのデフォルト値）。

6.2.10.3 関数のステップ

- 制御ブロックで音量を更新します。

6.2.11 SF_AUDIO_PLAYBACK_VersionGet

```
ssp_err_t  SF_AUDIO_PLAYBACK_VersionGet ( ssp_version_t  *const p_version )
```

6.2.11.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.2.11.2 詳細説明

表 128: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.2.12 モジュール

- ビルドタイム構成

6.2.12.1 ビルドタイム構成

定義

- #define SF_AUDIO_PLAYBACK_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES

  初期値 :(256)

  処理済みのデータのみが供給される場合は、この値を 1 に設定して制御ブロックの容量を節減します。各制御ブロックには 2 バッファが割り当てられるので、ここで 256 と設定した場合、合計 512 バイトが制御ブロックあたりのバッファとして予約されることになります。

- #define SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS

  初期値 :(2)

  システ内で一度にミキシングできる最大ストリーム数を指定します。ミキシングを無効にする場合は、1 に設定します。

6.3 DAC オーディオ再生フレームワーク

オーディオ再生インタフェースの RTOS 統合された DAC 実装。

オーディオ再生フレームワークの DAC 実装では、タイマを使用してサンプリング周波数でイベントを生成し、これらのイベントを使用して PCM サンプルを DAC に転送します。

6.3.1 Functions

- SF_AUDIO_PLAYBACK_HW_DAC_Open
- SF_AUDIO_PLAYBACK_HW_DAC_Start
- SF_AUDIO_PLAYBACK_HW_DAC_Stop
- SF_AUDIO_PLAYBACK_HW_DAC_Play
- SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet
- SF_AUDIO_PLAYBACK_HW_DAC_Close
- SF_AUDIO_PLAYBACK_HW_DAC_VersionGet

6.3.2 変数

- g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

6.3.3 定義

- #define SF_AUDIO_PLAYBACK_HW_DAC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_AUDIO_PLAYBACK_HW_DAC_CODE_VERSION_MINOR

  初期値 :(1)

6.3.4 SF_AUDIO_PLAYBACK_HW_DAC_Open

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Open ( sf_audio_playback_hw_ctrl_t *const p_ctrl ,
 sf_audio_playback_hw_cfg_t const *const p_cfg )
```

6.3.4.1 詳細説明

DAC HAL ドライバやヘルパー タイマなどの DAC オーディオ ドライバを開き、HAL ドライバを転送します。

表 129: 戻り値

名前	説明
SSP_SUCCESS	ローレベル ドライバの構成が正常に完了しました。
p_ctrl の可能性があります。	Null ポインタ。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.3.4.2 関数のステップ

- 選択された周波数でタイマ ドライバを開きます
- DTC が選択されている場合、オーディオ コールバックをタイマ ISR とともに登録します。転送が完了すると、DTC がアクティベーション ソース ISR を呼び出します。
- DAC モジュールを開きます
- 転送モジュールを開いて、バッファから DAC 出力レジスタに転送します
- 転送ドライバを開きます
- ドライバ データを格納します。
- 線形ランプ データを再生して、最大出力の半分までの DAC を取得します。

6.3.5 SF_AUDIO_PLAYBACK_HW_DAC_Start

```
ssp_err_t  SF_AUDIO_PLAYBACK_HW_DAC_Start ( sf_audio_playback_hw_ctrl_t  *const p_ctrl )
```

6.3.5.1 詳細説明

DAC とタイマ HAL ドライバを開始します。

表 130: 戻り値

名前	説明
SSP_SUCCESS	オーディオ再生ハードウェアが正常に開始されました。

表 130: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.3.5.2 関数のステップ

- DAC を開始します。
- タイマを開始します。

6.3.6 SF_AUDIO_PLAYBACK_HW_DAC_Stop

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Stop ( sf_audio_playback_hw_ctrl_t *const p_ctrl )
```

6.3.6.1 詳細説明

DAC とタイマ HAL ドライバを停止します。

表 131: 戻り値

名前	説明
SSP_SUCCESS	オーディオ再生ハードウェアが正常に停止しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.3.6.2 関数のステップ

- タイマを停止します。

- DAC を停止します。

6.3.7 SF_AUDIO_PLAYBACK_HW_DAC_Play

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Play (sf_audio_playback_hw_ctrl_t *const p_ctrl , int16_t const *const p_buffer , uint32_t length)

6.3.7.1 詳細説明

タイマによって構成されたサンプリング周波数で、入力サンプルによって単一のオーディオ バッファを DAC に再生します。

表 132: 戻り値

名前	説明
SSP_SUCCESS	バッファ再生が正常に開始されました。
p_ctrl の可能性があります。	パラメータ p_ctrl または p_buffer が NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.3.7.2 関数のステップ

- 転送をリセットします。

6.3.8 SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet (sf_audio_playback_hw_ctrl_t *const p_ctrl , sf_audio_playback_data_type_t *const p_data_type)

6.3.8.1 詳細説明

ポインタ p_data_type に、指定されたデータ型を提供します。

表 133: 戻り値

名前	説明
SSP_SUCCESS	データ型を p_data_type に格納しました。

表 133: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	パラメータ p_ctrl または p_data_type が NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.3.8.2 関数のステップ

- データ型を格納します。Synergy DAC は、12 ビット符号なしデータのみをサポートします。

6.3.9 SF_AUDIO_PLAYBACK_HW_DAC_Close

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Close ( sf_audio_playback_hw_ctrl_t *const p_ctrl )
```

6.3.9.1 詳細説明

開かれているオーディオ ドライバを閉じます。

表 134: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

6.3.9.2 関数のステップ

- タイマ ドライバを閉じます。

- DAC ドライバを閉じます。
- 転送ドライバを閉じます。

6.3.10 SF_AUDIO_PLAYBACK_HW_DAC_VersionGet

ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_VersionGet (ssp_version_t *const p_version)

6.3.10.1 詳細説明

ファームウェアと API のバージョンを指定されたポインタ p_version に格納します。

表 135: 戻り値

名前	説明
p_ctrl の可能性があります。	p_version パラメータは NULL です。
SSP_SUCCESS	モジュール バージョン情報を正常に p_version に格納しました。

I：この関数は再入可能です。

6.3.11 g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

sf_audio_playback_hw_api_t::g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

6.3.11.1 詳細説明

オーディオ再生 API の DAC 実装のための関数ポインタ。

6.3.12 拡張

6.3.12.1 sf_audio_playback_hw_dac_ctrl_t

sf_audio_playback_hw_dac_ctrl_t

詳細説明

DAC オーディオ ドライバの、ハードウェアに依存する制御ブロック。

変数

- dac_instance_t const * p_lower_lvl_dac

  DAC ハードウェアへのアクセスに使用される DAC API。
- timer_instance_t const * p_lower_lvl_timer

  サンプリング周波数の生成に使用されるタイマ API。
- transfer_instance_t const * p_lower_lvl_transfer

  各サンプリング周波数でデータを転送するために使用される転送 API。

6.3.12.2 sf_audio_playback_hw_dac_cfg_t

sf_audio_playback_hw_dac_cfg_t

詳細説明

DAC オーディオ ドライバの、ハードウェアに依存する構成。

変数

- dac_instance_t const * p_lower_lvl_dac

  DAC ハードウェアへのアクセスに使用される DAC API。
- timer_instance_t const * p_lower_lvl_timer

  サンプリング周波数の生成に使用されるタイマ API。
- transfer_instance_t const * p_lower_lvl_transfer

  各サンプリング周波数でデータを転送するために使用される転送 API。
- sf_audio_playback_hw_dac_ctrl_t * p_ctrl

  ハードウェア制御ブロックに割り当てられたメモリへのポインタ。初期化は必要ありません。

6.3.13 モジュール

- ビルドタイム構成

6.3.13.1 ビルドタイム構成

定義

- #define SF_AUDIO_PLAYBACK_HW_DAC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_AUDIO_PLAYBACK_HW_DAC_CFG_DMAC_SUPPORT_ENABLE

  初期値 :(0)

  転送インタフェースの DMAC 実装をサポートするかどうか指定します。有効な設定には次が含まれます。1 : DMAC 実装をサポート、0 : DMAC サポートをコンパイル アウト

6.4 I2S オーディオ再生フレームワーク

オーディオ再生インタフェースの RTOS 統合された I2S 実装。

オーディオ再生フレームワーク I2S 実装は、オーディオ再生用の I2S インタフェースを使用します。

6.4.1 Functions

- SF_AUDIO_PLAYBACK_HW_I2S_Open
- SF_AUDIO_PLAYBACK_HW_I2S_Start
- SF_AUDIO_PLAYBACK_HW_I2S_Stop
- SF_AUDIO_PLAYBACK_HW_I2S_Play
- SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet
- SF_AUDIO_PLAYBACK_HW_I2S_Close
- SF_AUDIO_PLAYBACK_HW_I2S_VersionGet

6.4.2 定義

- #define SF_AUDIO_PLAYBACK_HW_I2S_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_AUDIO_PLAYBACK_HW_I2S_CODE_VERSION_MINOR

  初期値 :(1)

6.4.3 SF_AUDIO_PLAYBACK_HW_I2S_Open

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Open ( sf_audio_playback_hw_ctrl_t *const p_ctrl ,
 sf_audio_playback_hw_cfg_t const *const p_cfg )
```

6.4.3.1 詳細説明

I2S HAL ドライバやヘルパー タイマなどの I2S オーディオ ドライバを開き、HAL ドライバを転送します。

表 136: 戻り値

名前	説明
SSP_SUCCESS	ローレベル ドライバの構成が正常に完了しました。

表 136: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	Null ポインタ。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.4.3.2 関数のステップ

- I2S モジュールを開きます
- ドライバ データを格納します。

6.4.4 SF_AUDIO_PLAYBACK_HW_I2S_Start

```
ssp_err_t  SF_AUDIO_PLAYBACK_HW_I2S_Start ( sf_audio_playback_hw_ctrl_t  *const p_ctrl )
```

6.4.4.1 詳細説明

I2S とタイマ HAL ドライバを開始します。

表 137: 戻り値

名前	説明
SSP_SUCCESS	オーディオ再生ハードウェアが正常に開始されました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.4.4.2 関数のステップ

- この API は使用されません。I2S は、SF_AUDIO_PLAYBACK_HW_I2S_Play から書き込みが呼び出された場合に開始します。

6.4.5 SF_AUDIO_PLAYBACK_HW_I2S_Stop

```
ssp_err_t  SF_AUDIO_PLAYBACK_HW_I2S_Stop ( sf_audio_playback_hw_ctrl_t  *const p_ctrl )
```

6.4.5.1 詳細説明

I2S とタイマ HAL ドライバを停止します。

表 138:　戻り値

名前	説明
SSP_SUCCESS	オーディオ再生ハードウェアが正常に停止しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.4.5.2 関数のステップ

- I2S を停止します。

6.4.6 SF_AUDIO_PLAYBACK_HW_I2S_Play

```
ssp_err_t  SF_AUDIO_PLAYBACK_HW_I2S_Play ( sf_audio_playback_hw_ctrl_t  *const p_ctrl ,
 int16_t const *const p_buffer ,  uint32_t length )
```

6.4.6.1 詳細説明

タイマによって構成されたサンプリング周波数で、入力サンプルによって単一のオーディオ バッファを I2S に再生します。

表 139: 戻り値

名前	説明
SSP_SUCCESS	バッファ再生が正常に開始されました。
p_ctrl の可能性があります。	パラメータ p_ctrl または p_buffer が NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.4.6.2 関数のステップ

- 転送をリセットします。

6.4.7 SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet

```
ssp_err_t  SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet ( sf_audio_playback_hw_ctrl_t
*const p_ctrl ,  sf_audio_playback_data_type_t  *const p_data_type )
```

6.4.7.1 詳細説明

ポインタ p_data_type に、指定されたデータ型を提供します。

表 140: 戻り値

名前	説明
SSP_SUCCESS	データ型を p_data_type に格納しました。
p_ctrl の可能性があります。	パラメータ p_ctrl または p_data_type が NULL です。

I：ローレベル ドライバ関数が再入可能な場合、この関数は再入可能です。

6.4.7.2 関数のステップ

- データ型を格納します。オーディオ フレームワークは 16 ビット符号付きデータのみをサポートします。

6.4.8 SF_AUDIO_PLAYBACK_HW_I2S_Close

`ssp_err_t` SF_AUDIO_PLAYBACK_HW_I2S_Close (`sf_audio_playback_hw_ctrl_t` *const p_ctrl)

6.4.8.1 詳細説明

開かれているオーディオ ドライバを閉じます。

表 141: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

6.4.8.2 関数のステップ

- I2S ドライバを閉じます。

6.4.9 SF_AUDIO_PLAYBACK_HW_I2S_VersionGet

`ssp_err_t` SF_AUDIO_PLAYBACK_HW_I2S_VersionGet (`ssp_version_t` *const p_version)

6.4.9.1 詳細説明

ファームウェアと API のバージョンを指定されたポインタ p_version に格納します。

表 142: 戻り値

名前	説明
p_ctrl の可能性があります。	p_version パラメータは NULL です。
SSP_SUCCESS	モジュール バージョン情報を正常に p_version に格納しました。

I：この関数は再入可能です。

6.4.10 拡張

6.4.10.1 sf_audio_playback_hw_i2s_ctrl_t

sf_audio_playback_hw_i2s_ctrl_t

詳細説明

I2S オーディオ ドライバの、ハードウェアに依存する制御ブロック。

変数

- i2s_instance_t const * p_lower_lvl_i2s

  I2S ハードウェアへのアクセスに使用される I2S API。

6.4.10.2 sf_audio_playback_hw_i2s_cfg_t

sf_audio_playback_hw_i2s_cfg_t

詳細説明

I2S オーディオ ドライバの、ハードウェアに依存する構成。

変数

- i2s_instance_t const * p_lower_lvl_i2s

  I2S ハードウェアへのアクセスに使用される I2S API。

- sf_audio_playback_hw_i2s_ctrl_t * p_ctrl

  ハードウェア制御ブロックに割り当てられたメモリへのポインタ。初期化は必要ありません。

6.4.11 モジュール

- ビルドタイム構成

6.4.11.1 ビルドタイム構成

定義

- #define SF_AUDIO_PLAYBACK_HW_I2S_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define  BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_AUDIO_PLAYBACK_HW_I2S_CFG_DMAC_SUPPORT_ENABLE

  初期値 :(0)

  転送インタフェースの DMAC 実装をサポートするかどうか指定します。有効な設定には次が含まれます。1 : DMAC 実装をサポート、0 : DMAC サポートをコンパイル アウト

6.5 BLOCK_MEDIA_SDMMC

SDMMC ドライバ用の RTOS 統合されたブロック メディア フレームワーク。

6.5.1 Functions

- SF_Block_Media_SDMMC_Open
- SF_Block_Media_SDMMC_Read
- SF_Block_Media_SDMMC_Write
- SF_Block_Media_SDMMC_Control
- SF_Block_Media_SDMMC_Close
- SF_Block_Media_SDMMC_VersionGet

6.5.2 定義

- #define BLOCK_MEDIA_SDMMC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define BLOCK_MEDIA_SDMMC_CODE_VERSION_MINOR

  初期値 :(1)

6.5.3 SF_Block_Media_SDMMC_Open

```
ssp_err_t SF_Block_Media_SDMMC_Open ( sf_block_media_ctrl_t *const p_ctrl ,
 sf_block_media_cfg_t const *const p_cfg )
```

6.5.3.1 概要説明

読み書きおよび制御のためにデバイスを開きます。

6.5.3.2 詳細説明

読み書きおよび制御のために SD または MMC デバイス ポートを開きます。この関数は、リセット以降に初めて呼び出されたときに SDMMC ドライバとハードウェアを初期化します。

表 143: 戻り値

名前	説明
SSP_SUCCESS	ポートが使用可能で、読み取り、書き込み、および制御アクセス用に開いています。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	1 つまたは複数の構成オプションが無効です。
チャネルは現在動作中です。	指定したチャネルはすでに開かれています。構成は変更されていません。関連する Close 関数を呼び出すか、関連する Control コマンドを使用してチャネルを再構成します。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

6.5.3.3 関数のステップ

- SDMMC イベント フラグを作成し、コンテキストに挿入します

6.5.4 SF_Block_Media_SDMMC_Read

`ssp_err_t` SF_Block_Media_SDMMC_Read (`sf_block_media_ctrl_t` *const p_ctrl , uint8_t *const p_dest , uint32_t const start_block , uint32_t const block_count)

6.5.4.1 概要説明

SD/MMC からデータを読み取ります。

6.5.4.2 詳細説明

SD または MMC デバイス ポートからデータを読み取ります。

表 144: 戻り値

名前	説明
SSP_SUCCESS	正常にデータが読み取られました。
p_ctrl の可能性があります。	NULL ポインタ。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

6.5.4.3 関数のステップ

- リード動作が完了するまで待機します。イベントは、イベント フラグ オブジェクトによって通知されます。

6.5.5 SF_Block_Media_SDMMC_Write

```
ssp_err_t SF_Block_Media_SDMMC_Write ( sf_block_media_ctrl_t *const p_ctrl , uint8_t const *const p_src , uint32_t const start_block , uint32_t const block_count )
```

6.5.5.1 概要説明

SDMMC チャネルにデータを書き込みます。

6.5.5.2 詳細説明

表 145: 戻り値

名前	説明
SSP_SUCCESS	カードの書き込みが正常に終了しました。
p_ctrl の可能性があります。	ハンドルまたはソース アドレスが NULL です。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。

表 145: 戻り値 (続き)

名前	説明
SSP_ERR_WRITE_PROTECTED	SD または MMC カードが書き込み保護されています。

I：この関数は別のチャネルに対して再入可能です。

6.5.5.3 関数のステップ

- 書き込み操作が完了するまで待機します。イベントは、イベント フラグ オブジェクトによって通知されます。

6.5.6 SF_Block_Media_SDMMC_Control

```
ssp_err_t  SF_Block_Media_SDMMC_Control ( sf_block_media_ctrl_t  *const p_ctrl ,
 ssp_command_t  const command ,  void * p_extend )
```

6.5.6.1 概要説明

制御コマンドを送信して、SD/MMC ポートのステータスを受信します。

6.5.6.2 詳細説明

SD/MMC ポートに制御コマンドを送信して、SD/MMC ポートのステータスを受信します。

表 146: 戻り値

名前	説明
SSP_SUCCESS	コマンドが正常に実行されました。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	コマンドが無効です。
SF_INFO_NOT_AVAILABLE	カードが取り外されているか、故障しているために情報が入手できない可能性があります。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

6.5.7 SF_Block_Media_SDMMC_Close

ssp_err_t SF_Block_Media_SDMMC_Close (sf_block_media_ctrl_t *const p_ctrl)

6.5.7.1 概要説明

開いているデバイス ポートを閉じます。

6.5.7.2 詳細説明

開いている SD/MMC デバイス ポートを閉じます。

表 147: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	パラメータ p_cfg が NULL です。
SSP_ERR_INVALID_POINTER	不正なチャネル番号。
タッチ パネルが設定されていません。	チャネルは開かれていません。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

6.5.8 SF_Block_Media_SDMMC_VersionGet

ssp_err_t SF_Block_Media_SDMMC_VersionGet (ssp_version_t *const p_version)

6.5.8.1 概要説明

ブロック メディア SD/MMC ドライバのバージョンを取得します。

6.5.8.2 詳細説明

フォームウェアと API のバージョンを返します。

表 148: 戻り値

名前	説明
p_ctrl の可能性があります。	Null ポインタ。
SSP_SUCCESS	関数が正常に実行されました。

I：この関数は再入可能です。

6.5.9 拡張

6.5.9.1 sf_block_media_on_sdmmc_cfg_t

sf_block_media_on_sdmmc_cfg_t

詳細説明

変数

- sdmmc_instance_t const *const p_lower_lvl_sdmmc

  SDMMC インスタンス構造体へのポインタ。

6.5.9.2 sf_block_media_on_sdmmc_ctrl_t

sf_block_media_on_sdmmc_ctrl_t

詳細説明

変数

- sdmmc_instance_t * p_lower_lvl_sdmmc

  SDMMC インスタンス構造体へのポインタ。

- TX_EVENT_FLAGS_GROUP eventflag

  SDMMC データ転送用のイベント フラグ オブジェクトへのポインタ。

6.5.10 モジュール

- ビルドタイム構成

6.5.10.1 ビルドタイム構成

定義

- #define BLOCK_MEDIA_SDMMC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.6 コンソールフレームワーク

RTOS 統合されたコンソール フレームワーク。

これは、SSP 通信フレームワークを使って実装された ThreadX 認識コンソール フレームワークです。

6.6.1 Functions

- SF_CONSOLE_Open
- SF_CONSOLE_Close
- SF_CONSOLE_Parse
- SF_CONSOLE_Prompt
- SF_CONSOLE_Read
- SF_CONSOLE_Write
- SF_CONSOLE_ArgumentFind
- SF_CONSOLE_CallbackNextMenu
- SF_CONSOLE_VersionGet

6.6.2 定義

- #define SF_CONSOLE_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_CONSOLE_CODE_VERSION_MINOR

  初期値 :(1)

6.6.3 SF_CONSOLE_Open

`ssp_err_t` SF_CONSOLE_Open (`sf_console_ctrl_t` *const p_ctrl , `sf_console_cfg_t` const *const p_cfg)

6.6.3.1 詳細説明

表 149: 戻り値

名前	説明
SSP_SUCCESS	コンソールチャネルが正常に開かれました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.3.2 関数のステップ

- UART ドライバを開きます
- コントロールブロックにエコー構成および初期メニューを格納します
- 入力自動スタートのプロンプトが true です

6.6.4 SF_CONSOLE_Close

`ssp_err_t` SF_CONSOLE_Close (`sf_console_ctrl_t` *const p_ctrl)

6.6.4.1 詳細説明

表 150: 戻り値

名前	説明
SSP_SUCCESS	コンソールが正常に閉じられました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.4.2 関数のステップ

- UART ドライバを閉じます

6.6.5 SF_CONSOLE_Parse

ssp_err_t SF_CONSOLE_Parse (sf_console_ctrl_t *const p_ctrl , sf_console_menu_t const *const p_menu , uint8_t const *const p_input , uint32_t const bytes)

6.6.5.1 詳細説明

表 151: 戻り値

名前	説明
SSP_SUCCESS	データが正常に解析され、コマンドが見つかり、コールバックが呼び出されました。
データ構造体が割り当てられませんでした。	コマンドが現在のメニューで見つかりませんでした。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.5.2 関数のステップ

- ヘルプコマンドを入力した場合はヘルプメニューが印刷されます
- 前のメニューコマンドを入力した場合は 1 つ前のメニューに戻ります。
- ルートメニューコマンドを入力した場合はルートメニューに戻ります。
- 一致するコマンドを探し、見つかった場合はコールバックを呼び出します。

6.6.6 SF_CONSOLE_Prompt

ssp_err_t SF_CONSOLE_Prompt (sf_console_ctrl_t *const p_ctrl , sf_console_menu_t const *const p_menu , UINT const timeout)

6.6.6.1 詳細説明

表 152: 戻り値

名前	説明
SSP_SUCCESS	有効なコマンドを受け取り、コールバックを呼び出しました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.6.2 関数のステップ

- 新しいポインタを指定した場合、保存されている現在のメニューポインタを更新します。
- プリント メニュー名の後に「>」を表示して、ユーザー入力をプロンプトします。
- コマンドが完了するまで専用アクセスを確保するために、コンソール UART フレームワークをロックします。
  I：他のスレッドでデバッグメッセージをプリントできるように、伝送はロックされません。

  入力を待ちます
- 入力およびコール関連ユーザコールバックを解析します。
- コマンドが完了したので、通信の受信をアンロックします。

6.6.7 SF_CONSOLE_Read

ssp_err_t SF_CONSOLE_Read (sf_console_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const bytes , uint32_t const timeout)

6.6.7.1 詳細説明

表 153: 戻り値

名前	説明
SSP_SUCCESS	データの読み取りが正常に完了しました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.7.2 関数のステップ

- キャリッジリターンを受け取るまで、コンソール UART フレームワークをロックします。
- 1 バイトごとに読み取り、キャリッジリターン、バックスペース、削除、エスケープコードがないかどうか確認します。

6.6.8 SF_CONSOLE_Write

```
ssp_err_t  SF_CONSOLE_Write ( sf_console_ctrl_t  *const p_ctrl ,  uint8_t const *const p_src ,
 uint32_t const timeout )
```

6.6.8.1 詳細説明

表 154: 戻り値

名前	説明
SSP_SUCCESS	データの書き込みが正常に完了しました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.8.2 関数のステップ

- NULL 停止文字列を書き込みます。長さを計算します。上限を超えていない場合、文字列全体をコンソールに書き込みます。

6.6.9 SF_CONSOLE_ArgumentFind

ssp_err_t SF_CONSOLE_ArgumentFind (uint8_t const *const p_arg , uint8_t const *const p_str , int32_t *const p_index , int32_t *const p_data)

6.6.9.1 詳細説明

表 155: 戻り値

名前	説明
SSP_SUCCESS	引数が正常に見つかりました

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.6.9.2 関数のステップ

- 単語の冒頭で最初の文字の一致を検索します。
- 入力文字列と入力引数が一致する場合、引数の直後にある文字のインデックスを p_index に保存し、インデックスのデータを p_data に保存します。その後、リターンします。

6.6.10 SF_CONSOLE_CallbackNextMenu

SF_CONSOLE_CallbackNextMenu (sf_console_callback_args_t * p_args)

6.6.10.1 概要説明

次のメニューを下に解析するためにコールバックが提供されています。

6.6.10.2 関数のステップ

- 次のレベルのメニューがユーザコンテキストパラメータとして受け渡されます
- 現在のメニューを更新します。
- 残りの文字列で次のメニューコマンドが入力であったかどうか確認します。

6.6.11 SF_CONSOLE_VersionGet

```
ssp_err_t  SF_CONSOLE_VersionGet ( ssp_version_t  *const p_version )
```

6.6.11.1 概要説明

コンソールバージョの取得関数。

6.6.11.2 詳細説明

表 156:　戻り値

名前	説明
SSP_SUCCESS	提供されたポインタに保管されたバージョンです。
p_ctrl の可能性があります。	p_version が NULL でした。

6.6.11.3 関数のステップ

- バージョンポインタをセットします

6.7 FX_IO フレームワーク

ブロック メディア デバイス ドライバの FileX 適応レイヤー。

SF_EL_FX FileX I/O は、FileX を Renesas Synergy Block Media デバイス ドライバに適応させる単一エントリ関数です。

6.7.1 概要

SF_EL_FX には API ファイルは含まれていません。

6.7.2 Functions

- SF_EL_FX_BlockDriver

6.7.3 定義

- #define SF_EL_FX_API_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義された API のバージョン
- #define SF_EL_FX_API_VERSION_MINOR

  初期値 :(0)
- #define SF_EL_FX_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_EL_FX_CODE_VERSION_MINOR

  初期値 :(1)

6.7.4 SF_EL_FX_BlockDriver

SF_EL_FX_BlockDriver (FX_MEDIA * media_ptr)

6.7.4.1 概要説明

Block Media デバイスをオープン、クローズ、読み取り、書き込み、および制御するための各関数にアクセスします。

6.7.4.2 詳細説明

SSP FileX をサポートします。このファイル システムを使用するには、ディレクトリやファイルを作成する前にメディアをフォーマットする必要があります。セクター サイズおよびセクター数は、メディア タイプとメディア数によって異なります。

メディア内では、予約セクターの後にファイル アロケーション テーブル （FAT）が続きます。FAT 領域は、基本的には 12 ビット、16 ビット、または 32 ビットのエントリの配列であり、そのクラスタが割り当てられているか、またサブディレクトリやファイルを構成するクラスタ チェーンの一部であるかどうかが記録されています。各 FAT エントリのサイズは、記録するクラスタの数によって決まります。クラスタの数（合計セクター数を、クラスタあたりのセクター数で割った値）が 4,086 未満の場合は、12 ビット FAT エントリが使用されます。合計クラスタ数が 4,086 より大きく、65,525 以下の場合は、16 ビット FAT エントリが使用されます。それ以外の、合計クラスタ数が 65,525 より大きい場合は、32 ビット FAT エントリが使用されます。R_FX_BlockDriver 関数は、FileX ファイル システム ドライバから呼び出され、Synergy Block Media インタフェースを介して Block Media デバイスに要求を発行します。

表 157: パラメータ

名前	方向	説明
p_fx_media	入力 / 出力	FileX メディア制御ブロック。開いている各メディア デバイスに関するすべての情報は、FX_MEDIA データ型で維持されます。
p_fx_media->fx_media_driver_status	out	I/O ドライバは、FX_MEDIA の fx_media_driver_status メンバーを介して、要求の成功または失敗を伝達します。FX_SUCCESS は、要求が正常に実行された場合に返されます。FX_MEDIA_INVALID が返された場合は、ブート レコードが無効であるか、セクター サイズが無効です。FX_IO_ERROR は、要求が無効であることを示します。

表 158: 戻り値

名前	説明
noneUses	アクセス用の Block Media ドライバ。

6.7.4.3 関数のステップ

- エラーに対し、FileX I/O の状態を初期化します。操作に失敗した場合を除き、FX_SUCCESS に変更されます。

- メディア制御ブロックで指定されたドライバ要求を処理します。
- FileX は、I/O ドライバに読み取り要求を発行して、1 つ以上のセクターをメモリに読み込みます。
- FileX は、I/O ドライバに書き込み要求を発行して、物理メディアに 1 つ以上のセクターを書き込みます。
- FileX は、I/O ドライバにフラッシュ要求を発行して、現在ドライバのセクター キャッシュにあるすべてのセクターを物理メディアにフラッシュします。Synergy ドライバは、現時点では、セクターをキャッシュしません。
- FileX は、I/O ドライバに中止要求を発行して、物理メディアとのそれ以降の物理的な I/O アクティビティを中止するようにドライバに通知します。 ドライバは、再度初期化されるまで、I/O を再実行してはなりません。
- ドライバの実際の初期化処理はアプリケーションによって異なりますが、通常はデータ構造の初期化に加え、場合によっては、一部の予備的なハードウェアの初期化が行われます。この要求はまず FileX によって行われ、fx_media_open サービスの中から実行されます。メディアの書き込み保護が検出された場合、FX_MEDIA のドライバ fx_media_driver_write_protect メンバーを FX_TRUE に設定する必要があります。
- FileX では、uninit コマンドを使用してメディアを閉じます。
- FileX でメディアのブート セクターを読み取るには、標準の読み取りコマンドを使用するのではなく、特定の要求を実行します。
- FileX でメディアのブート セクターに書き込むには、標準の書き込みコマンドを使用するのではなく、特定の要求を実行します。

6.7.5 拡張

6.7.5.1 sf_el_fx_t

sf_el_fx_t

詳細説明

Block Media 制御ブロック タイプ

変数

- sf_block_media_instance_t * p_lower_lvl_block_media

  ローレベル ブロック メディア ポインタ。

- void * p_extend

6.7.6 モジュール

- ビルドタイム構成

6.7.6.1 ビルドタイム構成

定義

- #define SF_EL_FX_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.8 GUIX™ フレームワーク

GUIX™ 適応レイヤー。

6.8.1 Functions

- SF_EL_GX_Open
- SF_EL_GX_Close
- SF_EL_GX_VersionGet
- SF_EL_GX_Setup
- SF_EL_GX_CanvasInit

6.8.2 SF_EL_GX_Open

```
ssp_err_t SF_EL_GX_Open ( sf_el_gx_ctrl_t *const p_ctrl , sf_el_gx_cfg_t const
*const p_cfg )
```

6.8.2.1 概要説明

Synergy 対応の GUIX™ 適用フレームワークであるドライバ構成。この関数は、以下を呼び出します。

6.8.2.2 詳細説明

- tx_mutex_create コンテキスト更新中にドライバをロックするためのミューテックスを作成します。
- tx_mutex_delete カーネル サービスの呼び出し処理中に失敗した場合、ミューテックスを削除します。
- tx_semaphore_create レンダリング同期と表示同期のためのセマフォを作成します。

表 159: 戻り値

名前	説明
SSP_SUCCESS	モジュールが正常に開かれました。
p_ctrl の可能性があります。	NULL ポインタのエラーが発生しました。
チャネルは現在動作中です。	SF_EL_GX が使用中です。
内部エラーが発生しました。	カーネル サービス呼び出し中にエラーが発生しました。

6.8.2.3 関数のステップ

- SF_EL_GX のグローバル ミューテックスを作成します
- SF_EL_GX_Setup によってドライバのセットアップが完了するまで、SF_EL_GX インスタンスをロックします。
- フレーム バッファのフリップに対してセマフォを作成します
- SF_EL_GX 制御ブロックを初期化します
- モジュール内のグローバル ポインタに、制御ブロックを一時的に保存します。格納したデータは、GUIX™ によって呼び出される sf_el_gx_driver_setup() で使用されます。このポインタは最後には有効になりますが、SF_EL_GX_Setup が完了するまで保護されます。
- ドライバの状態を変更します

6.8.3 SF_EL_GX_Close

```
ssp_err_t  SF_EL_GX_Close ( sf_el_gx_ctrl_t  *const p_ctrl )
```

6.8.3.1 概要説明

Synergy 対応 GUIX™ 適用フレームワークであるクローズ関数。この関数は、以下を呼び出します。

6.8.3.2 詳細説明

- tx_mutex_get デバイス アクセス中にミューテックスにドライバをロックさせます。
- tx_mutex_put デバイス アクセス中にミューテックスにドライバをロック解除させます。
- tx_mutex_delete カーネル サービスの呼び出し処理中に失敗した場合、ミューテックスを削除します。
- tx_semaphore_delete レンダリング同期と表示同期のためのセマフォを削除します。
- sf_el_gx_d2_close が 2D 描画エンジン ハードウェアを終了処理します。
- sf_el_gx_display_close 表示ハードウェアをを終了処理します。

表 160: 戻り値

名前	説明
SSP_SUCCESS	モジュールが正常に閉じられました。
p_ctrl の可能性があります。	NULL ポインタのエラーが発生しました。
タッチ パネルが設定されていません。	SF_EL_GX が開かれていません。

表 160: 戻り値 (続き)

名前	説明
内部エラーが発生しました。	カーネル サービス呼び出し中にエラーが発生しました。
SSP_ERR_TIMEOUT	ディスプレイ ドライバでエラーが発生しました。
SSP_ERR_D2D_ERROR_DEINIT	D/AVE 2D ドライバでエラーが発生しました。

I：この関数は再入可能です。

6.8.3.3 関数のステップ

- コンテキストを更新するために、ドライバをロックします。
- 表示ハードウェアを終了処理します
- ドライバの状態を変更します
- フレーム バッファのフリップに対してセマフォを作成します
- ドライバのセットアップ完了後、SF_EL_GX インスタンスをロック解除します
- ドライバのグローバル ミューテックスを削除します
- 制御ブロックへのポインタ用の一時ストレージをクリアします。これは、SF_EL_GX_Setup で予期される関数呼び出しシーケンスに応じて行なわれますが、ここでは予期しないシーケンスが発生した場合もクリアされます。つまり、SF_EL_GX_Setup は呼び出されません。

6.8.4 SF_EL_GX_VersionGet

```
ssp_err_t  SF_EL_GX_VersionGet ( ssp_version_t  * p_version )
```

6.8.4.1 概要説明

Synergy 対応の GUIX™ 適用フレームワークであるバージョン取得関数。

6.8.4.2 詳細説明

表 161: パラメータ

名前	方向	説明
p_version	入力 / 出力	バージョン番号。

表 162: 戻り値

名前	説明
SSP_SUCCESS	この関数は常にこの値を返します

I：この関数は再入可能です。

6.8.5 SF_EL_GX_Setup

SF_EL_GX_Setup (GX_DISPLAY * p_display)

6.8.5.1 概要説明

Synergy 対応の GUIX™ 適応フレームワークであり、表示および D/AVE 2D インタフェース用に GUIX ローレベル デバイス ドライバをセットアップします。この関数は、GUIX Studio 表示ドライバ セットアップ関数 gx_studio_display_configure() に渡す必要があります。これにより、GUIX™ が GUIX™ ローレベルデバイスドライバを構成できるようになります。この関数は、以下を呼び出します。

6.8.5.2 詳細説明

- tx_mutex_put ローレベル ドライバ設定が行われたとき、ドライバのグローバル ミューテックスを置きます
- tx_mutex_delete カーネル サービスの呼び出し処理中に失敗した場合、ミューテックスを削除します
- _gx_synergy_display_driver_565rgb_setup ディスプレイ フォーマットが RGB565 フォーマットである場合、RGB565 のデフォルト GUIX™ コールバック関数を設定します
- _gx_synergy_display_driver_24xrgb_setup ディスプレイ フォーマットが RGB888 フォーマット、アンパック フォーマットである場合、RGB565 のデフォルト GUIX™ コールバック関数を設定します

- _gx_display_driver_32argb_setup ディスプレイ フォーマットが ARGB8888 フォーマット、アンパック フォーマットである場合、RGB565 のデフォルト GUIX™ コールバック関数を設定します
- sf_el_gx_driver_setup ローレベル デバイス ドライバをセットアップし、GUIX™ のデフォルトのコールバック関数をハードウェアによって加速した関数で上書きします。

表 163: 戻り値

名前	説明
GX_SUCCESS	デバイス ドライバのセットアップが正常に完了しました。
GX_FAILURE	デバイス ドライバのセットアップが失敗しました。

I :GUIX™ によってこの関数をコールバックするときには、それより前に SF_EL_GX_Open が呼び出されていることを確認してください。この関数が GUIX™ で呼び出されていない場合の動作は不定です。

6.8.5.3 関数のステップ

- < RGB565、16 ビット カラー フォーマットの汎用コールバック関数をセットアップします。
- < RGB888、24 ビット、アンパック フォーマットの汎用コールバック関数をセットアップします。
- < RGB888、24 ビット、アンパック フォーマットの汎用コールバック関数をセットアップします。
- < ARGB4444、16 ビット カラー フォーマットの汎用コールバック関数をセットアップします。
- 後で使用できるように、GX_DISPLAY コンテキストをコピーします。
- GUIX™ ローレベルデバイスドライバをセットアップします
- ドライバの状態を変更します
- 制御ブロックへのポインタ用の一時ストレージをクリアします。
- ドライバのセットアップ完了後、SF_EL_GX インスタンスをロック解除します

6.8.6 SF_EL_GX_CanvasInit

```
ssp_err_t  SF_EL_GX_CanvasInit ( sf_el_gx_ctrl_t  *const p_ctrl ,  GX_WINDOW_ROOT
* p_window_root )
```

6.8.6.1 概要説明

Synergy 対応の GUIX™ 適応フレームワークであるキャンバスの初期化。レンダリングする最初のキャンバスのメモリアドレスをセットアップします。

6.8.6.2 詳細説明

表 164: 戻り値

名前	説明
SSP_SUCCESS	メモリ アドレスがキャンバスに正常に構成されました。
SSP_ERR_INVALID_CALL	ドライバが SF_EL_GX_CONFIGURED 状態でないときに、関数呼び出しが実行されます。
内部エラーが発生しました。	ミューテックス操作にエラーが発生しました。

6.8.6.3 関数のステップ

- コンテキストを更新するために、ドライバをロックします。
- GUIX™ に対して最初のキャンバスを指定します
- ドライバをロック解除します。

6.8.7 モジュール

- ビルドタイム構成

6.8.7.1 ビルドタイム構成

定義

- #define SF_EL_GX_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.9 Telnet 通信フレームワーク

RTOS 統合された通信フレームワーク NetX™ telnet サーバーの実装。

6.9.1 Functions

- SF_EL_NX_COMMS_Open
- SF_EL_NX_COMMS_Close
- SF_EL_NX_COMMS_Read
- SF_EL_NX_COMMS_Write
- SF_EL_NX_COMMS_Lock
- SF_EL_NX_COMMS_Unlock
- SF_EL_NX_COMMS_VersionGet

6.9.2 定義

- #define SF_EL_NX_COMMS_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_EL_NX_COMMS_CODE_VERSION_MINOR

  初期値 :(1)
- #define SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES

  初期値 :((1536 + 32 + sizeof(NX_PACKET)) * 50)
- #define SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES

  初期値 :(2048)
- #define SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES

  初期値 :(1024)
- #define SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES

  初期値 :(2048)
- #define SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES

  初期値 :(20)

6.9.3 SF_EL_NX_COMMS_Open

ssp_err_t SF_EL_NX_COMMS_Open (sf_comms_ctrl_t *const p_ctrl , sf_comms_cfg_t const *const p_cfg)

6.9.3.1 詳細説明

Telnet サーバーを初期化します。

表 165: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開かれました
p_ctrl の可能性があります。	UART 制御ブロックまたは設定構造体へのポインタが NULL です
内部エラーが発生しました。	USBX™ の呼び出しの初期化時にエラーが返されました。

I：この関数は再入可能です。

6.9.3.2 関数のステップ

- NetX™ システムを初期化します。
- パケット プールを作成します。
- IP インスタンスを作成します。
- ARP を有効化し、IP インスタンスに ARP キャッシュ メモリを提供します。
- TCP トラフィックを有効にします。
- ICMP を有効にします。
- NetX™ Telnet サーバーを作成します。
- Telnet サーバーを開始します。
- ミューテックスを作成します。
- 制御ブロックをオープンとしてマークします。

6.9.4 SF_EL_NX_COMMS_Close

ssp_err_t SF_EL_NX_COMMS_Close (sf_comms_ctrl_t *const p_ctrl)

6.9.4.1 詳細説明

Telnet サーバーを切断し、変数をクリーンアップします。

表 166: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました
p_ctrl の可能性があります。	制御ブロックへのポインタが NULL です

I：この関数は再入可能です。

6.9.4.2 関数のステップ

- サーバーの切断を開始します。
- RTOS オブジェクトを削除します。
- 制御ブロックをクローズとしてマークします。

6.9.5 SF_EL_NX_COMMS_Read

ssp_err_t SF_EL_NX_COMMS_Read (sf_comms_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const bytes , UINT const timeout)

6.9.5.1 詳細説明

Telnet サーバーからデータを読み取ります。

表 167: 戻り値

名前	説明
SSP_SUCCESS	データの読み取りが正常に終了しました。
内部エラーが発生しました。	NetX™ API 呼び出しでエラーが発生しました。

I：この API は再入可能です。

6.9.5.2 関数のステップ

- クライアントが接続されるまで待機します。
- ミューテックスを取得します。
- データを受信します。
- ミューテックスを解放します。

6.9.6 SF_EL_NX_COMMS_Write

```
ssp_err_t  SF_EL_NX_COMMS_Write ( sf_comms_ctrl_t  *const p_ctrl ,  uint8_t const *const p_src ,
 uint32_t const bytes ,  UINT const timeout )
```

6.9.6.1 詳細説明

USB CDC ACM ドライバにデータを書き込みます。

表 168: 戻り値

名前	説明
SSP_SUCCESS	データの送信が正常に終了しました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
内部エラーが発生しました。	USB デバイスが列挙されていないか、USBX™ 書き込み呼び出しがエラーを返しました。

I：この関数は再入可能です。

6.9.6.2 関数のステップ

- データにパケットを割り当てます。
- メッセージをビルドします。
- パケットをクライアントに送信します。

6.9.7 SF_EL_NX_COMMS_Lock

```
ssp_err_t  SF_EL_NX_COMMS_Lock ( sf_comms_ctrl_t  *const p_ctrl ,
 sf_comms_lock_t   lock_type ,  UINT timeout )
```

6.9.7.1 詳細説明

表 169:　戻り値

名前	説明
SSP_SUCCESS	UART リソースが正常にロックされました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
SSP_ERR_TIMEOUT	ミューテックスをタイムアウト内に使用できません。
チャネルは現在動作中です。	チャネルが使用中です。

6.9.7.2 関数のステップ

- Telnet クライアントが接続されるまで待機します。
- 要求されたミューテックスを取得します。

6.9.8 SF_EL_NX_COMMS_Unlock

```
ssp_err_t  SF_EL_NX_COMMS_Unlock ( sf_comms_ctrl_t  *const p_ctrl ,
 sf_comms_lock_t   lock_type )
```

6.9.8.1 詳細説明

表 170: 戻り値

名前	説明
SSP_SUCCESS	UART リソースが正常にロック解除されました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
チャネルは現在動作中です。	チャネルが使用中です。

6.9.8.2 関数のステップ

- ミューテックスを返します。

6.9.9 SF_EL_NX_COMMS_VersionGet

```
ssp_err_t SF_EL_NX_COMMS_VersionGet ( ssp_version_t *const p_version )
```

6.9.9.1 詳細説明

ドライバのバージョンを取得します

表 171: パラメータ

名前	方向	説明
p_version	out	バージョンがここに格納されます。

I：この関数は再入可能です。

6.9.10 拡張

6.9.10.1 sf_el_nx_comms_on_comms_ctrl_t

sf_el_nx_comms_on_comms_ctrl_t

詳細説明

NetX™ Telnet サーバー通信ドライバ構成

変数

- uint32_t open
- NX_PACKET * p_current_packet
- uint32_t packet_index
- TX_MUTEX mutex[2]
- NX_PACKET_POOL pool
- uint8_t pool_memory[SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES]
- NX_IP ip
- uint8_t ip_memory[SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES]
- uint8_t arp_memory[SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES]
- NX_TELNET_SERVER telnet_server
- uint8_t telnet_server_memory[SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES]
- UINT logical_connection
- TX_EVENT_FLAGS_GROUP available
  この接続が使用可能かどうかを示すフラグ。
- TX_QUEUE queue
  受信バイトのキュー。
- uint8_t queue_memory[SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES]

6.9.10.2 sf_el_nx_comms_on_comms_cfg_t

sf_el_nx_comms_on_comms_cfg_t

詳細説明

NetX™ Telnet サーバー デバイス通信ドライバ構成

変数

- sf_el_nx_comms_on_comms_ctrl_t * p_ctrl
  NetX™ Telnet サーバー デバイス制御ブロックに割り当てられたメモリ。
- uint32_t ip_address
- uint32_t subnet_mask

- VOID(* driver)(NX_IP_DRIVER *driver_req_ptr)

6.10 USB 通信フレームワーク

RTOS 統合された USBX™ CDC ACM デバイス実装。

6.10.1 Functions

- SF_EL_UX_COMMS_Open
- SF_EL_UX_COMMS_Close
- SF_EL_UX_COMMS_Read
- SF_EL_UX_COMMS_Write
- SF_EL_UX_COMMS_Lock
- SF_EL_UX_COMMS_Unlock
- SF_EL_UX_COMMS_VersionGet

6.10.2 定義

- #define SF_EL_UX_COMMS_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_EL_UX_COMMS_CODE_VERSION_MINOR

  初期値 :(1)

6.10.3 SF_EL_UX_COMMS_Open

`ssp_err_t` SF_EL_UX_COMMS_Open (`sf_comms_ctrl_t` *const p_ctrl , `sf_comms_cfg_t` const *const p_cfg)

6.10.3.1 詳細説明

USB チャネルを CDC ACM モードに初期化します。

表 172: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開かれました
p_ctrl の可能性があります。	UART 制御ブロックまたは設定構造体へのポインタが NULL です

表 172: 戻り値 (続き)

名前	説明
内部エラーが発生しました。	USBX™ の呼び出しの初期化時にエラーが返されました。

I：この関数は再入可能です。

6.10.3.2 関数のステップ

- ミューテックスを作成します。
- 制御ブロックをオープンとしてマークします。

6.10.4 SF_EL_UX_COMMS_Close

```
ssp_err_t  SF_EL_UX_COMMS_Close ( sf_comms_ctrl_t  *const p_ctrl )
```

6.10.4.1 詳細説明

USB チャネルの CDC ACM 操作を最終処理し、SCI デバイス制御ブロックのメンバーをクリアします。

表 173: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました
p_ctrl の可能性があります。	制御ブロックへのポインタが NULL です

I：この関数は再入可能です。

6.10.5 SF_EL_UX_COMMS_Read

ssp_err_t SF_EL_UX_COMMS_Read (sf_comms_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const bytes , UINT const timeout)

6.10.5.1 詳細説明

USB CDC ACM ドライバからデータを読み取ります。

表 174: 戻り値

名前	説明
SSP_SUCCESS	データの読み取りが正常に終了しました。
内部エラーが発生しました。	USB デバイスが列挙されていないか、USBX™ 読み込み呼び出しがエラーを返しました。

I：この API は再入可能です。

6.10.5.2 関数のステップ

- USB が挿入されるまで待機します。
- 前回のパケットのデータが残っている場合は、使用してください。
- CDC クラスから読み取ります。
- バッファ オーバーフローが発生しました。

6.10.6 SF_EL_UX_COMMS_Write

ssp_err_t SF_EL_UX_COMMS_Write (sf_comms_ctrl_t *const p_ctrl , uint8_t const *const p_src , uint32_t const bytes , UINT const timeout)

6.10.6.1 詳細説明

USB CDC ACM ドライバにデータを書き込みます。

表 175: 戻り値

名前	説明
SSP_SUCCESS	データの送信が正常に終了しました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
内部エラーが発生しました。	USB デバイスが列挙されていないか、USBX™ 書き込み呼び出しがエラーを返しました。

I：この関数は再入可能です。

6.10.6.2 関数のステップ

- USB が挿入されるまで待機します。

6.10.7 SF_EL_UX_COMMS_Lock

```
ssp_err_t  SF_EL_UX_COMMS_Lock ( sf_comms_ctrl_t  *const p_ctrl ,
 sf_comms_lock_t   lock_type ,  UINT timeout )
```

6.10.7.1 詳細説明

表 176: 戻り値

名前	説明
SSP_SUCCESS	UART リソースが正常にロックされました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
SSP_ERR_TIMEOUT	ミューテックスをタイムアウト内に使用できません。
チャネルは現在動作中です。	チャネルが使用中です。

6.10.8 SF_EL_UX_COMMS_Unlock

```
ssp_err_t  SF_EL_UX_COMMS_Unlock ( sf_comms_ctrl_t  *const p_ctrl ,
 sf_comms_lock_t   lock_type )
```

6.10.8.1 詳細説明

表 177:　戻り値

名前	説明
SSP_SUCCESS	UART リソースが正常にロック解除されました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
チャネルは現在動作中です。	チャネルが使用中です。

6.10.9 SF_EL_UX_COMMS_VersionGet

```
ssp_err_t  SF_EL_UX_COMMS_VersionGet ( ssp_version_t  *const p_version )
```

6.10.9.1 詳細説明

ドライバのバージョンを取得します

表 178:　パラメータ

名前	方向	説明
p_version	out	バージョンがここに格納されます。

I：この関数は再入可能です。

6.10.10 拡張

6.10.10.1 sf_el_ux_comms_on_comms_ctrl_t

sf_el_ux_comms_on_comms_ctrl_t

詳細説明

USBX™ CDC ACM デバイスの通信ドライバ設定

変数

- uint32_t open
- TX_MUTEX mutex[2]
- UX_SLAVE_CLASS_CDC_ACM * p_cdc
- uint32_t leftover_length
- uint32_t index
- uint8_t memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]
- uint8_t rx_memory[SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH]

6.10.10.2 sf_el_ux_comms_on_comms_cfg_t

sf_el_ux_comms_on_comms_cfg_t

詳細説明

USBX™ CDC ACM デバイスの通信ドライバ設定

変数

- sf_el_ux_comms_on_comms_ctrl_t * p_ctrl
  USBX™ CDC ACM デバイス制御ブロックに割り当てられたメモリ。

6.10.11 モジュール

- ビルドタイム構成

6.10.11.1 ビルドタイム構成

定義

- #define SF_EL_UX_COMMS_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES

  初期値 :(65536)

  USBX™ に割り当てるメモリ サイズを指定します。デバイスのアプリケーションでは、65536 が推奨サイズです。

- #define SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH

  初期値 :(128)

  リード入力バッファに割り当てるメモリ サイズを指定します。

6.11 外部 IRQ フレームワーク

RTOS に統合された外部 IRQ フレームワーク。

6.11.1 概要

このモジュールは、スイッチなどのバイナリ信号を含む外部入力のための、ThreadX 対応の外部 IRQ フレームワークです。

6.11.2 Functions

- SF_EXTERNAL_IRQ_Open
- SF_EXTERNAL_IRQ_Wait
- SF_EXTERNAL_IRQ_VersionGet
- SF_EXTERNAL_IRQ_Close

6.11.3 定義

- #define SF_EXTERNAL_IRQ_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_EXTERNAL_IRQ_CODE_VERSION_MINOR

  初期値 :(1)

6.11.4 SF_EXTERNAL_IRQ_Open

```
ssp_err_t SF_EXTERNAL_IRQ_Open ( sf_external_irq_ctrl_t *const p_ctrl ,
 sf_external_irq_cfg_t const *const p_cfg )
```

6.11.4.1 概要説明

外部 IRQ を設定し、オプションとして外部 IRQ コールバックを有効にします。open を実装します。

6.11.4.2 詳細説明

SF_EXTERNAL_IRQ_Open 関数は、使用する外部 IRQ チャネル用のミューテックスを取得した後、HAL ドライバの open 関数を呼び出します。初期化に成功すると、外部 IRQ が使用できるようになります。

表 179: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、外部割り込みが開始しました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg、p_api、p_api->open のいずれかのパラメータが NULL の可能性があります。考えられる他の原因については、HAL ドライバを参照してください。
チャネルは現在動作中です。	このチャネルはすでに開かれています。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数はあらゆるチャネルで再入可能です。

6.11.4.3 関数のステップ

- 関数のステップ
- 他のフレームワーク レイヤー関数で使用するためのドライバ構造体を保存します
- HW レジスタにアクセスする下位レイヤーを呼び出す前に、ミューテックスを取得します。
- HW レジスタにアクセスする下位レイヤーを呼び出す前に、ミューテックスを取得します。
- 下位レイヤーのための設定を準備します
- 制御ブロックをオープンとしてマークし、それが有効であることを他のタスクに示します

6.11.5 SF_EXTERNAL_IRQ_Wait

ssp_err_t SF_EXTERNAL_IRQ_Wait (sf_external_irq_ctrl_t *const p_ctrl , ULONG const timeout)

6.11.5.1 概要説明

ミューテックスを取得し、外部割り込みの期限切れを待ってミューテックスを解放します。wait を実装します。

6.11.5.2 詳細説明

表 180: 戻り値

名前	説明
SSP_SUCCESS	外部割り込みが正常に停止しました。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_EXTERNAL_IRQ_Open を呼び出して構成します。
SSP_ERR_TIMEOUT	セマフォを待っている間にタイムアウトが発生します。
SSP_ERR_WAIT_ABORTED	他のスレッドによりサスペンションが中断されました。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません(p_ctrl->drv.stop が NULL)。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.11.5.3 関数のステップ

- ISR からセマフォがポストされるまで待機します

6.11.6 SF_EXTERNAL_IRQ_VersionGet

```
ssp_err_t SF_EXTERNAL_IRQ_VersionGet ( ssp_version_t *const p_version )
```

6.11.6.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。versionGet を実装します。

6.11.6.2 詳細説明

表 181: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.11.7 SF_EXTERNAL_IRQ_Close

ssp_err_t SF_EXTERNAL_IRQ_Close (sf_external_irq_ctrl_t *const p_ctrl)

6.11.7.1 概要説明

チャネル ミューテックスを解放し、HAL レイヤーでチャネルを閉じます。close を実装します。

6.11.7.2 詳細説明

表 182: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	パラメータ ctrl が NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.11.7.3 関数のステップ

- これは後にハードウェア レジスタにアクセスするので、ミューテックスを取得します
- HAL レイヤーのチャネルを閉じます
- 制御ブロックの情報を閉じることによって、このブロックが閉じていることを他の関数に示します
- 使用された RTOS サービスを削除します

6.11.8 モジュール

- ビルドタイム構成

6.11.8.1 ビルドタイム構成

定義

- #define SF_EXTERNAL_IRQ_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータ チェック用のコードを含めるかどうかを指定します。
  BSP_CFG_PARAM_CHECKING_ENABLE に設定した場合は、システムのデフォルト設定が使用されます。1 に設定した場合はパラメータ チェックが含まれます。0 はパラメータ チェックをコンパイルアウトします

6.12 I²C フレームワーク

RTOS 統合された I²C フレームワーク。

SSP I²C フレームワーク ドライバ API。

6.12.1 概要

ThreadX 対応の I²C ドライバ API です。この API は、I2C フレームワーク インタフェースを実装し、HAL レイヤーで複数のハードウェア周辺機能にアクセスできます。HAL レイヤーへの接続は、SF_I2C_Open 内でドライバ構造体を渡して確立されます。

6.12.2 使用されるインタフェース

I2C フレームワーク

6.12.3 Functions

- SF_I2C_Open
- SF_I2C_Read
- SF_I2C_Write
- SF_I2C_Reset
- SF_I2C_Close
- SF_I2C_Lock
- SF_I2C_Unlock
- SF_I2C_VersionGet

6.12.4 定義

- #define SF_I2C_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_I2C_CODE_VERSION_MINOR

  初期値 :(1)

6.12.5 SF_I2C_Open

```
ssp_err_t SF_I2C_Open ( sf_i2c_ctrl_t *const p_ctrl , sf_i2c_cfg_t const *const p_cfg )
```

6.12.5.1 詳細説明

表 183: 戻り値

名前	説明
SSP_SUCCESS	I2C チャネルが正常に開かれました
未サポートまたは不正確なモードです。	指定された I2C モードが不正です
SSP_ERR_INVALID_CHANNEL	指定された I2C チャネルは除外されています
チャネルは現在動作中です。	I2C チャネルがすでに開かれています
SSP_ERR_INVALID_POINTER	引数が、事前に定義された値に含まれていません
Null ポインタが指定されました。	Null ポインタが指定されました
内部エラーが発生しました。	重要なアサーションが失敗しました
p_ctrl の可能性があります。	この関数を呼び出す前に、呼び出し側が制御ハンドルをクリアする必要があります。

I：この関数はあらゆるチャネルで再入可能です。

I：この関数を呼び出す前に、呼び出し側が制御ハンドルをクリアする必要があります。

6.12.5.2 関数のステップ

- 制御対象の chip_select をコピーします
- フレームワーク レベルのコールバック関数を設定します
- コンテキストを ISR で使用できるように保存します。
- デバイスのオープンで、バス チャネルを使用します

- バスの上の最初のデバイスに対してのみ開きます。
- バスの上の最初のデバイスに対してのみ開きます。デバイスでバスの再設定が必要な場合、後で読み取り / 書き込みが発生したときに再設定が行われます。
- このバスに対するミューテックスを作成します。
- イベント フラグを作成します。
- ローレベルを開きます。
- このバスで最後に使用されたデバイス レートを割り当てます。
- 再設定に備えて、デバイス設定を保存します。
- デバイス状態を Opened に設定します。
- 再開始フラグを false に初期化します。
- デバイス カウントをインクリメントします。

6.12.6 SF_I2C_Read

ssp_err_t SF_I2C_Read (sf_i2c_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const bytes , bool const restart , uint32_t const timeout)

6.12.6.1 詳細説明

表 184: 戻り値

名前	説明
SSP_SUCCESS	関数が問題なく実行されました。
Null ポインタが指定されました。	p_ctrl または p_dest が NULL です。
SSP_ERR_INVALID_POINTER	バイト数が 0 です。
チャネルは現在動作中です。	別の転送が進行中でした。
SSP_ERR_TIMEOUT	タイムアウト エラーが発生しました。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。
SSP_ERR_TRANSFER_ABORTED	データ転送が中止されました。
タッチ パネルが設定されていません。	デバイス インスタンスが開かれていません。
p_ctrl の可能性があります。	重要なアサーションが失敗しました。

6.12.6.2 関数のステップ

- close() によってデバイスを閉じる必要があるかどうかを確認します。
- 転送プロセスを開始します （ロックの確認、再設定の確認、ミューテックスの取得)。
- 読み取りを実行します。
- コールバックがイベント フラグを設定するまで待機します。
- 転送を完了します。

6.12.7 SF_I2C_Write

ssp_err_t SF_I2C_Write (sf_i2c_ctrl_t *const p_ctrl , uint8_t *const p_src , uint32_t const bytes , bool const restart , uint32_t const timeout)

6.12.7.1 詳細説明

表 185: 戻り値

名前	説明
SSP_SUCCESS	関数が問題なく実行されました。
Null ポインタが指定されました。	p_ctrl または p_src が NULL です。
SSP_ERR_INVALID_POINTER	バイト数が 0 です。
チャネルは現在動作中です。	別の転送が進行中でした。
SSP_ERR_TIMEOUT	タイムアウト エラーが発生しました。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。
p_ctrl の可能性があります。	重要なアサーションが失敗しました。

6.12.7.2 関数のステップ

- close() によってデバイスを閉じる必要があるかどうかを確認します。
- 転送プロセスを開始します （ロックの確認、再設定の確認、ミューテックスの取得)。
- 転送を完了します。

6.12.8 SF_I2C_Reset

ssp_err_t SF_I2C_Reset (sf_i2c_ctrl_t *const p_ctrl , uint32_t const timeout)

6.12.8.1 詳細説明

表 186: 戻り値

名前	説明
SSP_SUCCESS	チャネルは問題なくリセットされました。
Null ポインタが指定されました。	p_ctrl が NULL です。
SSP_ERR_ABORTED	ハードウェアのリセット中に転送が中止されました。
タッチ パネルが設定されていません。	デバイスが開かれていません。*
SSP_ERR_TIMEOUT	タイムアウト エラーが発生しました。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。
p_ctrl の可能性があります。	重要なアサーションが失敗しました。

6.12.8.2 関数のステップ

- close() によってデバイスを閉じる必要があるかどうかを確認します。
- 完了し、転送に使用されたミューテックスを解放します。

6.12.9 SF_I2C_Close

ssp_err_t SF_I2C_Close (sf_i2c_ctrl_t *const p_ctrl)

6.12.9.1 詳細説明

表 187: 戻り値

名前	説明
SSP_SUCCESS	I2C チャネルが正常に閉じられました。
Null ポインタが指定されました。	NULL ポインタ。

表 187: 戻り値 (続き)

名前	説明
チャネルは現在動作中です。	使用中エラー。
内部エラーが発生しました。	内部エラー。
p_ctrl の可能性があります。	重要なアサーションが失敗しました。

I：この関数はあらゆるデバイスで再入可能です。

6.12.9.2 関数のステップ

- close() によってデバイスを閉じる必要があるかどうかを確認します。
- close() によってデバイスを閉じる必要があるかどうかを確認します。これが現在のデバイスである場合のみデバイスを閉じます。それ以外の場合、デバイスは再設定時に閉じられているか、open() によって開かれていません。
- これは後にハードウェア レジスタにアクセスするので、ミューテックスを取得します。
- ローレベル ドライバを閉じます。
- ミューテックスを解放します。
- デバイス カウントをデクリメントします。
- バスを使用しているデバイスがあるかどうかを確認します。
- バスによって使用された RTOS サービスを削除します。
- デバイスを閉じた状態に設定します。

6.12.10 SF_I2C_Lock

```
ssp_err_t  SF_I2C_Lock (  sf_i2c_ctrl_t *const p_ctrl )
```

6.12.10.1 詳細説明

表 188: 戻り値

名前	説明
SSP_SUCCESS	I2C チャネルが正常に閉じられました。
タッチ パネルが設定されていません。	デバイスは開かれていません。
チャネルは現在動作中です。	使用中エラー。
内部エラーが発生しました。	内部エラー。
p_ctrl の可能性があります。	重要なアサーションが失敗しました。

I：この関数はあらゆるデバイスで再入可能です。

6.12.10.2 関数のステップ

- close() によってデバイスを閉じる必要があるかどうかを確認します。
- ロックがすでに使用されているかどうかを確認します。
- このデバイスに対してミューテックスを取得します。
- ロック フラグを設定します。

6.12.11 SF_I2C_Unlock

```
ssp_err_t  SF_I2C_Unlock (  sf_i2c_ctrl_t *const p_ctrl )
```

6.12.11.1 詳細説明

表 189: 戻り値

名前	説明
SSP_SUCCESS	I2C チャネルが正常に閉じられました。

表 189: 戻り値 (続き)

名前	説明
タッチ パネルが設定されていません。	デバイスは開かれていません。
チャネルは現在動作中です。	使用中エラー。
内部エラーが発生しました。	内部エラー。
p_ctrl の可能性があります。	重要なアサーションが失敗しました。

I：この関数はあらゆるデバイスで再入可能です。

6.12.11.2 関数のステップ

- close() によってデバイスを閉じる必要があるかどうかを確認します。
- ロック フラグをクリアします。
- ミューテックスを解放して、他のスレッドがバスを使用できるようにします。

6.12.12 SF_I2C_VersionGet

```
ssp_err_t  SF_I2C_VersionGet ( ssp_version_t  *const p_version )
```

6.12.12.1 詳細説明

表 190: 戻り値

名前	説明
SSP_SUCCESS	正常な復帰。

6.12.12.2 関数のステップ

- エラーを確認します。それ以上のパラメータ チェックは、ドライバ レイヤーで行うことができます。

6.12.13 モジュール

- ビルドタイム構成

6.12.13.1 ビルドタイム構成

定義

- #define I2C_CH_MAX

  初期値 :(4)

- #define SF_I2C_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータ チェック用のコードを含めるかどうかを指定します。
  BSP_CFG_PARAM_CHECKING_ENABLE に設定した場合は、システムのデフォルト設定が使用されます。1 に設定した場合はパラメータ チェックが含まれます。0 はパラメータ チェックをコンパイルアウトします。

- #define SF_HAL_SCI0

  初期値 :(0)

- #define SF_HAL_SCI1

  初期値 :(1)

- #define SF_HAL_SCI2

  初期値 :(2)

- #define SF_HAL_SCI3

  初期値 :(3)

- #define SF_HAL_SCI4

  初期値 :(4)

- #define SF_HAL_SCI5

  初期値 :(5)

- #define SF_HAL_SCI6

  初期値 :(6)

- #define SF_HAL_SCI7

  初期値 :(7)

- #define SF_HAL_SCI8

  初期値 :(8)

- #define SF_HAL_SCI9

  初期値 :(9)
- #define SF_UART_CH0

  初期値 :(SF_HAL_SCI9)
- #define SF_UART_CH1

  初期値 :(SF_HAL_SCI1)

6.13 JPEG フレームワーク

RTOS 統合された JPEG フレームワーク。

6.13.1 Functions

- SF_JPEG_Decode_Open
- SF_JPEG_Decode_InputBufferSet
- SF_JPEG_Decode_LinesDecodedGet
- SF_JPEG_Decode_HorizontalStrideSet
- SF_JPEG_Decode_ImageSubsampleSet
- SF_JPEG_Decode_OutputBufferSet
- SF_JPEG_Decode_Wait
- SF_JPEG_Decode_StatusGet
- SF_JPEG_Decode_PixelFormatGet
- SF_JPEG_Decode_ImageSizeGet
- SF_JPEG_Decode_Close
- SF_JPEG_Decode_VersionGet

6.13.2 定義

- #define SF_JPEG_DECODE_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_JPEG_DECODE_CODE_VERSION_MINOR

  初期値 :(1)

6.13.3 SF_JPEG_Decode_Open

```
ssp_err_t  SF_JPEG_Decode_Open ( sf_jpeg_decode_ctrl_t  *const p_ctrl ,
 sf_jpeg_decode_cfg_t  const *const p_cfg )
```

6.13.3.1 概要説明

ミューテックスを取得した後、HAL レイヤーでドライバの初期化を処理します。この関数は呼び出しスレッドに復帰する前に、ミューテックスを解放します。

6.13.3.2 詳細説明

表 191: 戻り値

名前	説明
SSP_SUCCESS	JPEG デコード ドライバが正常に開かれました。
p_ctrl の可能性があります。	p_ctrl または p_cfg、p_cfg->p_lower_lvl_api、p_cfg->p_lower_lvl_api->open のいずれかのパラメータが NULL の可能性があります。または、デバイスがすでに開かれています。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.3.3 関数のステップ

- 他のフレームワーク レイヤー関数で使用するためのドライバ構造体ポインタを保存します
- 待機関数で使用するためのイベント フラグを作成します
- フレームワーク コールバック関数をインストールします。
- HW レジスタにアクセスする下位レイヤーを呼び出す前に、HAL JPEG ドライバにミューテックスを取得します。
- JPEG デバイスを設定するローレベル ドライバを呼び出します。
- HAL JEPG ドライバにミューテックスを解放します。
- open 呼び出しからの戻り値を確認します。
- 制御ブロックをオープンとしてマークし、それがオープンであることを後続の呼び出しに示します。

6.13.4 SF_JPEG_Decode_InputBufferSet

```
ssp_err_t SF_JPEG_Decode_InputBufferSet ( sf_jpeg_decode_ctrl_t *const p_ctrl , void *const p_buffer , uint32_t const buffer_size )
```

6.13.4.1 概要説明

JPEG コードされた入力データを設定します。

6.13.4.2 詳細説明

この API は、デコード入力バッファのアドレス レジスタを設定します。入力バッファのアドレスが設定された後、ドライバは、出力バッファのアドレスが設定されているかどうかを確認するとともに、出力バッファ サイズが 8 ライン以上の出力データを保持できるかどうかを検証します。入力バッファと出力バッファの両方が適切に設定されている場合は、ドライバによってデコード プロセスが自動的に開始されます。

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 192: 戻り値

名前	説明
SSP_SUCCESS	デコード入力バッファが正常に設定されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.4.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- HAL ドライバ レイヤーの inputBufferSet ルーチンを呼び出します。
- ミューテックスを解放します

6.13.5 SF_JPEG_Decode_LinesDecodedGet

```
ssp_err_t SF_JPEG_Decode_LinesDecodedGet ( sf_jpeg_decode_ctrl_t *const p_ctrl , uint32_t *const p_lines )
```

6.13.5.1 概要説明

コーデックによってデコードされたライン数を取得します。

6.13.5.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 193: 戻り値

名前	説明
SSP_SUCCESS	デコードされたライン数の値が正常に取得されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.5.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.6 SF_JPEG_Decode_HorizontalStrideSet

```
ssp_err_t  SF_JPEG_Decode_HorizontalStrideSet ( sf_jpeg_decode_ctrl_t  *const p_ctrl ,
 uint32_t horizontal_stride )
```

6.13.6.1 概要説明

水平ストライド値を設定します。

6.13.6.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 194: 戻り値

名前	説明
SSP_SUCCESS	水平ストライド値が正常に設定されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.6.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.7 SF_JPEG_Decode_ImageSubsampleSet

```
ssp_err_t  SF_JPEG_Decode_ImageSubsampleSet ( sf_jpeg_decode_ctrl_t  *const p_ctrl ,
 jpeg_decode_subsample_t   horizontal_subsample ,
 jpeg_decode_subsample_t   vertical_subsample )
```

6.13.7.1 概要説明

これにより、デコードされた画像サイズをアプリケーションによってランタイムに縮小できます。これにより、デコードされた画像サイズをアプリケーションによってランタイムに縮小できます。

6.13.7.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成

してください。

表 195: 戻り値

名前	説明
SSP_SUCCESS	画像サブサンプル値が正常に設定されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.7.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.8 SF_JPEG_Decode_OutputBufferSet

```
ssp_err_t SF_JPEG_Decode_OutputBufferSet ( sf_jpeg_decode_ctrl_t *const p_ctrl , void
* p_buffer , uint32_t buffer_size )
```

6.13.8.1 概要説明

詳細説明

6.13.8.2 詳細説明

出力バッファのアドレスが設定された後、ドライバは、バッファが保持できる出力ライン数を計算します。ハードウェアで処理するためには、一度にデコードする出力ライン数が、8 の倍数であることが必要です。入力バッファと出力バッファの両方が適切に設定されている場合は、ドライバによってデコード プロセスが自動的に開始されます。入力バッファと出力バッファの両方が適切に設定されている場合は、ドライバによってデコード プロセスが自動的に開始されます。

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成

してください。

表 196: 戻り値

名前	説明
SSP_SUCCESS	出力バッファが正常に設定されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.8.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.9 SF_JPEG_Decode_Wait

```
ssp_err_t  SF_JPEG_Decode_Wait ( sf_jpeg_decode_ctrl_t   *const p_ctrl ,
 jpeg_decode_status_t   *const p_status ,   uint32_t timeout )
```

6.13.9.1 概要説明

現在の JPEG コーデック操作が完了するまで待機します。

6.13.9.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 197: 戻り値

名前	説明
SSP_SUCCESS	待機関数が正常に復帰しました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
SSP_ERR_TIMEOUT	待機操作がタイムアウトし、基礎となるドライバが時間内に応答しませんでした。
SSP_ERR_WAIT_ABORTED	システム内部エラーが発生しました。

6.13.9.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。

6.13.10 SF_JPEG_Decode_StatusGet

```
ssp_err_t SF_JPEG_Decode_StatusGet ( sf_jpeg_decode_ctrl_t *const p_ctrl ,
 jpeg_decode_status_t *const p_status )
```

6.13.10.1 概要説明

この関数は、デバイスのポーリングに使用できます。この関数を使用すると、JPEG 操作で SF_JPEG_Decode_Wait を使用してブロックする代わりに、デバイスをポーリングすることができます。

6.13.10.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 198: 戻り値

名前	説明
SSP_SUCCESS	JPEG ステータス情報が取得されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。

表 198: 戻り値 (続き)

名前	説明
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.10.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.11 SF_JPEG_Decode_PixelFormatGet

```
ssp_err_t SF_JPEG_Decode_PixelFormatGet ( sf_jpeg_decode_ctrl_t *const p_ctrl ,
 jpeg_decode_color_space_t *const p_color_space )
```

6.13.11.1 概要説明

この関数は、JPEG 画像をデコードする場合のみ機能します。詳細説明

6.13.11.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 199: 戻り値

名前	説明
SSP_SUCCESS	JPEG 画像のサイズが取得されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。

表 199: 戻り値 (続き)

名前	説明
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.11.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.12 SF_JPEG_Decode_ImageSizeGet

`ssp_err_t` SF_JPEG_Decode_ImageSizeGet (`sf_jpeg_decode_ctrl_t` *const p_ctrl , uint16_t * p_horizontal_size , uint16_t * p_vertical_size)

6.13.12.1 概要説明

この関数は、JPEG 画像をデコードする場合のみ機能します。詳細説明

6.13.12.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出して JPEG コーデック ブロックを構成してください。

表 200: 戻り値

名前	説明
SSP_SUCCESS	JPEG 画像のサイズが取得されました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。

表 200: 戻り値 (続き)

名前	説明
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.12.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します

6.13.13 SF_JPEG_Decode_Close

```
ssp_err_t  SF_JPEG_Decode_Close ( sf_jpeg_decode_ctrl_t  *const p_ctrl )
```

6.13.13.1 概要説明

JPEG コーデック デバイスを閉じます。未完了のコーデック操作が中断され、出力データが破棄されます。

6.13.13.2 詳細説明

I：この関数を使用する前に、SF_JPEG_Decode_Open を呼び出してタイマを構成してください。

表 201: 戻り値

名前	説明
SSP_SUCCESS	JPEG デコード デバイスが正常に閉じました。
p_ctrl の可能性があります。	p_ctrl が NULL であるか、デバイスが開かれていません。
チャネルは現在動作中です。	ミューテックスが、このデバイスで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.13.13.3 関数のステップ

- HAL レベル ドライバ呼び出しを行う前に、ミューテックスを取得します。
- ミューテックスを解放します
- 制御ブロックの情報を閉じることによって、このブロックが閉じていることを他の関数に示します
- open 呼び出しで割り当てられた RTOS サービスを削除します。

6.13.14 SF_JPEG_Decode_VersionGet

```
ssp_err_t  SF_JPEG_Decode_VersionGet ( ssp_version_t  *const p_version )
```

6.13.14.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.13.14.2 詳細説明

表 202:　戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.13.15 モジュール

- ビルドタイム構成

6.13.15.1 ビルドタイム構成

定義

- #define SF_JPEG_DECODE_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define  BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.14 メッセージングフレームワーク

RTOS 統合されたメッセージング フレームワーク実装。

6.14.1 Functions

- SF_MESSAGE_Open
- SF_MESSAGE_Close
- SF_MESSAGE_BufferAcquire
- SF_MESSAGE_BufferRelease
- SF_MESSAGE_Post
- SF_MESSAGE_Pend
- SF_MESSAGE_VersionGet

6.14.2 定義

- #define SF_MESSAGE_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_MESSAGE_CODE_VERSION_MINOR

  初期値 :(1)
- #define SF_MESSAGE_QUEUE_MESSAGE_WORDS

  初期値 :(1)

  ワード単位のメッセージ サイズ

6.14.3 SF_MESSAGE_Open

```
ssp_err_t SF_MESSAGE_Open ( sf_message_buffer_ctrl_t *const p_ctrl , sf_message_cfg_t const *const p_cfg )
```

6.14.3.1 概要説明

メッセージ フレームワークを初期化します。この関数は、メッセージング フレームワーク制御ブロックを開始し、設定パラメータに対応する作業メモリを設定します。

6.14.3.2 詳細説明

表 203: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_cfg または p_mem_start が NULL です。
内部エラーが発生しました。	OS サービス呼び出しが失敗しました。
チャネルは現在動作中です。	メッセージング フレームワークが使用中です。
SSP_ERR_INVALID_WORKBUFFER_SIZE	作業バッファ サイズが無効です。
SSP_ERR_INVALID_MSG_BUFFER_SIZE	メッセージバッファサイズが無効です。
SSP_ERR_TOO_MANY_BUFFERS	メッセージ バッファが多すぎます。
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	不正なメッセージサブスクライバリストです。

I：この API 関数は、1 つのインスタンスあたり 1 回呼び出すことができます。2 回呼び出したときの動作は不定です。

I：この API 関数は、スレッド コンテキストからのみ呼び出すことができます。

6.14.3.3 関数のステップ

- メッセージング フレームワークが開かれているかどうかを返します
- 作業メモリ領域に、メモリ プールを作成します
- サブスクライバー リストを登録します
- フレームワークのステータスを CLOSED から OPENED に変更します

6.14.4 SF_MESSAGE_Close

ssp_err_t SF_MESSAGE_Close (sf_message_buffer_ctrl_t *const p_ctrl)

6.14.4.1 概要説明

メッセージ フレームワークを終了処理します。

6.14.4.2 詳細説明

表 204: 戻り値

名前	説明
SSP_SUCCESS	修了処理が正常に終了しました。
p_ctrl の可能性があります。	p_cfg または p_mem_start が NULL です。
タッチ パネルが設定されていません。	メッセージ フレームワーク モジュールが開かれていません。
内部エラーが発生しました。	OS サービス呼び出しが失敗しました

I：この API 関数は、スレッド コンテキストからのみ呼び出すことができます。

6.14.4.3 関数のステップ

- サブスクライバーを見つけ、サブスクライバーのキューをフラッシュします
- 作業メモリ領域に割り当てられたメモリ プールを削除します

6.14.5 SF_MESSAGE_BufferAcquire

ssp_err_t SF_MESSAGE_BufferAcquire (sf_message_buffer_ctrl_t const *const p_ctrl ,
sf_message_header_t ** pp_buffer , sf_message_acquire_cfg_t const *const p_acquire_cfg ,
uint32_t const wait_option)

6.14.5.1 概要説明

ブロックから渡されるメッセージ用に、バッファを取得します。

6.14.5.2 詳細説明

表 205: 戻り値

名前	説明
SSP_SUCCESS	バッファが正常に取得されました。
p_ctrl の可能性があります。	p_ctrl または p_buffer_ctrl が NULL です。
タッチ パネルが設定されていません。	メッセージ フレームワーク モジュールが開かれていません。
SSP_ERR_NO_MORE_BUFFER	メモリブロックプールにバッファがこれ以上見つかりません。
SSP_ERR_TIMEOUT	OS サービス呼び出しがタイムアウトを返しました。
内部エラーが発生しました。	OS サービス呼び出しが失敗しました。

I：この API 関数は、スレッドからだけではなく、ISR からも呼び出すことができます。

6.14.5.3 関数のステップ

- ブロック メモリ プールにバッファを割り当てます。
- バッファ制御ブロックをクリアします
- 使用中バッファのフラグを設定します
- 割り当てられたバッファのアドレスを "pp_buffer" に設定します
- バッファ内のイベント クラスとイベント コードをクリアします。これは、バッファ制御ブロック内の最初の値が不明であり、安全でないためです。
- "option" 引数に SF_MESSAGE_ACQUIRE_OPTION_KEEP が設定されている場合、バッファ制御ブロック内で "buffer_keep" フラグを設定します

6.14.6 SF_MESSAGE_BufferRelease

```
ssp_err_t SF_MESSAGE_BufferRelease ( sf_message_buffer_ctrl_t *const p_ctrl ,
 sf_message_header_t *const p_buffer , sf_message_release_option_t const option )
```

6.14.6.1 概要説明

SF_MESSAGE_BufferAcquire で取得されたバッファを解放します。

6.14.6.2 詳細説明

表 206: 戻り値

名前	説明
SSP_SUCCESS	バッファが正常に解放されました。
タッチ パネルが設定されていません。	メッセージ フレームワーク モジュールが開かれていません。
p_ctrl の可能性があります。	p_ctrl または p_buffer_ctrl が NULL です。

I：この API 関数は、スレッドから呼び出すことはできますが、ISR からは呼び出せません。

6.14.6.3 関数のステップ

- バッファ制御ブロックのアドレスを計算します
- SF_MESSAGE_RELEASE_OPTION_NAK オプションが設定されている場合、バッファ制御ブロック内で NAK フラグを設定します。NAK フラグは、サブスクライバーが複数存在する場合の論理和（OR）ロジックです。SF_MESSAGE_RELEASE_OPTION_ACK と SF_MESSAGE_RELEASE_OPTION_NAK の両方が設定されている場合は、SF_MESSAGE_RELEASE_OPTION_NAK が採用されます。
- カウンティング セマフォが 0 より小さい場合は、セマフォを低減します。
- 以下の場合は、バッファを解放します。（1）カウンティング セマフォが 0 で、バッファ キープ オプションが指定されていない。（2）"option" が SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE に設定されている。
- バッファ制御ブロックのフラグをクリアします。

- バックアップされた割り込みマスク レベルに戻します。
- 以下の場合、ユーザー コールバック関数 （登録されている場合）を呼び出します。(1) カウンティング セマフォが 0。(2) "option" が SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE に設定されている。
- メッセージのいずれかのサブスクライバーから NAK 応答が返された場合は、SF_MESSAGE_CALLBACK_EVENT_NAK を設定します
- メッセージのすべてのサブスクライバーから ACK 応答が返された場合は、SF_MESSAGE_CALLBACK_EVENT_ACK を設定します
- バッファ制御ブロックに維持されるコンテキストへのポインタを設定します
- 登録されているユーザー定義コールバック関数を呼び出します。

6.14.7 SF_MESSAGE_Post

```
ssp_err_t  SF_MESSAGE_Post ( sf_message_buffer_ctrl_t  *const p_ctrl ,
 sf_message_header_t  const *const p_buffer , sf_message_post_cfg_t  const *const p_post_cfg ,
 sf_message_post_err_t  *const p_post_err ,  uint32_t const wait_option )
```

6.14.7.1 概要説明

サブスクライバーに対してメッセージをポストします。

6.14.7.2 詳細説明

表 207:　戻り値

名前	説明
SSP_SUCCESS	メッセージのポストが正常に完了しました。
p_ctrl の可能性があります。	p_ctrl または p_buffer_ctrl が NULL です。
タッチ パネルが設定されていません。	メッセージ フレームワーク モジュールが開かれていません。
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	サブスクライバー リストが破損しています
SSP_ERR_NO_SUBSCRIBER_FOUND	サブスクライバーが見つかりません。
SSP_ERR_MESSAGE_QUEUE_FULL	キューが一杯です （タイムアウト オプションが指定されている場合は、メッセージの送信前にタイムアウトが発生します）

表 207: 戻り値 (続き)

名前	説明
内部エラーが発生しました。	OS サービス呼び出しが失敗しました

I：この API 関数は、スレッドからだけではなく、ISR からも呼び出すことができます （wait_option が TX_NO_WAIT の場合）。

I：メッセージ コンシューマーによるメッセージ読み取りの前に、バッファへの別のバッファ書き込みが行われるとデータが上書きされます。

6.14.7.3 関数のステップ

- 指定されたイベント クラスのサブスクライバー数を確認します
- バッファ制御ブロックのアドレスを計算します
- バッファ制御ブロック内でカウンティング セマフォをカウント アップします
- ユーザーから渡されたユーザー コールバック関数とコンテキストを登録します
- イベント クラスを受け取り、メッセージ キューにメッセージを送信するサブスクライバーを見つけます
- 共通メッセージ ヘッダー内のイベント クラス コードと、サブスクライバー リストのイベント クラスを比較します。比較した結果、一致しない場合は、そのサブスクライバー グループをスキップします。
- 共通メッセージ ヘッダー内のクラス インスタンス値が、サブスクライバー リスト内のイベント クラス インスタンスの範囲であることを確認します。
- メッセージが失敗したため、カウンティング セマフォをカウント ダウンします
- エラー サブスクライバー情報を設定します

6.14.8 SF_MESSAGE_Pend

ssp_err_t SF_MESSAGE_Pend (sf_message_buffer_ctrl_t const *const p_ctrl , TX_QUEUE const *const p_queue , sf_message_header_t ** pp_buffer , uint32_t const wait_option)

6.14.8.1 概要説明

メッセージを保留にします。

6.14.8.2 詳細説明

表 208: 戻り値

名前	説明
SSP_SUCCESS	メッセージが正常に保留されました。
p_ctrl の可能性があります。	p_ctrl または p_buffer_ctrl が NULL です。
タッチ パネルが設定されていません。	メッセージ フレームワーク モジュールが開かれていません。
SSP_ERR_MESSAGE_QUEUE_EMPTY	キューが空です （タイムアウト オプションが指定されている場合は、メッセージの受信前にタイムアウトが発生します）
SSP_ERR_TIMEOUT	OS サービス呼び出しがタイムアウトを返しました。
内部エラーが発生しました。	OS サービス呼び出しが失敗しました

I：この API 関数は、スレッドからだけではなく、ISR からも呼び出すことができます （wait_option が TX_NO_WAIT の場合）。

6.14.8.3 関数のステップ

- ここでメッセージを受け取ります。受け取るデータはメッセージそれ自体ではなく、バッファへのポインタです
- メッセージ キューにデータが存在せず、TX_NO_WAIT が wait_option に指定されている場合、SSP_ERR_MESSAGE_QUEUE_EMPTY エラー コードが即座に返されます

6.14.9 SF_MESSAGE_VersionGet

```
ssp_err_t  SF_MESSAGE_VersionGet ( ssp_version_t  *const p_version )
```

6.14.9.1 概要説明

メッセージング フレームワークのバージョンを取得します。指定されたポインタにバージョン情報を格納します。

6.14.9.2 詳細説明

表 209: 戻り値

名前	説明
SSP_SUCCESS	バージョン番号が正常に処理されました
p_ctrl の可能性があります。	p_version が NULL です

6.14.10 モジュール

- ビルドタイム構成

6.14.10.1 ビルドタイム構成

定義

- #define SF_MESSAGE_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータ チェック用のコードを含めるかどうかを指定します。
  BSP_CFG_PARAM_CHECKING_ENABLE に設定した場合は、システムのデフォルト設定が使用されます。1 に設定した場合はパラメータ チェックが含まれます。0 はパラメータ チェックをコンパイルアウトします

6.15 パワープロファイルフレームワーク

パワー プロファイル フレームワーク。

6.15.1 Functions

- SF_POWER_PROFILES_Open
- SF_POWER_PROFILES_Sleep
- SF_POWER_PROFILES_Close
- SF_POWER_PROFILES_VersionGet

6.15.2 定義

- #define SF_POWER_PROFILES_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_POWER_PROFILES_CODE_VERSION_MINOR

  初期値 :(1)

6.15.3 SF_POWER_PROFILES_Open

```
ssp_err_t SF_POWER_PROFILES_Open ( sf_power_profiles_ctrl_t  *const p_ctrl ,
 sf_power_profiles_cfg_t  const *const p_cfg )
```

6.15.3.1 概要説明

電力プロファイル フレームワークを構成し、使用される必要な HAL レイヤー ドライバを開きます。

6.15.3.2 詳細説明

SF_POWER_PROFILES_Open 関数は、電力プロファイル モジュール用のミューテックスを取得した後、必要なローレベル ドライバ （RTC、LPM）の open/init 関数を呼び出し、最後に電力プロファイル モジュール用のドライバの .open 関数を呼び出します。ミューテックスは、ドライバ レイヤーの open 関数の後に解放されます。

表 210: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、HAL ドライバを参照してください。
チャネルは現在動作中です。	ミューテックスが、要求されたユニットで使用できない可能性があります。考えられる他の原因については、HAL ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.15.3.3 関数のステップ

- 他のフレームワーク レイヤー関数で使用するためのドライバ構造体ポインタを保存します
- すでに開かれている RTC モジュールと関連する制御ポインタを維持します
- コールバック関数へのポインタを維持します
- スリープおよびスリープ解除の I/O ポート テーブルへのポインタを維持します
- それを変更するために、LPM 構成構造体のローカル コピーを作成します
- LPM HAL ドライバを初期化します
- 制御ブロックをオープンとしてマークし、それが有効であることを他のタスクに示します

6.15.4 SF_POWER_PROFILES_Sleep

```
ssp_err_t  SF_POWER_PROFILES_Sleep ( sf_power_profiles_ctrl_t  *const p_ctrl )
```

6.15.4.1 概要説明

MCU をソフトウェア スタンバイ モードにします。

6.15.4.2 詳細説明

表 211: 戻り値

名前	説明
SSP_SUCCESS	スリープに移行した後スリープを解除または == SF_POWER_PROFILES_MODE_RUN モードで呼び出しました。
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。SF_POWER_PROFILES_Open を呼び出して構成します。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません(p_ctrl->p_api->start が NULL)。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.15.4.3 関数のステップ

- スリープモードの準備
- 指定されたスリープ構成ごとに、I/O ポート端子を設定します
- LOCO や Subclk が現在起動しているかどうかを確認します。
- 後で復元するために現在の WUPEN を取得します
- RTC に関する情報を取得します
- RTC 周期やアラームが、それらが MCU の割り込みやスリープ解除を可能にするように WUPEN ビットを設定します
- LOCO を停止します
- サブクロックを停止します
- RTC 周期やアラームが、それらが MCU の割り込みやスリープ解除を可能にするように WUPEN ビットを設定します
- これらのクロックは、ソフトウェア スタンバイ モードの間オンにすることができます。ユーザーが RTC モードを使用していない場合、これらのクロックは自動的にオフになります。
- LOCO を停止します
- サブクロックを停止します

- S7G2
- 一部の周辺機能は、ソフトウェア スタンバイ モードでも動作するように設定できます。それらの機能がオンで、ウェイクアップ時に復元される場合は、低消費電力を実現するため、スリープ状態に移行する前にそれらの機能がオフになります。このような機能には、SCI0 （MSTPCRB b31）IIC0 （MSTPCRB b09）が含まれます
- それらの機能がオンで、ウェイクアップ時に復元される場合は、低消費電力を実現するため、スリープ状態に移行する前にそれらの機能がオフになります。
- 上記は MSTPCRB に適したビット設定です。このため、当社の MSTBCRB マスクは 0x7FFFFDFF となっています
- D/A コンバータ 0 （MSTPCRD b20）、AGT1 （MSTPCRD b1）AGT0 （MSTPCRD b3）コンパレータ - OC5 （MSTPCRD b23）コンパレータ - OC4 （MSTPCRD b24）コンパレータ - OC3 （MSTPCRD b25）コンパレータ - OC2 （MSTPCRD b26）コンパレータ - OC1 （MSTPCRD b27）コンパレータ - OC0 （MSTPCRD b28）
- MSTPCRD 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 1 1 1 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
- 上記は MSTPCRD に最適なビット設定です。このため、当社の MSTBCRD マスクは 0xE06FFFF3 となっています
- S124
- 一部の周辺機能は、ソフトウェア スタンバイ モードでも動作するように設定できます。それらの機能がオンで、ウェイクアップ時に復元される場合は、低消費電力を実現するため、スリープ状態に移行する前にそれらの機能がオフになります。このような機能には、SCI0 （MSTPCRB b31）IIC0 （MSTPCRB b09）が含まれます
- それらの機能がオンで、ウェイクアップ時に復元される場合は、低消費電力を実現するため、スリープ状態に移行する前にそれらの機能がオフになります。
- 上記は MSTPCRB に適したビット設定です。このため、当社の MSTBCRB マスクは 0x7FFFFDFF となっています
- D/A コンバータ 0 （MSTPCRD b20）、AGT1 （MSTPCRD b2）AGT0 （MSTPCRD b3）コンパレータ - OC0 （MSTPCRD b28）ローパワー コンパレータ （MSTPCRD b29）
- MSTPCRD 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
- 上記は MSTPCRD に最適なビット設定です。このため、当社の MSTBCRD マスクは 0xCFEFFFF3 となっています
- < LCD を無効化します
- スリープ前コールバックを発行します
- コールバック引数フィールドを入力します
- ユーザー コールバックを呼び出します
- ローパワー モードに移行するようにセットアップします

- スリープモードへの移行
- スリープモードの解除
- 停止されたクロックを復元します
- 元の mstp モードを復元します
- < LCD を再度有効にします
- 指定されたスリープ解除構成ごとに、I/O ポート端子を設定します
- スリープ後コールバックを発行します
- コールバック引数フィールドを入力します
- ユーザー コールバックを呼び出します
- ソフトウェア スタンバイ モードに移行する前に WUPEN コンテンツがあった場所に復元します

6.15.5 SF_POWER_PROFILES_Close

```
ssp_err_t  SF_POWER_PROFILES_Close ( sf_power_profiles_ctrl_t  *const p_ctrl )
```

6.15.5.1 概要説明

このクローズ関数は、ユニットのミューテックスを取得し、ドライバのクローズ関数を呼び出した後、ミューテックスを解放します。

6.15.5.2 詳細説明

表 212: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_POWER_PROFILES_Open を呼び出して構成します。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません(p_ctrl->p_api->close が NULL)。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.15.5.3 関数のステップ

- 制御ブロックの情報を閉じることによって、このブロックが閉じていることを他の関数に示します

6.15.6 SF_POWER_PROFILES_VersionGet

`ssp_err_t` SF_POWER_PROFILES_VersionGet (`ssp_version_t` *const p_version)

6.15.6.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.15.6.2 詳細説明

表 213: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.15.7 モジュール

- ビルドタイム構成

6.15.7.1 ビルドタイム構成

定義

- #define SF_POWER_PROFILES_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.16 SPI フレームワーク

RTOS 統合された SPI フレームワーク。

6.16.1 Functions

- SF_SPI_Open
- SF_SPI_Read
- SF_SPI_Write
- SF_SPI_WriteRead
- SF_SPI_Close
- SF_SPI_Lock
- SF_SPI_Unlock
- SF_SPI_VersionGet

6.16.2 定義

- #define SF_SPI_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_SPI_CODE_VERSION_MINOR

  初期値 :(1)

6.16.3 SF_SPI_Open

`ssp_err_t` SF_SPI_Open (sf_spi_ctrl_t *const p_ctrl , `sf_spi_cfg_t` const *const p_cfg)

6.16.3.1 詳細説明

表 214: 戻り値

名前	説明
SSP_SUCCESS	SPI チャネルが正常に開かれました。
未サポートまたは不正確なモードです。	指定された SPI チャネルが誤っています。
SSP_ERR_INVALID_CHANNEL	指定された SPI チャネルは除外されています。

表 214: 戻り値 (続き)

名前	説明
チャネルは現在動作中です。	SPI チャネルがすでに開かれています。
未サポートまたは不正確なモードです。	SSP_ERR_INVALID_ARGUMENT
SSP_ERR_INVALID_POINTER	SSP_ERR_QUEUE_UNAVAILABLE
SSP_ERR_QUEUE_UNAVAILABLE	SSP_ERR_INVALID_POINTER
Null ポインタが指定されました。	SSP_ERR_INTERNAL
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。

I : この関数はあらゆるチャネルで再入可能です。

l : この関数を呼び出す前に、呼び出し側が制御ブロックをクリアする必要があります。

6.16.3.2 関数のステップ

- 制御対象の chip_select をコピーします
- 制御対象の chip_select レベルをコピーします
- バス ロックを false に初期化します
- フレームワーク レベルのコールバック関数を設定します
- コンテキストを ISR で使用できるように保存します
- デバイスのオープンで、バス チャネルを使用します
- バスの上の最初のデバイスに対してのみ開きます。
- バスの上の最初のデバイスに対してのみ開きます。デバイスでバスの再設定が必要な場合、後で読み取り / 書き込みが発生したときに再設定が行われます。
- このバスに対するミューテックスを作成します。
- イベント フラグを作成します。
- このバスで最後に使用されたデバイス ctrl を割り当てます。
- 再設定に備えて、デバイス設定を保存します

- デバイス状態を Opened に設定します
- デバイス カウントをインクリメントします
- チップ選択を初期化します。

6.16.4 SF_SPI_Read

```
ssp_err_t  SF_SPI_Read (  sf_spi_ctrl_t *const p_ctrl ,  void *const p_dest ,  uint32_t const length ,
 spi_bit_width_t  const bit_width ,  uint32_t const timeout )
```

6.16.4.1 詳細説明

表 215:　戻り値

名前	説明
SSP_SUCCESS	データの読み取りが正常に終了しました。
Null ポインタが指定されました。	NULL ポインタ。
チャネルは現在動作中です。	チャネルが実行中です。
SSP_ERR_TIMEOUT	タイムアウト エラーが発生しました。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。
SSP_ERR_TRANSFER_ABORTED	データ転送が中止されました。
SSP_ERR_MODE_FAULT	モード障害エラーが発生しました。
SSP_ERR_READ_OVF	読み取りオーバーフローが発生しました。
SSP_ERR_PARITY	パリティエラーが発生しました。
SSP_ERR_OVERRUN	オーバーラン エラーが発生しました。
SSP_ERR_UNDEF	不明なエラーが発生しました。

6.16.4.2 関数のステップ

- デバイスが開いているかどうかを確認します
- 転送プロセスを開始します（ロックの確認、再設定の確認、バスの互換性の確認、チップ選択の有効化）
- 読み取りを実行します。

- コールバックがイベント フラグを設定するまで待機します。
- 転送を完了します

6.16.5 SF_SPI_Write

```
ssp_err_t  SF_SPI_Write (  sf_spi_ctrl_t *const p_ctrl ,  void *const p_src ,  uint32_t const length ,
 spi_bit_width_t  const bit_width ,  uint32_t const timeout )
```

6.16.5.1 詳細説明

表 216:　戻り値

名前	説明
SSP_SUCCESS	データの書き込みが正常に終了しました。
Null ポインタが指定されました。	NULL ポインタ。
SSP_ERR_TIMEOUT	タイムアウト エラーが発生しました。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。

6.16.5.2 関数のステップ

- デバイスが開いていることを確認します
- 転送プロセスを開始します （ロックの確認、再設定の確認、バスの互換性の確認、チップ選択の有効化）
- 書き込みを実行します
- 転送を完了します

6.16.6 SF_SPI_WriteRead

```
ssp_err_t  SF_SPI_WriteRead (  sf_spi_ctrl_t *const p_ctrl ,  void *const p_src ,  void *const p_dest ,
 uint32_t const length ,  spi_bit_width_t  const bit_width ,  uint32_t const timeout )
```

6.16.6.1 詳細説明

表 217: 戻り値

名前	説明
SSP_SUCCESS	データの書き込みが正常に終了しました。
Null ポインタが指定されました。	NULL ポインタ。
チャネルは現在動作中です。	チャネルが実行中です。
SSP_ERR_TIMEOUT	タイムアウト エラーが発生しました。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。

6.16.6.2 関数のステップ

- デバイスが開いているかどうかを確認します
- 転送プロセスを開始します （ロックの確認、再設定の確認、バスの互換性の確認、チップ選択の有効化）
- 転送を完了します

6.16.7 SF_SPI_Close

```
ssp_err_t  SF_SPI_Close (  sf_spi_ctrl_t *const p_ctrl )
```

6.16.7.1 詳細説明

表 218: 戻り値

名前	説明
SSP_SUCCESS	SPI チャネルが正常に閉じられました。
Null ポインタが指定されました。	NULL ポインタ。
チャネルは現在動作中です。	使用中エラー。
内部エラーが発生しました。	内部エラー。

I：この関数はあらゆるデバイスで再入可能です。

6.16.7.2 関数のステップ

- デバイスが開いているかどうかを確認します
- close() によってデバイスを閉じる必要があるかどうかを確認します。これが現在のデバイスである場合のみデバイスを閉じます。それ以外の場合、デバイスは再設定時に閉じられているか、open() によって開かれていません。
- これは後にハードウェア レジスタにアクセスするので、ミューテックスを取得します
- ローレベル ドライバを閉じます
- ミューテックスを解放します
- デバイス カウントをデクリメントします
- バスを使用しているデバイスがあるかどうかを確認します
- バスによって使用された RTOS サービスを削除します
- デバイスを閉じた状態に設定します

6.16.8 SF_SPI_Lock

```
ssp_err_t  SF_SPI_Lock (  sf_spi_ctrl_t *const p_ctrl )
```

6.16.8.1 詳細説明

表 219:　戻り値

名前	説明
SSP_SUCCESS	SPI チャネルが正常に閉じられました。
タッチ パネルが設定されていません。	デバイスは開かれていません。
チャネルは現在動作中です。	使用中エラー。
内部エラーが発生しました。	内部エラー。

I：この関数はあらゆるデバイスで再入可能です。

6.16.8.2 関数のステップ

- デバイスが開いているかどうかを確認します
- デバイスがすでにロックされているかどうかを確認します
- このデバイスに対してミューテックスを取得します
- ロック フラグを設定します
- スレーブを有効にします。

6.16.9 SF_SPI_Unlock

```
ssp_err_t  SF_SPI_Unlock (  sf_spi_ctrl_t *const p_ctrl )
```

6.16.9.1 詳細説明

表 220:　戻り値

名前	説明
SSP_SUCCESS	SPI チャネルが正常に閉じられました。
タッチ パネルが設定されていません。	デバイスは開かれていません。
チャネルは現在動作中です。	使用中エラー。
内部エラーが発生しました。	内部エラー。

I：この関数はあらゆるデバイスで再入可能です。

6.16.9.2 関数のステップ

- デバイスが開いているかどうかを確認します
- ロック フラグをクリアします
- ミューテックスを解放して、他のスレッドがバスを使用できるようにします
- スレーブを無効にします

6.16.10 SF_SPI_VersionGet

```
ssp_err_t  SF_SPI_VersionGet ( ssp_version_t  *const p_version )
```

6.16.10.1 詳細説明

表 221: 戻り値

名前	説明
SSP_SUCCESS	正常な復帰。

6.16.10.2 関数のステップ

- エラーを確認します。それ以上のパラメータ チェックは、ドライバ レイヤーで行うことができます。

6.16.11 モジュール

- ビルドタイム構成

6.16.11.1 ビルドタイム構成

定義

- #define SF_SPI_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define  BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータ チェック用のコードを含めるかどうかを指定します。
  BSP_CFG_PARAM_CHECKING_ENABLE に設定した場合は、システムのデフォルト設定が使用されます。1 に設定した場合はパラメータ チェックが含まれます。0 はパラメータ チェックをコンパイル アウトします

6.17 スレッド監視フレームワーク

スレッドの監視機能を提供するフレームワーク モジュール。

スレッドが不正な動作を行った場合に、デバイスをリセットします。このフレームワーク モジュールは、WDT と IWDT HAL モジュールの両方をサポートしています。

6.17.1 使用されるインタフェース

WDT インタフェース

6.17.2 Functions

- SF_THREAD_MONITOR_Thread
- SF_THREAD_MONITOR_Open
- SF_THREAD_MONITOR_Close
- SF_THREAD_MONITOR_ThreadRegister
- SF_THREAD_MONITOR_ThreadUnregister
- SF_THREAD_MONITOR_CountIncrement
- SF_THREAD_MONITOR_VersionGet

6.17.3 定義

- #define SF_THREAD_MONITOR_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_THREAD_MONITOR_CODE_VERSION_MINOR

  初期値 :(1)

6.17.4 SF_THREAD_MONITOR_Thread

SF_THREAD_MONITOR_Thread (ULONG thread_input)

6.17.4.1 概要説明

SF_THREAD_MONITOR_thread() は、周期的に実行されるメインのスレッド監視スレッドです。

6.17.4.2 詳細説明

期間は、ウォッチドッグ ハードウェアのタイムアウト期間から決定されます。このスレッドの期間は、ウォッチドッグ ハードウェアの期間の半分に設定する必要があります。これにより、ウォッチドッグがカウント値の 50% でリフレッシュされます。

表 222: パラメータ

名前	方向	説明
thread_input	複数のビットを書き換えることもできます。	スレッド監視モジュールの制御ブロックのアドレスです。これにより、制御構造体によって管理されたデータを使用し、ハードウェアとインタフェースして、このスレッドを複数回作成することができます。

これは ThreadX スレッドであるため、この関数は復帰しません。

I：この関数は再入可能ではありません。

I：監視対象スレッドが登録されていない場合、監視スレッドがウォッチドッグをリフレッシュ / ティクル / キックすることで、ウォッチドッグによるデバイスのリセットが回避されます。

6.17.5 SF_THREAD_MONITOR_Open

```
ssp_err_t  SF_THREAD_MONITOR_Open ( sf_thread_monitor_ctrl_t  *const p_ctrl ,
 sf_thread_monitor_cfg_t  const *const p_cfg )
```

6.17.5.1 概要説明

p_lower_lvl_wdt パラメータ内でドライバの .open 関数を呼び出します。

6.17.5.2 詳細説明

HAL ドライバを開いて構成した後、SF_THREAD_MONITOR_Open 関数によってスレッド監視スレッドが開始します。このスレッドは、スレッド カウント変数によってスレッド実行をチェックし、実装されたウォッチドッグ タイマ周辺機能をリフレッシュします。

表 223: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、ウォッチドッグ監視スレッドが開始しました。
p_ctrl の可能性があります。	p_ctrl、p_cfg、p_cfg->p_lower_lvl_wdt の各ポインタおよび / または p_cfg->p_lower_lvl_wdt が NULL です。
チャネルは現在動作中です。	スレッド監視がすでに開かれています。
未サポートまたは不正確なモードです。	オープン時にローレベル ウォッチドッグ周辺機器がエラーを返しました。
データ構造体が割り当てられませんでした。	データ構造体が割り当てられませんでした。
SSP_ERR_INVALID_POINTER	ローレベル ドライバの 1 つ以上の設定オプションが無効です。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数は再入可能ではありません。

I：監視対象スレッドが登録されていない場合、監視スレッドがウォッチドッグをリフレッシュすることで、ウォッチドッグによるデバイスのリセットが回避されます。

I：スレッド監視は、SF_THREAD_MONITOR_Close で閉じた後、SF_THREAD_MONITOR_Open を使用して再度呼び出すことができます。. ただし、WDT/IWDT など、基礎となるウォッチドッグ タイマ ハードウェアが終了または停止できない場合、好ましくない結果が生じる可能性があります。

6.17.6 SF_THREAD_MONITOR_Close

ssp_err_t SF_THREAD_MONITOR_Close (sf_thread_monitor_ctrl_t *const p_ctrl)

6.17.6.1 概要説明

スレッド監視スレッドを停止します。すべてのスレッドの登録が解除され、監視が行われなくなります。

6.17.6.2 詳細説明

表 224: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じられ、ウォッチドッグ監視スレッドが停止しました。
p_ctrl の可能性があります。	p_ctrl ポインタが NULL です。
タッチ パネルが設定されていません。	SF_THREAD_MONITOR_Open が呼び出されなかったか、正常に呼び出されませんでした。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数は再入可能ではありません。

I：この関数は、ウォッチドッグ タイマ （WDT や IWDT など）を停止しません。ただし、これらのタイマをリフレッシュするスレッドは停止します。ハードウェア ウォッチドッグがデバイスをリセットしないようにするには、ウォッチドッグを別の場所でリフレッシュする必要があります。

I：スレッド監視は SF_THREAD_MONITOR_Close で閉じることができます。. ただし、WDT/IWDT など、基礎となるウォッチドッグ タイマ ハードウェアが終了または停止できない場合、好ましくない結果が生じる可能性があります。

6.17.7 SF_THREAD_MONITOR_ThreadRegister

```
ssp_err_t  SF_THREAD_MONITOR_ThreadRegister ( sf_thread_monitor_ctrl_t  *const p_ctrl ,
 sf_thread_monitor_counter_min_max_t  const * p_counter_min_max )
```

6.17.7.1 概要説明

ウォッチドッグ監視スレッドによって監視されるスレッドを登録します。

6.17.7.2 詳細説明

このスレッドは、スレッドで想定される最小値と最大値を提供する必要があります。最小値と最大値は、監視モードを使用して判定できます。

表 225: 戻り値

名前	説明
SSP_SUCCESS	スレッドが正常に登録されました。
p_ctrl の可能性があります。	p_ctrl または p_counter_min_max ポインタが NULL です。
SSP_ERR_INSUFFICIENT_SPACE	監視対象のスレッドの配列に十分なエントリが存在しないため、このスレッドに追加できません。sf_thread_moinitor_cfg.h にある THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS の値を上昇させます。
タッチ パネルが設定されていません。	SF_THREAD_MONITOR_Open が呼び出されなかったか、正常に呼び出されませんでした。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数は再入可能です。

6.17.8 SF_THREAD_MONITOR_ThreadUnregister

ssp_err_t SF_THREAD_MONITOR_ThreadUnregister (sf_thread_monitor_ctrl_t *const p_ctrl)

6.17.8.1 概要説明

ウォッチドッグ監視スレッドによる監視からスレッドを除外します。

6.17.8.2 詳細説明

表 226: 戻り値

名前	説明
SSP_SUCCESS	スレッドが正常に登録解除されました。
p_ctrl の可能性があります。	p_ctrl ポインタが NULL です。
タッチ パネルが設定されていません。	SF_THREAD_MONITOR_Open が呼び出されなかったか、正常に呼び出されませんでした。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数は再入可能です。

6.17.9 SF_THREAD_MONITOR_CountIncrement

ssp_err_t SF_THREAD_MONITOR_CountIncrement (sf_thread_monitor_ctrl_t *const p_ctrl)

6.17.9.1 概要説明

スレッドのカウンタを安全にインクリメントします。破損なしに、確実にこのインクリメントを行うために、ミューテックスが使用されます。

6.17.9.2 詳細説明

表 227: 戻り値

名前	説明
SSP_SUCCESS	スレッド カウンタが正常にインクリメントされました。
p_ctrl の可能性があります。	p_ctrl ポインタが NULL です。
SSP_ERR_INSUFFICIENT_SPACE	監視対象のスレッドの配列に十分なエントリが存在しないため、このスレッドに追加できません。
タッチ パネルが設定されていません。	SF_THREAD_MONITOR_Open が呼び出されなかったか、正常に呼び出されませんでした。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

I：この関数は再入可能です。

6.17.10 SF_THREAD_MONITOR_VersionGet

```
ssp_err_t  SF_THREAD_MONITOR_VersionGet ( ssp_version_t  *const p_version )
```

6.17.10.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。versionGet を実装します。

6.17.10.2 詳細説明

表 228: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.17.11 モジュール

- ビルドタイム構成

6.17.11.1 ビルドタイム構成

定義

- #define THREAD_MONITOR_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS

  初期値 :(5)

  スレッド監視によって同時に監視される最大スレッド数を指定します。各スレッドの監視には RAM が必要なので、この値を小さく設定すれば RAM の使用量を抑えることができます。

6.18 CTSU フレームワーク

RTOS 統合された CTSU フレームワーク。

6.18.1 Functions

- SF_TOUCH_CTSU_Open
- SF_TOUCH_CTSU_Read
- SF_TOUCH_CTSU_Close
- SF_TOUCH_CTSU_VersionGet

6.18.2 定義

- #define SF_TOUCH_CTSU_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_TOUCH_CTSU_CODE_VERSION_MINOR

  初期値 :(1)

6.18.3 SF_TOUCH_CTSU_Open

```
ssp_err_t SF_TOUCH_CTSU_Open ( sf_touch_ctsu_ctrl_t *const p_ctrl ,
 sf_touch_ctsu_cfg_t const *const p_cfg )
```

6.18.3.1 概要説明

タッチ CTSU フレームワーク構成します。

6.18.3.2 詳細説明

SF_TOUCH_CTSU_Open 関数はすでに初期化が行われているかどうかを確認し、初期化されていない場合は CTSU HAL レイヤーを初期化し、呼び出しレイヤーのコールバックを保存し内部スレッドを作成します。フレームワークが同じウィジェット レイヤーによってすでに初期化されている場合、エラーを返します。呼び出しウィジェットが初めての呼び出し側である場合、コールバックとコンテキストは内部に格納されます。

表 229: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、HAL ドライバを参照してください。
チャネルは現在動作中です。	フレームワークはこの特定のウィジェットによってすでに初期化されています。
内部エラーが発生しました。	ハードウェアの初期化またはスレッド / セマフォが失敗しました。
SSP_ERR_INVALID_POINTER	入力パラメータがサポートされている範囲から外れていました。

I : 戻り値の説明についてはローレベル ドライバを参照してください。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.18.3.3 関数のステップ

- フレームワークが一つ以上のウィジェット レイヤーによってすでに開かれている場合、これはウィジェット レイヤーからのコールバックがすでに格納されているものと同じかどうかを確認することによって検証されます。
- 呼び出しレイヤーがすでにこのレイヤーを初期化済みかどうかを確認します。true の場合は、同じウィジェット レイヤーがエラーでもう一度初期化関数を呼び出します。渡されたコールバックが NULL の場合、これはウィジェット レイヤーがコールバックを必要としていないことを意味します。
- これがフレームワークを初めてオープンにしたときである場合、要素を NULL に初期化します
- ウィジェット コールバックとコンテキストをフレームワーク コールバック配列に保存します
- コールバックおよび / またはコンテキストを保存するスペースがあるか確認します
- フレームワークが一つ以上のウィジェット レイヤーによって開かれていない場合、低レイヤーを初期化し内部スレッドを作成します
- セマフォを作成し、ローレベル ドライバへのアクセスを制限します

- semaphore_ctsu_scan_complete セマフォを削除します
- エラーを返します
- 制御構造体のローレベルに対しインスタンスを割り当てます
- CTSU HAL ドライバのローカル版を作成し、このファイルで定義されたコールバックを使用するために HAL コールバックを修正できるようにします。
- このファイルの秘密機能へのコールバック関数を変更します。
- 動作用の CTSU を開きます
- 構成とローレベル API 情報を保存します
- ローレベル ドライバを閉じてハードウェア ロックおよびメモリをクリーンアップします
- コールバックをクリアします
- コールバックをクリアします
- セマフォを削除します
- エラーを返します
- 内部 CTSU スレッドをドライブ スキャンに作成します。
- セマフォを削除します
- このフレームワークが前のコールによりすでに初期化済みである場合は、現在の呼び出しレイヤーの p_ctrl を更新します
- 現在のフレームワークでローレベルを閉じることができるよう、ローカルに保存されたローレベル制御へのポインタを、現在の呼び出しフレームワークローレベル制御に保存します
- コールバック インデックスを保存します
- 制御ブロックをオープンとしてマークし、それが有効であることを他のタスクに示します
- ローカル フラグを初期化済みとしてマークします

6.18.4 SF_TOUCH_CTSU_Read

```
ssp_err_t  SF_TOUCH_CTSU_Read ( sf_touch_ctsu_ctrl_t  *const p_ctrl ,  void * p_dest ,
 ctsu_read_t  read_options , ctsu_channel_pair_t  const * channels ,  const uint16_t count )
```

6.18.4.1 概要説明

タッチ CTSU フレームワークからデータを読み取ります。

6.18.4.2 詳細説明

The SF_TOUCH_CTSU_Read 関数により、ローレベル レイヤーから読み戻すことができます。ローレベル データへのアクセスは、スキャンや処理の進行中にデータのリードを防ぐためにセマフォを介して制御されます。

表 230: 戻り値

名前	説明
SSP_SUCCESS	読み取りが正常に完了しました。
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_TOUCH_CTSU_Open を呼び出して構成します。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません（p_ctrl->p_lower_lvl_api->read が NULL）。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.18.4.3 関数のステップ

- TX ステータスを保持します
- CTSU がスキャン中の場合、スキャンの完了を待ってください。
- 低レイヤー ドライバからデータを読み取ります
- セマフォを返します
- 低レイヤー ドライバから変換されたデータを読み取ります。このデータは進行中のスキャンによって更新されないため、データのリードにはセマフォは必要はありません。
- ThreadX エラーがある場合はこれを返すか、またはローレベル ドライバのリターン コードを返します

6.18.5 SF_TOUCH_CTSU_Close

```
ssp_err_t  SF_TOUCH_CTSU_Close ( sf_touch_ctsu_ctrl_t  *const p_ctrl )
```

6.18.5.1 概要説明

この close 関数は、他のウィジェットの登録されたコールバックが他にあるかどうかを確認します。ある場合は、この関数はこのウィジェットのコールバックを単純に NULL に設定し、返します。他に登録された関数がない場合は、この関数は内部スキャン スレッドを終了し、ローレベル close 関数を呼び出します。

6.18.5.2 詳細説明

表 231: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_ctrl->p_api が NULL です。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_TOUCH_CTSU_Open を呼び出して構成します。
内部エラーが発生しました。	ThreadX の呼び出しに失敗しました。
データ構造体が割り当てられませんでした。	この関数は、HAL ドライバではサポートされていません(p_ctrl->p_lower_lvl_api->close が NULL)。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.18.5.3 関数のステップ

- ステータスを保持します
- このウィジェットのコールバックを NULL に設定します
- 制御ブロックの情報を閉じることによって、このブロックが閉じていることを他の関数に示します
- テーブルが、他のウィジェット レイヤーもこのフレームワークを使用していることを示す NULL でないコールバックを格納しているかどうかを確認します。NULL でないコールバックがテーブルにない場合、内部のスレッドは終了し、HAL は閉じます。
- 正常終了を返します
- Thread の削除前にまずセマフォを取得します。TX の実装によっては不要となる場合があります
- 内部スキャン スレッドをまず終了します
- 終了したら、内部スキャン スレッドを削除します
- 低レイヤー ドライバを閉じます
- セマフォを削除します
- 制御ブロックの情報を閉じることによって、このブロックが閉じていることを他の関数に示します
- フレームワーク モジュールが閉じていることを示すローカル フラグをクリアします
- ThreadX エラーがある場合はこれを返すか、またはローレベル ドライバのリターン コードを返します

6.18.6 SF_TOUCH_CTSU_VersionGet

ssp_err_t SF_TOUCH_CTSU_VersionGet (ssp_version_t *const p_version)

6.18.6.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.18.6.2 詳細説明

表 232: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.18.7 モジュール

- ビルドタイム構成

6.18.7.1 ビルドタイム構成

定義

- #define SF_TOUCH_CTSU_CFG_PARAM_CHECKING_ENABLE

  初期値 :(#define BSP_CFG_PARAM_CHECKING_ENABLE)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_TOUCH_CTSU_CFG_MAX_WIDGET_TYPES

  初期値 :(3)

  このフレームワークを使用する上位レベルのウィジェットのレイヤーの最大数を指定します。デフォルトでは、スライダが 2 つとボタンレイヤーが 1 つで、3 つに設定されています。

6.19 CTSU ボタンフレームワーク

RTOS 統合された CTSU ボタン フレームワーク。

6.19.1 Functions

- SF_TOUCH_CTSU_Button_Open
- SF_TOUCH_CTSU_Button_Close
- SF_TOUCH_CTSU_Button_Enable
- SF_TOUCH_CTSU_Button_Disable
- SF_TOUCH_CTSU_Button_VersionGet

6.19.2 定義

- #define SF_TOUCH_CTSU_BUTTON_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_TOUCH_CTSU_BUTTON_CODE_VERSION_MINOR

  初期値 :(1)
- #define SF_TOUCH_CTSU_BUTTON_MAX_CHANNELS

  初期値 :CTSU_MAX_CHANNELS

  サポートされている最大ボタン チャネル
- #define SF_TOUCH_CTSU_BUTTON_BIT_MASK_ARRAY_SIZE

  初期値 :6

6.19.3 SF_TOUCH_CTSU_Button_Open

```
ssp_err_t SF_TOUCH_CTSU_Button_Open ( sf_touch_ctsu_button_ctrl_t  *const p_ctrl ,
 sf_touch_ctsu_button_cfg_t  const *const p_cfg )
```

6.19.3.1 概要説明

タッチ CTSU ボタン フレームワークを構成します。

6.19.3.2 詳細説明

SF_TOUCH_CTSU_Button_Open が構成構造体内にあるすべてのボタンを初期化し、ローレベル フレームワーク モジュールも初期化します。

表 233: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、低ドライバを参照してください。
チャネルは現在動作中です。	フレームワークはこの特定のウィジェットによってすでに初期化されています。

その他のリターン コードや原因については、HAL ドライバを参照してください。

6.19.4 SF_TOUCH_CTSU_Button_Close

```
ssp_err_t  SF_TOUCH_CTSU_Button_Close ( sf_touch_ctsu_button_ctrl_t  * p_ctrl )
```

6.19.4.1 概要説明

タッチ CTSU ボタン フレームワークを閉じます。

6.19.4.2 詳細説明

SF_TOUCH_CTSU_Button_Close はボタン フレームワークを閉じます。

表 234: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。SF_TOUCH_CTSU_Button_Open を呼び出して構成します。

表 234: 戻り値 (続き)

名前	説明
内部エラーが発生しました。	ハードウェアの初期化またはスレッド / ミューテックス / セマフォが失敗しました。

その他のリターン コードや原因については、HAL ドライバを参照してください。

6.19.4.3 関数のステップ

- ローレベル モジュールを閉じます。
- モジュールを未初期化としてマークします

6.19.5 SF_TOUCH_CTSU_Button_Enable

```
ssp_err_t SF_TOUCH_CTSU_Button_Enable ( sf_touch_ctsu_button_ctrl_t * p_ctrl ,
 sf_touch_ctsu_button_id id )
```

6.19.5.1 概要説明

特定のボタンのイベントを有効にします。

6.19.5.2 詳細説明

SF_TOUCH_CTSU_Button_Enable 関数はボタン ステータスをリセットし、そのボタンのアクションからイベントを生成できるようにします。

表 235: 戻り値

名前	説明
SSP_SUCCESS	ボタンの有効化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl のパラメータが NULL である可能性があります。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。SF_TOUCH_CTSU_Button_Open を呼び出して構成します。
内部エラーが発生しました。	ハードウェアの初期化またはスレッド / ミューテックス / セマフォが失敗しました。

その他のリターン コードや原因については、HAL ドライバを参照してください。

6.19.5.3 関数のステップ

- ボタン識別子のリストを反復処理してボタンを見つけ、有効にします

6.19.6 SF_TOUCH_CTSU_Button_Disable

```
ssp_err_t SF_TOUCH_CTSU_Button_Disable ( sf_touch_ctsu_button_ctrl_t * p_ctrl ,
 sf_touch_ctsu_button_id id )
```

6.19.6.1 概要説明

特定のボタンを無効にします。

6.19.6.2 詳細説明

The SF_TOUCH_CTSU_Button_Disable 関数は特定のボタンのイベントを無効にします。

表 236: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl のパラメータが NULL である可能性があります。
タッチ パネルが設定されていません。	ドライバ制御ブロックが無効です。 SF_TOUCH_CTSU_Button_Open を呼び出して構成します。

その他のリターン コードや原因については、HAL ドライバを参照してください。

6.19.6.3 関数のステップ

- ボタン識別子のリストを反復処理してボタンを見つけ、無効にします。

6.19.7 SF_TOUCH_CTSU_Button_VersionGet

```
ssp_err_t SF_TOUCH_CTSU_Button_VersionGet ( ssp_version_t *const p_version )
```

6.19.7.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.19.7.2 詳細説明

表 237: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.19.8 モジュール

- ビルドタイム構成

6.19.8.1 ビルドタイム構成

定義

- #define SF_TOUCH_CTSU_BUTTON_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_TOUCH_CTSU_BUTTON_CFG_USER_SUPPORTED_BUTTONS

  初期値 :(3)

  使用するボタンの最大数を指定します

- #define SF_TOUCH_CTSU_BUTTON_CFG_SHORT_HOLD_DEBOUNCE_MULTIPLIER

  初期値 :(1)

  このコードは、ボタンが押された状態にあると決定された最初の時点から、SHORT_HOLD_DEBOUNCE_MULTIPLIER*debounce_threshold のインターバル後の通知を生成します。

- #define SF_TOUCH_CTSU_BUTTON_CFG_LONG_HOLD_DEBOUNCE_MULTIPLIER

  初期値 :(5)

  このコードは、ボタンが押された状態にあると決定された最初の時点から、LONG_HOLD_DEBOUNCE_MULTIPLIER*debounce_threshold のインターバル後の通知を生成します。注記 : SF_TOUCH_CTSU_BUTTON_CFG_LONG_HOLD_DEBOUNCE_MULTIPLIER > SF_TOUCH_CTSU_BUTTON_CFG_SHORT_HOLD_DEBOUNCE_MULTIPLIER を保持してください

- #define SF_TOUCH_CTSU_BUTTON_CFG_STUCK_IN_DEBOUNCE_MULTIPLIER

  初期値 :(10)

  CapTouch ボタンには、ノイズやその他の障害が原因でタッチの有無に変動が生じる、ジッターが発生する場合があります。このコードはボタンがこの状態にある時間数を監視します。ボタンが SF_TOUCH_CTSU_BUTTON_CFG_STUCK_IN_DEBOUNCE_MULTIPLIER*debounce_threshold を超過した場合、そのボタンの通知イベントが生成されます。すべてのボタンの乗算は同じですが、デバウンス期間は異なる可能性があります。注記 : 監視の対象は、押された状態と放された状態の間の移行のみとなります。

- #define SF_TOUCH_CTSU_BUTTON_CFG_MULTI_TOUCH_ENABLE

  初期値 :(0)

  マルチタッチを有効化 / 無効化します

- #define SF_TOUCH_CTSU_BUTTON_DETECT_STUCK_AT

  初期値 :(0)

  スタックの状態を検出します。

6.20 CTSU スライダフレームワーク

RTOS 統合された CTSU スライダ フレームワーク。

6.20.1 Functions

- SF_TOUCH_CTSU_Slider_VersionGet
- SF_TOUCH_CTSU_Slider_Open
- SF_TOUCH_CTSU_Slider_Close
- SF_TOUCH_CTSU_Slider_Enable
- SF_TOUCH_CTSU_Slider_Disable

6.20.2 定義

- #define SF_TOUCH_CTSU_SLIDER_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_TOUCH_CTSU_SLIDER_CODE_VERSION_MINOR

  初期値 :(1)
- #define SF_TOUCH_CTSU_SLIDER_MAX_CHANNELS

  初期値 :CTSU_MAX_CHANNELS

  サポートされている最大 CTSU チャネル
- #define SF_TOUCH_CTSU_SLIDER_MAX_CHANNELS_PER_SLIDER

  初期値 :10

  2016 年 2 月時点での、Workbench6 に基づきスライダごとにサポートされている最大チャネル
- #define SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE

  初期値 :6

6.20.3 SF_TOUCH_CTSU_Slider_VersionGet

```
ssp_err_t SF_TOUCH_CTSU_Slider_VersionGet ( ssp_version_t *const p_version )
```

6.20.3.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

6.20.3.2 詳細説明

表 238: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.20.4 SF_TOUCH_CTSU_Slider_Open

```
ssp_err_t  SF_TOUCH_CTSU_Slider_Open ( sf_touch_ctsu_slider_ctrl_t  *const p_ctrl ,
 sf_touch_ctsu_slider_cfg_t  const *const p_cfg )
```

6.20.4.1 概要説明

タッチ CTSU スライダ フレームワークを構成します。

6.20.4.2 詳細説明

SF_TOUCH_CTSU_Slider_Open が構成構造体内にあるすべてのスライダ / ホイールを初期化し、ローレベル フレームワーク モジュールも初期化します。

表 239: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、低ドライバを参照してください。
チャネルは現在動作中です。	フレームワークはこの特定のウィジェットによってすでに初期化されています。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.20.4.3 関数のステップ

- アクティブなチャネルを検出します

6.20.5 SF_TOUCH_CTSU_Slider_Close

ssp_err_t SF_TOUCH_CTSU_Slider_Close (sf_touch_ctsu_slider_ctrl_t *const p_ctrl)

6.20.5.1 概要説明

タッチ CTSU スライダ フレームワークを閉じます。

6.20.5.2 詳細説明

SF_TOUCH_CTSU_Slider_Close

表 240: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、低ドライバを参照してください。
タッチ パネルが設定されていません。	フレームワークはこの特定のウィジェットによって初期化されていません。
データ構造体が割り当てられませんでした。	p_ctrl->p_lower_lvl_api または p_ctrl->p_lower_lvl_api->close が NULL でした

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.20.5.3 関数のステップ

- ローレベル モジュールを閉じます。
- モジュールを未初期化としてマークします

6.20.6 SF_TOUCH_CTSU_Slider_Enable

ssp_err_t SF_TOUCH_CTSU_Slider_Enable (sf_touch_ctsu_slider_ctrl_t *const p_ctrl , sf_touch_ctsu_slider_id_t id)

6.20.6.1 概要説明

SF_TOUCH_CTSU_Slider_Enable。

6.20.6.2 詳細説明

表 241: 戻り値

名前	説明
SSP_SUCCESS	有効化が正常に行われました。
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、低ドライバを参照してください。
タッチ パネルが設定されていません。	フレームワークはこの特定のウィジェットによって初期化されていません。
SSP_ERR_INVALID_POINTER	スライダの無効な識別子。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.20.6.3 関数のステップ

- スライダ識別子のリストを反復処理してスライダを見つけ、有効にします

6.20.7 SF_TOUCH_CTSU_Slider_Disable

```
ssp_err_t SF_TOUCH_CTSU_Slider_Disable ( sf_touch_ctsu_slider_ctrl_t *const p_ctrl ,
 sf_touch_ctsu_slider_id_t id )
```

6.20.7.1 概要説明

SF_TOUCH_CTSU_Slider_Disable。

6.20.7.2 詳細説明

表 242: 戻り値

名前	説明
SSP_SUCCESS	無効化が正常に行われました。

表 242: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	p_ctrl または p_api、p_cfg のいずれかのパラメータが NULL である可能性があります。考えられる他の原因については、低ドライバを参照してください。
タッチ パネルが設定されていません。	フレームワークはこの特定のウィジェットによって初期化されていません。
チャネルは現在動作中です。	フレームワークはこの特定のウィジェットによってすでに初期化されています。

その他のリターン コードや原因については、一般的なエラーコードまたは HAL ドライバを参照してください。

6.20.7.3 関数のステップ

- スライダ識別子のリストを反復処理してスライダを見つけ、無効にします

6.20.8 拡張

6.20.8.1 sf_slider_on_ctsu_cfg_t

sf_slider_on_ctsu_cfg_t

詳細説明

Workbench により生成される個々のスライダ構造

変数

- sf_slider_type_tconst type
  線形または円形 （ホイール）
- uint32_t num_slider_channels
- ctsu_channel_pair_t const * p_slider_channels
  スライダを構成するチャネル / チャネル ペアを定義します。
- int32_t const * p_normalization
- int32_t * p_channel_average
- uint32_t * p_offset
- uint16_t *const p_slider_scount

- uint16_t *const p_slider_baseline
- int32_t *const p_slider_delta
- sf_touch_ctsu_slider_id_t id

  スライダの一意の識別子
- int32_t max_slider_value

  スライダの最大値、0 より大きい値でなければならず、最小値は常に 0
- const uint16_t slider_norm_max

  ユーザーにより変更可能な個々のスライダ設定。
  st_sf_touch_ctsu_slider_hdlTOUCH_SLIDER_CFG_NORM_MAX と同一でなければなりません
- const int32_t slider_threshold

  タッチのしきい値。値が 0 より大きいことを確認してください
- const int32_t channel_average_weight

  各チャネルに対するカウントの加重移動平均
- const int32_t prev_sum_weight

  位置計算における前回の合計の加重移動平均 （0 より大きい値でなければなりません）
- const int32_t cutoff

  prev_sum 移動平均をどの程度下回った場合に「SF_TOUCH_SLIDER_STATE_RELEASED」を検出するか定義します

6.20.8.2 st_sf_touch_ctsu_slider_hdl

st_sf_touch_ctsu_slider_hdl

詳細説明

各スライダのハンドル

sf_touch_ctsu_slider_hdl_t への前方参照に必要となります

変数

- uint32_t open

  スライダが開かれていることを示します
- sf_touch_ctsu_slider_id_t id

  スライダの一意の識別子。
- sf_touch_ctsu_slider_state_t state

  スライダの現在の状態を表します。

- sf_slider_type_t type

  線形または円形 （ホイール）
- uint32_t num_slider_channels
- ctsu_channel_pair_t const * p_slider_channels

  スライダを構成するチャネル / チャネル ペアを定義します。
- int32_t const * p_normalization
- int32_t * p_channel_average
- uint32_t * p_offset
- uint16_t * p_slider_scount
- uint16_t * p_slider_baseline
- int32_t * p_slider_delta
- uint32_t position

  計算済みの位置。
- int32_t prev_sum

  位置計算における前回の合計の移動平均の保存に使用します
- int32_t max_slider_value

  スライダの最大値、0 より大きい値でなければならず、最小値は常に 0
- ssp_err_t(* p_update)(*const hdl, const *const p_lower_lvl_touch_framework, uint32_t *p_pos, *const p_state)

  タッチの位置を計算する関数。
- uint64_t bit_mask[SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE]

  関数の更新に使用するビット マスク
- uint16_t slider_norm_max

  ユーザーにより変更可能な個々のスライダ設定。
  sf_slider_on_ctsu_cfg_tTOUCH_SLIDER_CFG_NORM_MAX と同一でなければなりません
- int32_t slider_threshold

  タッチのしきい値。値が 0 より大きいことを確認してください
- int32_t channel_average_weight

  各チャネルに対するカウントの加重移動平均
- int32_t prev_sum_weight

  位置計算における前回の合計の加重移動平均 （0 より大きい値でなければなりません）

- int32_t cutoff

  prev_sum 移動平均をどの程度下回った場合に「SF_TOUCH_SLIDER_STATE_RELEASED」を検出するか定義します

6.20.9 モジュール

- ビルドタイム構成

6.20.9.1 ビルドタイム構成

定義

- #define SF_TOUCH_CTSU_SLIDER_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SF_TOUCH_CTSU_SLIDER_CFG_USER_SUPPORTED_SLIDERS

  初期値 :(2)

  スライダおよびホイールの合計カウントを指定します

- #define SF_TOUCH_CTSU_SLIDER_CFG_MULTI_TOUCH_ENABLE

  初期値 :(1)

  マルチタッチを有効化 / 無効化します

6.21 I2C タッチパネルフレームワーク

外部 I2C タッチ チップのための RTOS 統合されたタッチ パネル フレームワーク実装。

6.21.1 概要

これは、IRQ ピンを持つ外部 I2C タッチ コントローラを使用して、新しいデータが使用可能になったときにアプリケーションに通知するための ThreadX タッチ パネル フレームワーク実装です。

6.21.2 Functions

- SF_TOUCH_PANEL_I2C_Open
- SF_TOUCH_PANEL_I2C_Calibrate
- SF_TOUCH_PANEL_I2C_Start
- SF_TOUCH_PANEL_I2C_Stop
- SF_TOUCH_PANEL_I2C_Reset
- SF_TOUCH_PANEL_I2C_Close
- SF_TOUCH_PANEL_I2C_VersionGet

6.21.3 定義

- #define SF_TOUCH_PANEL_I2C_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SF_TOUCH_PANEL_I2C_CODE_VERSION_MINOR

  初期値 :(1)
- #define SF_TOUCH_PANEL_I2C_RESET_PIN_UNUSED

  初期値 :(0xFFFF)
- #define SF_TOUCH_PANEL_I2C_STACK_SIZE

  初期値 :(1024)

  I2C タッチ パネル スレッドのスタック サイズ。

6.21.4 SF_TOUCH_PANEL_I2C_Open

```
ssp_err_t SF_TOUCH_PANEL_I2C_Open ( sf_touch_panel_ctrl_t *const p_ctrl ,
 sf_touch_panel_cfg_t const *const p_cfg )
```

6.21.4.1 概要説明

open を実装します。

6.21.4.2 詳細説明

表 243: 戻り値

名前	説明
SSP_SUCCESS	作成されたタッチ パネル スレッドおよびローレベル ドライバが正常に開きました。
p_ctrl の可能性があります。	ポインタ パラメータが NULL であったか、ローレベル ドライバがこのエラーをレポートしました。
内部エラーが発生しました。	タッチ パネル スレッドまたはイベント フラグが作成できなかったか、ローレベル ドライバがこのエラーをレポートしました。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるパネルで再入可能です。

6.21.4.3 関数のステップ

- API を格納します。
- ユーザー設定を制御ブロックに格納します。
- 前のイベントを初期化します。
- HW レジスタにアクセスする下位レイヤーを呼び出す前に、ミューテックスを取得します。
- ローレベル I^2C ドライバを開きます。
- タッチ チップをリセットします。
- ローレベル外部 IRQ ドライバを開きます。
- ミューテックスを返します
- 内部通信用のイベント フラグを作成します。

- メインのタッチ パネル スレッドを作成します。
- 制御ブロックをオープンとしてマークし、それが有効であることを他のタスクに示します

6.21.5 SF_TOUCH_PANEL_I2C_Calibrate

ssp_err_t SF_TOUCH_PANEL_I2C_Calibrate (sf_touch_panel_ctrl_t *const p_ctrl , sf_touch_panel_calibrate_t const *const p_expected , sf_touch_panel_payload_t const *const p_actual , ULONG timeout)

6.21.5.1 概要説明

calibrate を実装します。

6.21.5.2 詳細説明

表 244: 戻り値

名前	説明
SSP_SUCCESS	タッチ パネルが正常に較正されました。
p_ctrl の可能性があります。	ポインタ パラメータが NULL であったか、ローレベル ドライバがこのエラーをレポートしました。
SSP_ERR_CALIBRATE_FAILED	タッチの実測値が期待範囲外です。
タッチ パネルが設定されていません。	タッチ パネルが設定されていません。 SF_TOUCH_PANEL_I2C_Open を呼び出してください。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるパネルで再入可能です。

6.21.5.3 関数のステップ

- タイムアウトはこの実装では使用されません。
- タッチ メッセージ データが期待された値と一致しているかどうかを判定します。

6.21.6 SF_TOUCH_PANEL_I2C_Start

ssp_err_t SF_TOUCH_PANEL_I2C_Start (sf_touch_panel_ctrl_t *const p_ctrl)

6.21.6.1 概要説明

start を実装します。

6.21.6.2 詳細説明

表 245: 戻り値

名前	説明
SSP_SUCCESS	タッチ パネルの開始フラグが、タッチ スレッドにポストされました。
タッチ パネルが設定されていません。	タッチ パネルが設定されていません。 SF_TOUCH_PANEL_I2C_Open を呼び出してください。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるパネルで再入可能です。

6.21.6.3 関数のステップ

- 開始フラグを設定します。

6.21.7 SF_TOUCH_PANEL_I2C_Stop

ssp_err_t SF_TOUCH_PANEL_I2C_Stop (sf_touch_panel_ctrl_t *const p_ctrl)

6.21.7.1 概要説明

stop を実装します。

6.21.7.2 詳細説明

表 246: 戻り値

名前	説明
SSP_SUCCESS	タッチ パネルは、タッチ イベントに応答しなくなります。
p_ctrl の可能性があります。	パラメータ p_ctrl が NULL であったか、ローレベル ドライバがこのエラーをレポートしました。
タッチ パネルが設定されていません。	タッチ パネルが設定されていません。 SF_TOUCH_PANEL_I2C_Open を呼び出してください。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるパネルで再入可能です。

6.21.7.3 関数のステップ

- 停止フラグを設定します。

6.21.8 SF_TOUCH_PANEL_I2C_Reset

```
ssp_err_t  SF_TOUCH_PANEL_I2C_Reset ( sf_touch_panel_ctrl_t  *const p_ctrl )
```

6.21.8.1 概要説明

reset を実装します。

6.21.8.2 詳細説明

表 247: 戻り値

名前	説明
SSP_SUCCESS	タッチ パネルおよびローレベル I2C ドライバが正常にリセットされました。
p_ctrl の可能性があります。	パラメータ p_ctrl が NULL であったか、ローレベル ドライバがこのエラーをレポートしました。
チャネルは現在動作中です。	ミューテックスが使用できなかったか、ローレベル ドライバがこのエラーをレポートしました。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数はあらゆるパネルで再入可能です。

6.21.8.3 関数のステップ

- これは共有リソースへのアクセスなので、ミューテックスを取得します。
- ハードウェアに固有のリセット関数を呼び出します。
- ミューテックスを解放します。

6.21.9 SF_TOUCH_PANEL_I2C_Close

```
ssp_err_t SF_TOUCH_PANEL_I2C_Close ( sf_touch_panel_ctrl_t *const p_ctrl )
```

6.21.9.1 概要説明

close を実装します。

6.21.9.2 詳細説明

表 248: 戻り値

名前	説明
SSP_SUCCESS	タッチ パネル インスタンスが正常に閉じられました。
p_ctrl の可能性があります。	パラメータ p_ctrl が NULL であったか、ローレベル ドライバがこのエラーをレポートしました。
タッチ パネルが設定されていません。	タッチ パネルが設定されていません。 SF_TOUCH_PANEL_I2C_Open を呼び出してください。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

I：この関数は再入可能です。

6.21.9.3 関数のステップ

- ローレベル外部 IRQ ドライバを閉じます。
- ローレベル I^2C ドライバを閉じます。
- メッセージング フレームワーク バッファが使用中の場合は、それを解放します。
- 使用された RTOS サービスを削除します
- 制御ブロックをクローズとしてマークします

6.21.10 SF_TOUCH_PANEL_I2C_VersionGet

```
ssp_err_t  SF_TOUCH_PANEL_I2C_VersionGet ( ssp_version_t  *const p_version )
```

6.21.10.1 概要説明

versionGet を実装します。

6.21.10.2 詳細説明

表 249: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に返されました。
p_ctrl の可能性があります。	パラメータ p_version が NULL でした。

6.21.11 拡張

6.21.11.1 sf_touch_panel_i2c_chip_t

sf_touch_panel_i2c_chip_t

詳細説明

チップの定義。

変数

- ssp_err_t(* payloadGet)(*const p_ctrl, *const p_payload)

  タッチ チップを読み取り、タッチ ペイロード データを書き込みます。この制御構造体は、この関数で初期化されます。p_payload データ構造体へのペイロードへのポインタ。提供されたタッチ データは、論理ピクセル値に変換する必要があります。

- ssp_err_t(* reset)(*const p_ctrl)

  タッチ チップをリセットします。この制御構造体は、この関数で初期化されます。

6.21.11.2 sf_touch_panel_i2c_ctrl_t

sf_touch_panel_i2c_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、SF_TOUCH_PANEL_I2C_Open の呼び出し時に実行されます

変数

- ioport_port_pin_t pin

  タッチ コントローラ チップ上のラインをリセットするために接続されたポート ピン。

- i2c_master_instance_t const * p_lower_lvl_i2c

  ローレベル I^2C。

- sf_external_irq_instance_t const * p_lower_lvl_irq

  ローレベル外部 IRQ。

- uint8_t stack[SF_TOUCH_PANEL_I2C_STACK_SIZE]

  タッチ パネル スレッド用のスタック。

- sf_touch_panel_i2c_chip_t const * p_chip

  チップ固有の関数および定義。

6.21.11.3 sf_touch_panel_i2c_cfg_t

sf_touch_panel_i2c_cfg_t

詳細説明

RTOS タッチ パネル ドライバの設定。

変数

- i2c_master_instance_t const * p_lower_lvl_i2c

  ローレベル I^2C へのポインタ。

- sf_external_irq_instance_t const * p_lower_lvl_irq

  ローレベル外部 IRQ へのポインタ。

- sf_touch_panel_i2c_ctrl_t *const p_lower_lvl_ctrl

  ローレベル制御ブロックに対して割り当てられたメモリへのポインタ。初期化しないでください。初期化は、SF_TOUCH_PANEL_I2C_Open の呼び出し時に実行されます。

- ioport_port_pin_t pin

  タッチ コントローラ チップ上のラインをリセットするために接続されたポート ピン。使用しない場合は、SF_TOUCH_PANEL_I2C_RESET_PIN_UNUSED に設定されます。

- sf_touch_panel_i2c_chip_t const *const p_chip

  選択されたタッチ コントローラ チップ。

6.21.12 モジュール

- ビルドタイム構成

6.21.12.1 ビルドタイム構成

定義

- #define SF_TOUCH_PANEL_I2C_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

6.22 UART フレームワークインスタンス

RTOS 統合された通信フレームワーク UART 実装。

6.22.1 Functions

- SF_UART_COMMS_Open
- SF_UART_COMMS_Close
- SF_UART_COMMS_Read
- SF_UART_COMMS_Write
- SF_UART_COMMS_Lock
- SF_UART_COMMS_Unlock
- SF_UART_COMMS_VersionGet

6.22.2 定義

- #define SF_UART_COMMS_CODE_VERSION_MAJOR

  初期値 :(1)

  このファイルで定義されている API を実装するコードのバージョン
- #define SF_UART_COMMS_CODE_VERSION_MINOR

  初期値 :(1)

6.22.3 SF_UART_COMMS_Open

`ssp_err_t` SF_UART_COMMS_Open (`sf_comms_ctrl_t` *const p_ctrl , `sf_comms_cfg_t` const *const p_cfg)

6.22.3.1 詳細説明

表 250: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開かれました。
チャネルは現在動作中です。	チャネルはすでに使用中です。

表 250: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	UART 制御ブロックまたは設定構造体へのポインタが NULL です。
SSP_ERR_INVALID_ARGUMENT	チャネルがロックされています。
未サポートまたは不正確なモードです。	チャネルが非 UART モード用に使用されているか、設定されているモードが誤っています。
SSP_ERR_INVALID_POINTER	設定構造体に無効なパラメータ設定値が見つかりました。
SSP_ERR_QUEUE_UNAVAILABLE	送信キューと受信キューのいずれかまたは両方が開かれていません。
内部エラーが発生しました。	この関数はあらゆるチャネルで再入可能です。

I：この関数はあらゆるチャネルで再入可能です。この関数を呼び出す前に、呼び出し側がハンドルをクリアする必要があります。

6.22.3.2 関数のステップ

- エラーを確認します。それ以上のパラメータ チェックは、ドライバ レイヤーで行うことができます。
- UART ミューテックスを作成し、UART コンテキストに挿入します
- UART イベント フラグを作成し、UART コンテキストに挿入します
- リード データをバッファするためのキューを作成します。
- UART HAL ドライバのオープン関数を呼び出します

6.22.4 SF_UART_COMMS_Close

```
ssp_err_t  SF_UART_COMMS_Close ( sf_comms_ctrl_t  *const p_ctrl )
```

6.22.4.1 詳細説明

表 251: 戻り値

名前	説明
SSP_SUCCESS	UART チャネルが正常に閉じられました。
p_ctrl の可能性があります。	Null ポインタが提供されました。
SSP_ERR_TIMEOUT	タイムアウトエラー。
内部エラーが発生しました。	内部エラー。

I：この関数はあらゆるチャネルで再入可能です。

6.22.4.2 関数のステップ

- エラーを確認します。それ以上のパラメータ チェックは、ドライバ レイヤーで行うことができます。
- UART HAL ドライバのクローズ関数を呼び出します
- RTOS オブジェクトを削除します。

6.22.5 SF_UART_COMMS_Read

```
ssp_err_t  SF_UART_COMMS_Read ( sf_comms_ctrl_t  *const p_ctrl ,  uint8_t *const p_dest ,
 uint32_t const bytes ,  UINT const timeout )
```

6.22.5.1 詳細説明

表 252: 戻り値

名前	説明
SSP_SUCCESS	データの読み取りが正常に終了しました。
SSP_ERR_INVALID_ARGUMENT	チャネルがロックされています。

表 252: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
未サポートまたは不正確なモードです。	チャネルが非 UART モード用に使用されています。
SSP_ERR_INVALID_POINTER	宛先アドレスまたはデータ サイズが、データ長に対して無効です。
SSP_ERR_INSUFFICIENT_DATA	受信循環バッファに十分なデータがありません。
SSP_ERR_RXBUF_OVERFLOW	キューオーバーフローを受け取ります。
SSP_ERR_OVERFLOW	ハードウェアオーバフロー。
SSP_ERR_FRAMING	フレーミング エラー。
SSP_ERR_PARITY	パリティ エラー。
SSP_ERR_TIMEOUT	タイムアウトエラー。
内部エラーが発生しました。	内部エラー。

6.22.5.2 関数のステップ

- エラーを確認します。それ以上のパラメータ チェックは、ドライバ レイヤーで行うことができます。
- UART ハードウェア リソースをロックします
- UART HAL ドライバの読み取り関数を呼び出します
- リード動作が完了するまで待機します。イベントは、イベント フラグ オブジェクトによって通知されます。
- UART ハードウェア リソースをロック解除します

6.22.6 SF_UART_COMMS_Write

```
ssp_err_t  SF_UART_COMMS_Write ( sf_comms_ctrl_t  *const p_ctrl ,  uint8_t const *const p_src ,
 uint32_t const bytes ,  UINT const timeout )
```

6.22.6.1 詳細説明

表 253: 戻り値

名前	説明
SSP_SUCCESS	データの送信が正常に終了しました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
未サポートまたは不正確なモードです。	チャネルが非 UART モード用に使用されているか、ハンドルで設定されているモードが誤っています。
SSP_ERR_INVALID_POINTER	ソース アドレスまたはデータ サイズが、データ長に対して無効です。
SSP_ERR_INSUFFICIENT_SPACE	伝送循環バッファに十分なスペースがありません。
SSP_ERR_INVALID_ARGUMENT	ハードウェアをロックできませんでした。
SSP_ERR_TIMEOUT	タイムアウトエラー。
内部エラーが発生しました。	内部エラー。

6.22.6.2 関数のステップ

- エラーを確認します。それ以上のパラメータ チェックは、ドライバ レイヤーで行うことができます。
- UART ハードウェア リソースをロックします
- イベント フラグをクリアします。
- UART HAL ドライバの書き込み関数を呼び出します
- 書き込み操作が完了するまで待機します。イベントは、イベント フラグ オブジェクトによって通知されます
- UART ハードウェア リソースをロック解除します

6.22.7 SF_UART_COMMS_Lock

```
ssp_err_t  SF_UART_COMMS_Lock ( sf_comms_ctrl_t  *const p_ctrl ,
 sf_comms_lock_t   lock_type ,  UINT timeout )
```

6.22.7.1 詳細説明

表 254: 戻り値

名前	説明
SSP_SUCCESS	UART リソースが正常にロックされました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
SSP_ERR_TIMEOUT	ミューテックスをタイムアウト内に使用できません。
チャネルは現在動作中です。	チャネルが使用中です。

6.22.8 SF_UART_COMMS_Unlock

```
ssp_err_t  SF_UART_COMMS_Unlock ( sf_comms_ctrl_t  *const p_ctrl ,
 sf_comms_lock_t   lock_type )
```

6.22.8.1 詳細説明

表 255: 戻り値

名前	説明
SSP_SUCCESS	UART リソースが正常にロック解除されました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
チャネルは現在動作中です。	チャネルが使用中です。

6.22.9 SF_UART_COMMS_VersionGet

```
ssp_err_t  SF_UART_COMMS_VersionGet ( ssp_version_t  *const p_version )
```

6.22.9.1 詳細説明

表 256: 戻り値

名前	説明
SSP_SUCCESS	バージョン番号が正常に取得されました。
p_ctrl の可能性があります。	

6.22.10 API データ

6.22.10.1 sf_uart_comms_state_t

sf_uart_comms_state_t

詳細説明

フレームワーク UART の状態

列挙値

名前	説明
SF_UART_COMMS_STATE_CLOSED	UART ポートは閉じています。
SF_UART_COMMS_STATE_OPENED	UART ポートは開いています。
SF_UART_COMMS_STATE_READING	UART ポートはデータを受信中です。
SF_UART_COMMS_STATE_WRITING	UART ポートはデータを送信中です。

6.22.11 拡張

6.22.11.1 sf_uart_comms_ctrl_t

sf_uart_comms_ctrl_t

詳細説明

UART デバイス コンテキスト。初期化しないでください。初期化は、SF_UART_COMMS_Open の呼び出し時に実行されます

変数

- uint32_t state

  UART のステータス。

- uart_api_t * p_lower_lvl_api

  UART インタフェースへのポインタ （cfg からコピー）。

- uart_ctrl_t uart_ctrl

  UART ペリフェラル制御ブロック。

- TX_MUTEX mutex[2]

  UART リソース相互排除のためのミューテックス オブジェクトへのポインタ。

- TX_EVENT_FLAGS_GROUP eventflag[2]

  UART データ転送用のイベント フラグ オブジェクトへのポインタ。

- TX_QUEUE queue

  リード用のキュー。

- uint32_t queue_mem[SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS]

  キューのメモリ。

6.22.11.2 sf_uart_comms_cfg_t

sf_uart_comms_cfg_t

詳細説明

RTOS 統合された UART ドライバの設定

変数

- uart_instance_t const * p_lower_lvl_uart

  UART ドライバ インスタンスへのポインタ

- sf_uart_comms_ctrl_t * p_ctrl

  制御ブロックに割り当てられたメモリへのポインタ。制御構造体は初期化しないでください。

6.22.12 モジュール

- ビルドタイム構成

6.22.12.1 ビルドタイム構成

定義

- #define SF_UART_COMMS_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define  BSP_CFG_PARAM_CHECKING_ENABLE`)
- #define SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS

  初期値 :(5)

第 7 章 API リファレンス : HAL インタフェース

HAL インタフェースは、関数ユースケース用に共通の API を提供します。1 つまたは複数の HAL レイヤー ドライバによって実装できます。フレームワーク レイヤー ドライバは、インタフェース レイヤーを通じて HAL ドライバに接続します。

- ADC インタフェース
- CAC インタフェース
- CAN インタフェース
- CGC インタフェース
- CRC インタフェース
- 暗号インタフェース
- CTSU インタフェース
- DAC インタフェース
- ディスプレイインタフェース
- DOC インタフェース
- ELC インタフェース
- 外部 IRQ インタフェース
- フラッシュインタフェース
- FMI インタフェース
- I2C インタフェース
- I2S インタフェース
- 入力キャプチャインタフェース
- I/O ポートインタフェース
- JPEG デコードインタフェース
- Key Matrix インタフェース
- 低電力モードインタフェース
- 低電圧検出ドライバインタフェース
- PDC インタフェース

参考資料

- クワッド SPI フラッシュインタフェース
- RTC インタフェース
- SDMMC インタフェース
- SLCDC インタフェース
- SPI インタフェース
- タイマインタフェース
- 転送インタフェース
- UART インタフェース
- WDT インタフェース

7.1 ADC インタフェース

A/D コンバータのインタフェース。

7.1.1 概要

ADC インタフェースは、ワンショット モード （シングル スキャン）、連続スキャン、およびグループ スキャンを含む標準の ADC 機能を提供します。このインタフェースを使用すれば、ハードウェアとソフトウェアの設定でスキャンの開始をトリガーすることもできます。変換が完了するたびに割り込みをトリガーすることができ、コールバック関数が提供されていれば、適切なイベント情報とともにコールバックが呼び出されます。

以下によって実装されます。ADC

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

ADC インタフェースの説明 : ADC ドライバ

7.1.2 インタフェース API

adc_api_t

関数名	説明
.open	電源を適用し、すべてのチャネルとセンターに共通の動作モード、トリガー ソース、割り込みの優先度、および設定値を指定します。
.scanCfg	open 呼び出しで初期化されたユニットに使用されるチャネル、グループ、およびスキャン トリガーを含むスキャンを設定します。
.scanStart	スキャンを開始する （ソフトウェア トリガーの場合）か、ハードウェア トリガーを有効にします。
.scanStop	ADC スキャンを停止する （ソフトウェア トリガーの場合）か、ハードウェア トリガーを無効にします。
.scanStatusGet	スキャン ステータスをチェックします。
.read	ADC 変換結果を読み取ります。

関数名	説明
.sampleStateCountSet	指定されたチャネルのサンプル状態カウントを設定します。
.close	進行中のスキャンを停止して、割り込みを無効にし、指定された A/D ユニットへの電源を遮断することによって、指定された ADC ユニットを閉じます。
.infoGet	構成されたすべてのチャネルの変換結果を DTC/DMAC によって読み取るために、先頭 （最も小さい番号の）チャネルの ADC データのレジスタアドレスと読み取る合計バイト数を返します。
.versionGet	API バージョンを取得します。

7.1.3 データ構造体

- adc_sample_state_t
- adc_callback_args_t
- adc_info_t
- adc_channel_cfg_t
- adc_cfg_t
- adc_ctrl_t
- adc_instance_t

7.1.4 列挙

- adc_mode_t
- adc_resolution_t
- adc_alignment_t
- adc_add_t
- adc_clear_t
- adc_trigger_t
- adc_sample_state_reg_t
- adc_cb_event_t
- adc_group_a_t

- adc_register_t

7.1.5 型定義

- adc_data_size_t

7.1.6 定義

- #define ADC_API_VERSION_MAJOR

  初期値 :(1)

  ボードと MCU 関連のヘッダー ファイルを追加します。API のバージョン番号。
- #define ADC_API_VERSION_MINOR

  初期値 :(1)

7.1.7 API データ

7.1.7.1 adc_mode_t

adc_mode_t

詳細説明

ADC 動作モードの定義

列挙値

名前	説明
ADC_MODE_SINGLE_SCAN	シングル スキャン - 1 つ以上のチャネル。
ADC_MODE_CONTINUOUS_SCAN	連続スキャン - 1 つ以上のチャネル。
ADC_MODE_GROUP_SCAN	1 つ以上のチャネルを含む 2 つのグループに対するスキャンをトリガーするための 2 つのトリガー ソース。

7.1.7.2 adc_resolution_t

adc_resolution_t

詳細説明

ADC データ分解能の定義

列挙値

名前	説明
ADC_RESOLUTION_12_BIT	12 ビットの分解能
ADC_RESOLUTION_10_BIT	10 ビットの分解能
ADC_RESOLUTION_8_BIT	8 ビットの分解能
ADC_RESOLUTION_14_BIT	14 ビットの分解能

7.1.7.3 adc_alignment_t

adc_alignment_t

詳細説明

ADC データ配置の定義

列挙値

名前	説明
ADC_ALIGNMENT_RIGHT	データ右揃え。
ADC_ALIGNMENT_LEFT	データ左揃え。

7.1.7.4 adc_add_t

adc_add_t

詳細説明

ADC データ サンプルの追加オプションと平均化オプション

列挙値

名前	説明
ADC_ADD_OFF	チャネル / センサーの追加がオフになります。
ADC_ADD_TWO	2 つのサンプルを追加します。

名前	説明
ADC_ADD_THREE	3 つのサンプルを追加します。
ADC_ADD_FOUR	4 つのサンプルを追加します。
ADC_ADD_AVERAGE_TWO	2 つのサンプルを平均化します。
ADC_ADD_AVERAGE_FOUR	4 つのサンプルを平均化します。
ADC_ADD_AVERAGE_SIXTEEN	16 のサンプルを平均化します。

7.1.7.5 adc_clear_t

adc_clear_t

詳細説明

ADC 読み取り後クリアの定義

列挙値

名前	説明
ADC_CLEAR_AFTER_READ_OFF	読み取り後クリア オフ。
ADC_CLEAR_AFTER_READ_ON	読み取り後クリア オン。

7.1.7.6 adc_trigger_t

adc_trigger_t

詳細説明

ADC トリガー モードの定義

列挙値

名前	説明
ADC_TRIGGER_ASYNC_EXT_TRG0	外部非同期トリガー。グループ モード以外。
ADC_TRIGGER_SYNC_ELC	ELC 経由の同期トリガー。
ADC_TRIGGER_SOFTWARE	ソフトウェア トリガー。グループ モード以外。

7.1.7.7 adc_sample_state_reg_t

adc_sample_state_reg_t

詳細説明

ADC サンプル状態レジスタ

列挙値

名前	説明
ADC_SAMPLE_STATE_CHANNEL_0	サンプル状態レジスタ チャネル 0。
ADC_SAMPLE_STATE_CHANNEL_1	サンプル状態レジスタ チャネル 1。
ADC_SAMPLE_STATE_CHANNEL_2	サンプル状態レジスタ チャネル 2。
ADC_SAMPLE_STATE_CHANNEL_3	サンプル状態レジスタ チャネル 3。
ADC_SAMPLE_STATE_CHANNEL_4	サンプル状態レジスタ チャネル 4。
ADC_SAMPLE_STATE_CHANNEL_5	サンプル状態レジスタ チャネル 5。
ADC_SAMPLE_STATE_CHANNEL_6	サンプル状態レジスタ チャネル 6。
ADC_SAMPLE_STATE_CHANNEL_7	サンプル状態レジスタ チャネル 7。
ADC_SAMPLE_STATE_CHANNEL_8	サンプル状態レジスタ チャネル 8。
ADC_SAMPLE_STATE_CHANNEL_9	サンプル状態レジスタ チャネル 9。
ADC_SAMPLE_STATE_CHANNEL_10	サンプル状態レジスタ チャネル 10。
ADC_SAMPLE_STATE_CHANNEL_11	サンプル状態レジスタ チャネル 11。
ADC_SAMPLE_STATE_CHANNEL_12	サンプル状態レジスタ チャネル 12。
ADC_SAMPLE_STATE_CHANNEL_13	サンプル状態レジスタ チャネル 13。
ADC_SAMPLE_STATE_CHANNEL_14	サンプル状態レジスタ チャネル 14。
ADC_SAMPLE_STATE_CHANNEL_15	サンプル状態レジスタ チャネル 15。
ADC_SAMPLE_STATE_CHANNEL_16_TO_21	S7G2 上のユニット 0 のサンプル状態レジスタ チャネル 16 ～ 21。

名前	説明
ADC_SAMPLE_STATE_CHANNEL_16_TO_20	S7G2 上のユニット 1 のサンプル状態レジスタ チャネル 16 ～ 20。
ADC_SAMPLE_STATE_CHANNEL_16_TO_27	S3A7 上のユニット 0 のサンプル状態レジスタ チャネル 16 ～ 27。
ADC_SAMPLE_STATE_TEMPERATURE	サンプル状態レジスタ チャネル温度。
ADC_SAMPLE_STATE_VOLTAGE	サンプル状態レジスタ チャネル電圧。

7.1.7.8 adc_cb_event_t

adc_cb_event_t

詳細説明

ADC コールバック イベントの定義

列挙値

名前	説明
ADC_EVENT_SCAN_COMPLETE	標準 / グループ A スキャン完了。
ADC_EVENT_SCAN_COMPLETE_GROUP_B	グループ B スキャン完了。

7.1.7.9 adc_group_a_t

adc_group_a_t

詳細説明

グループ A の ADC アクションがグループ B スキャンを中断します。この列挙は、グループ モードでのグループ A と B の優先順位を指定するために使用されます。

列挙値

名前	説明
ADC_GROUP_A_PRIORITY_OFF	グループ A は無視され、実行中であるグループ B のスキャンは中断されません。

名前	説明
ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER	グループ B （シングル スキャン）がグループ A により中断され、その後次のグループ B トリガにおいて再開します。
ADC_GROUP_A_GROUP_B_RESTART_SCAN	グループ B （シングル スキャン）がグループ A により中断され、グループ A のスキャンが完了した直後に再開します。
ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN	グループ B （シングル スキャン）がグループ A により中断され、新たなグループ B のトリガなしでもスキャンが続行されます。

7.1.7.10 adc_register_t

adc_register_t

詳細説明

パラメータ p_data

列挙値

名前	説明
ADC_REG_CHANNEL_0	ADC チャネル 0。
ADC_REG_CHANNEL_1	ADC チャネル 1。
ADC_REG_CHANNEL_2	ADC チャネル 2。
ADC_REG_CHANNEL_3	ADC チャネル 3。
ADC_REG_CHANNEL_4	ADC チャネル 4。
ADC_REG_CHANNEL_5	ADC チャネル 5。
ADC_REG_CHANNEL_6	ADC チャネル 6。
ADC_REG_CHANNEL_7	ADC チャネル 7。
ADC_REG_CHANNEL_8	ADC チャネル 8。
ADC_REG_CHANNEL_9	ADC チャネル 9。

名前	説明
ADC_REG_CHANNEL_10	ADC チャネル 10。
ADC_REG_CHANNEL_11	ADC チャネル 11。
ADC_REG_CHANNEL_12	ADC チャネル 12。
ADC_REG_CHANNEL_13	ADC チャネル 13。
ADC_REG_CHANNEL_14	ADC チャネル 14。
ADC_REG_CHANNEL_15	ADC チャネル 15。
ADC_REG_CHANNEL_16	ADC チャネル 16。
ADC_REG_CHANNEL_17	ADC チャネル 17。
ADC_REG_CHANNEL_18	ADC チャネル 18。
ADC_REG_CHANNEL_19	ADC チャネル 19。
ADC_REG_CHANNEL_20	ADC チャネル 20。
ADC_REG_CHANNEL_21	ADC チャネル 21 （ユニット 0 のみ）
ADC_REG_CHANNEL_22	ADC チャネル 22。
ADC_REG_CHANNEL_23	ADC チャネル 23。
ADC_REG_CHANNEL_24	ADC チャネル 24。
ADC_REG_CHANNEL_25	ADC チャネル 25。
ADC_REG_CHANNEL_26	ADC チャネル 26。
ADC_REG_CHANNEL_27	ADC チャネル 27。
ADC_REG_TEMPERATURE	ADC チャネル温度。
ADC_REG_VOLT	ADC チャネル電圧。

7.1.7.11 adc_data_size_t

typedef uint16_t adc_data_size_t

詳細説明

最大サポート ADC 分解能サイズ。

7.1.8 API 構造

7.1.8.1 adc_sample_state_t

adc_sample_state_t

詳細説明

ADC サンプル状態設定

変数

- adc_sample_state_reg_t reg_id
  サンプル状態レジスタ ID。
- uint8_t num_states
  変換のサンプリング状態数。Ch16-20/21 は、同じ値を使用します。

7.1.8.2 adc_callback_args_t

adc_callback_args_t

詳細説明

ADC コールバック引数の定義

変数

- uint16_t unit
  使用中の ADC デバイス。
- adc_cb_event_t event
  ADC コールバック イベント。
- void const * p_context
  ユーザー データのプレースホルダー。

7.1.8.3 adc_info_t

adc_info_t

詳細説明

転送インタフェースの ADC 情報構造体へのポインタ

変数

- uint16_t * p_address

  データの読み取りを開始するアドレス。

- uint32_t length

  読み取る合計バイト数。

- elc_peripheral_t elc_peripheral

  ELC リストの周辺機能名。

- elc_event_t elc_event

  周辺機能の ELC イベント名。

7.1.8.4 adc_channel_cfg_t

adc_channel_cfg_t

詳細説明

変数

変数

- uint32_t scan_mask

  ビット 0 は ch0、ビット 15 は ch15 です。r_adc.h の #define ADC_MASK_xxx を使用します。

- uint32_t scan_mask_group_b

  グループ モードで有効です。r_adc.h の #define ADC_MASK_xxx を使用します。

- adc_group_a_t priority_group_a

  グループ モードで有効です。

- uint32_t add_mask

  Open() で追加が有効な場合に有効です。r_adc.h の #define ADC_MASK_xxx を使用します。

- uint8_t sample_hold_mask

  チャネル / ビット 0-2。r_adc.h の #define ADC_MASK_xxx を使用します。

- uint8_t sample_hold_states

  サンプル アンド ホールドに使用される状態数。チャネル 0-2 に影響を与えます。

7.1.8.5 adc_cfg_t

adc_cfg_t

詳細説明

ADC の一般的な設定

変数

- uint16_t unit

  使用される ADC ユニット。

- adc_mode_t mode

  8、10、12 ビットの ADC 分解能。

- adc_resolution_t resolution

  追加が使用される場合は無視されます。

- adc_alignment_t alignment

  追加が使用される場合は無視されます。

- adc_add_t add_average_count

  サンプルを追加または平均化します。

- adc_clear_t clearing

  読み取り後クリアします。

- adc_trigger_t trigger

  デフォルトおよびグループ A のトリガー ソース。

- adc_trigger_t trigger_group_b

  グループ モードでのみ有効です。

- void(* p_callback)(adc_callback_args_t *p_args)

  使用しない場合は NULL に設定します。

- void const * p_context

  ユーザー データのプレースホルダー。adc_api_t::adc_callback_args_t 内のユーザー コールバックに渡されます。

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.1.8.6 adc_ctrl_t

adc_ctrl_t

詳細説明

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

変数

- uint16_t unit

  使用中の ADC 単位。

- void const * p_context

  ユーザー データのプレースホルダー。

7.1.8.7 adc_api_t

adc_api_t

詳細説明

HAL レイヤーに実装された ADC 関数は、この API に従います。

7.1.8.8 open

ssp_err_t(* adc_api_t::open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

詳細説明

電源を適用し、すべてのチャネルとセンターに共通の動作モード、トリガー ソース、割り込みの優先度、および設定値を指定します。また電力消費を低減できます。

- R_ADC_Open

  I：この関数を呼び出す前に、ペリフェラルのクロック、ADC ピン、および IRQ を設定します。

表 257: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

定義 :

定義 : adc_cfg_t const *const p_cfg

ADC の一般的な設定

- adc_cfg_t::unit

  使用される ADC ユニット。

- adc_cfg_t::adc_mode_t

  8、10、12 ビットの ADC 分解能。

  列挙値 :

  – ADC_MODE_SINGLE_SCAN

  – ADC_MODE_CONTINUOUS_SCAN

  – ADC_MODE_GROUP_SCAN

- adc_cfg_t::adc_resolution_t

  追加が使用される場合は無視されます。

  列挙値 :

  – ADC_RESOLUTION_12_BIT

  – ADC_RESOLUTION_10_BIT

  – ADC_RESOLUTION_8_BIT

  – ADC_RESOLUTION_14_BIT

- adc_cfg_t::adc_alignment_t

  追加が使用される場合は無視されます。

  列挙値 :

  – ADC_ALIGNMENT_RIGHT

  – ADC_ALIGNMENT_LEFT

- adc_cfg_t::adc_add_t

  サンプルを追加または平均化します。

  列挙値 :

  – ADC_ADD_OFF

  – ADC_ADD_TWO

  – ADC_ADD_THREE

  – ADC_ADD_FOUR

  – ADC_ADD_AVERAGE_TWO

  - ADC_ADD_AVERAGE_FOUR
  - ADC_ADD_AVERAGE_SIXTEEN
- adc_cfg_t::adc_clear_t

  読み取り後クリアします。

  列挙値 :
  - ADC_CLEAR_AFTER_READ_OFF
  - ADC_CLEAR_AFTER_READ_ON
- adc_cfg_t::adc_trigger_t

  デフォルトおよびグループ A のトリガー ソース。

  列挙値 :
  - ADC_TRIGGER_ASYNC_EXT_TRG0
  - ADC_TRIGGER_SYNC_ELC
  - ADC_TRIGGER_SOFTWARE
- adc_cfg_t::adc_trigger_t

  グループ モードでのみ有効です。

  列挙値 :
  - ADC_TRIGGER_ASYNC_EXT_TRG0
  - ADC_TRIGGER_SYNC_ELC
  - ADC_TRIGGER_SOFTWARE
- adc_cfg_t::p_callback

  使用しない場合は NULL に設定します。
- adc_cfg_t::p_context

  ユーザー データのプレースホルダー。adc_api_t::adc_callback_args_t 内のユーザー コールバックに渡されます。
- adc_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.1.8.9 scanCfg

ssp_err_t(* adc_api_t::scanCfg)(adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)

詳細説明

open 呼び出しで初期化されたユニットに使用されるチャネル、グループ、およびスキャン トリガーを含むスキャンを設定します。また電力消費を低減できます。

- R_ADC_ScanConfigure

表 258: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_channel_cfg	複数のビットを書き換えることもできます。	スキャン設定構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

パラメータ p_channel_cfg

定義 : adc_channel_cfg_t const *const p_channel_cfg

変数

- adc_channel_cfg_t::scan_mask

  ビット 0 は ch0、ビット 15 は ch15 です。r_adc.h の #define ADC_MASK_xxx を使用します。
- adc_channel_cfg_t::scan_mask_group_b

  グループ モードで有効です。r_adc.h の #define ADC_MASK_xxx を使用します。
- adc_channel_cfg_t::adc_group_a_t

  グループ モードで有効です。

  列挙値 :
  - ADC_GROUP_A_PRIORITY_OFF
  - ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER
  - ADC_GROUP_A_GROUP_B_RESTART_SCAN
  - ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN
- adc_channel_cfg_t::add_mask

  Open() で追加が有効な場合に有効です。r_adc.h の #define ADC_MASK_xxx を使用します。
- adc_channel_cfg_t::sample_hold_mask

  チャネル / ビット 0-2。r_adc.h の #define ADC_MASK_xxx を使用します。

- adc_channel_cfg_t::sample_hold_states

  サンプル アンド ホールドに使用される状態数。チャネル 0-2 に影響を与えます。

7.1.8.10 scanStart

ssp_err_t(* adc_api_t::scanStart)(adc_ctrl_t *const p_ctrl)

詳細説明

スキャンを開始する （ソフトウェア トリガーの場合）か、ハードウェア トリガーを有効にします。また電力消費を低減できます。

- R_ADC_ScanStart

表 259: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

7.1.8.11 scanStop

ssp_err_t(* adc_api_t::scanStop)(adc_ctrl_t *const p_ctrl)

詳細説明

ADC スキャンを停止する （ソフトウェア トリガーの場合）か、ハードウェア トリガーを無効にします。また電力消費を低減できます。

- R_ADC_ScanStop

表 260: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

7.1.8.12 scanStatusGet

ssp_err_t(* adc_api_t::scanStatusGet)(adc_ctrl_t *const p_ctrl)

詳細説明

スキャン ステータスをチェックします。また電力消費を低減できます。

- R_ADC_CheckScanDone

表 261: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

7.1.8.13 read

ssp_err_t(* adc_api_t::read)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, adc_data_size_t *const p_data)

詳細説明

ADC 変換結果を読み取ります。また電力消費を低減できます。

- R_ADC_Read

表 262: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
reg_id	複数のビットを書き換えることもできます。	読み取るレジスタの ID （列挙値 adc_register_t を参照）
p_data	複数のビットを書き換えることもできます。	値の読み込み先の変数へのポインタ。

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

パラメータ reg_id

定義 : adc_register_tconst reg_id

パラメータ p_data

パラメータ p_data

定義 : adc_data_size_t*const p_data

最大サポート ADC 分解能サイズ。

7.1.8.14 sampleStateCountSet

ssp_err_t(* adc_api_t::sampleStateCountSet)(adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)

詳細説明

指定されたチャネルのサンプル状態カウントを設定します。また電力消費を低減できます。

- R_ADC_SetSampleStateCount

表 263: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_sample	複数のビットを書き換えることもできます。	設定する ADC チャネルおよび対応するサンプル状態へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

定義 :

定義 : adc_sample_state_t*p_sample

ADC サンプル状態設定

- adc_sample_state_t::reg_id

  サンプル状態レジスタ ID。

- adc_sample_state_t::num_states

  変換のサンプリング状態数。Ch16-20/21 は、同じ値を使用します。

7.1.8.15 close

ssp_err_t(* adc_api_t::close)(adc_ctrl_t *const p_ctrl)

詳細説明

進行中のスキャンを停止して、割り込みを無効にし、指定された A/D ユニットへの電源を遮断することによって、指定された ADC ユニットを閉じます。また電力消費を低減できます。

- R_ADC_Close

表 264: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

7.1.8.16 infoGet

ssp_err_t(* adc_api_t::infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)

詳細説明

構成されたすべてのチャネルの変換結果を DTC/DMAC によって読み取るために、先頭 （最も小さい番号の) チャネルの ADC データのレジスタアドレスと読み取る合計バイト数を返します。また電力消費を低減できます。

- R_ADC_InfoGet

表 265: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_adc_info	out	ADC 情報構造体へのポインタ

定義 : adc_ctrl_t

ADC 制御ブロック。初期化しないでください。初期化は open で実行されます。

パラメータ p_adc_info

定義 : adc_info_t*const p_adc_info

転送インタフェースの ADC 情報構造体へのポインタ

- adc_info_t::p_address

  データの読み取りを開始するアドレス。

- adc_info_t::length

  読み取る合計バイト数。

- adc_info_t::elc_peripheral

  ELC リストの周辺機能名。

- adc_info_t::elc_event

  周辺機能の ELC イベント名。

7.1.8.17 versionGet

ssp_err_t(* adc_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

API バージョンを取得します。また電力消費を低減できます。

- R_ADC_VersionGet

  I：この関数は、API バージョンを取得します。

表 266:　パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン構造体へのポインタ

パラメータ p_version

7.1.8.18 adc_instance_t

adc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- adc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- adc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- adc_channel_cfg_t const * p_channel_cfg

  このインスタンスのチャネル設定構造体へのポインタ。

- adc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.2 CAC インタフェース

クロック周波数精度測定用インタフェース。

定義、共通サービス、およびエラー コードを登録します。

7.2.1 概要

クロック周波数精度測定回路 （CAC）ペリフェラル用のインタフェースは、測定するクロック （システム クロック）のパルス数をカウントすることによって、参照クロック信号を使用してシステム クロック周波数をチェックするために使用されます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

CAC インタフェースの説明 : クロック精度測定回路ドライバ

7.2.2 インタフェース API

cac_api_t

関数名	説明
.open	CAC デバイス用の関数を開きます。
.read	CAC ペリフェラル用の関数を読み取ります。
.close	CAC デバイス用の関数を閉じます。
.stopMeasurement	CAC ペリフェラルの測定を終了します。
.startMeasurement	CAC ペリフェラルの測定を開始します。
.reset	CAC デバイス用の関数をリセットします。
.versionGet	CAC API とコード バージョンに関する情報を取得します。

7.2.3 データ構造体

- cac_ctrl_t

- cac_ref_clock_config_t
- cac_meas_clock_config_t
- cac_callback_args_t
- cac_cfg_t
- cac_instance_t

7.2.4 列挙

- cac_event_t
- cac_clock_type_t
- cac_clock_source_t
- cac_ref_divider_t
- cac_ref_digfilter_t
- cac_ref_edge_t
- cac_meas_divider_t

7.2.5 型定義

- cb_context_t

7.2.6 定義

- #define CAC_API_VERSION_MAJOR

  初期値 :(1)
- #define CAC_API_VERSION_MINOR

  初期値 :(1)

7.2.7 API データ

7.2.7.1 cac_event_t

cac_event_t

詳細説明

CAC 割り込みモードで使用された ISR コールバックが返すイベントタイプ

列挙値

名前	説明
CAC_EVENT_FREQUENCY_ERROR	周波数エラー。
CAC_EVENT_MEASUREMENT_COMPLETE	測定完了。
CAC_EVENT_COUNTER_OVERFLOW	カウンタ オーバフロー。

7.2.7.2 cac_clock_type_t

cac_clock_type_t

詳細説明

2 つの使用可能なクロックの列挙。

列挙値

名前	説明
CAC_CLOCK_MEASURED	測定クロック。
CAC_CLOCK_REFERENCE	参照クロック。

7.2.7.3 cac_clock_source_t

cac_clock_source_t

詳細説明

参照クロックおよび測定クロックの両方で使用可能なクロック ソースの列挙。

列挙値

名前	説明
CAC_CLOCK_SOURCE_MAIN_OSC	メイン クロック発振器。
CAC_CLOCK_SOURCE_SUBCLOCK	サブクロック。
CAC_CLOCK_SOURCE_HOCO	HOCO （高速オンチップ発振器）
CAC_CLOCK_SOURCE_MOCO	MOCO （中速オンチップ発振器）

名前	説明
CAC_CLOCK_SOURCE_LOCO	LOCO （中速オンチップ発振器）
CAC_CLOCK_SOURCE_PCLKB	PCLKB （周辺クロック B）
CAC_CLOCK_SOURCE_IWDT	IWDT - 専用オンチップ発振器。
CAC_CLOCK_SOURCE_MEAS_MAX	最大測定クロック。
CAC_CLOCK_SOURCE_EXTERNAL	CACREF ピンの外部供給測定クロック。
CAC_CLOCK_SOURCE_REF_MAX	最大参照クロック。

7.2.7.4 cac_ref_divider_t

cac_ref_divider_t

詳細説明

参照クロックで使用可能なディバイダーの列挙。

列挙値

名前	説明
CAC_REF_DIV_32	32 で分割された参照クロック。
CAC_REF_DIV_128	128 で分割された参照クロック。
CAC_REF_DIV_1024	1024 で分割された参照クロック。
CAC_REF_DIV_8192	8192 で分割された参照クロック。

7.2.7.5 cac_ref_digfilter_t

cac_ref_digfilter_t

詳細説明

外部参照クロックで使用可能なデジタル フィルタ設定の列挙。

列挙値

名前	説明
CAC_REF_DIGITAL_FILTER_OFF	参照クロックで CACREF ピンにデジタル フィルタがありません。
CAC_REF_DIGITAL_FILTER_1	デジタル フィルタのサンプル クロック = 測定周波数。
CAC_REF_DIGITAL_FILTER_4	デジタル フィルタのサンプル クロック = 測定周波数 /4。
CAC_REF_DIGITAL_FILTER_16	デジタル フィルタのサンプル クロック = 測定周波数 /16。

7.2.7.6 cac_ref_edge_t

cac_ref_edge_t

詳細説明

参照クロックで使用可能なエッジ検出設定の列挙。

列挙値

名前	説明
CAC_REF_EDGE_RISE	参照クロックでの上昇エッジ検出。
CAC_REF_EDGE_FALL	参照クロックでの下降エッジ検出。
CAC_REF_EDGE_BOTH	参照クロックでの上昇および下降エッジ検出。

7.2.7.7 cac_meas_divider_t

cac_meas_divider_t

詳細説明

測定クロックで使用可能なディバイダーの列挙。

列挙値

名前	説明
CAC_MEAS_DIV_1	1 で分割された測定クロック。

名前	説明
CAC_MEAS_DIV_4	4 で分割された測定クロック。
CAC_MEAS_DIV_8	8 で分割された測定クロック。
CAC_MEAS_DIV_32	32 で分割された測定クロック。

7.2.7.8 cb_context_t

typedef void const* cb_context_t

詳細説明

ユーザーが選択してコールバックに受け渡されたコールバック コンテキスト

7.2.8 API 構造

7.2.8.1 cac_ctrl_t

cac_ctrl_t

詳細説明

パラメータ cac_cfg_t

変数

- void const * p_extend

7.2.8.2 cac_ref_clock_config_t

cac_ref_clock_config_t

詳細説明

参照クロック設定に適用する設定値を定義する構造体。

変数

- cac_ref_divider_t divider
  参照クロック用の除算値の指定。
- cac_clock_source_t clock
  参照クロック用のクロック ソース。

- cac_ref_digfilter_t digfilter

  CACREF 拡張クロック用のデジタル フィルタ選択。

- cac_ref_edge_t edge

  参照クロック用のエッジ検出。

7.2.8.3 cac_meas_clock_config_t

cac_meas_clock_config_t

詳細説明

測定クロック設定に適用する設定値を定義する構造体。

変数

- cac_meas_divider_t divider

  測定クロック用の除算値の指定。

- cac_clock_source_t clock

  測定クロック用のクロック ソース。

7.2.8.4 cac_callback_args_t

cac_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- cac_event_t event

  このイベントを使用して、コールバックの原因 （cac レディまたはエラー）を特定できます。

- void const * p_context

  ユーザー データのプレースホルダー。cac_cfg_t 内の cac_api_t::open 関数で設定されます。

7.2.8.5 cac_cfg_t

cac_cfg_t

詳細説明

CAC の設定

変数

- cac_ref_clock_config_t cac_ref_clock

  参照クロック固有の設定

- cac_meas_clock_config_t cac_meas_clock

  測定クロック固有の設定

- uint16_t cac_upper_limit

  上限カウンタしきい値

- uint16_t cac_lower_limit

  下限カウンタしきい値

- bool mei_interrupt_enabled

  Measurement Complete 割り込みが有効な場合は True。

- bool ovf_interrupt_enabled

  Overflow 割り込みが有効な場合は True。

- bool ferr_interrupt_enabled

  Frequency Error 割り込みが有効な場合は True。

- bool continuous_mode

  測定が完了した後、連続的に再開始する場合は True。

- void(* p_callback)(cac_callback_args_t *p_args)

  CAC 割り込み ISR の発生時に提供されるコールバック。

- void const * p_extend

  CAC ハードウェアに依存する設定 */。

- void const * p_context

  ユーザー データのプレースホルダー。cac_callback_args_t 内のユーザー コールバックに渡されます。

7.2.8.6 cac_api_t

cac_api_t

詳細説明

HAL レイヤー API に実装された CAC 関数

7.2.8.7 open

ssp_err_t(* cac_api_t::open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

詳細説明

CAC デバイス用の関数を開きます。

表 267: パラメータ

名前	方向	説明
p_ctrl	out	CAC デバイス制御へのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
cac_cfg_t	複数のビットを書き換えることもできます。	CAC 設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : cac_ctrl_t

パラメータ cac_cfg_t

定義 :

定義 : cac_cfg_t const *const p_cfg

CAC の設定

- cac_cfg_t::cac_ref_clock_config_t
  参照クロック固有の設定
- cac_cfg_t::cac_meas_clock_config_t
  測定クロック固有の設定
- cac_cfg_t::cac_upper_limit
  上限カウンタしきい値
- cac_cfg_t::cac_lower_limit
  下限カウンタしきい値
- cac_cfg_t::mei_interrupt_enabled
  Measurement Complete 割り込みが有効な場合は True。
- cac_cfg_t::ovf_interrupt_enabled
  Overflow 割り込みが有効な場合は True。
- cac_cfg_t::ferr_interrupt_enabled
  Frequency Error 割り込みが有効な場合は True。
- cac_cfg_t::continuous_mode
  測定が完了した後、連続的に再開始する場合は True。

- cac_cfg_t::p_callback

  CAC 割り込み ISR の発生時に提供されるコールバック。

- cac_cfg_t::p_extend

  CAC ハードウェアに依存する設定 */。

- cac_cfg_t::p_context

  ユーザー データのプレースホルダー。cac_callback_args_t 内のユーザー コールバックに渡されます。

7.2.8.8 read

ssp_err_t(* cac_api_t::read)(cac_ctrl_t *const p_ctrl, uint8_t *const p_status, uint16_t *const p_counter)

詳細説明

CAC ペリフェラル用の関数を読み取ります。

表 268: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAC デバイスのコンテキストの制御。
p_status	複数のビットを書き換えることもできます。	現在の CASTR レジスタの内容を格納する変数へのポインタ。
p_counter	複数のビットを書き換えることもできます。	現在の CACNTBR レジスタの内容を格納する変数へのポインタ。

定義 : cac_ctrl_t

パラメータ cac_cfg_t

パラメータ p_status

uint8_t

パラメータ p_counter

uint16_t

7.2.8.9 close

ssp_err_t(* cac_api_t::close)(cac_ctrl_t *const p_ctrl)

詳細説明

CAC デバイス用の関数を閉じます。

表 269: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAC デバイス制御へのポインタ。

定義 : cac_ctrl_t

パラメータ cac_cfg_t

7.2.8.10 stopMeasurement

ssp_err_t(* cac_api_t::stopMeasurement)(cac_ctrl_t *const p_ctrl)

詳細説明

CAC ペリフェラルの測定を終了します。

表 270: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAC デバイス制御へのポインタ。

定義 : cac_ctrl_t

パラメータ cac_cfg_t

7.2.8.11 startMeasurement

ssp_err_t(* cac_api_t::startMeasurement)(cac_ctrl_t *const p_ctrl)

詳細説明

CAC ペリフェラルの測定を開始します。

表 271: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAC デバイス制御へのポインタ。

定義 : cac_ctrl_t

パラメータ cac_cfg_t

7.2.8.12 reset

ssp_err_t(* cac_api_t::reset)(cac_ctrl_t *const p_ctrl)

詳細説明

CAC デバイス用の関数をリセットします。

表 272: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAC デバイス制御へのポインタ。

定義 : cac_ctrl_t

パラメータ cac_cfg_t

7.2.8.13 versionGet

ssp_err_t(* cac_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

CAC API とコード バージョンに関する情報を取得します。

表 273: パラメータ

名前	方向	説明
p_version	out	戻り値。

パラメータ p_version

7.2.8.14 cac_instance_t

cac_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- cac_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- cac_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- cac_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.3 CAN インタフェース

CAN 周辺機器用のインタフェース。

7.3.1 概要

CAN インタフェースは、CAN HAL ドライバ用の共通 API を提供します。CAN インタフェースは、以下の機能をサポートします。

- 全二重 CAN 通信
- 汎用 CAN パラメータ設定値
- 割り込み駆動型の送信 / 受信処理
- イベント コードを返すコールバック関数サポート
- トランザクション中のハードウェア リソース ロック

7.3.2 インタフェース API

can_api_t

関数名	説明
.open	CAN デバイス用の open 関数
.read	CAN デバイスのリード関数。ブロッキング関数ではありません。
.write	CAN デバイス用の書き込み関数
.close	CAN デバイス用の close 関数
.control	CAN デバイス用の制御関数
.infoGet	CAN チャネルの情報を取得します。
.versionGet	CAN デバイス用のバージョン取得関数

7.3.3 データ構造体

- can_status_t
- can_error_t

- can_info_t
- can_callback_args_t
- can_bit_timing_cfg_t
- can_frame_t
- can_mailbox_t
- can_cfg_t
- can_ctrl_t
- can_instance_t

7.3.4 列挙

- can_event_t
- can_mode_t
- can_id_mode_t
- can_frame_type_t
- can_message_mode_t
- can_clock_source_t
- can_time_segment1_t
- can_time_segment2_t
- can_sync_jump_width_t
- can_mailbox_send_receive_t
- can_command_t

7.3.5 型定義

- can_id_t

7.3.6 定義

- #define CAN_API_VERSION_MAJOR

  初期値 :(01UL)
- #define CAN_API_VERSION_MINOR

  初期値 :(00UL)

7.3.7 API データ

7.3.7.1 can_event_t

can_event_t

詳細説明

CAN イベント コード

列挙値

名前	説明
CAN_EVENT_RX_COMPLETE	完全なイベントを受信します。
CAN_EVENT_TX_COMPLETE	完全なイベントを送信します。
CAN_EVENT_ERR_BUS_OFF	バス オフ イベント。
CAN_EVENT_BUS_RECOVERY	バス オフ復帰イベント。
CAN_EVENT_ERR_PASSIVE	エラー パッシブ イベント。
CAN_EVENT_MAILBOX_OVERWRITE_OVERRUN	メールボックスが上書きまたはオーバーランされました。

7.3.7.2 can_mode_t

can_mode_t

詳細説明

CAN 動動作モード

列挙値

名前	説明
CAN_MODE_NORMAL	CAN ノーマル モード。
CAN_MODE_HALT	CAN halt モード。
CAN_MODE_SLEEP	CAN スリープ モード。
CAN_MODE_EXIT_SLEEP	CAN スリープ解除モード。

名前	説明
CAN_MODE_RESET	CAN リセット モード。
CAN_MODE_LISTEN	CAN リッスン モード。
CAN_MODE_LOOPBACK_INTERNAL	CAN 内部ループバック モード。
CAN_MODE_LOOPBACK_EXTERNAL	CAN 外部ループバック モード。

7.3.7.3 can_id_mode_t

can_id_mode_t

詳細説明

CAN ID モード

列挙値

名前	説明
CAN_ID_MODE_STANDARD	11 ビットに対する標準 ID が使用されています。
CAN_ID_MODE_EXTENDED	29 ビットに対する標準 ID が使用されています。

7.3.7.4 can_frame_type_t

can_frame_type_t

詳細説明

CAN フレーム タイプ

列挙値

名前	説明
CAN_FRAME_TYPE_DATA	データのフレーム タイプ。
CAN_FRAME_TYPE_REMOTE	リモート フレーム タイプ。

7.3.7.5 can_message_mode_t

can_message_mode_t

詳細説明

CAN メッセージ モード

列挙値

名前	説明
CAN_MESSAGE_MODE_OVERWRITE	次のフレームの前に読み込まれなかった場合、受信データは上書きされます。
CAN_MESSAGE_MODE_OVERRUN	受信データは、読み込まれるまで保持されます。

7.3.7.6 can_clock_source_t

can_clock_source_t

詳細説明

CAN ソース クロック

列挙値

名前	説明
CAN_CLOCK_SOURCE_PCLKB	PCLB が CAN クロックのソースです。
CAN_CLOCK_SOURCE_CANMCLK	CANMCLK が CAN クロックのソースです。

7.3.7.7 can_time_segment1_t

can_time_segment1_t

詳細説明

CAN タイム セグメント 1 タイム クォンタム

列挙値

名前	説明
CAN_TIME_SEGMENT1_TQ4	4 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ5	5 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ6	6 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ7	7 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ8	8 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ9	9 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ10	10 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ11	11 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ12	12 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ13	13 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ14	14 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ15	15 タイム クォンタムに対するタイム セグメント 1 設定。
CAN_TIME_SEGMENT1_TQ16	16 タイム クォンタムに対するタイム セグメント 1 設定。

7.3.7.8 can_time_segment2_t

can_time_segment2_t

詳細説明

CAN タイム セグメント 2 タイム クォンタム

列挙値

名前	説明
CAN_TIME_SEGMENT2_TQ2	2 タイム クォンタムに対するタイム セグメント 2 設定。
CAN_TIME_SEGMENT2_TQ3	3 タイム クォンタムに対するタイム セグメント 2 設定。
CAN_TIME_SEGMENT2_TQ4	4 タイム クォンタムに対するタイム セグメント 2 設定。

名前	説明
CAN_TIME_SEGMENT2_TQ5	5 タイム クォンタムに対するタイム セグメント 2 設定。
CAN_TIME_SEGMENT2_TQ6	6 タイム クォンタムに対するタイム セグメント 2 設定。
CAN_TIME_SEGMENT2_TQ7	7 タイム クォンタムに対するタイム セグメント 2 設定。
CAN_TIME_SEGMENT2_TQ8	8 タイム クォンタムに対するタイム セグメント 2 設定。

7.3.7.9 can_sync_jump_width_t

can_sync_jump_width_t

詳細説明

CAN 同期ジャンプ幅タイム クォンタム

列挙値

名前	説明
CAN_SYNC_JUMP_WIDTH_TQ1	1 タイム クォンタムに対する同期ジャンプ幅設定。
CAN_SYNC_JUMP_WIDTH_TQ2	2 タイム クォンタムに対する同期ジャンプ幅設定。
CAN_SYNC_JUMP_WIDTH_TQ3	3 タイム クォンタムに対する同期ジャンプ幅設定。
CAN_SYNC_JUMP_WIDTH_TQ4	4 タイム クォンタムに対する同期ジャンプ幅設定。

7.3.7.10 can_mailbox_send_receive_t

can_mailbox_send_receive_t

詳細説明

CAN メールボックス タイプ

列挙値

名前	説明
CAN_MAILBOX_RECEIVE	メールボックスは受信用です。
CAN_MAILBOX_TRANSMIT	メールボックスは送信用です。

7.3.7.11 can_command_t

can_command_t

詳細説明

CAN 制御コマンド

列挙値

名前	説明
CAN_COMMAND_MODE_SWITCH	CAN の動作モードを切り替え …

7.3.7.12 can_id_t

typedef uint32_t can_id_t

詳細説明

CAN ID

7.3.8 API 構造

7.3.8.1 can_status_t

詳細説明

CAN のステータス

変数

status

status_b

7.3.8.2 status

uint32_t can_status_t::status

7.3.8.3 status_b

struct can_status_t::st_status_b can_status_t::status_b

7.3.8.4 can_error_t

詳細説明

CAN エラー コード

変数

error

error_b

7.3.8.5 error

uint32_t can_error_t::error

7.3.8.6 error_b

struct can_error_t::st_error_b can_error_t::error_b

7.3.8.7 can_info_t

can_info_t

詳細説明

変数

- can_mode_t operation_mode

  CAN 動作モード。
- can_status_t status

  CAN のステータス。
- uint32_t bit_rate

  CAN ビット レート。
- uint8_t error_count_transmit

  送信エラー カウント。
- uint8_t error_count_receive

  受信エラー カウント。
- can_error_t error_code

  エラー コード、リード後削除されます。

7.3.8.8 can_callback_args_t

can_callback_args_t

詳細説明

CAN コールバック パラメータ定義

変数

- uint32_t channel

  デバイス チャネル番号。

- can_event_t event

  イベント コード。

- uint32_t mailbox

  割り込みソースのメールボックス番号。

- void const * p_context

  コールバック時にユーザーに提供されるコンテキスト。

7.3.8.9 can_bit_timing_cfg_t

can_bit_timing_cfg_t

詳細説明

CAN ビット レート構成。

変数

- uint32_t baud_rate_prescaler

  CAN チャネルのビット レート。

- can_time_segment1_t time_segment_1

  タイム セグメント 1 コントローラ。

- can_time_segment2_t time_segment_2

  タイム セグメント 2 コントローラ。

- can_sync_jump_width_t synchronization_jump_width

  同期ジャンプ幅。

7.3.8.10 can_frame_t

can_frame_t

詳細説明

CAN データ フレーム

変数

- can_id_t id

  CAN ID。

- uint8_t data_length_code

  CAN データ長コード、メッセージ内のバイト数。

- uint8_t data[8]

  CAN データ、最大 8 バイト。

- can_frame_type_t type

  フレーム タイプ、データまたはリモート フレーム。

7.3.8.11 can_mailbox_t

can_mailbox_t

詳細説明

CAN メールボックス

変数

- can_id_t mailbox_id

  メールボックス ID。

- can_mailbox_send_receive_t mailbox_type

  受信または送信メールボックスのタイプ。

- can_frame_type_t frame_type

  受信メールボックスのフレーム タイプ。

7.3.8.12 can_cfg_t

can_cfg_t

詳細説明

CAN の設定

変数

- uint32_t channel

  CAN チャネル。

- can_bit_timing_cfg_t * p_bit_timing

  CAN ビット タイミング。

- can_id_mode_t id_mode

  標準または拡張 ID モード。

- uint32_t mailbox_count

  メールボックスの数。

- can_mailbox_t * p_mailbox

  メールボックスへのポインタ。

- can_message_mode_t message_mode

  上書きメッセージまたはオーバーラン。

- void(* p_callback)(can_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void const * p_context

  ユーザー定義のコールバック コンテキスト。

- void const * p_extend

  CAN ハードウェアに依存する設定。

7.3.8.13 can_ctrl_t

can_ctrl_t

詳細説明

CAN 制御ブロック

変数

- void * p_instance_ctrl

  インスタンス制御情報へのポインタ。

7.3.8.14 can_api_t

can_api_t

詳細説明

CAN の共有インタフェース定義

7.3.8.15 open

ssp_err_t(* can_api_t::open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)

詳細説明

CAN デバイス用の open 関数。以下として実装されます。

- R_CAN_Open

表 274: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	CAN 制御ブロックへのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
can_cfg_t	複数のビットを書き換えることもできます。	CAN 設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : can_ctrl_t

CAN 制御ブロック

パラメータ can_cfg_t

定義 : can_cfg_t const *const p_cfg

CAN の設定

- can_cfg_t::channel

  CAN チャネル。

- can_cfg_t::can_bit_timing_cfg_t

  CAN ビット タイミング。

- can_cfg_t::can_id_mode_t

  標準または拡張 ID モード。

  列挙値 :

  – CAN_ID_MODE_STANDARD

  – CAN_ID_MODE_EXTENDED

- can_cfg_t::mailbox_count

  メールボックスの数。

- can_cfg_t::can_mailbox_t

  メールボックスへのポインタ。

- can_cfg_t::can_message_mode_t

  上書きメッセージまたはオーバーラン。

  列挙値 :

  – CAN_MESSAGE_MODE_OVERWRITE

  – CAN_MESSAGE_MODE_OVERRUN

- can_cfg_t::p_callback

  コールバック関数へのポインタ。

- can_cfg_t::p_context

  ユーザー定義のコールバック コンテキスト。

- can_cfg_t::p_extend

  CAN ハードウェアに依存する設定。

7.3.8.16 read

ssp_err_t(* can_api_t::read)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)

詳細説明

CAN デバイスのリード関数。ブロッキング関数ではありません。また電力消費を低減できます。

- R_CAN_Read

表 275: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの CAN 制御ブロックへのポインタ。
メールボックス	複数のビットを書き換えることもできます。	読み取り用のメールボックス。
p_frame	out	CAN ID、DLC、データおよびフレーム タイプへのポインタ。

定義 : can_ctrl_t

CAN 制御ブロック

パラメータ メールボックス

uint32_t

Parameter p_frame

定義 : can_frame_t*const p_frame

CAN データ フレーム

- can_frame_t::id

  CAN ID。
- can_frame_t::data_length_code

  CAN データ長コード、メッセージ内のバイト数。
- can_frame_t::data

  CAN データ、最大 8 バイト。
- can_frame_t::type

  フレーム タイプ、データまたはリモート フレーム。

7.3.8.17 write

ssp_err_t(* can_api_t::write)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)

詳細説明

CAN デバイス用の書き込み関数。以下として実装されます。

- R_CAN_Write

表 276: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAN 制御ブロックへのポインタ。
メールボックス	複数のビットを書き換えることもできます。	書き込み用のメールボックス。
p_frame	複数のビットを書き換えることもできます。	書き込み先となる CAN ID、DLC、データおよびフレーム タイプへのポインタ。

定義 : can_ctrl_t

CAN 制御ブロック

パラメータ メールボックス

uint32_t

Parameter p_frame

定義 : can_frame_t*const p_frame

CAN データ フレーム

- can_frame_t::id

  CAN ID。
- can_frame_t::data_length_code

  CAN データ長コード、メッセージ内のバイト数。
- can_frame_t::data

  CAN データ、最大 8 バイト。
- can_frame_t::type

  フレーム タイプ、データまたはリモート フレーム。

7.3.8.18 close

ssp_err_t(* can_api_t::close)(can_ctrl_t *const p_ctrl)

詳細説明

CAN デバイス用の close 関数。以下として実装されます。

- R_CAN_Close

表 277: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAN 制御ブロックへのポインタ。

定義 : can_ctrl_t

CAN 制御ブロック

7.3.8.19 control

ssp_err_t(* can_api_t::control)(can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data)

詳細説明

CAN デバイス用の制御関数。以下として実装されます。

- R_CAN_Control

表 278: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CAN 制御ブロックへのポインタ。
mode	複数のビットを書き換えることもできます。	切り替え後の動作モード。

定義 : `can_ctrl_t`

CAN 制御ブロック

パラメータ mode

7.3.8.20 infoGet

ssp_err_t(* `can_api_t`::infoGet)(`can_ctrl_t` *const p_ctrl, `can_info_t` *const p_info)

詳細説明

CAN チャネルの情報を取得します。また電力消費を低減できます。

- R_CAN_InfoGet

表 279: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）
p_info	out	チャネル固有のデータを返す先のメモリ アドレス。

定義 : `can_ctrl_t`

CAN 制御ブロック

パラメータ p_info

定義 : `can_info_t`*const p_info

- can_info_t::operation_mode

  CAN 動作モード。

- can_info_t::status

  CAN のステータス。

- can_info_t::bit_rate

  CAN ビット レート。

- can_info_t::error_count_transmit

  送信エラー カウント。

- can_info_t::error_count_receive

  受信エラー カウント。

- can_info_t::error_code

  エラー コード、リード後削除されます。

7.3.8.21 versionGet

ssp_err_t(* can_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

CAN デバイス用のバージョン取得関数。以下として実装されます。

- R_CAN_VersionGet

表 280: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン情報を格納するメモリへのポインタ

パラメータ p_version

7.3.8.22 can_instance_t

can_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- can_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- can_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- can_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.4 CGC インタフェース

クロック生成のインタフェース。

7.4.1 概要

CGC インタフェースでは、CGC モジュールのすべての機能を構成し、使用することができます。たとえば、複数のクロック ソースをシステム クロック ソースとして選択できます。また、システム クロックを分周することで、システムや周辺機能のニーズに適した多様な周波数を提供できます。

クロックの安定性を確認し、クロックが不要なときは停止して電力を節減することができます。API には、実行時にシステムおよびシステム周辺クロックの周波数を返す関数があります。メイン発振器の停止を検出する機能があり、それにはユーザー定義のコールバック関数を呼び出すオプションも用意されています。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

CGC インタフェースの説明 : CGC ドライバ

7.4.2 インタフェース API

cgc_api_t

関数名	説明
.init	初期設定
.clockStart	クロックを開始します。
.clockStop	クロックを停止します。
.systemClockSet	システム クロックを設定します。
.systemClockGet	システム クロック情報を取得します。
.systemClockFreqGet	選択されたクロックの周波数を返します。
.clockCheck	選択されたクロックの安定性をチェックします。
.oscStopDetect	メイン オシレーター停止検出を設定します。
.oscStopStatusClear	オシレーター停止検出フラグをクリアします。

関数名	説明
.busClockOutCfg	バス クロック出力セカンダリ除算値。プライマリ分周器は、BSP クロック構成と systemClockSet 関数によって設定されます （S7G2 および S3A7 のみ)。
.busClockOutEnable	バス クロック出力を有効にします （S7G2 および S3A7 のみ)。
.busClockOutDisable	バス クロック出力を無効にします （S7G2 および S3A7 のみ)。
.clockOutCfg	clockOut を設定します。
.clockOutEnable	CLKOUT ピンでの クロック出力を有効にします。クロック ソースは、clockOutCfg によって制御されます。
.clockOutDisable	CLKOUT ピンでの クロック出力を無効にします。クロック ソースは、clockOutCfg によって制御されます。
.lcdClockCfg	セグメント LCD クロックを設定します （S3A7 および S124 のみ)。
.lcdClockEnable	LCD クロックを有効にします （S3A7 および S124 のみ)。
.lcdClockDisable	LCD クロックを無効にします （S3A7 および S124 のみ)。
.sdramClockOutEnable	SDRAM クロック出力を有効にします （S7G2 のみ)。
.sdramClockOutDisable	SDRAM クロックを無効にします （S7G2 のみ)。
.usbClockCfg	USB クロックを設定します （S7G2 のみ)。
.systickUpdate	Systick タイマを更新します。
.versionGet	CGC ドライバのバージョンを取得します。

7.4.3 データ構造体

- cgc_callback_args_t
- cgc_clock_cfg_t
- cgc_system_clock_cfg_t
- cgc_instance_t

7.4.4 列挙

- cgc_event_t
- cgc_clock_t
- cgc_pll_div_t
- cgc_sys_clock_div_t
- cgc_system_clocks_t
- cgc_clockout_dividers_t
- cgc_bclockout_dividers_t
- cgc_usb_clock_div_t
- cgc_systick_period_units_t

7.4.5 定義

- #define CGC_API_VERSION_MAJOR

  初期値 :(1)
- #define CGC_API_VERSION_MINOR

  初期値 :(1)

7.4.6 API データ

7.4.6.1 cgc_event_t

cgc_event_t

詳細説明

コールバック関数をトリガー可能なイベント

列挙値

名前	説明
CGC_EVENT_OSC_STOP_DETECT	発振器停止検出によって、イベントが発生しました。

7.4.6.2 cgc_clock_t

cgc_clock_t

詳細説明

システム クロック ソース識別子 - システム クロックの除算値の前に記載されている ICLK、BCLK、FCLK、PCLKS A-D、UCLK のソース

列挙値

名前	説明
CGC_CLOCK_HOCO	高速オンチップ発振器。
CGC_CLOCK_MOCO	中速オンチップ発振器。
CGC_CLOCK_LOCO	低速オンチップ発振器。
CGC_CLOCK_MAIN_OSC	メイン発振器。
CGC_CLOCK_SUBCLOCK	サブクロック発振器。
CGC_CLOCK_PLL	PLL 発振器。

7.4.6.3 cgc_pll_div_t

cgc_pll_div_t

詳細説明

PLL 分周器の値

列挙値

名前	説明
CGC_PLL_DIV_1	PLL 分周器の値は 1 です。
CGC_PLL_DIV_2	PLL 分周器の値は 2 です。
CGC_PLL_DIV_3	PLL 分周器の値は 3 です （S7G2 のみ)。
CGC_PLL_DIV_4	PLL 分周器の値は 4 です （S3A7 のみ)。

7.4.6.4 cgc_sys_clock_div_t

cgc_sys_clock_div_t

詳細説明

システム クロックの分周器の値 - ICLK、BCLK、FCLK、および PCLKS A ～ D の各システム クロックに対して個別に選択可能な分周器の値

列挙値

名前	説明
CGC_SYS_CLOCK_DIV_1	システム クロックは、1 分周されます。
CGC_SYS_CLOCK_DIV_2	システム クロックは、2 分周されます。
CGC_SYS_CLOCK_DIV_4	システム クロックは、4 分周されます。
CGC_SYS_CLOCK_DIV_8	システム クロックは、8 分周されます。
CGC_SYS_CLOCK_DIV_16	システム クロックは、16 分周されます。
CGC_SYS_CLOCK_DIV_32	システム クロックは、32 分周されます。
CGC_SYS_CLOCK_DIV_64	システム クロックは、64 分周されます。

7.4.6.5 cgc_system_clocks_t

cgc_system_clocks_t

詳細説明

システム クロック識別子 - systemClockFreqGet 関数への入力パラメータとして使用されます。

列挙値

名前	説明
CGC_SYSTEM_CLOCKS_PCLKA	PCLKA - 周辺モジュール クロック A。
CGC_SYSTEM_CLOCKS_PCLKB	PCLKB - 周辺モジュール クロック B。
CGC_SYSTEM_CLOCKS_PCLKC	PCLKC - 周辺モジュール クロック C。
CGC_SYSTEM_CLOCKS_PCLKD	PCLKD - 周辺モジュール クロック D。
CGC_SYSTEM_CLOCKS_BCLK	BCLK - 外部バス クロック。
CGC_SYSTEM_CLOCKS_FCLK	FCLK - FlashIF クロック。

名前	説明
CGC_SYSTEM_CLOCKS_ICLK	ICLK - システム クロック。

7.4.6.6 cgc_clockout_dividers_t

cgc_clockout_dividers_t

詳細説明

CLKOUT 出力の除算値。

列挙値

名前	説明
CGC_CLOCKOUT_DIV_1	クロックアウト ソースは、1 分周されています。
CGC_CLOCKOUT_DIV_2	クロックアウト ソースは、2 分周されています。
CGC_CLOCKOUT_DIV_4	クロックアウト ソースは、4 分周されています。
CGC_CLOCKOUT_DIV_8	クロックアウト ソースは、8 分周されています。
CGC_CLOCKOUT_DIV_16	クロックアウト ソースは、16 分周されています。
CGC_CLOCKOUT_DIV_32	クロックアウト ソースは、32 分周されています。
CGC_CLOCKOUT_DIV_64	クロックアウト ソースは、64 分周されています。
CGC_CLOCKOUT_DIV_128	クロックアウト ソースは、128 分周されています。

7.4.6.7 cgc_bclockout_dividers_t

cgc_bclockout_dividers_t

詳細説明

外部バス クロック出力の除算値

列挙値

名前	説明
CGC_BCLOCKOUT_DIV_1	外部バス クロック ソースは 1 分周されています。

名前	説明
CGC_BCLOCKOUT_DIV_2	外部バス クロック ソースは 2 分周されています。

7.4.6.8 cgc_usb_clock_div_t

cgc_usb_clock_div_t

詳細説明

USB クロックの除算値

列挙値

名前	説明
CGC_USB_CLOCK_DIV_3	USB ソース クロックを 3 分周します。
CGC_USB_CLOCK_DIV_4	USB ソース クロックを 4 分周します。
CGC_USB_CLOCK_DIV_5	USB ソース クロックを 5 分周します。

7.4.6.9 cgc_systick_period_units_t

cgc_systick_period_units_t

詳細説明

に対して利用できる期間の単位 R_CGC_SystickUpdate

列挙値

名前	説明
CGC_SYSTICK_PERIOD_UNITS_MILLISECONDS	期間はミリ秒単位で要求されます。
CGC_SYSTICK_PERIOD_UNITS_MICROSECONDS	期間はマイクロ秒単位で要求されます。

7.4.7 API 構造

7.4.7.1 cgc_callback_args_t

cgc_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- cgc_event_t event

  このイベントは、コールバックの原因を特定するために使用できます。

- void const * p_context

  ユーザー データのプレースホルダー。

7.4.7.2 cgc_clock_cfg_t

cgc_clock_cfg_t

詳細説明

クロック構成構造体 - PLL クロックの clockStart 関数への入力パラメータとして使用されます。

変数

- cgc_clock_t source_clock

  PLL ソース クロック （S7G2 のみ）。

- cgc_pll_div_t divider

  PLL 除算値。

- float multiplier

  PLL 乗算値。

7.4.7.3 cgc_system_clock_cfg_t

cgc_system_clock_cfg_t

詳細説明

クロック構成構造体 - systemClockSet 関数および systemClockGet 関数への入力パラメータとして使用されます。

変数

- cgc_sys_clock_div_t pclka_div

  PCLKA の除算値。

- cgc_sys_clock_div_t pclkb_div

  PCLKB の除算値。

- cgc_sys_clock_div_t pclkc_div

  PCLKC の除算値。
- cgc_sys_clock_div_t pclkd_div

  PCLKD の除算値。
- cgc_sys_clock_div_t bclk_div

  BCLK の除算値。
- cgc_sys_clock_div_t fclk_div

  FCLK の除算値。
- cgc_sys_clock_div_t iclk_div

  ICLK の除算値。

7.4.7.4 cgc_api_t

cgc_api_t

詳細説明

HAL レイヤーに実装された CGC 関数は、この API に従います。

7.4.7.5 init

ssp_err_t(* `cgc_api_t`::init)(void)

詳細説明

初期設定。以下として実装されます。

- R_CGC_Init

I :BSP モジュールは、開始時にこの関数を呼び出します。それ以上の初期化は不要です。

7.4.7.6 clockStart

ssp_err_t(* `cgc_api_t`::clockStart)(`cgc_clock_t` clock_source, `cgc_clock_cfg_t` *p_clock_cfg)

詳細説明

クロックを開始します。また電力消費を低減できます。

- R_CGC_ClockStart

I：開始されるクロックは、この関数を呼び出す前に動作していてはなりません。すでに動作している場合はエラーが返されます。

表 281:　パラメータ

名前	方向	説明
clock_source	複数のビットを書き換えることもできます。	初期化するクロック ソース。
p_clock_cfg	複数のビットを書き換えることもできます。	PLL の設定時に使用される除算値または乗算値が含まれている構造体へのポインタ。

パラメータ clock_source

定義 : `cgc_clock_t`clock_source

システム クロック ソース識別子 - システム クロックの除算値の前に記載されている ICLK、BCLK、FCLK、PCLKS A-D、UCLK のソース

パラメータ p_clock_cfg

定義 : `cgc_clock_cfg_t` *p_clock_cfg

クロック構成構造体 - PLL クロックの clockStart 関数への入力パラメータとして使用されます。

- `cgc_clock_cfg_t`::`cgc_clock_t`

  PLL ソース クロック （S7G2 のみ）。

  列挙値 :

  – CGC_CLOCK_HOCO
  – CGC_CLOCK_MOCO
  – CGC_CLOCK_LOCO
  – CGC_CLOCK_MAIN_OSC
  – CGC_CLOCK_SUBCLOCK
  – CGC_CLOCK_PLL

- `cgc_clock_cfg_t`::`cgc_pll_div_t`

  PLL 除算値。

  列挙値 :

– CGC_PLL_DIV_1
– CGC_PLL_DIV_2
– CGC_PLL_DIV_3
– CGC_PLL_DIV_4

- cgc_clock_cfg_t::multiplier

  PLL 乗算値。

7.4.7.7 clockStop

ssp_err_t(* cgc_api_t::clockStop)(cgc_clock_t clock_source)

詳細説明

クロックを停止します。また電力消費を低減できます。

- R_CGC_ClockStop

  I：停止されるクロックは、この関数の呼び出しよりも前に停止していてはなりません。すでに停止している場合はエラーが返されます。

表 282:　パラメータ

名前	方向	説明
clock_source	複数のビットを書き換えることもできます。	停止するクロック ソース。

パラメータ clock_source

定義：cgc_clock_tclock_source

システム クロック ソース識別子 - システム クロックの除算値の前に記載されている ICLK、BCLK、FCLK、PCLKS A-D、UCLK のソース

7.4.7.8 systemClockSet

ssp_err_t(* cgc_api_t::systemClockSet)(cgc_clock_t clock_source, cgc_system_clock_cfg_t *p_clock_cfg)

詳細説明

システム クロックを設定します。また電力消費を低減できます。

- R_CGC_SystemClockSet

I：システム クロックとして設定されるクロックは、この関数の呼び出し前に動作している必要があります。

表 283:　パラメータ

名前	方向	説明
clock_source	複数のビットを書き換えることもできます。	システム クロックとして設定するクロック ソース
p_clock_cfg	複数のビットを書き換えることもできます。	呼び出し側から渡されるクロック除算値設定へのポインタ。

パラメータ clock_source

定義 : `cgc_clock_t`clock_source

システム クロック ソース識別子 - システム クロックの除算値の前に記載されている ICLK、BCLK、FCLK、PCLKS A-D、UCLK のソース

パラメータ p_clock_cfg

定義 : `cgc_system_clock_cfg_t` *p_clock_cfg

クロック構成構造体 - systemClockSet 関数および systemClockGet 関数への入力パラメータとして使用されます。

- `cgc_system_clock_cfg_t`::`cgc_sys_clock_div_t`

  PCLKA の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  PCLKB の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  PCLKC の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  PCLKD の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  BCLK の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  FCLK の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  ICLK の除算値。

  列挙値 :

  – CGC_SYS_CLOCK_DIV_1
  – CGC_SYS_CLOCK_DIV_2
  – CGC_SYS_CLOCK_DIV_4
  – CGC_SYS_CLOCK_DIV_8
  – CGC_SYS_CLOCK_DIV_16
  – CGC_SYS_CLOCK_DIV_32
  – CGC_SYS_CLOCK_DIV_64

7.4.7.9 systemClockGet

ssp_err_t(* cgc_api_t::systemClockGet)(cgc_clock_t *p_clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg)

詳細説明

システム クロック情報を取得します。また電力消費を低減できます。

- R_CGC_SystemClockGet

表 284: パラメータ

名前	方向	説明
clock_source	out	現在のシステム クロックを返します。
p_clock_cfg	out	現在のシステム クロックの除算値を返します。

パラメータ clock_source

パラメータ p_clock_cfg

7.4.7.10 systemClockFreqGet

ssp_err_t(* cgc_api_t::systemClockFreqGet)(cgc_system_clocks_t clock, uint32_t *p_freq_hz)

詳細説明

選択されたクロックの周波数を返します。また電力消費を低減できます。

- R_CGC_SystemClockFreqGet

表 285: パラメータ

名前	方向	説明
clock	複数のビットを書き換えることもできます。	周波数が返される内部クロックを指定します。
p_freq_hz	out	このポインタによって参照される周波数 （ヘルツ単位）を返します。

パラメータ clock

定義 : cgc_system_clocks_tclock

システム クロック識別子 - cgc_api_t::systemClockFreqGet 関数への入力パラメータとして使用されます。

パラメータ p_freq_hz

uint32_t

7.4.7.11 clockCheck

ssp_err_t(* cgc_api_t::clockCheck)(cgc_clock_t clock_source)

詳細説明

選択されたクロックの安定性をチェックします。また電力消費を低減できます。

- R_CGC_ClockCheck

表 286: パラメータ

名前	方向	説明
clock_source	複数のビットを書き換えることもできます。	安定性をチェックするクロック ソースを選択します。

パラメータ clock_source

定義 : cgc_clock_tclock_source

システム クロック ソース識別子 - システム クロックの除算値の前に記載されている ICLK、BCLK、FCLK、PCLKS A-D、UCLK のソース

7.4.7.12 oscStopDetect

ssp_err_t(* cgc_api_t::oscStopDetect)(void(*p_callback)(cgc_callback_args_t *p_args), bool enable)

詳細説明

メイン オシレーター停止検出を設定します。また電力消費を低減できます。

- R_CGC_OscStopDetect

表 287: パラメータ

名前	方向	説明
p_callback	複数のビットを書き換えることもできます。	発振停止検出時に、NMI 割り込みによって呼び出されるコールバック関数。2 番目の引数が "false" の場合は、この引数を NULL に設定できます。

表 287: パラメータ (続き)

名前	方向	説明
enable	複数のビットを書き換えることもできます。	発振停止検出機能の有効 / 無効を設定します。

パラメータ **p_callback**

パラメータ **enable**

const

7.4.7.13 oscStopStatusClear

ssp_err_t(* cgc_api_t::oscStopStatusClear)(void)

詳細説明

オシレーター停止検出フラグをクリアします。また電力消費を低減できます。

- R_CGC_OscStopStatusClear

7.4.7.14 busClockOutCfg

ssp_err_t(* cgc_api_t::busClockOutCfg)(cgc_bclockout_dividers_t divider)

詳細説明

バス クロック出力セカンダリ除算値。プライマリ分周器は、BSP クロック構成と systemClockSet 関数によって設定されます （S7G2 および S3A7 のみ)。

また電力消費を低減できます。

- R_CGC_BusClockOutCfg

表 288: パラメータ

名前	方向	説明
divider	複数のビットを書き換えることもできます。	クロック ソースの 1 または 2 の除算値。

パラメータ **divider**

定義 : cgc_bclockout_dividers_tdivider

外部バス クロック出力の除算値。

7.4.7.15 busClockOutEnable

ssp_err_t(* cgc_api_t::busClockOutEnable)(void)

詳細説明

バス クロック出力を有効にします （S7G2 および S3A7 のみ）。また電力消費を低減できます。

- R_CGC_BusClockOutEnable

7.4.7.16 busClockOutDisable

ssp_err_t(* cgc_api_t::busClockOutDisable)(void)

詳細説明

バス クロック出力を無効にします （S7G2 および S3A7 のみ）。また電力消費を低減できます。

- R_CGC_BusClockOutDisable

7.4.7.17 clockOutCfg

ssp_err_t(* cgc_api_t::clockOutCfg)(cgc_clock_t clock, cgc_clockout_dividers_t divider)

詳細説明

clockOut を設定します。また電力消費を低減できます。

- R_CGC_ClockOutCfg

表 289: パラメータ

名前	方向	説明
clock	複数のビットを書き換えることもできます。	クロック ソース。
divider	複数のビットを書き換えることもできます。	クロック ソースの 1 ～ 128 の除算値。

パラメータ clock

パラメータ divider

定義 : cgc_clockout_dividers_tdivider

CLKOUT 出力の除算値。

7.4.7.18 clockOutEnable

ssp_err_t(* cgc_api_t::clockOutEnable)(void)

詳細説明

CLKOUT ピンでの クロック出力を有効にします。クロック ソースは、clockOutCfg によって制御されます。また電力消費を低減できます。

- R_CGC_ClockOutEnable

7.4.7.19 clockOutDisable

ssp_err_t(* cgc_api_t::clockOutDisable)(void)

詳細説明

CLKOUT ピンでの クロック出力を無効にします。クロック ソースは、clockOutCfg によって制御されます。また電力消費を低減できます。

- R_CGC_ClockOutDisable

7.4.7.20 lcdClockCfg

ssp_err_t(* cgc_api_t::lcdClockCfg)(cgc_clock_t clock)

詳細説明

セグメント LCD クロックを設定します （S3A7 および S124 のみ)。また電力消費を低減できます。

- R_CGC_LCDClockCfg

表 290: パラメータ

名前	方向	説明
clock	複数のビットを書き換えることもできます。	セグメント LCD クロック ソース。

パラメータ clock

7.4.7.21 lcdClockEnable

ssp_err_t(* cgc_api_t::lcdClockEnable)(void)

詳細説明

LCD クロックを有効にします （S3A7 および S124 のみ)。また電力消費を低減できます。

- R_CGC_LCDClockEnable

7.4.7.22 lcdClockDisable

ssp_err_t(* cgc_api_t::lcdClockDisable)(void)

詳細説明

LCD クロックを無効にします （S3A7 および S124 のみ）。また電力消費を低減できます。

- R_CGC_LCDClockDisable

7.4.7.23 sdramClockOutEnable

ssp_err_t(* cgc_api_t::sdramClockOutEnable)(void)

詳細説明

SDRAM クロック出力を有効にします （S7G2 のみ）。また電力消費を低減できます。

- R_CGC_SDRAMClockOutEnable

7.4.7.24 sdramClockOutDisable

ssp_err_t(* cgc_api_t::sdramClockOutDisable)(void)

詳細説明

SDRAM クロックを無効にします （S7G2 のみ）。また電力消費を低減できます。

- R_CGC_SDRAMClockOutDisable

7.4.7.25 usbClockCfg

ssp_err_t(* cgc_api_t::usbClockCfg)(cgc_usb_clock_div_t divider)

詳細説明

USB クロックを設定します （S7G2 のみ）。また電力消費を低減できます。

- R_CGC_USBClockCfg

表 291:　パラメータ

名前	方向	説明
divider	複数のビットを書き換えることもできます。	クロック ソースの 3、4、5 のいずれかの除算値。

パラメータ divider

定義 : cgc_usb_clock_div_tdivider

USB クロックの除算値

7.4.7.26 systickUpdate

ssp_err_t(* cgc_api_t::systickUpdate)(uint32_t period_count, cgc_systick_period_units_t units)

詳細説明

Systick タイマを更新します。また電力消費を低減できます。

- R_CGC_SystickUpdate

表 292: パラメータ

名前	方向	説明
period_count	複数のビットを書き換えることもできます。	systick 期間の長さ。
units	複数のビットを書き換えることもできます。	指定された期間の単位。

Parameter period_count

uint32_t

Parameter units

7.4.7.27 versionGet

ssp_err_t(* cgc_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

CGC ドライバのバージョンを取得します。また電力消費を低減できます。

- R_CGC_VersionGet

表 293: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ **p_version**

7.4.7.28 cgc_instance_t

cgc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- cgc_clock_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- cgc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.5 CRC インタフェース

巡回冗長検査用インタフェース。

7.5.1 概要

CRC （巡回冗長検査）カルキュレータは、8 ビット、16 ビット、および 32 ビットのバリエーションを含む 5 つの異なる多項式を使用して、CRC コードを生成します。計算は、CPU を使用してブロックにデータを送信するか、10 個ある SCI チャネルのいずれか 1 つで読み取りまたは書き込みアクティビティをスヌーピングすることで実行します。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

CRC インタフェースの説明 : CRC ドライバ

7.5.2 インタフェース API

crc_api_t

関数名	説明
.open	CRC ドライバ モジュールを開きます。
.close	CRC モジュール ドライバを閉じます
.crcResultGet	現在の計算値を返します。
.snoopEnable	スヌーピングを有効にします。
.snoopDisable	スヌーピングを無効にします。
.snoopCfg	スヌープ チャネルと方向を設定します。
.calculate	データ ブロックに対して CRC 計算を実行します。
.versionGet	コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

7.5.3 データ構造体

- crc_ctrl_t
- crc_cfg_t
- crc_snoop_cfg_t
- crc_instance_t

7.5.4 列挙

- crc_polynomial_t
- crc_bit_order_t
- crc_snoop_direction_t

7.5.5 定義

- #define CRC_API_VERSION_MAJOR

  初期値 :(1)
- #define CRC_API_VERSION_MINOR

  初期値 :(1)

7.5.6 API データ

7.5.6.1 crc_polynomial_t

crc_polynomial_t

詳細説明

CRC 生成多項式切り替え （GPS）

列挙値

名前	説明
CRC_POLYNOMIAL_CRC_8	8 ビット CRC-8 ($X^8 + X^2 + X + 1$)
CRC_POLYNOMIAL_CRC_16	16 ビット CRC-16 ($X^{16} + X^{15} + X^2 + 1$)
CRC_POLYNOMIAL_CRC_CCITT	16 ビット CRC-CCITT ($X^{16} + X^{12} + X^5 + 1$)

名前	説明
CRC_POLYNOMIAL_CRC_32	32 ビット CRC-32 ($X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$)
CRC_POLYNOMIAL_CRC_32C	32 ビット CRC-32C ($X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$)

7.5.6.2 crc_bit_order_t

crc_bit_order_t

詳細説明

CRC 演算切り替え （LMS）

列挙値

名前	説明
CRC_BIT_ORDER_LMS_LSB	LSB ファースト通信用に CRC を生成します。
CRC_BIT_ORDER_LMS_MSB	MSB ファースト通信用に CRC を生成します。

7.5.6.3 crc_snoop_direction_t

crc_snoop_direction_t

詳細説明

スヌープオンライト / リード切り替え （CRCSWR）

列挙値

名前	説明
CRC_SNOOP_DIRECTION_RECEIVE	読み込み時スヌープ。
CRC_SNOOP_DIRECTION_TRANSMIT	書き込み時スヌープ。

7.5.7 API 構造

7.5.7.1 crc_ctrl_t

crc_ctrl_t

詳細説明

ドライバ ハンドル構造体。

変数

- crc_polynomial_t polynomial
  CRC 生成多項式切り替え （GPS）
- crc_bit_order_t bit_order
  CRC 演算切り替え （LMS）

7.5.7.2 crc_cfg_t

crc_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- crc_polynomial_t polynomial
  CRC 生成多項式切り替え（GPS）
- crc_bit_order_t bit_order
  CRC 演算切り替え （LMS）
- void const * p_extend
  CRC のハードウェアに依存する設定。

7.5.7.3 crc_snoop_cfg_t

crc_snoop_cfg_t

詳細説明

スヌープ設定

変数

- uint32_t snoop_channel

  レジスタスヌープアドレス（CRCSA）

- crc_snoop_direction_t snoop_direction

  スヌープオンライト / リード切り替え（CRCSWR）

7.5.7.4 crc_api_t

crc_api_t

詳細説明

CRC ドライバ構造体。HAL レイヤーに実装された汎用 CRC 関数は、この API に従います。

7.5.7.5 open

ssp_err_t(* crc_api_t::open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)

詳細説明

CRC ドライバ モジュールを開きます。また電力消費を低減できます。

- R_CRC_Open

表 294: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。

定義 : crc_ctrl_t

ドライバ ハンドル構造体。

定義 :

定義 : crc_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- crc_cfg_t::crc_polynomial_t

  CRC 生成多項式切り替え（GPS）

  列挙値 :

- CRC_POLYNOMIAL_CRC_8
- CRC_POLYNOMIAL_CRC_16
- CRC_POLYNOMIAL_CRC_CCITT
- CRC_POLYNOMIAL_CRC_32
- CRC_POLYNOMIAL_CRC_32C

- `crc_cfg_t`::`crc_bit_order_t`

  CRC 演算切り替え（LMS）

  列挙値 :

  - CRC_BIT_ORDER_LMS_LSB
  - CRC_BIT_ORDER_LMS_MSB

- `crc_cfg_t`::p_extend

  CRC のハードウェアに依存する設定。

7.5.7.6 close

ssp_err_t(* `crc_api_t`::close)(`crc_ctrl_t` *const p_ctrl)

詳細説明

CRC モジュール ドライバを閉じます。以下として実装されます。

- R_CRC_Close

表 295: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ

表 296: 戻り値

名前	説明
SSP_SUCCESS	構成が正常に行われました。

定義 : `crc_ctrl_t`

ドライバ ハンドル構造体。

パラメータ **SSP_SUCCESS**

7.5.7.7 crcResultGet

ssp_err_t(* crc_api_t::crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)

詳細説明

現在の計算値を返します。また電力消費を低減できます。

- R_CRC_CalculatedValueGet

表 297: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ。
crc_result	out	最後の CRC 計算の計算値。

定義 : crc_ctrl_t

ドライバ ハンドル構造体。

パラメータ crc_result

uint32_t

7.5.7.8 snoopEnable

ssp_err_t(* crc_api_t::snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

詳細説明

スヌーピングを有効にします。また電力消費を低減できます。

- R_CRC_SnoopEnable

表 298: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ。
crc_seed	複数のビットを書き換えることもできます。	CRC シード。

定義 : crc_ctrl_t

ドライバ ハンドル構造体。

パラメータ crc_seed

uint32_t

7.5.7.9 snoopDisable

ssp_err_t(* crc_api_t::snoopDisable)(crc_ctrl_t *const p_ctrl)

詳細説明

スヌーピングを無効にします。また電力消費を低減できます。

- R_CRC_SnoopDisable

表 299: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ。

定義 : crc_ctrl_t

ドライバ ハンドル構造体。

7.5.7.10 snoopCfg

ssp_err_t(* crc_api_t::snoopCfg)(crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const p_snoop_cfg)

詳細説明

スヌープ チャネルと方向を設定します。また電力消費を低減できます。

- R_CRC_SnoopCfg

表 300: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ。
p_snoopCfg	複数のビットを書き換えることもできます。	スヌープ設定。

定義 : crc_ctrl_t

ドライバ ハンドル構造体。

パラメータ p_snoopCfg

7.5.7.11 calculate

ssp_err_t(* crc_api_t::calculate)(crc_ctrl_t *const p_ctrl, void *p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)

詳細説明

データ ブロックに対して CRC 計算を実行します。また電力消費を低減できます。

- R_CRC_Calculate

表 301: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	CRC デバイス ハンドルへのポインタ。
input_buffer	複数のビットを書き換えることもできます。	データ値の配列へのポインタ。
num_bytes	複数のビットを書き換えることもできます。	配列内のバイト数 （要素数ではありません）。
crc_seed	複数のビットを書き換えることもできます。	CRC 計算のシード値。
crc_result	out	CRC 計算の計算値。

定義 : crc_ctrl_t

ドライバ ハンドル構造体。

パラメータ input_buffer

パラメータ num_bytes

uint32_t

パラメータ crc_seed

uint32_t

パラメータ **crc_result**

7.5.7.12 versionGet

ssp_err_t(* crc_api_t::versionGet)(ssp_version_t *version)

詳細説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。また電力消費を低減できます。

- R_CRC_VersionGet

7.5.7.13 crc_instance_t

crc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- crc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- crc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- crc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.6 暗号インタフェース

暗号化 / 暗号解除、サイン / 認証、およびハッシングの暗号アルゴリズム API です。

7.6.1 インタフェース API

crypto_api_t

関数名	説明
.open	指定された設定を使用して、暗号化モジュールを開きます
.close	指定された制御構造体 p_ctrl に対する暗号化インタフェース モジュールを閉じます
.statusGet	SCE 初期化のステータスを取得します
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.6.2 データ構造体

- crypto_ctrl_t
- crypto_cfg_t
- crypto_instance_t

7.6.3 モジュール

- AES インタフェース
- DSA インタフェース
- HASH アルゴリズムインタフェース
- RSA インタフェース
- TDES インタフェース
- 乱数生成

7.6.4 定義

- #define CRYPTO_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define CRYPTO_API_VERSION_MINOR

  初期値 :(00)

7.6.5 API 構造

7.6.5.1 crypto_ctrl_t

crypto_ctrl_t

詳細説明

ここでユーザーが必要とする Crypto_Interface Add API 定義。

変数

- uint32_t state

  SCE/SCE-Lite ドライバの状態 （初期化済みかどうか）を示します
- uint32_t cb_data
- void(* p_sce_long_plg_start_callback)(void)
- void(* p_sce_long_plg_end_callback)(void)

7.6.5.2 crypto_cfg_t

crypto_cfg_t

詳細説明

暗号化エンジンの設定パラメータ

変数

- void(* p_sce_long_plg_start_callback)(void)
- void(* p_sce_long_plg_end_callback)(void)

  ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

7.6.5.3 crypto_api_t

crypto_api_t

詳細説明

HAL レイヤーに実装された Crypto_Interface SCE 関数は、この API に従います。

7.6.5.4 open

uint32_t(* crypto_api_t::open)(crypto_ctrl_t *const p_ctrl, crypto_cfg_t const *const p_cfg)

詳細説明

指定された設定を使用して、暗号化モジュールを開きます

表 302: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります

定義 : crypto_ctrl_t

ここでユーザーが必要とする Crypto_Interface Add API 定義。

定義 :

定義 : crypto_cfg_t const *const p_cfg

暗号化エンジンの設定パラメータ

- crypto_cfg_t::p_sce_long_plg_start_callback
- crypto_cfg_t::p_sce_long_plg_end_callback

  ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

7.6.5.5 close

uint32_t(* crypto_api_t::close)(crypto_ctrl_t *const p_ctrl)

詳細説明

指定された制御構造体 p_ctrl に対する暗号化インタフェース モジュールを閉じます

表 303: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ

定義 : crypto_ctrl_t

ここでユーザーが必要とする Crypto_Interface Add API 定義。

7.6.5.6 statusGet

uint32_t(* crypto_api_t::statusGet)(uint32_t *p_status)

詳細説明

SCE 初期化のステータスを取得します

表 304: パラメータ

名前	方向	説明
p_status	out	SCE モジュールの初期化ステータスは、p_status がポイントするメモリに書き込まれます

パラメータ p_status

uint32_t

7.6.5.7 versionGet

uint32_t(* crypto_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 305: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ **p_version**

7.6.5.8 crypto_instance_t

crypto_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- crypto_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- crypto_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- crypto_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.6.6 モジュール

- AES インタフェース
- DSA インタフェース
- HASH アルゴリズムインタフェース
- RSA インタフェース
- TDES インタフェース
- 乱数生成

7.6.6.1 AES インタフェース

AES 暗号化 API および復号化 API。

インタフェース API

aes_api_t

参考資料

関数名	説明
.open	AES モジュールのオープン関数です。暗号化 / 復号化操作を実行する前に呼び出す必要があります。
.createKey	暗号化 / 復号化操作の AES キーを生成します。
.encrypt	aes.open() 関数呼び出しで指定されたチェーン モードとパディング モードを使用した AES 暗号化。128 ビット AES キーを使用して ECB モードで入力データを暗号化します
.addAdditionalAuthenticationData	新しい認証データ （暗号化または復号化操作を開始する前に呼び出される）を追加します。
.encryptFinal	aes.open() 関数呼び出しで指定されたチェーン モードとパディング モードを使用した AES 最終暗号化。
.decrypt	ECB モードでの AES 128 復号化。128 ビット AES キーを使用して ECB モードで入力データを復号化します
.setGcmTag	モード固有のパラメータを設定します
.getGcmTag	認証タグ データを取得します。
.close	AES モジュールを閉じます。
.zeroPaddingEncrypt	指定されたチェーン モードとパディング モードを使用した AES 暗号化。
.zeroPaddingDecrypt	指定されたチェーン モードとパディング モードを使用した AES 復号化。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

データ構造体

- aes_ctrl_t
- aes_cfg_t
- aes_instance_t

変数

- g_sce_crypto_api

- g_aes128ecb_on_sce
- g_aes128cbc_on_sce
- g_aes128ctr_on_sce
- g_aes128gcm_on_sce
- g_aes128xts_on_sce
- g_aes192ecb_on_sce
- g_aes192cbc_on_sce
- g_aes192ctr_on_sce
- g_aes192gcm_on_sce
- g_aes256ecb_on_sce
- g_aes256cbc_on_sce
- g_aes256ctr_on_sce
- g_aes256gcm_on_sce
- g_aes256xts_on_sce
- g_aes128gcm_on_sceHrk
- g_aes192gcm_on_sceHrk
- g_aes256gcm_on_sceHrk

定義

- #define AES_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define AES_API_VERSION_MINOR

  初期値 :(00)
- #define DRV_AES_CONTEXT_BUFFER_SIZE

  初期値 :(64)

aes_ctrl_t

aes_ctrl_t

詳細説明

AES インタフェース制御構造体

変数

- crypto_ctrl_t * p_crypto_ctrl

  暗号化エンジン制御構造体へのポインタ

- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

- uint32_t work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]

  例 : AES-GCM モードは、認証タグなどの格納にこれを使用します。

  暗号化のコンテキスト / 状態の格納に使用します

aes_cfg_t

aes_cfg_t

詳細説明

AES インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

変数

- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

aes_api_t

aes_api_t

詳細説明

HAL レイヤーに実装された AES_Interface SCE 関数は、この API に従います。

open

uint32_t(* aes_api_t::open)(aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)

詳細説明

AES モジュールのオープン関数です。暗号化 / 復号化操作を実行する前に呼び出す必要があります。

表 306: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	AES インタフェースの制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。

表 306: パラメータ (続き)

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	AES 設定の制御構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : aes_ctrl_t

AES インタフェース制御構造体

定義 :

定義 : aes_cfg_t const *const p_cfg

AES インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

- aes_cfg_t::crypto_api_t

  暗号化エンジン API へのポインタ

createKey

uint32_t(* aes_api_t::createKey)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key)

概要説明

暗号化 / 復号化操作の AES キーを生成します。

詳細説明

表 307: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	AES インタフェースの制御構造体へのポインタ。
num_words	複数のビットを書き換えることもできます。	バッファ p_key 内のワード数
p_key	out	キー バッファへのポインタ。生成されたキーは、この場所に格納されます。

定義 : aes_ctrl_t

AES インタフェース制御構造体

パラメータ num_words

uint32_t

パラメータ p_key

uint32_t

encrypt

uint32_t(* aes_api_t::encrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

概要説明

aes.open() 関数呼び出しで指定されたチェーン モードとパディング モードを使用した AES 暗号化。

詳細説明

128 ビット AES キーを使用して ECB モードで入力データを暗号化します

表 308: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	AES プレーンテキスト キーへのポインタ
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ
*p_dest	out	出力データ バッファ

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

addAdditionalAuthenticationData

uint32_t(* aes_api_t::addAdditionalAuthenticationData)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source)

概要説明

新しい認証データ （暗号化または復号化操作を開始する前に呼び出される）を追加します。

詳細説明

表 309: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	AES プレーンテキスト キーへのポインタ
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

encryptFinal

uint32_t(* aes_api_t::encryptFinal)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t output_num_words, uint32_t *p_dest)

概要説明

aes.open() 関数呼び出しで指定されたチェーン モードとパディング モードを使用した AES 最終暗号化。

decrypt

uint32_t(* aes_api_t::decrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)

概要説明

ECB モードでの AES 128 復号化。

詳細説明

128 ビット AES キーを使用して ECB モードで入力データを復号化します

表 310: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	128 ビット プレーン キー
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用
num_words	複数のビットを書き換えることもできます。	p_source および p_dest データ バッファのサイズ （ワード単位）。1 ワードは 4 バイトです。4 ワードの倍数である必要があります。
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ
*p_dest	out	出力データ バッファ

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

setGcmTag

uint32_t(* aes_api_t::setGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source)

概要説明

モード固有のパラメータを設定します

詳細説明

表 311: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ
num_words	複数のビットを書き換えることもできます。	p_source バッファ内のワード数。4 ワード以上である必要があります
p_source	複数のビットを書き換えることもできます。	認証タグ データ バッファへのポインタは、4 ワードのサイズである必要があります。

定義 : aes_ctrl_t

AES インタフェース制御構造体

パラメータ num_words

uint32_t

パラメータ p_source

uint32_t

getGcmTag

uint32_t(* aes_api_t::getGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest)

概要説明

認証タグ データを取得します。

詳細説明

表 312: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ
num_words	複数のビットを書き換えることもできます。	バッファ p_dest 内のワード数。4 ワード以上である必要があります

表 312: パラメータ (続き)

名前	方向	説明
p_dest	複数のビットを書き換えることもできます。	データ バッファのポインタ、4 ワードのサイズである必要があります。

定義 : aes_ctrl_t

AES インタフェース制御構造体

パラメータ num_words

uint32_t

パラメータ p_dest

uint32_t

close

uint32_t(* aes_api_t::close)(aes_ctrl_t *const p_ctrl)

詳細説明

AES モジュールを閉じます。

表 313: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ

定義 : aes_ctrl_t

AES インタフェース制御構造体

zeroPaddingEncrypt

uint32_t(* aes_api_t::zeroPaddingEncrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)

概要説明

指定されたチェーン モードとパディング モードを使用した AES 暗号化。

詳細説明

表 314: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	AES プレーンテキスト キーへのポインタ、バッファ サイズはキーの長さと同じである必要があります
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用、バッファ サイズは 16 バイトである必要があります
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ、
*p_dest	out	出力データ バッファ

I：この関数はスレッド セーフではありません。

定義 : aes_ctrl_t

AES インタフェース制御構造体

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

zeroPaddingDecrypt

uint32_t(* aes_api_t::zeroPaddingDecrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)

概要説明

指定されたチェーン モードとパディング モードを使用した AES 復号化。

詳細説明

表 315: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	AES プレーンテキスト キーへのポインタ、バッファ サイズはキーの長さと同じである必要があります
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用、バッファ サイズは 16 バイトである必要があります
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ、
*p_dest	out	出力データ バッファ

I：この関数はスレッド セーフではありません。

定義 : aes_ctrl_t

AES インタフェース制御構造体

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words
パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

versionGet

uint32_t(* aes_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 316: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

aes_instance_t

aes_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- aes_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- aes_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- aes_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

g_sce_crypto_api

crypto_api_t::g_sce_crypto_api

g_aes128ecb_on_sce

aes_api_t::g_aes128ecb_on_sce

詳細説明

AES 128 ビット ECB モード実装

g_aes128cbc_on_sce

aes_api_t::g_aes128cbc_on_sce

詳細説明

AES 128 ビット CBC モード実装

g_aes128ctr_on_sce

aes_api_t::g_aes128ctr_on_sce

詳細説明

AES 128 ビット CTR モード実装

g_aes128gcm_on_sce

aes_api_t::g_aes128gcm_on_sce

詳細説明

AES 128 ビット GCM モード実装

g_aes128xts_on_sce

aes_api_t::g_aes128xts_on_sce

詳細説明

AES 128 ビット CCM モード実装

g_aes192ecb_on_sce

aes_api_t::g_aes192ecb_on_sce

詳細説明

AES 192 ビット ECB モード実装

g_aes192cbc_on_sce

aes_api_t::g_aes192cbc_on_sce

詳細説明

AES 192 ビット CBC モード実装

g_aes192ctr_on_sce

aes_api_t::g_aes192ctr_on_sce

詳細説明

AES 192 ビット CTR モード実装

g_aes192gcm_on_sce

aes_api_t::g_aes192gcm_on_sce

詳細説明

AES 192 ビット GCM モード実装

g_aes256ecb_on_sce

aes_api_t::g_aes256ecb_on_sce

詳細説明

AES 256 ビット ECB モード実装

g_aes256cbc_on_sce

aes_api_t::g_aes256cbc_on_sce

詳細説明

AES 256 ビット CBC モード実装

g_aes256ctr_on_sce

aes_api_t::g_aes256ctr_on_sce

詳細説明

AES 256 ビット CTR モード実装

g_aes256gcm_on_sce

aes_api_t::g_aes256gcm_on_sce

詳細説明

AES 256 ビット GCM モード実装

g_aes256xts_on_sce

aes_api_t::g_aes256xts_on_sce

詳細説明

AES 256 ビット XTS モード実装

g_aes128gcm_on_sceHrk

aes_api_t::g_aes128gcm_on_sceHrk

g_aes192gcm_on_sceHrk

aes_api_t::g_aes192gcm_on_sceHrk

g_aes256gcm_on_sceHrk

aes_api_t::g_aes256gcm_on_sceHrk

7.6.6.2 DSA インタフェース

DSA （デジタル署名アルゴリズム）による署名の生成と検証 API。

インタフェース API

dsa_api_t

関数名	説明
.open	DSA モジュールのオープン関数です。暗号化 / 復号化操作を実行する前に呼び出す必要があります。
.verify	DSA 公開キーを使用した DSA 署名の検証。指定された DSA 公開キーと入力メッセージ ハッシュのドメイン パラメータを使用して、バッファの DSA 署名を検証します
.hashVerify	DSA 公開キーを使用した DSA 署名の検証。指定された DSA 公開キーと入力メッセージ ハッシュのドメイン パラメータを使用して、バッファの DSA 署名を検証します
.sign	DSA 秘密キーを使用した DSA 署名の生成。ドメイン パラメータに対して指定された DSA 秘密キーを使用して、バッファに対する署名を生成します。結果はバッファに書き込まれます
.hashSign	DSA 秘密キーを使用した DSA 署名の生成。ドメイン パラメータに対して指定された DSA 秘密キーを使用して、バッファに対する署名を生成します。結果はバッファに書き込まれます
.close	DSA モジュールを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

データ構造体

- dsa_ctrl_t
- dsa_cfg_t
- dsa_instance_t

変数

- g_sce_crypto_api
- g_dsa1024_160_on_sce
- g_dsa2048_224_on_sce
- g_dsa2048_256_on_sce

定義

- #define DSA_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define DSA_API_VERSION_MINOR

  初期値 :(00)

dsa_ctrl_t

dsa_ctrl_t

詳細説明

DSA インタフェース制御構造体

変数

- crypto_ctrl_t * p_crypto_ctrl

  暗号化エンジン制御構造体へのポインタ
- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

dsa_cfg_t

dsa_cfg_t

詳細説明

DSA インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

変数

- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

dsa_api_t

dsa_api_t

詳細説明

HAL レイヤーに実装された DSA_Interface SCE 関数は、この API に従います。

open

uint32_t(* dsa_api_t::open)(dsa_ctrl_t *const p_ctrl, dsa_cfg_t const *const p_cfg)

詳細説明

DSA モジュールのオープン関数です。暗号化 / 復号化操作を実行する前に呼び出す必要があります。

表 317: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	DSA インタフェースの制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	DSA 設定の制御構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : dsa_ctrl_t

DSA インタフェース制御構造体

定義 :

定義 : dsa_cfg_t const *const p_cfg

DSA インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

- dsa_cfg_t::crypto_api_t

  暗号化エンジン API へのポインタ

verify

uint32_t(* dsa_api_t::verify)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash)

概要説明

DSA 公開キーを使用した DSA 署名の検証。

詳細説明

指定された DSA 公開キー p_key と、入力メッセージ ハッシュ p_paddedHash 用の p_domain からのドメイン パラメータを使用して、バッファ p_signature からの DSA 署名を検証します

表 318: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	DSA プレーン テキスト キーへのポインタ。
*p_domain	複数のビットを書き換えることもできます。	DSA ドメイン パラメータへのポインタ。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_signature	複数のビットを書き換えることもできます。	検証する署名データ バッファ
*p_paddedHash	複数のビットを書き換えることもできます。	入力メッセージのパディングされたハッシュ

非推奨の

この関数は非推奨です。代わりに関数 hashVerify を使用する必要があります。

パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_signature

uint32_t

パラメータ *p_paddedHash

uint32_t

hashVerify

uint32_t(* dsa_api_t::hashVerify)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash)

概要説明

DSA 公開キーを使用した DSA 署名の検証。

詳細説明

指定された DSA 公開キー p_key と、入力メッセージ ハッシュ p_paddedHash 用の p_domain からのドメイン パラメータを使用して、バッファ p_signature からの DSA 署名を検証します

表 319: パラメータ

名前	方向	説明
*p_ctrl	複数のビットを書き換えることもできます。	DSA 制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	DSA プレーン テキスト キーへのポインタ。
*p_domain	複数のビットを書き換えることもできます。	DSA ドメイン パラメータへのポインタ。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_signature	複数のビットを書き換えることもできます。	検証する署名データ バッファ
*p_paddedHash	複数のビットを書き換えることもできます。	入力メッセージのパディングされたハッシュ

パラメータ *p_ctrl

パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_signature

uint32_t

パラメータ *p_paddedHash

uint32_t

sign

uint32_t(* dsa_api_t::sign)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)

概要説明

DSA 秘密キーを使用した DSA 署名の生成。

詳細説明

ドメイン パラメータ p_domain に対して指定された DSA 秘密キー p_key を使用して、バッファ p_paddedHash に対する署名を生成します。結果はバッファ p_dest に書き込まれます

表 320: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	DSA プレーン テキスト秘密キー
*p_domain	複数のビットを書き換えることもできます。	DSA ドメイン パラメータへのポインタ。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_paddedHash	複数のビットを書き換えることもできます。	入力データ バッファ
*p_dest	out	出力データ バッファ

非推奨の

この関数は非推奨です。代わりに関数 hashSign を使用する必要があります。

パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_paddedHash

uint32_t

パラメータ *p_dest

uint32_t

hashSign

uint32_t(* dsa_api_t::hashSign)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)

概要説明

DSA 秘密キーを使用した DSA 署名の生成。

詳細説明

ドメイン パラメータ p_domain に対して指定された DSA 秘密キー p_key を使用して、バッファ p_paddedHash に対する署名を生成します。結果はバッファ p_dest に書き込まれます

表 321: パラメータ

名前	方向	説明
*p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	DSA プレーン テキスト秘密キー
*p_domain	複数のビットを書き換えることもできます。	DSA ドメイン パラメータへのポインタ。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_paddedHash	複数のビットを書き換えることもできます。	入力データ バッファ
*p_dest	out	出力データ バッファ

パラメータ *p_ctrl
パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_paddedHash

uint32_t

パラメータ *p_dest

uint32_t

close

uint32_t(* dsa_api_t::close)(dsa_ctrl_t *const p_ctrl)

詳細説明

DSA モジュールを閉じます。

表 322: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ

定義 : dsa_ctrl_t

DSA インタフェース制御構造体

versionGet

uint32_t(* dsa_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 323: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

dsa_instance_t

dsa_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- dsa_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- dsa_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- dsa_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

g_sce_crypto_api

crypto_api_t::g_sce_crypto_api

g_dsa1024_160_on_sce

dsa_api_t::g_dsa1024_160_on_sce

詳細説明

DSA API の SCE/DSA 実装。

g_dsa2048_224_on_sce

dsa_api_t::g_dsa2048_224_on_sce

詳細説明

DSA API の SCE/DSA 実装。

g_dsa2048_256_on_sce

dsa_api_t::g_dsa2048_256_on_sce

詳細説明

DSA API の SCE/DSA 実装。

7.6.6.3 HASH アルゴリズムインタフェース

HASH_Interface ハッシュアルゴリズム API。

インタフェース API

hash_api_t

参考資料

関数名	説明
.open	HASH インタフェース：初期設定
.updateHash	ソース バッファのワードのハッシュを更新します。
.hashUpdate	ソース バッファのワードのハッシュを更新します。
.close	
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

データ構造体

- hash_cfg_t
- hash_ctrl_t
- hash_instance_t

変数

- g_sha1_hash_on_sce
- g_sha256_hash_on_sce

定義

- #define HASH_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define HASH_API_VERSION_MINOR

  初期値 :(00)
- #define HASH_MESSAGE_BLOCK_SIZE_WORDS

  初期値 :(16)
- #define HASH_MESSAGE_BLOCK_SIZE_BYTES

  初期値 :((HASH_MESSAGE_BLOCK_SIZE_WORDS) * 4)
- #define HASH_MAX_DIGEST_SIZE_WORDS

  初期値 :(8)
- #define HASH_MAX_DIGEST_SIZE_BYTES

  初期値 :((HASH_DIGEST_SIZE_WORDS) * 4)

hash_cfg_t

hash_cfg_t

詳細説明

HASH_Interface 設定構造体

変数

- crypto_ctrl_t * p_crypto_ctrl

  暗号化エンジン制御構造体へのポインタ

- crypto_api_t const * p_crypto_api

  暗号化エンジン API 構造体へのポインタ

hash_ctrl_t

hash_ctrl_t

詳細説明

HASH_Interface 制御構造体

変数

- uint32_t msgbuf[HASH_MESSAGE_BLOCK_SIZE_WORDS]

  ハッシュされるメッセージ バッファ

- uint32_t hash[HASH_MAX_DIGEST_SIZE_WORDS]

  現在のハッシュ値

- uint64_t length

  64 ビット メッセージ長 （ビット数）

hash_api_t

hash_api_t

詳細説明

HAL レイヤーに実装された HASH_Interface SCE 関数は、この API に従います。

open

uint32_t(* `hash_api_t`::open)(`hash_ctrl_t` *const p_ctrl, `hash_cfg_t` const *const p_cfg)

詳細説明

HASH インタフェース：初期設定

l :HASH_Interface: この関数を呼び出す前に、周辺クロックおよび必要なすべての出力ピンを構成する必要があります。

表 324: パラメータ

名前	方向	説明
HASH_Interface	入力 / 出力	p_ctrl 制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
HASH_Interface	複数のビットを書き換えることもできます。	p_cfg 構成構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

パラメータ HASH_Interface

パラメータ HASH_Interface

updateHash

uint32_t(* hash_api_t::updateHash)(const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)

詳細説明

ソース バッファ p_source の num_words ワードのハッシュを更新します。

表 325: パラメータ

名前	方向	説明
p_source	複数のビットを書き換えることもできます。	入力メッセージ バッファへのポインタ。サイズは 64 バイトの倍数である必要があります
num_words	複数のビットを書き換えることもできます。	バッファ p_source からハッシュされるワード数
p_dest	入力 / 出力	メッセージ ダイジェストへのポインタ。入力時に初期価値が含まれます。

非推奨の

: この関数は非推奨です。代わりに関数 hashUpdate を使用する必要があります。

パラメータ p_source

uint32_t

パラメータ num_words

uint32_t

パラメータ p_dest

uint32_t

hashUpdate

uint32_t(* hash_api_t::hashUpdate)(hash_ctrl_t *const p_ctrl, const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)

詳細説明

ソース バッファ p_source の num_words ワードのハッシュを更新します。

表 326: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	hash_ctrl_t 制御構造体へのポインタ。
p_source	複数のビットを書き換えることもできます。	入力メッセージ バッファへのポインタ。サイズは 64 バイトの倍数である必要があります
num_words	複数のビットを書き換えることもできます。	バッファ p_source からハッシュされるワード数
p_dest	入力 / 出力	メッセージ ダイジェストへのポインタ。入力時に初期価値が含まれます。

定義 : hash_ctrl_t

HASH_Interface 制御構造体

パラメータ p_source

uint32_t

パラメータ num_words

uint32_t

パラメータ p_dest

uint32_t

close

uint32_t(* hash_api_t::close)(hash_ctrl_t *const p_ctrl)

versionGet

uint32_t(* hash_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 327: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

hash_instance_t

hash_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- hash_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- hash_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- hash_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

g_sha1_hash_on_sce

hash_api_t::g_sha1_hash_on_sce

詳細説明

HASH API の SHA1 実装。

g_sha256_hash_on_sce

hash_api_t::g_sha256_hash_on_sce

詳細説明

HASH API の SHA256 実装。

7.6.6.4 RSA インタフェース

署名の生成、確認、暗号化、および復号化用の RSA 暗号関数。

インタフェース API

rsa_api_t

関数名	説明
.open	RSA モジュールのオープン関数。暗号化 / 復号化操作または署名 / 検証操作を実行する前に、呼び出す必要があります。
.encrypt	p_key の RSA 公開キーを使用して、p_source のソースデータを暗号化し、結果を宛先バッファ p_dest に書き込みます。
.decrypt	p_key の RSA 秘密キーを使用して、p_source のソースデータを復号化し、結果を宛先バッファ p_dest に書き込みます。
.decryptCrt	p_key の RSA 秘密キーを使用して、p_source のソースデータを復号化し、結果を宛先バッファ p_dest に書き込みます。RSA 秘密キー データは、CRT 形式で指定されます。
.verify	バッファ p_padded_hash 内のパディングされたメッセージ ハッシュに関するバッファ p_signature 内の署名を、RSA 公開キー p_key を使用して検証します。
.sign	RSA 秘密キー p_key を使用して、パディングされたハッシュ バッファ p_padded_hash に対する署名を生成します。結果をバッファ p_dest に書き込みます。
.signCrt	RSA 秘密キー p_key を使用して、パディングされたハッシュ バッファ p_padded_hash に対する署名を生成します。RSA 秘密キー p_key は、CRT 形式と想定されます。結果をバッファ p_dest に書き込みます。
.close	RSA モジュールを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

データ構造体

- rsa_ctrl_t

- rsa_cfg_t
- rsa_instance_t

変数

- g_sce_crypto_api
- g_rsa1024_on_sce
- g_rsa2048_on_sce

定義

- #define RSA_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define RSA_API_VERSION_MINOR

  初期値 :(00)

rsa_ctrl_t

rsa_ctrl_t

詳細説明

RSA インタフェース制御構造体

変数

- crypto_ctrl_t * p_crypto_ctrl

  暗号化エンジン制御構造体へのポインタ
- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ
- uint32_t stage_num

  処理段階

rsa_cfg_t

rsa_cfg_t

詳細説明

RSA インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

変数

- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

rsa_api_t

rsa_api_t

詳細説明

HAL レイヤーに実装された RSA_Interface SCE 関数は、この API に従います。

open

uint32_t(* rsa_api_t::open)(rsa_ctrl_t *const p_ctrl, rsa_cfg_t const *const p_cfg)

詳細説明

RSA モジュールのオープン関数です。暗号化 / 復号化操作または署名 / 検証操作を実行する前に、呼び出す必要があります。

表 328: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	RSA インタフェースの制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	RSA 設定の制御構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : rsa_ctrl_t

RSA インタフェース制御構造体

定義 :

定義 : rsa_cfg_t const *const p_cfg

RSA インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

- rsa_cfg_t::crypto_api_t

  暗号化エンジン API へのポインタ

encrypt

uint32_t(* rsa_api_t::encrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

概要説明

p_key の RSA 公開キーを使用して、p_source のソース データを暗号化し、結果を宛先バッファ p_dest に書き込みます。

詳細説明

表 329: パラメータ

名前	方向	説明
*p_ctrl	複数のビットを書き換えることもできます。	RSA インタフェースの制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	RSA プレーンテキスト公開キーへのポインタ。
*p_domain	複数のビットを書き換えることもできます。	このパラメータは、RSA 暗号化では使用されません。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトです。
*p_source	複数のビットを書き換えることもできます。	暗号化するソース データ バッファ。
*p_dest	out	書き込み先のデータ バッファ。暗号化の結果は、ここに格納されます。

パラメータ *p_ctrl
パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

decrypt

uint32_t(* rsa_api_t::decrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

概要説明

p_key の RSA 秘密キーを使用して、p_source のソース データを復号化し、結果を宛先バッファ p_dest に書き込みます。

詳細説明

表 330: パラメータ

名前	方向	説明
*p_ctrl	複数のビットを書き換えることもできます。	RSA インタフェースの制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	RSA プレーンテキスト秘密キーへのポインタ。
*p_domain	複数のビットを書き換えることもできます。	このパラメータは、RSA 暗号化では使用されません。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトです。
*p_source	複数のビットを書き換えることもできます。	復号化する入力データ バッファ。
*p_dest	out	出力の書き込み先のデータ バッファ。復号化の結果は、ここに格納されます。

パラメータ *p_ctrl
パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

decryptCrt

uint32_t(* rsa_api_t::decryptCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

概要説明

p_key の RSA 秘密キーを使用して、p_source のソース データを復号化し、結果を宛先バッファ p_dest に書き込みます。RSA 秘密キー データは、CRT 形式で指定されます。

詳細説明

表 331: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	CRT 形式の RSA プレーンテキスト秘密キーへのポインタ。
*p_domain	複数のビットを書き換えることもできます。	このパラメータは、RSA 暗号化では使用されません。
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトです。
*p_source	複数のビットを書き換えることもできます。	復号化する入力データ バッファ。
*p_dest	out	出力の書き込み先のデータ バッファ。復号化の結果は、ここに格納されます。

パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

verify

uint32_t(* rsa_api_t::verify)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash)

概要説明

バッファ p_padded_hash 内のパディングされたメッセージ ハッシュに関するバッファ p_signature 内の署名を、RSA 公開キー p_key を使用して検証します。

詳細説明

表 332: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	RSA プレーンテキスト公開キーへのポインタ。
*p_domain	複数のビットを書き換えることもできます。	このパラメータは、RSA 暗号化機能では不要です
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトです。
*p_signature	複数のビットを書き換えることもできます。	検証が必要な署名データ
*p_paddedHash	複数のビットを書き換えることもできます。	入力メッセージ バッファのパディングされたハッシュ値

パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_signature

uint32_t

パラメータ *p_paddedHash

sign

uint32_t(* rsa_api_t::sign)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest)

概要説明

RSA 秘密キー p_key を使用して、パディングされたハッシュ バッファ p_padded_hash に対する署名を生成します。結果をバッファ p_dest に書き込みます。

詳細説明

表 333: パラメータ

名前	方向	説明
*p_ctrl	複数のビットを書き換えることもできます。	RSA インタフェースの制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	RSA 秘密キーへのポインタ
*p_domain	複数のビットを書き換えることもできます。	このパラメータは使用されません
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_padded_hash	複数のビットを書き換えることもできます。	RSA 署名の対象となる入力メッセージ用のパディングされたハッシュ
*p_dest	out	生成された署名データは、ここに書き込まれます。

パラメータ *p_ctrl
パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_padded_hash

uint32_t

パラメータ *p_dest

uint32_t

signCrt

uint32_t(* rsa_api_t::signCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest)

概要説明

RSA 秘密キー p_key を使用して、パディングされたハッシュ バッファ p_padded_hash に対する署名を生成します。RSA 秘密キー p_key は、CRT 形式と想定されます。結果をバッファ p_dest に書き込みます。

詳細説明

表 334: パラメータ

名前	方向	説明
*p_ctrl	複数のビットを書き換えることもできます。	RSA インタフェースの制御構造体へのポインタ
*p_key	複数のビットを書き換えることもできます。	RSA 秘密キーへのポインタ
*p_domain	複数のビットを書き換えることもできます。	このパラメータは使用されません
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_padded_hash	複数のビットを書き換えることもできます。	RSA 署名の対象となる入力メッセージ用のパディングされたハッシュ
*p_dest	out	生成された署名データは、ここに書き込まれます。

パラメータ *p_ctrl
パラメータ *p_key

uint32_t

パラメータ *p_domain

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_padded_hash

uint32_t

パラメータ *p_dest

uint32_t

close

uint32_t(* rsa_api_t::close)(rsa_ctrl_t *const p_ctrl)

詳細説明

RSA モジュールを閉じます。

表 335: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ

定義 : rsa_ctrl_t

RSA インタフェース制御構造体

versionGet

uint32_t(* rsa_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 336: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

rsa_instance_t

rsa_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- rsa_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- rsa_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- rsa_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

g_sce_crypto_api

crypto_api_t::g_sce_crypto_api

g_rsa1024_on_sce

rsa_api_t::g_rsa1024_on_sce

詳細説明

RSA API の SCE/RSA 実装。

g_rsa2048_on_sce

rsa_api_t::g_rsa2048_on_sce

7.6.6.5 TDES インタフェース

TDES 暗号化 / 復号化 API。

インタフェース API

tdes_api_t

関数名	説明
.open	TDES モジュールのオープン関数。暗号化 / 復号化操作を実行する前に呼び出す必要があります。
.encrypt	tdes.open() 関数呼び出しで指定されたチェーン モードとパディング モードを使用した TDES 暗号化。192 ビット TDES キーを使用して ECB モードで入力データを暗号化します
.decrypt	ECB モードでの TDES 復号化。192 ビット TDES キーを使用して ECB モードで入力データを復号化します
.close	TDES モジュールを閉じます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

データ構造体

- tdes_ctrl_t
- tdes_cfg_t
- tdes_instance_t

変数

- g_sce_crypto_api
- g_tdes192ecb_on_sce
- g_tdes192cbc_on_sce
- g_tdes192ctr_on_sce

定義

- #define TDES_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define TDES_API_VERSION_MINOR

  初期値 :(00)

tdes_ctrl_t

tdes_ctrl_t

詳細説明

TDES インタフェース制御構造体

変数

- crypto_ctrl_t crypto_ctrl

  暗号化制御構造体へのポインタ
- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

tdes_cfg_t

tdes_cfg_t

詳細説明

TDES インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

変数

- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ

tdes_api_t

tdes_api_t

詳細説明

HAL レイヤーに実装された TDES_Interface SCE 関数は、この API に従います。

open

uint32_t(* tdes_api_t::open)(tdes_ctrl_t *const p_ctrl, tdes_cfg_t const *const p_cfg)

詳細説明

TDES モジュールのオープン関数です。暗号化 / 復号化操作を実行する前に呼び出す必要があります。

表 337: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	TDES インタフェースの制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	TDES 設定の制御構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : tdes_ctrl_t

TDES インタフェース制御構造体

定義 :

定義 : tdes_cfg_t const *const p_cfg

TDES インタフェース設定構造体。ユーザーは、open() 関数を呼び出す前に、これらの値を指定する必要があります

- tdes_cfg_t::crypto_api_t

  暗号化エンジン API へのポインタ

encrypt

uint32_t(* tdes_api_t::encrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

概要説明

tdes.open() 関数呼び出しで指定されたチェーン モードとパディング モードを使用した TDES 暗号化。

詳細説明

192 ビット TDES キーを使用して ECB モードで入力データを暗号化します

表 338: パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	TDES プレーンテキスト キーへのポインタ
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ
*p_dest	out	出力データ バッファ

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

decrypt

uint32_t(* tdes_api_t::decrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)

概要説明

ECB モードでの TDES 復号化。

詳細説明

192 ビット TDES キーを使用して ECB モードで入力データを復号化します

表 339:　パラメータ

名前	方向	説明
*p_key	複数のビットを書き換えることもできます。	192 ビット プレーン キー
*p_iv	複数のビットを書き換えることもできます。	ECB モードでは不使用
num_words	複数のビットを書き換えることもできます。	ワード単位のデータ バッファ サイズ。1 ワードは 4 バイトであるため、4 の倍数
*p_source	複数のビットを書き換えることもできます。	入力データ バッファ
*p_dest	out	出力データ バッファ

パラメータ *p_key

uint32_t

パラメータ *p_iv

uint32_t

パラメータ num_words

uint32_t

パラメータ *p_source

uint32_t

パラメータ *p_dest

uint32_t

close

uint32_t(* tdes_api_t::close)(tdes_ctrl_t *const p_ctrl)

詳細説明

TDES モジュールを閉じます。

表 340: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ

定義 : tdes_ctrl_t

TDES インタフェース制御構造体

versionGet

uint32_t(* tdes_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 341: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

tdes_instance_t

tdes_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- tdes_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- tdes_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- tdes_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

g_sce_crypto_api

crypto_api_t::g_sce_crypto_api

g_tdes192ecb_on_sce

tdes_api_t::g_tdes192ecb_on_sce

詳細説明

TDES API の SCE/TDES 実装。

g_tdes192cbc_on_sce

tdes_api_t::g_tdes192cbc_on_sce

g_tdes192ctr_on_sce

tdes_api_t::g_tdes192ctr_on_sce

7.6.6.6 乱数生成

RNG_Interface 乱数発生。

インタフェース API

trng_api_t

関数名	説明
.open	TRNG ドライバを開いて、ハードウェア TRNG モジュールから乱数データを読み取ります
.read	乱数バイトを生成して、バッファに保存します
.close	TRNG インタフェース ドライバを閉じます
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

データ構造体

- trng_ctrl_t
- trng_cfg_t
- trng_instance_t

定義

- #define TRNG_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define TRNG_API_VERSION_MINOR

  初期値 :(00)
- #define TRNG_REGISTER_SIZE_WORDS

  初期値 :(4)
- #define TRNG_REGISTER_SIZE_BYTES

  初期値 :((TRNG_REGISTER_SIZE_WORDS) * 4)

trng_ctrl_t

trng_ctrl_t

詳細説明

TRNG_Interface 制御構造体。

変数

- uint32_t nattempts

  リトライ回数
- crypto_ctrl_t * p_crypto_ctrl

  暗号化制御構造体へのポインタ
- crypto_api_t const * p_crypto_api

  暗号化エンジン API へのポインタ
- uint32_t prevbuf[TRNG_REGISTER_SIZE_WORDS]

  前の乱数データ
- uint32_t currbuf[TRNG_REGISTER_SIZE_WORDS]

  現在の乱数データ

trng_cfg_t

trng_cfg_t

詳細説明

TRNG インタフェース設定パラメータ

変数

- crypto_api_t const * p_crypto_api

  暗号化 API へのポインタ

- uint32_t nattempts

  連続してテストに失敗した場合のリトライ回数

trng_api_t

trng_api_t

詳細説明

HAL レイヤーに実装された TRNG_Interface SCE 関数は、この API に従います。

open

uint32_t(* trng_api_t::open)(trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)

詳細説明

TRNG ドライバを開いて、ハードウェア TRNG モジュールから乱数データを読み取ります

表 342: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御構造体へのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : trng_ctrl_t

TRNG_Interface 制御構造体。

定義 :

定義 : trng_cfg_t const *const p_cfg

TRNG インタフェース設定パラメータ

- trng_cfg_t::crypto_api_t

  暗号化 API へのポインタ

- trng_cfg_t::nattempts

  連続してテストに失敗した場合のリトライ回数

read

uint32_t(* trng_api_t::read)(trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)

詳細説明

nbytes の乱数バイトを生成し、p_rngbuf バッファに格納します

表 343: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	TRNG 制御構造体へのポインタ
p_rngbuf	out	生成された乱数は、バッファ p_rngbuf に格納されます
nbytes	複数のビットを書き換えることもできます。	生成する乱数バイトの数

定義 : trng_ctrl_t

TRNG_Interface 制御構造体。

パラメータ p_rngbuf

uint32_t

パラメータ nbytes

close

uint32_t(* trng_api_t::close)(trng_ctrl_t *const p_ctrl)

詳細説明

TRNG インタフェース ドライバを閉じます

表 344: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	TRNG インタフェース制御構造体へのポインタ

定義 : trng_ctrl_t

TRNG_Interface 制御構造体。

versionGet

uint32_t(* trng_api_t::versionGet)(*const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。

表 345: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

trng_instance_t

trng_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- trng_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- trng_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- trng_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.7 CTSU インタフェース

静電容量方式タッチ コントローラのインタフェース。

7.7.1 概要

CTSU インタフェースには、引数として渡される構成に応じて CTSU をオープン、クローズ、実行、および制御するための機能が用意されています。

以下によって実装されます。CTSU

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

CTSU インタフェースの説明 : CTSU ドライバ

7.7.2 インタフェース API

ctsu_api_t

関数名	説明
.open	CTSU を初期化します。電源とクロックを有効化し、レジスタ構成を設定します。
.close	進行中のスキャンを停止し、割り込みを無効化し、周辺機能への電源を遮断し、オプションに従って構成を保存して、指定された CTSU を閉じます。
.scan	1 回の CTSU スキャンを開始します。
.update	スキャン結果に従って、タッチ判定やその他の派生データなどの CTSU 内部データを更新します。
.read	選択したオプションに従って、未加工データ、バイナリデータ、その他の派生データなどの結果を CTSU から読み取ります。
.versionGet	API バージョンを取得します。

7.7.3 データ構造体

- ctsu_callback_args_t
- ctsu_channel_pair_t
- ctsu_channel_setting_t
- ctsu_channel_data_self_t
- ctsu_channel_data_mutual_t
- ctsu_hw_cfg_t
- ctsu_functions_t
- ctsu_cfg_t
- ctsu_ctrl_t
- ctsu_instance_t

7.7.4 列挙

- ctsu_event_t
- ctsu_read_t
- ctsu_process_option_t
- ctsu_close_option_t
- ctsu_action_t

7.7.5 定義

- #define CTSU_API_VERSION_MAJOR

  初期値 :(01)

  ボードと MCU 関連のヘッダー ファイルを追加します。このファイルで定義された API のバージョン
- #define CTSU_API_VERSION_MINOR

  初期値 :(1)

7.7.6 API データ

7.7.6.1 ctsu_event_t

ctsu_event_t

詳細説明

CTSU コールバック イベントの定義

列挙値

名前	説明
CTSU_EVENT_SCAN_COMPLETE	スキャン サイクルが完了しました。このコールバックは、ISR から実行されます。
CTSU_EVENT_PARAMETERS_UPDATED	未加工 CTSU データから得られたすべての派生パラメータが更新されます。コールバックは ISR からではありません。

7.7.6.2 ctsu_read_t

ctsu_read_t

詳細説明

ユーザーが CTSU スキャン結果の照会に使用できる複数のオプション。これらのオプションは、論理和（OR）を使用して組み合わせることはできません。

列挙値

名前	説明
CTSU_READ_BINARY_DATA_ALL	タッチ判定のバイナリ文字列を読み取ります。
CTSU_READ_BINARY_DATA_SEL	選択したチャネルのみについて、タッチ判定のバイナリ文字列を読み取ります。
CTSU_READ_RAW_SENSOR_OUTPUT_ALL	すべてのアクティブ チャネルについて、未加工の CTSU 出力を読み取ります。
CTSU_READ_RAW_SENSOR_OUTPUT_SEL	選択したチャネルについて、未加工の CTSU 出力を読み取ります。
CTSU_READ_FILTERED_SENSOR_VALUES_ALL	すべてのアクティブ チャネルについて、フィルタから得た scount 値出力を読み取ります。
CTSU_READ_FILTERED_SENSOR_VALUES_SEL	すべてのアクティブ チャネルについて、フィルタから得た scount 値出力を読み取ります。

名前	説明
CTSU_READ_DELTA_COUNT_ALL	すべてのアクティブ センサーについて、センサー カウントとベースライン カウントの差を p_dest 配列に書き出します。
CTSU_READ_DELTA_COUNT_SEL	選択したチャネルについて、センサー カウントとベースライン カウントの差を p_dest 配列に書き出します。
CTSU_READ_BASELINE_COUNT_ALL	すべてのアクティブ チャネルについて、チャネルのタッチ判定に使用される現在のセンサー ベースラインを読み取ります。
CTSU_READ_BASELINE_COUNT_SEL	選択したチャネルについて、チャネルのタッチ判定に使用される現在のセンサー ベースラインを読み取ります。
CTSU_READ_FILTERED_REF_ICO_VALUES_ALL	すべてのアクティブ チャネルについて、フィルタから得た rcount 値出力を読み取ります。
CTSU_READ_FILTERED_REF_ICO_VALUES_SEL	選択したチャネルのみについて、フィルタから得た rcount 値出力を読み取ります。

7.7.6.3 ctsu_process_option_t

ctsu_process_option_t

詳細説明

プロセス関数で指定可能な複数のオプション。

列挙値

名前	説明
CTSU_PROCESS_OPTION_DEFAULT_SETTING	ほとんどのユース ケースで推奨されるオプションです。
CTSU_PROCESS_OPTION_AUTO_SCAN	すべての内部変数が更新された後、自動的にスキャンを開始します。
CTSU_PROCESS_OPTION_ENABLE_DRIFT_COMP	ドリフト補正を実行し、内部変数が更新された時点で新しいスキャンを開始します。
CTSU_PROCESS_OPTION_ENABLE_AUTO_CALIB	自動較正を実行し、内部変数が更新された時点で新しいスキャンを開始します。
CTSU_PROCESS_OPTION_NONE	すべてのオプションのアクションを省略します。

7.7.6.4 ctsu_close_option_t

ctsu_close_option_t

詳細説明

クローズ関数に渡すことができるオプション。これらのオプションは、論理和 （OR）を使用して組み合わせることができます。

列挙値

名前	説明
CTSU_CLOSE_OPTION_SUSPEND	CTSUSNZ ビットを設定して CTSU をサスペンドします。
CTSU_CLOSE_OPTION_SAVE_CONFIG	クローズ関数に対して指定された 2 番目の引数に構成を保存します。
CTSU_CLOSE_OPTION_RESET_SFRS	SFR をその P-O-RS 値に戻します。
CTSU_CLOSE_OPTION_POWER_DOWN	CTSU へのクロック供給を無効化し、CTSU をローパワー モードに設定します。

7.7.6.5 ctsu_action_t

ctsu_action_t

詳細説明

CTSU センサー データ処理 / 更新関数の状態

列挙値

名前	説明
CTSU_ACTION_START_NEW_SCAN	新しいスキャンを開始します。
CTSU_ACTION_WAITING_FOR_SCAN_COMPLETE	スキャンの完了まで待機します。
CTSU_ACTION_CHECK_FOR_ERRORS	エラーがないかを確認します。
CTSU_ACTION_FILTER_DATA	データをフィルタリングします。
CTSU_ACTION_UPDATE_TOUCH_INFO	タッチ情報を更新します。
CTSU_ACTION_UPDATE_BUTTONS	ボタンを更新します。

名前	説明
CTSU_ACTION_UPDATE_SLIDERS	スライダを更新します。
CTSU_ACTION_RUN_DRIFT_COMP	ドリフト補正を実行します。
CTSU_ACTION_RUN_AUTO_TUNE	オート チューニングを実行します。
CTSU_ACTION_RESTART_STATE_MACHINE	状態マシンを再開始します。

7.7.7 API 構造

7.7.7.1 ctsu_callback_args_t

ctsu_callback_args_t

詳細説明

CTSU コールバック引数の定義

変数

- ctsu_event_t event

  CTSU コールバック イベント。

- void const * p_context

  ユーザー データのプレースホルダー。

7.7.7.2 ctsu_channel_pair_t

ctsu_channel_pair_t

詳細説明

タッチ センサー要素に関する情報を格納する構造体

変数

- int8_t rx

  プライマリ チャネルを示します。

- int8_t tx

  セカンダリ チャネルを示します （相互容量モードでのみ使用されます）

7.7.7.3 ctsu_channel_setting_t

ctsu_channel_setting_t

詳細説明

WR ISR で SFR に書き込まれる値を保持している構造体

変数

- uint16_t ctsussc

  CTSUSSC レジスタ用の値を保持します。

- uint16_t ctsuso0

  CTSUSO0 レジスタ用の値を保持します。

- uint16_t ctsuso1

  CTSUSO1 レジスタ用の値を保持します。

7.7.7.4 ctsu_channel_data_self_t

ctsu_channel_data_self_t

詳細説明

自己容量モードで、CTSU レジスタ CTSUSC （センサー カウント）および CTSURC （参照カウント）から即座に未加工データを読み取ります。

変数

- uint16_t sensor_count

  未加工センサー カウント。

- uint16_t reference_count

  未加工参照カウント。

7.7.7.5 ctsu_channel_data_mutual_t

ctsu_channel_data_mutual_t

詳細説明

相互容量モードで、CTSU レジスタ CTSUSC （センサー カウント）および CTSURC （参照カウント）から即座に未加工データを読み取ります。

変数

- uint16_t sen_cnt_1

  未加工センサー カウントのプライマリ リード。

- uint16_t ref_cnt_1

  未加工参照 ICO カウントのプライマリ リード。

- uint16_t sen_cnt_2

  未加工センサー ICO カウントのセカンダリ リード。

- uint16_t ref_cnt_2

  未加工参照 ICO カウントのセカンダリ リード。

7.7.7.6 ctsu_hw_cfg_t

ctsu_hw_cfg_t

詳細説明

Open 関数に渡されるハードウェア構成を定義する構造体

変数

- R_CTSU_Type ctsu_settings

  CR0、CR1、SDPRS、SST、CHACn、CHTRCn、DCLKC に対するユーザー定義の SFR 設定。

- ctsu_channel_setting_t * write_settings

  各アクティブ チャネルのしきい値に対するユーザー定義の初期設定。しきい値は、センサー カウントの実行時ベースラインとフィルタリングされた出力の差です。

  各アクティブ チャネルの SSC、SO0、SO1 に対するユーザー定義の設定。

- uint16_t * threshold

- uint16_t * hysteresis

  カウント値の許容誤差に対するユーザー定義の設定。

- uint16_t * baseline

  タッチしていない場合に、各アクティブ チャネルに対して想定されるカウントのベースライン。

- void * raw_result

  CTSU 測定の未加工の結果を保持するバッファへのポインタ。

- void * filter_output

  未加工の結果をフィルタリングした後の出力を保持するバッファへのポインタ。

- void * binary_result

  バイナリ データを保存できる場所へのポインタ。

- ctsu_channel_pair_t * excluded

  無視する必要があるチャネル ペアを rx、tx の順に昇順に並べたリストが含まれている配列へのポインタ。

- int8_t num_excluded

  除外された配列内の要素数。

- const uint16_t * series_resistance

  チャネルの抵抗 （調整時に RC 定数の判定に使用されます）

7.7.7.7 ctsu_functions_t

ctsu_functions_t

詳細説明

内部でドライバによって使用されるデフォルトの処理関数をオーバーライドするために使用できるユーザー定義の関数。高度な使用のみ。

変数

- int32_t(* preFilter)(void *p_args)

  データのフィルタリング前に、未加工データ SNR を計算するために使用されます。

- int32_t(* filter)(volatile uint16_t *output, volatile uint16_t *input)

  CTSU_CFG_FILTER_DEPTH を使用した加重平均。

- int32_t(* postFilter)(void *p_args)

  フィルタリング結果の処理。

- int32_t(* ctsuDecode)(void *p_args)

  チャネルがタッチされたかどうかを判定するアルゴリズム。

- int32_t(* otDriftComp)(void *p_args)

  初期化時にベースライン、エンベロープ、およびしきい値への操作を行うアルゴリズム。

- int32_t(* rtDriftComp)(void *p_args)

  実行時にベースライン、エンベロープ、およびしきい値への操作を行うアルゴリズム。

- int32_t(* otAutoTune)(void *p_args)

  システム初期化時に一度呼び出される関数。

- int32_t(* rtAutoTune)(void *p_args)

  システムの実行中にセンサーの自動調整のために呼び出される関数。

7.7.7.8 ctsu_cfg_t

ctsu_cfg_t

詳細説明

CTSU ドライバの構成構造体を定義する構造体

変数

- transfer_instance_t const *const p_lower_lvl_transfer_read

  リード結果への転送インスタンスへのポインタ。

- transfer_instance_t const *const p_lower_lvl_transfer_write

  write cfg への転送インスタンスへのポインタ。

- ctsu_hw_cfg_t * p_ctsu_hw_cfg

  CTSU 構成へのポインタ。

- ctsu_functions_t * p_ctsu_functions

  カスタム データ関数のプレース ホルダーへのポインタ。

- void(* p_callback)(ctsu_callback_args_t *p_args)

  スキャン完了時に使用する関数へのコールバック。

- void * p_context

  update_complete 通知に渡すデータへのポインタ。

- ctsu_process_option_t ctsu_soft_option

  Open および Process の実行時に使用するソフトウェア オプション。

- ctsu_close_option_t ctsu_close_option

  タッチ動作終了時に使用するソフトウェア オプション。

7.7.7.9 ctsu_ctrl_t

ctsu_ctrl_t

詳細説明

制御構造体

変数

- transfer_api_t const * p_api_transfer

  ローレベル転送ドライバ関数ポインタへのポインタ。

- transfer_ctrl_t * p_lowerl_lvl_transfer_read_ctrl

  転送リード制御へのポインタ。

- transfer_ctrl_t * p_lowerl_lvl_transfer_write_ctrl

  転送書き込み制御へのポインタ。

- bool ctsu_opened

  初期化状態を格納します。

- uint8_t ctsu_unit

  使用中の CTSU 単位。

- ctsu_hw_cfg_t * p_ctsu_hw_cfg

  CTSU 構成へのポインタ。

- ctsu_process_option_t ctsu_open_option

  Open および Process の実行時に使用するソフトウェア オプション。

- ctsu_process_option_t ctsu_update_option

  パラメータ更新プロセスの実行時に使用するソフトウェア オプション。

- ctsu_close_option_t ctsu_close_option

  タッチ動作終了時に使用するソフトウェア オプション。

- ctsu_action_t ctsu_process_state

  CTSU 処理状態のマシン動作を観察するための変数。

- void(* p_callback)(ctsu_callback_args_t *p_args)

  従属パラメータの更新完了時に使用する関数へのコールバック。

- void * p_context

  update_complete 通知に渡すデータへのポインタ。

7.7.7.10 ctsu_api_t

ctsu_api_t

詳細説明

CTSU HAL ドライバ API 構造体。HAL レイヤーに実装された関数は、この API に従います。

7.7.7.11 open

ssp_err_t(* `ctsu_api_t`::open)(`ctsu_ctrl_t` *p_ctrl, `ctsu_cfg_t` *p_cfg)

詳細説明

CTSU を初期化します。電源とクロックを有効化し、レジスタ構成を設定します。また電力消費を低減できます。

- R_CTSU_Open

表 346: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : ctsu_ctrl_t

制御構造体

定義 :

定義 : ctsu_cfg_t *p_cfg

CTSU ドライバの構成構造体を定義する構造体

- ctsu_cfg_t::transfer_instance_t

  リード結果への転送インスタンスへのポインタ。

- ctsu_cfg_t::transfer_instance_t

  write cfg への転送インスタンスへのポインタ。

- ctsu_cfg_t::ctsu_hw_cfg_t

  CTSU 構成へのポインタ。

- ctsu_cfg_t::ctsu_functions_t

  カスタム データ関数のプレース ホルダーへのポインタ。

- ctsu_cfg_t::p_callback

  スキャン完了時に使用する関数へのコールバック。

- ctsu_cfg_t::p_context

  update_complete 通知に渡すデータへのポインタ。

- ctsu_cfg_t::ctsu_process_option_t

  Open および Process の実行時に使用するソフトウェア オプション。

  列挙値 :

  – CTSU_PROCESS_OPTION_DEFAULT_SETTING

  – CTSU_PROCESS_OPTION_AUTO_SCAN

  – CTSU_PROCESS_OPTION_ENABLE_DRIFT_COMP

  – CTSU_PROCESS_OPTION_ENABLE_AUTO_CALIB

– CTSU_PROCESS_OPTION_NONE

- ctsu_cfg_t::ctsu_close_option_t

  タッチ動作終了時に使用するソフトウェア オプション。

  列挙値 :

  – CTSU_CLOSE_OPTION_SUSPEND

  – CTSU_CLOSE_OPTION_SAVE_CONFIG

  – CTSU_CLOSE_OPTION_RESET_SFRS

  – CTSU_CLOSE_OPTION_POWER_DOWN

7.7.7.12 close

ssp_err_t(* ctsu_api_t::close)(ctsu_ctrl_t *p_ctrl, ctsu_close_option_t opts)

詳細説明

進行中のスキャンを停止し、割り込みを無効化し、周辺機能への電源を遮断し、オプションに従って構成を保存して、指定された CTSU を閉じます。また電力消費を低減できます。

- R_CTSU_Close

表 347: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
opts	複数のビットを書き換えることもできます。	クローズ オプション

定義 : ctsu_ctrl_t

制御構造体

パラメータ opts

定義 : ctsu_close_option_topts

クローズ関数に渡すことができるオプション。これらのオプションは、論理和 （OR）を使用して組み合わせることができます。

7.7.7.13 scan

ssp_err_t(* ctsu_api_t::scan)(ctsu_ctrl_t *p_ctrl)

詳細説明

1 回の CTSU スキャンを開始します。また電力消費を低減できます。

- R_CTSU_Start_Scan

表 348: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : `ctsu_ctrl_t`

制御構造体

7.7.7.14 update

ssp_err_t(* `ctsu_api_t`::update)(`ctsu_ctrl_t` *p_ctrl)

詳細説明

スキャン結果に従って、タッチ判定やその他の派生データなどの CTSU 内部データを更新します。また電力消費を低減できます。

- R_CTSU_Update_Parameters

表 349: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ

定義 : `ctsu_ctrl_t`

制御構造体

7.7.7.15 read

ssp_err_t(* `ctsu_api_t`::read)(`ctsu_ctrl_t` *p_ctrl, void *p_dest, `ctsu_read_t` opts, const `ctsu_channel_pair_t` *channels, const uint16_t count)

詳細説明

選択したオプションに従って、未加工データ、バイナリ データ、その他の派生データなどの結果を CTSU から読み取ります。また電力消費を低減できます。

- R_CTSU_Read_Results

表 350:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_dest	out	読み取ったデータの書き込み先の場所へのポインタ
opts	複数のビットを書き換えることもできます。	リード オプション、リードの呼び出し 1 点につきオプション 1 点のみを使用、論理 OR 演算を行わない
channels	複数のビットを書き換えることもできます。	データを読み取るチャネル / チャネル ペアを指定します
count	複数のビットを書き換えることもできます。	データを読み取るチャネル / チャネル ペアの数を指定します

定義 : ctsu_ctrl_t

制御構造体

パラメータ p_dest

const

パラメータ opts

定義 : ctsu_read_topts

ユーザーが CTSU スキャン結果の照会に使用できる複数のオプション。これらのオプションは、論理和（OR）を使用して組み合わせることはできません。

パラメータ channels

定義 : ctsu_channel_pair_tctsu_channel_pair_t *channels

タッチ センサー要素に関する情報を格納する構造体

- ctsu_channel_pair_t::rx

  プライマリ チャネルを示します。

- ctsu_channel_pair_t::tx

  セカンダリ チャネルを示します （相互容量モードでのみ使用されます）

パラメータ カウント

uint16_t

7.7.7.16 versionGet

ssp_err_t(* ctsu_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

API バージョンを取得します。また電力消費を低減できます。

- R_CTSU_VersionGet()

  I：この関数は、API バージョンを取得します。

表 351: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン構造体へのポインタ

パラメータ p_version

7.7.7.17 ctsu_instance_t

ctsu_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- ctsu_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- ctsu_cfg_t * p_cfg

  イベント クラスのインスタンス範囲の始点。

- ctsu_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.8 DAC インタフェース

D/A コンバータのインタフェース。

7.8.1 概要

DAC インタフェースは、標準のデジタル / アナログ変換機能を提供します。DAC アプリケーションは、デジタル サンプル データをデバイスに書き込み、DAC 出力ピン上でアナログ出力を生成します。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

DAC インタフェースの説明 : DAC ドライバ

7.8.2 インタフェース API

dac_api_t

関数名	説明
.open	初期設定。
.close	D/A コンバータを閉じます。
.write	D/A コンバータにサンプル値を書き込みます。
.start	D/A コンバータが動作していない場合は開始します。
.stop	D/A コンバータが動作している場合は停止します。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.8.3 データ構造体

- dac_cfg_t
- dac_ctrl_t
- dac_instance_t

7.8.4 列挙

- dac_data_format_t

7.8.5 型定義

- dac_size_t

7.8.6 API データ

7.8.6.1 dac_data_format_t

dac_data_format_t

詳細説明

DAC オープン API の AD/DA データ フォーマット設定。

列挙値

名前	説明
DAC_DATA_FORMAT_FLUSH_RIGHT	データの LSB を右方向にフラッシュし、上位 4 ビットを未使用にします。
DAC_DATA_FORMAT_FLUSH_LEFT	データの MSB を左方向にフラッシュして、下位 4 ビットを未使用にします。

7.8.6.2 dac_size_t

typedef uint16_t dac_size_t

詳細説明

DAC 出力値を格納するデータ型。

7.8.7 API 構造

7.8.7.1 dac_cfg_t

dac_cfg_t

詳細説明

DAC Open API 設定パラメータ

変数

- uint8_t channel

  この DAC チャネルに関連付けられた ID。

- bool ad_da_synchronized

  AD/DA 同期。

- dac_data_format_t data_format

  データ形式。

- bool output_amplifier_enabled

  出力増幅器有効化。

- void const * p_extend

7.8.7.2 dac_ctrl_t

dac_ctrl_t

詳細説明

DAC チャネル制御ブロック

変数

- uint8_t channel

  この DAC チャネルに関連付けられた ID。

- uint8_t channel_started

  開始されたチャネル上の DAC 動作。

- uint8_t channel_opened

  開いている DAC チャネル。

- uint8_t reserved1

7.8.7.3 dac_api_t

dac_api_t

詳細説明

DAC ドライバ構造体。HAL レイヤーに実装された汎用 DAC 関数は、この API に従います。

7.8.7.4 open

ssp_err_t(* `dac_api_t`::open)(`dac_ctrl_t` *p_ctrl, `dac_cfg_t` const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_DAC_Open

表 352: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : dac_ctrl_t

DAC チャネル制御ブロック

定義 :

定義 : dac_cfg_t const *const p_cfg

DAC Open API 設定パラメータ

- dac_cfg_t::channel

  この DAC チャネルに関連付けられた ID。
- dac_cfg_t::ad_da_synchronized

  AD/DA 同期。
- dac_cfg_t::dac_data_format_t

  データ形式。

  列挙値 :
  - DAC_DATA_FORMAT_FLUSH_RIGHT
  - DAC_DATA_FORMAT_FLUSH_LEFT
- dac_cfg_t::output_amplifier_enabled

  出力増幅器有効化。
- dac_cfg_t::p_extend

7.8.7.5 close

ssp_err_t(* dac_api_t::close)(dac_ctrl_t *p_ctrl)

詳細説明

D/A コンバータを閉じます。また電力消費を低減できます。

- R_DAC_Close

表 353: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : `dac_ctrl_t`

DAC チャネル制御ブロック

7.8.7.6 write

ssp_err_t(* `dac_api_t`::write)(`dac_ctrl_t` *p_ctrl, `dac_size_t` value)

詳細説明

D/A コンバータにサンプル値を書き込みます。また電力消費を低減できます。

- R_DAC_Write

表 354: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。
value	複数のビットを書き換えることもできます。	D/A コンバータに書き込むサンプル値。

定義 : `dac_ctrl_t`

DAC チャネル制御ブロック

パラメータ value

定義 : `dac_size_t`value

DAC 出力値を格納するデータ型。

7.8.7.7 start

ssp_err_t(* dac_api_t::start)(dac_ctrl_t *p_ctrl)

詳細説明

D/A コンバータが動作していない場合は開始します。また電力消費を低減できます。

- R_DAC_Start

表 355: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : dac_ctrl_t

DAC チャネル制御ブロック

7.8.7.8 stop

ssp_err_t(* dac_api_t::stop)(dac_ctrl_t *p_ctrl)

詳細説明

D/A コンバータが動作している場合は停止します。また電力消費を低減できます。

- R_DAC_Stop

表 356: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : dac_ctrl_t

DAC チャネル制御ブロック

7.8.7.9 versionGet

ssp_err_t(* dac_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_DAC_VersionGet

表 357: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.8.7.10 dac_instance_t

dac_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- dac_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- dac_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- dac_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.9 ディスプレイインタフェース

LCD パネル ディスプレイのインタフェース。

7.9.1 概要

表示インタフェースは、標準的な表示機能を提供します。

- RGB インタフェースを備えた LCD パネル用の信号タイミング構成。
- ドット クロック ソース選択 （内部または外部）および周波数分周器。
- 背景画面における複数のグラフィックス レイヤーのブレンド。
- 色補正 （明るさ / 構成 / ガンマ補正)。
- 割り込みおよびコールバック関数。

以下によって実装されます。GLCDC

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

表示インタフェースの説明 : ディスプレイ ドライバ

7.9.2 インタフェース API

display_api_t

関数名	説明
.open	表示デバイスを開きます。
.close	表示デバイスを閉じます。
.start	表示の開始。
.stop	表示の停止。
.layerChange	ランタイムにレイヤー パラメータを変更します。
.correction	色補正。
.clut	表示デバイスに対して CLUT を設定します。

関数名	説明
.statusGet	表示デバイスのステータスを取得します。
.versionGet	バージョンを取得します。

7.9.3 データ構造体

- display_timing_t
- display_color_t
- display_coordinate_t
- display_brightness_t
- display_contrast_t
- display_correction_t
- gamma_correction_t
- display_gamma_correction_t
- display_clut_cfg_t
- display_input_cfg_t
- display_output_cfg_t
- display_layer_t
- display_callback_args_t
- display_cfg_t
- display_runtime_cfg_t
- display_clut_cfg_t
- display_ctrl_t
- display_status_t
- display_instance_t

7.9.4 列挙

- display_frame_layer_t
- display_state_t
- display_event_t

- display_in_format_t
- display_out_format_t
- display_endian_t
- display_color_order_t
- display_signal_polarity_t
- display_sync_edge_t
- display_fade_control_t
- display_fade_status_t

7.9.5 定義

- #define DISPLAY_API_VERSION_MAJOR

  初期値 :(1)
- #define DISPLAY_API_VERSION_MINOR

  初期値 :(1)
- #define DISPLAY_GAMMA_CURVE_ELEMENT_NUM

  初期値 :(16)

7.9.6 API データ

7.9.6.1 display_frame_layer_t

display_frame_layer_t

詳細説明

表示フレーム番号

列挙値

名前	説明
DISPLAY_FRAME_LAYER_1	フレーム レイヤー 1。
DISPLAY_FRAME_LAYER_2	フレーム レイヤー 2。

7.9.6.2 display_state_t

display_state_t

詳細説明

表示インタフェース操作の状態

列挙値

名前	説明
DISPLAY_STATE_CLOSED	表示が閉じています。
DISPLAY_STATE_OPENED	表示が開いています。
DISPLAY_STATE_DISPLAYING	表示中です。

7.9.6.3 display_event_t

display_event_t

詳細説明

表示イベント コード

列挙値

名前	説明
DISPLAY_EVENT_GR1_UNDERFLOW	グラフィックス フレーム 1 のアンダーフローが発生しました。
DISPLAY_EVENT_GR2_UNDERFLOW	グラフィックス フレーム 2 のアンダーフローが発生しました。
DISPLAY_EVENT_LINE_DETECTION	指定したラインが処理されました。

7.9.6.4 display_in_format_t

display_in_format_t

詳細説明

入力フォーマット設定

列挙値

名前	説明
DISPLAY_IN_FORMAT_32BITS_ARGB8888	ARGB8888、32 ビット。
DISPLAY_IN_FORMAT_32BITS_RGB888	RGB888、32 ビット。
DISPLAY_IN_FORMAT_16BITS_RGB565	RGB565、16 ビット。
DISPLAY_IN_FORMAT_16BITS_ARGB1555	ARGB1555、16 ビット。
DISPLAY_IN_FORMAT_16BITS_ARGB4444	ARGB4444、16 ビット。
DISPLAY_IN_FORMAT_CLUT8	CLUT8。
DISPLAY_IN_FORMAT_CLUT4	CLUT4。
DISPLAY_IN_FORMAT_CLUT1	CLUT1。

7.9.6.5 display_out_format_t

display_out_format_t

詳細説明

出力フォーマット設定

列挙値

名前	説明
DISPLAY_OUT_FORMAT_24BITS_RGB888	RGB888、24 ビット。
DISPLAY_OUT_FORMAT_18BITS_RGB666	RGB666、18 ビット。
DISPLAY_OUT_FORMAT_16BITS_RGB565	RGB565、16 ビット。
DISPLAY_OUT_FORMAT_8BITS_SERIAL	SERIAL、8 ビット。

7.9.6.6 display_endian_t

display_endian_t

詳細説明

データ エンディアン選択

列挙値

名前	説明
DISPLAY_ENDIAN_LITTLE	リトルエンディアン。
DISPLAY_ENDIAN_BIG	ビッグエンディアン。

7.9.6.7 display_color_order_t

display_color_order_t

詳細説明

RGB の色順序選択

列挙値

名前	説明
DISPLAY_COLOR_ORDER_RGB	色順序は RGB。
DISPLAY_COLOR_ORDER_BGR	色順序は BGR。

7.9.6.8 display_signal_polarity_t

display_signal_polarity_t

詳細説明

信号極性の選択

列挙値

名前	説明
DISPLAY_SIGNAL_POLARITY_LOACTIVE	Low アクティブ信号。
DISPLAY_SIGNAL_POLARITY_HIACTIVE	High アクティブ信号。

7.9.6.9 display_sync_edge_t

display_sync_edge_t

詳細説明

信号同期エッジ選択

列挙値

名前	説明
DISPLAY_SIGNAL_SYNC_EDGE_RISING	信号は立ち上がりエッジに同期します。
DISPLAY_SIGNAL_SYNC_EDGE_FALLING	信号は立ち下がりエッジに同期します。

7.9.6.10 display_fade_control_t

display_fade_control_t

詳細説明

フェード制御

列挙値

名前	説明
DISPLAY_FADE_CONTROL_NONE	フェード制御を適用しません。
DISPLAY_FADE_CONTROL_FADEIN	フェードイン制御を適用します。
DISPLAY_FADE_CONTROL_FADEOUT	フェードアウト制御を適用します。

7.9.6.11 display_fade_status_t

display_fade_status_t

詳細説明

フェードステータス

列挙値

名前	説明
DISPLAY_FADE_STATUS_NOT_UNDERWAY	フェードイン / フェードアウトが実行中でありません。
DISPLAY_FADE_STATUS_FADING_UNDERWAY	フェードイン / フェードアウトが実行中です。

名前	説明
DISPLAY_FADE_STATUS_UNCERTAIN	ハードウェアの実行直前であるため、フェードイン / フェードアウトのステータスが不明です。

7.9.7 API 構造

7.9.7.1 display_timing_t

display_timing_t

詳細説明

表示シグナル時間設定

変数

- uint16_t total_cyc

  1 ラインの合計サイクルまたは 1 フレームの合計ライン。

- uint16_t display_cyc

  アクティブ ビデオ サイクルまたはライン。

- uint16_t back_porch

  バック ポーチ サイクルまたはライン。

- uint16_t sync_width

  同期信号アサーション幅。

- display_signal_polarity_t sync_polarity

  同期信号極性。

7.9.7.2 display_color_t

display_color_t

詳細説明

RGB 色設定値

変数

- uint32_t argb
- uint8_t b

  青

- uint8_t g

  緑

- uint8_t r

  赤

- uint8_t a

  a

- struct{} byte

  このメンバーの定義については、ソース コードを参照してください。

- union{} union{}

  このメンバーの定義については、ソース コードを参照してください。

7.9.7.3 display_coordinate_t

display_coordinate_t

詳細説明

コントラスト （ゲイン）補正設定値

変数

- int16_t x

  X 座標、単一の値を設定できます。

- int16_t y

  Y 座標、単一の値を設定できます。

7.9.7.4 display_brightness_t

display_brightness_t

詳細説明

明るさ （DC）補正設定値

変数

- bool enable

  明るさ補正オン / オフ。

- uint16_t r

  R チャネルの明るさ （DC）調整。

- uint16_t g

  G チャネルの明るさ （DC）調整。

- uint16_t b

  B チャネルの明るさ （DC）調整。

7.9.7.5 display_contrast_t

display_contrast_t

詳細説明

コントラスト （ゲイン）補正設定値

変数

- bool enable

  コントラスト補正オン / オフ。

- uint8_t r

  R チャネルのコントラスト （ゲイン）調整。

- uint8_t g

  G チャネルのコントラスト （ゲイン）調整。

- uint8_t b

  B チャネルのコントラスト （ゲイン）調整。

7.9.7.6 display_correction_t

display_correction_t

詳細説明

色補正設定値

変数

- display_brightness_t brightness

  明るさ。

- display_contrast_t contrast

  コントラスト。

7.9.7.7 gamma_correction_t

gamma_correction_t

詳細説明

各色のガンマ補正設定値

変数

- bool enable

  ガンマ補正オン / オフ。

- uint16_t gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]

  ゲイン調整。

- uint16_t threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]

  開始しきい値。

7.9.7.8 display_gamma_correction_t

display_gamma_correction_t

詳細説明

ガンマ補正設定値

変数

- gamma_correction_t r

  R チャネルのガンマ補正。

- gamma_correction_t g

  G チャネルのガンマ補正。

- gamma_correction_t b

  B チャネルのガンマ補正。

7.9.7.9 display_clut_cfg_t

display_clut_t

詳細説明

CLUT 設定値

変数

- uint32_t color_num

  CLUT 内の色数。

- const uint32_t * p_clut

  CLUT データを格納する領域のアドレス （ARGB8888 形式）

7.9.7.10 display_input_cfg_t

display_input_cfg_t

詳細説明

グラフィクス プレーン入力設定構造体

変数

- uint32_t * p_base

  フレーム バッファへのベース アドレス。

- uint16_t hsize

  1 ラインの水平ピクセル サイズ。

- uint16_t vsize

  1 フレームの垂直ピクセル サイズ。

- uint32_t hstride

  1 ラインのメモリ ストライド （バイト単位）。

- display_in_format_t format

  入力形式設定。

- bool line_descending_enable

  ライン降順有効化。

- bool lines_repeat_enable

  ライン繰り返し有効化。

- uint16_t lines_repeat_times

  期待されるライン繰り返し回数。

7.9.7.11 display_output_cfg_t

display_output_cfg_t

詳細説明

表示出力設定構造体

変数

- display_timing_t htiming

  水平表示のサイクル設定値。
- display_timing_t vtiming

  垂直表示のサイクル設定値。
- display_out_format_t format

  出力形式設定。
- display_endian_t endian

  出力データのビット順序。
- display_color_order_t color_order

  カラー オーダー （ピクセル単位）。
- display_signal_polarity_t data_enable_polarity

  データ有効信号極性。
- display_sync_edge_t sync_edge

  信号同期エッジ選択。
- display_color_t bg_color

  バックグラウンド カラー。
- display_brightness_t brightness

  明るさの設定値。
- display_contrast_t contrast

  コントラスト設定値。
- display_gamma_correction_t * p_gamma_correction

  ガンマ補正設定値へのポインタ。
- bool dithering_on

  ディザリングのオン / オフ。

7.9.7.12 display_layer_t

display_layer_t

詳細説明

グラフィクス レイヤーのブレンド設定パラメータ構造体

変数

- display_coordinate_t coordinate

  ブレンド位置 （画像の開始ポイント）

- display_color_t bg_color

  領域外の色。

- display_fade_control_t fade_control

  レイヤー フェードイン / アウト制御オン / オフ。

- uint8_t fade_speed

  レイヤー フェードイン / アウト フレーム レート。

7.9.7.13 display_callback_args_t

display_callback_args_t

詳細説明

表示コールバック パラメータ定義

変数

- display_event_t event

  イベント コード。

- void const * p_context

  コールバック時にユーザーに提供されるコンテキスト。

7.9.7.14 display_cfg_t

display_cfg_t

詳細説明

表示メイン設定構造体

変数

- display_input_cfg_t input[DISPLAY_FRAME_LAYER_2+1]

  グラフィックス入力フレーム設定値。

  表示デバイス用の汎用設定

- display_output_cfg_t output

  グラフィックス出力フレーム設定値。

- display_layer_t layer[DISPLAY_FRAME_LAYER_2+1]
  グラフィクス レイヤーのブレンド設定値。
- void(* p_callback)(display_callback_args_t *p_args)
  コールバック関数へのポインタ。
  表示イベント処理用の設定
- void const * p_context
  コールバック関数に渡されるユーザー定義のコンテキスト。
- void const * p_extend
  表示ハードウェアに依存する設定。
  表示ペリフェラルに固有の設定へのポインタ

7.9.7.15 display_runtime_cfg_t

display_runtime_cfg_t

詳細説明

表示メイン設定構造体

変数

- display_input_cfg_t input
  グラフィックス入力フレーム設定値。
  表示デバイス用の汎用設定
- display_layer_t layer
  グラフィクス レイヤーのアルファ ブレンド設定値。

7.9.7.16 display_clut_cfg_t

display_clut_cfg_t

詳細説明

表示 CLUT 設定構造体

変数

- uint32_t * p_base
  CLUT ソース データへのポインタ。
- uint16_t start
  更新される CLUT エントリの開始。

- uint16_t size

  更新される CLUT エントリのサイズ。

7.9.7.17 display_ctrl_t

display_ctrl_t

詳細説明

表示制御ブロック

変数

- display_state_t state

  GLCD モジュールのステータス。

- void(* p_callback)(display_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void const * p_context

  ハイレベルのデバイス コンテキストへのポインタ。

7.9.7.18 display_status_t

display_status_t

詳細説明

表示ステータス

変数

- display_state_t state

  GLCD モジュールのステータス。

- display_fade_status_t fade_status[DISPLAY_FRAME_LAYER_2+1]

  フェードイン / フェードアウトのステータス。

7.9.7.19 display_api_t

display_api_t

詳細説明

表示ペリフェラルの共有インタフェース定義

7.9.7.20 open

ssp_err_t(* display_api_t::open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)

詳細説明

表示デバイスを開きます。また電力消費を低減できます。

- R_GLCD_Open

表 358: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	表示インタフェース制御ブロックへのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	表示設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : display_ctrl_t

表示制御ブロック

定義 :

定義 : display_cfg_t const *const p_cfg

表示メイン設定構造体

- display_cfg_t::display_input_cfg_t

  グラフィックス入力フレーム設定値。

  表示デバイス用の汎用設定
- display_cfg_t::display_output_cfg_t

  グラフィックス出力フレーム設定値。
- display_cfg_t::display_layer_t

  グラフィクス レイヤーのブレンド設定値。
- display_cfg_t::p_callback

  コールバック関数へのポインタ。

  表示イベント処理用の設定
- display_cfg_t::p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。

- display_cfg_t::p_extend

  表示ハードウェアに依存する設定。

  表示ペリフェラルに固有の設定へのポインタ

7.9.7.21 close

ssp_err_t(* display_api_t::close)(display_ctrl_t *const p_ctrl)

詳細説明

表示デバイスを閉じます。また電力消費を低減できます。

- R_GLCD_Close

表 359: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : display_ctrl_t

表示制御ブロック

7.9.7.22 start

ssp_err_t(* display_api_t::start)(display_ctrl_t *const p_ctrl)

詳細説明

表示の開始。また電力消費を低減できます。

- R_GLCD_Start

表 360: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : display_ctrl_t

表示制御ブロック

7.9.7.23 stop

ssp_err_t(* display_api_t::stop)(display_ctrl_t *const p_ctrl)

詳細説明

表示の停止。また電力消費を低減できます。

- R_GLCD_Stop

表 361: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : display_ctrl_t

表示制御ブロック

7.9.7.24 layerChange

ssp_err_t(* display_api_t::layerChange)(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)

詳細説明

ランタイムにレイヤー パラメータを変更します。また電力消費を低減できます。

- R_GLCD_LayerChange

表 362: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
p_cfg	複数のビットを書き換えることもできます。	ランタイム レイヤー設定構造体へのポインタ。
frame	複数のビットを書き換えることもできます。	グラフィック フレーム数。

定義 : display_ctrl_t

表示制御ブロック

定義 :

定義 : `display_runtime_cfg_t` const *const p_cfg

表示メイン設定構造体

- `display_runtime_cfg_t`::`display_input_cfg_t`

  グラフィックス入力フレーム設定値。

  表示デバイス用の汎用設定

- `display_runtime_cfg_t`::`display_layer_t`

  グラフィクス レイヤーのアルファ ブレンド設定値。

パラメータ frame

7.9.7.25 correction

ssp_err_t(* `display_api_t`::correction)(`display_ctrl_t` const *const p_ctrl, `display_correction_t` const *const p_param)

詳細説明

色補正。また電力消費を低減できます。

- R_GLCD_ColorCorrection

表 363: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
param	複数のビットを書き換えることもできます。	色補正設定構造体へのポインタ。

定義 : `display_ctrl_t`

表示制御ブロック

パラメータ param

7.9.7.26 clut

ssp_err_t(* `display_api_t`::clut)(`display_ctrl_t` const *const p_ctrl, `display_clut_cfg_t` const *const p_clut_cfg, `display_frame_layer_t` frame)

詳細説明

表示デバイスに対して CLUT を設定します。また電力消費を低減できます。

- R_GLCD_ClutUpdate

表 364: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
p_clut_cfg	複数のビットを書き換えることもできます。	CLUT 設定構造体へのポインタ。
frame	複数のビットを書き換えることもできます。	CLUT に対応するフレーム バッファの数。

定義 : `display_ctrl_t`

表示制御ブロック

パラメータ p_clut_cfg

定義 : `display_clut_cfg_t` const *const p_clut_cfg

表示 CLUT 設定構造体

- `display_clut_cfg_t`::p_base

  CLUT ソース データへのポインタ。

- `display_clut_cfg_t`::start

  更新される CLUT エントリの開始。

- `display_clut_cfg_t`::size

  更新される CLUT エントリのサイズ。

パラメータ frame

7.9.7.27 statusGet

ssp_err_t(* `display_api_t`::statusGet)(`display_ctrl_t` const *const p_ctrl, `display_status_t` *const p_status)

詳細説明

表示デバイスのステータスを取得します。また電力消費を低減できます。

- R_GLCD_StatusGet

表 365: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
status	複数のビットを書き換えることもできます。	表示インタフェース ステータス構造体へのポインタ。

定義 : display_ctrl_t

表示制御ブロック

パラメータ status

7.9.7.28 versionGet

ssp_err_t(* display_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

バージョンを取得します。また電力消費を低減できます。

- R_GLCD_VersionGet

表 366: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン情報を格納するメモリへのポインタ。

パラメータ p_version

7.9.7.29 display_instance_t

display_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- display_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- display_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- display_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.10 DOC インタフェース

データ演算回路のインタフェース。

データ演算回路 （DOC）インタフェースの DOC 実装に対し、API とデータ構造体を定義します。

7.10.1 概要

このモジュールは、データ演算回路 （DOC）を使用して DOC_API を実装します。

以下によって実装されます。DOC

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

DOC インタフェースの説明 : データ操作回路ドライバ

7.10.2 インタフェース API

doc_api_t

関数名	説明
.open	初期設定。
.close	ドライバを再設定できるようになります。電力消費を低減できます。
.statusGet	DOC のステータスを取得し、指定されたポインタ p_status に格納します。
.statusClear	DOPCF ステータス フラグをクリアします。
.write	DODIR レジスタと DODSR レジスタに書き込みます。
.inputRegisterWrite	DODIR レジスタに書き込みます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.10.3 データ構造体

- doc_callback_args_t
- doc_ctrl_t
- doc_data_t
- doc_cfg_t
- doc_instance_t

7.10.4 列挙

- doc_event_t
- doc_status_t

7.10.5 型定義

- doc_size_t

7.10.6 定義

- #define DOC_API_VERSION_MAJOR

  初期値 :(1)

  定義、共通サービス、およびエラー コードを登録します。
- #define DOC_API_VERSION_MINOR

  初期値 :(1)

7.10.7 API データ

7.10.7.1 doc_event_t

doc_event_t

詳細説明

コールバック関数をトリガできるイベント。

列挙値

名前	説明
DOC_EVENT_COMPARISON_MISMATCH	データの比較の結果、不一致がありました。
DOC_EVENT_ADDITION	データの加算の結果、値が H'FFFF より大きくなりました。
DOC_EVENT_SUBTRACTION	データの減算の結果、値が H'0000 より小さくなりました。
DOC_EVENT_COMPARISON_MATCH	データの比較の結果、一致がありました。

7.10.7.2 doc_status_t

doc_status_t

詳細説明

データ比較演算のステータス。

列挙値

名前	説明
DOC_STATUS_CONDITION_FALSE	データ比較条件が満たされず（一致または不一致）、加算結果が > H'FFFF でなく、減算結果が < H'0000 でありません。
DOC_STATUS_CONDITION_TRUE	データ比較条件が満たされ（一致または不一致）、加算結果が > H'FFFF で、減算結果が < H'0000 です。

7.10.7.3 doc_size_t

typedef uint16_t doc_size_t

詳細説明

データ オペレーション回路 （DOC）によってサポートされる比較データのサイズ

7.10.8 API 構造

7.10.8.1 doc_callback_args_t

doc_callback_args_t

詳細説明

コールバック関数のパラメータ データ。

変数

- doc_event_t event

  このイベントは、コールバックの原因を特定するために使用されます。

- void const * p_context

  ユーザー データのプレースホルダー。doc_cfg_t 内の doc_api_t::open 関数で設定されます。

7.10.8.2 doc_ctrl_t

doc_ctrl_t

詳細説明

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

変数

- uint32_t open

  ドライバによって使用され、制御構造体が有効かどうかを確認します。

- void(* p_callback)(doc_callback_args_t *p_args)

  DOC ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL を使用します。

- doc_event_t event

  イベント DOC の設定対象です。ISR コールバックに渡されます。

- void const * p_context

  ユーザー データのプレースホルダー。doc_callback_args_t 内のユーザー コールバックに渡されます。

7.10.8.3 doc_data_t

doc_data_t

詳細説明

比較 / 加算 / 減産のために DOC レジスタに書き込まれるデータ。

変数

- doc_size_t dodir

  DOC DODIR に書き込まれる値。

- doc_size_t dodsr

  DOC DODSR に書き込まれる値。

7.10.8.4 doc_cfg_t

doc_cfg_t

詳細説明

オープン関数によって使用されるユーザー設定構造体。

変数

- doc_event_t event

  doc_event_t から列挙値を選択します。

- void(* p_callback)(doc_callback_args_t *p_args)

  DOC ISR の発生時に提供されるコールバック。

- void const * p_context

  ユーザー データのプレースホルダー。doc_callback_args_t 内のユーザー コールバックに渡されます。

7.10.8.5 doc_api_t

doc_api_t

詳細説明

データ オペレーション回路 （DOC）API 構造体。HAL レイヤーに実装された DOC 関数は、この API に従います。

7.10.8.6 open

ssp_err_t(* doc_api_t::open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_DOC_Open

  I：この関数を呼び出す前に、ペリフェラルのクロックを設定する必要があります。

表 367: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : doc_ctrl_t

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

定義 :

定義 : doc_cfg_t const *const p_cfg

オープン関数によって使用されるユーザー設定構造体。

- doc_cfg_t::doc_event_t

  doc_event_t から列挙値を選択します。

  列挙値 :

  – DOC_EVENT_COMPARISON_MISMATCH

  – DOC_EVENT_ADDITION

  – DOC_EVENT_SUBTRACTION

  – DOC_EVENT_COMPARISON_MATCH

- doc_cfg_t::p_callback

  DOC ISR の発生時に提供されるコールバック。

- doc_cfg_t::p_context

  ユーザー データのプレースホルダー。doc_callback_args_t 内のユーザー コールバックに渡されます。

7.10.8.7 close

ssp_err_t(* doc_api_t::close)(doc_ctrl_t *const p_ctrl)

詳細説明

ドライバを再設定できるようになります。電力消費を低減できます。また電力消費を低減できます。

- R_DOC_Close

表 368: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。

定義 : doc_ctrl_t

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

7.10.8.8 statusGet

ssp_err_t(* doc_api_t::statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)

詳細説明

DOC のステータスを取得し、指定されたポインタ p_status に格納します。また電力消費を低減できます。

- R_DOC_StatusGet

I : この関数を使用する前に、open を呼び出して DOC を構成してください。

表 369: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
p_status	out	比較 / 加算 / 減算演算のステータスを示します。結果は doc_status_t のいずれかです。

定義 : doc_ctrl_t

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

パラメータ p_status

定義 : doc_status_t*p_status

データ比較演算のステータス。

7.10.8.9 statusClear

ssp_err_t(* doc_api_t::statusClear)(doc_ctrl_t *const p_ctrl)

詳細説明

DOPCF ステータス フラグをクリアします。また電力消費を低減できます。

- R_DOC_StatusClear

I：この関数を使用する前に、open を呼び出して DOC を構成してください。

表 370:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。

定義 : doc_ctrl_t

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

7.10.8.10 write

ssp_err_t(* doc_api_t::write)(doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)

詳細説明

DODIR レジスタと DODSR レジスタに書き込みます。また電力消費を低減できます。

- R_DOC_Write

I：この関数を使用する前に、open を呼び出して DOC を構成してください。

表 371:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。

表 371: パラメータ (続き)

名前	方向	説明
p_data	複数のビットを書き換えることもできます。	DOC DODIR レジスタおよび DODSR レジスタに書き込まれるデータへのポインタ。

定義 : doc_ctrl_t

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

パラメータ p_data

定義 : doc_data_t*const p_data

比較 / 加算 / 減産のために DOC レジスタに書き込まれるデータ。

- doc_data_t::dodir

  DOC DODIR に書き込まれる値。

- doc_data_t::dodsr

  DOC DODSR に書き込まれる値。

7.10.8.11 inputRegisterWrite

ssp_err_t(* doc_api_t::inputRegisterWrite)(doc_ctrl_t *const p_ctrl, doc_size_t data)

詳細説明

DODIR レジスタに書き込みます。また電力消費を低減できます。

- R_DOC_InputRegisterWrite

  I : この関数を使用する前に、open を呼び出して DOC を構成してください。

表 372: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
data	複数のビットを書き換えることもできます。	DOC DODIR レジスタに書き込まれるデータ。

定義 : doc_ctrl_t

DOC 制御ブロック。初期化しないでください。初期化は、open 関数の呼び出し時に実行されます。

パラメータ data

定義 : doc_size_tdata

データ オペレーション回路 （DOC）によってサポートされる比較データのサイズ

7.10.8.12 versionGet

ssp_err_t(* doc_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_DOC_VersionGet

表 373: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.10.8.13 doc_instance_t

doc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- doc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- doc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- doc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.11 ELC インタフェース

イベント リンク コントローラのインタフェース。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

イベント リンク コントローラ インタフェースの説明 : ELC ドライバ

7.11.1 インタフェース API

elc_api_t

関数名	説明
.init	イベント リンク コントローラ内のすべてのリンクを初期化します。
.softwareEventGenerate	イベント リンク コントローラでソフトウェア イベントを生成します。
.linkSet	1 つのイベント リンクを作成します。
.linkBreak	イベント リンクを切断します。
.enable	イベント リンク コントローラの動作を有効にします。
.disable	イベント リンク コントローラの動作を無効にします。
.versionGet	コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

7.11.2 データ構造体

- elc_link_t
- elc_cfg_t
- elc_instance_t

7.11.3 列挙

- elc_software_event_t

7.11.4 定義

- #define ELC_API_VERSION_MAJOR

  初期値 :(1)
- #define ELC_API_VERSION_MINOR

  初期値 :(1)

7.11.5 API データ

7.11.5.1 elc_software_event_t

elc_software_event_t

詳細説明

ソフトウェア イベント番号

列挙値

名前	説明
ELC_SOFTWARE_EVENT_0	ソフトウェア イベント 0。
ELC_SOFTWARE_EVENT_1	ソフトウェア イベント 1。

7.11.6 API 構造

7.11.6.1 elc_link_t

elc_link_t

詳細説明

個々のイベント リンク。実際の周辺機器の定義は、MCU 固有の （つまり /mcu/S124/bsp_elc.h) bsp_elc.h ファイルに記載されています。

変数

- elc_peripheral_t peripheral
  信号を受信するペリフェラル。
- elc_event_t event
  ペリフェラルに送信される信号。

7.11.6.2 elc_cfg_t

elc_cfg_t

詳細説明

イベント リンク コントローラのメイン設定構造体

変数

- bool autostart
  open() の実行中に操作を開始し、割り込みを有効にします。
- uint32_t link_count
  イベント リンク数。
- elc_link_t const * link_list
  イベント リンク。

7.11.6.3 elc_api_t

elc_api_t

詳細説明

ELC ドライバ構造体。HAL レイヤーに実装された汎用 ELC 関数は、この API に従います。

7.11.6.4 init

ssp_err_t(* `elc_api_t`::init)(`elc_cfg_t` const *const p_cfg)

詳細説明

イベント リンク コントローラ内のすべてのリンクを初期化します。また電力消費を低減できます。

- R_ELC_Init

表 374: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。

定義 :

定義 : elc_cfg_t const *const p_cfg

イベント リンク コントローラのメイン設定構造体

- elc_cfg_t::autostart

  open() の実行中に操作を開始し、割り込みを有効にします。

- elc_cfg_t::link_count

  イベント リンク数。

- elc_cfg_t::elc_link_t

  イベント リンク。

7.11.6.5 softwareEventGenerate

ssp_err_t(* elc_api_t::softwareEventGenerate)(elc_software_event_t event_num)

詳細説明

イベント リンク コントローラでソフトウェア イベントを生成します。また電力消費を低減できます。

- R_ELC_SoftwareEventGenerate

表 375: パラメータ

名前	方向	説明
eventNum	複数のビットを書き換えることもできます。	生成されるソフトウェア イベント番号。

パラメータ eventNum

7.11.6.6 linkSet

ssp_err_t(* elc_api_t::linkSet)(elc_peripheral_t peripheral, elc_event_t signal)

詳細説明

1 つのイベント リンクを作成します。また電力消費を低減できます。

- R_ELC_LinkSet

表 376: パラメータ

名前	方向	説明
peripheral	複数のビットを書き換えることもできます。	イベント信号を受信するペリフェラル ブロック。
signal	複数のビットを書き換えることもできます。	イベント信号。

パラメータ peripheral

パラメータ signal

7.11.6.7 linkBreak

ssp_err_t(* elc_api_t::linkBreak)(elc_peripheral_t peripheral)

詳細説明

イベント リンクを切断します。また電力消費を低減できます。

- R_ELC_LinkBreak

表 377: パラメータ

名前	方向	説明
peripheral	複数のビットを書き換えることもできます。	リンクを切断するペリフェラル。

パラメータ peripheral

7.11.6.8 enable

ssp_err_t(* elc_api_t::enable)(void)

詳細説明

イベント リンク コントローラの動作を有効にします。また電力消費を低減できます。

- R_ELC_Enable

7.11.6.9 disable

ssp_err_t(* elc_api_t::disable)(void)

詳細説明

イベント リンク コントローラの動作を無効にします。また電力消費を低減できます。

- R_ELC_Disable

7.11.6.10 versionGet

ssp_err_t(* elc_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。また電力消費を低減できます。

- R_ELC_VersionGet

表 378: パラメータ

名前	方向	説明
p_version	out	戻り値。

パラメータ p_version

7.11.6.11 elc_instance_t

elc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- elc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- elc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.12 外部 IRQ インタフェース

外部割り込み検出のインタフェース。

7.12.1 概要

外部 IRQ インタフェースは、ピン入力や静電容量方式タッチ ボタンからの入力などの外部入力をサポートします。入力トリガが検出されると、ユーザーが指定したコールバック関数が呼び出されます。

以下によって実装されます。ICU

関連するインタフェース :Key Matrix インタフェース

関連する SSP アーキテクチャのトピック :

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

外部 IRQ インタフェースの説明 : 外部 IRQ ドライバ

7.12.2 インタフェース API

external_irq_api_t

関数名	説明
.open	初期設定。
.enable	外部 IRQ の発生時にコールバックを有効にします。
.disable	外部 IRQ の発生時にコールバックを無効にします。
.triggerSet	トリガーを設定します。
.filterEnable	ノイズ フィルタを有効にします。
.filterDisable	ノイズ フィルタを無効にします。
.close	ドライバを再設定できるようになります。電力消費を低減できます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.12.3 データ構造体

- external_irq_callback_args_t
- external_irq_ctrl_t
- external_irq_cfg_t
- external_irq_instance_t

7.12.4 列挙

- external_irq_trigger_t
- external_irq_pclk_div_t

7.12.5 定義

- #define EXTERNAL_IRQ_API_VERSION_MAJOR

  初期値 :(1)

  外部 IRQ API バージョン番号（メジャー）
- #define EXTERNAL_IRQ_API_VERSION_MINOR

  初期値 :(1)

  外部 IRQ API バージョン番号（マイナー）

7.12.6 API データ

7.12.6.1 external_irq_trigger_t

external_irq_trigger_t

詳細説明

立ち上がりエッジ、立ち下がりエッジ、両エッジ、ローレベル。

列挙値

名前	説明
EXTERNAL_IRQ_TRIG_FALLING	立ち下がりエッジ トリガー。
EXTERNAL_IRQ_TRIG_RISING	立ち上がりエッジ トリガー。

名前	説明
EXTERNAL_IRQ_TRIG_BOTH_EDGE	両エッジ トリガ。
EXTERNAL_IRQ_TRIG_LEVEL_LOW	ローレベル トリガ。

7.12.6.2 external_irq_pclk_div_t

external_irq_pclk_div_t

詳細説明

外部 IRQ 入力ピンにおけるデジタル フィルタリングのサンプル クロック分周器の設定。

列挙値

名前	説明
EXTERNAL_IRQ_PCLK_DIV_BY_1	1 分周の PCLK を使用したフィルタ。
EXTERNAL_IRQ_PCLK_DIV_BY_8	8 分周の PCLK を使用したフィルタ。
EXTERNAL_IRQ_PCLK_DIV_BY_32	32 分周の PCLK を使用したフィルタ。
EXTERNAL_IRQ_PCLK_DIV_BY_64	64 分周の PCLK を使用したフィルタ。

7.12.7 API 構造

7.12.7.1 external_irq_callback_args_t

external_irq_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- void const * p_context

  ユーザー データのプレースホルダー。external_irq_cfg_t 内の external_irq_api_t::open 関数で設定されます。

- uint32_t channel

  割り込みを発生させた物理ハードウェア チャネル。

7.12.7.2 external_irq_ctrl_t

external_irq_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- uint32_t channel

  チャネル。

- void(* p_callback)(external_irq_callback_args_t *p_args)

  外部 IRQ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

- void const * p_context

  ユーザー データのプレースホルダー。external_irq_callback_args_t のユーザー コールバックに渡されます。

7.12.7.3 external_irq_cfg_t

external_irq_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- uint32_t channel

  使用されるハードウェア チャネル。

- external_irq_trigger_t trigger

  トリガー設定値。

- bool filter_enable

  デジタル フィルタ有効 / 無効設定。

- external_irq_pclk_div_t pclk_div

  デジタル フィルタ クロック除算値設定。

- bool autostart

  open() の実行中に操作を開始し、割り込みを有効にします。

- void(* p_callback)(external_irq_callback_args_t *p_args)

  外部入力トリガーの発生時に提供されるコールバック。

- void const * p_context

  ユーザー データのプレースホルダー。external_irq_callback_args_t のユーザー コールバックに渡されます。

- void const * p_extend

  外部 IRQ ハードウェアに依存する設定。

7.12.7.4 external_irq_api_t

external_irq_api_t

詳細説明

外部割り込みドライバ構造体。HAL レイヤーに実装された外部割り込み関数は、この API に従います。

7.12.7.5 open

ssp_err_t(* external_irq_api_t::open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_ICU_ExternalIrqOpen

表 379: パラメータ

名前	方向	説明
p_ctrl	out	制御ブロックへのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : external_irq_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : external_irq_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- external_irq_cfg_t::channel

  使用されるハードウェア チャネル。

- external_irq_cfg_t::external_irq_trigger_t

  トリガー設定値。

  列挙値 :

  – EXTERNAL_IRQ_TRIG_FALLING

  – EXTERNAL_IRQ_TRIG_RISING

  – EXTERNAL_IRQ_TRIG_BOTH_EDGE

  – EXTERNAL_IRQ_TRIG_LEVEL_LOW

- external_irq_cfg_t::filter_enable

  デジタル フィルタ有効 / 無効設定。

- external_irq_cfg_t::external_irq_pclk_div_t

  デジタル フィルタ クロック除算値設定。

  列挙値 :

  – EXTERNAL_IRQ_PCLK_DIV_BY_1

  – EXTERNAL_IRQ_PCLK_DIV_BY_8

  – EXTERNAL_IRQ_PCLK_DIV_BY_32

  – EXTERNAL_IRQ_PCLK_DIV_BY_64

- external_irq_cfg_t::autostart

  open() の実行中に操作を開始し、割り込みを有効にします。

- external_irq_cfg_t::p_callback

  外部入力トリガーの発生時に提供されるコールバック。

- external_irq_cfg_t::p_context

  ユーザー データのプレースホルダー。external_irq_callback_args_t のユーザー コールバックに渡されます。

- external_irq_cfg_t::p_extend

  外部 IRQ ハードウェアに依存する設定。

7.12.7.6 enable

ssp_err_t(* external_irq_api_t::enable)(external_irq_ctrl_t *const p_ctrl)

詳細説明

外部 IRQ の発生時にコールバックを有効にします。また電力消費を低減できます。

- R_ICU_ExternalIrqEnable

表 380: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで設定された制御ブロック。

定義 : `external_irq_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.12.7.7 disable

ssp_err_t(* `external_irq_api_t`::disable)(`external_irq_ctrl_t` *const p_ctrl)

詳細説明

外部 IRQ の発生時にコールバックを無効にします。また電力消費を低減できます。

- R_ICU_ExternalIrqDisable

表 381: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで設定された制御ブロック。

定義 : `external_irq_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.12.7.8 triggerSet

ssp_err_t(* `external_irq_api_t`::triggerSet)(`external_irq_ctrl_t` *const p_ctrl, `external_irq_trigger_t` const trigger)

詳細説明

トリガーを設定します。また電力消費を低減できます。

- R_ICU_ExternalIrqTriggerSet

表 382: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで設定された制御ブロック。
trigger	複数のビットを書き換えることもできます。	トリガー タイプ

定義 : `external_irq_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ **trigger**

7.12.7.9 filterEnable

ssp_err_t(* `external_irq_api_t`::filterEnable)(`external_irq_ctrl_t` *const p_ctrl)

詳細説明

ノイズ フィルタを有効にします。また電力消費を低減できます。

- R_ICU_ExternalIrqFilterEnable

表 383: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで設定された制御ブロック。

定義 : `external_irq_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.12.7.10 filterDisable

ssp_err_t(* `external_irq_api_t`::filterDisable)(`external_irq_ctrl_t` *const p_ctrl)

詳細説明

ノイズ フィルタを無効にします。また電力消費を低減できます。

- R_ICU_ExternalIrqFilterDisable

表 384: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで設定された制御ブロック。

定義 : external_irq_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.12.7.11 close

ssp_err_t(* external_irq_api_t::close)(external_irq_ctrl_t *const p_ctrl)

詳細説明

ドライバを再設定できるようになります。電力消費を低減できます。また電力消費を低減できます。

- R_ICU_ExternalIrqClose

表 385: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この外部割り込みの Open 呼び出しで設定された制御ブロック。

定義 : external_irq_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.12.7.12 versionGet

ssp_err_t(* external_irq_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_ICU_ExternalIrqVersionGet

表 386: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.12.7.13 external_irq_instance_t

external_irq_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- external_irq_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- external_irq_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- external_irq_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.13 フラッシュインタフェース

フラッシュ コントローラのインタフェース。

7.13.1 概要

フラッシュ インタフェースは、フラッシュ メモリの読み取り、書き込み、消去、およびブランク チェックに必要な機能を備えています。加えて、フラッシュ メモリのアクセス ウィンドウおよびスワップ領域を構成するための機能もあります。

以下によって実装されます。

- 高性能フラッシュ
- ローパワー フラッシュ

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

フラッシュ インタフェースの説明：フラッシュ ドライバ

7.13.2 インタフェース API

flash_api_t

関数名	説明
.open	FLASH デバイスを開きます。
.write	FLASH デバイスに書き込みます。
.read	FLASH デバイスから読み取ります。
.erase	FLASH デバイスを消去します。
.blankCheck	FLASH デバイスのブランク チェックを実行します。
.close	FLASH デバイスを閉じます。
.statusGet	FLASH デバイスのステータスを取得します。
.accessWindowSet	FLASH デバイスのアクセス ウィンドウを設定します。

関数名	説明
.accessWindowClear	FLASH デバイスの既存のコード フラッシュ アクセス ウィンドウをクリアします。
.reset	FLASH デバイス用の関数をリセットします。
.updateFlashClockFreq	フラッシュ クロック周波数 （FCLK）を更新し、タイムアウト値を再計算します
.startupAreaSelect	Default （ブロック 0）と Alternate （ブロック 1）のどちらのブロックをスタートアップ領域ブロックとして使用するかを選択します。swap_type | is_temporary | Operation FLASH_STARTUP_AREA_BLOCK0: false 次回のリセットにより、スタートアップ領域がブロック 0 になります。FLASH_STARTUP_AREA_BLOCK0 | false | 次回のリセット時に、スタートアップ領域がブロック 0 になります。ブロック 0。 FLASH_STARTUP_AREA_BLOCK1: false 次回のリセット時に、スタートアップ領域がブロック 1 になります。 FLASH_STARTUP_AREA_BLOCK1 | true | スタートアップ領域が、即座に、しかし一時的にブロック 1 に切り替わります。ブロック 1。 FLASH_STARTUP_AREA_BTFLG | true | スタートアップ領域が、即座に、しかし一時的に設定 BTFLG によって指定されたブロックに切り替わります。
.versionGet	フラッシュ ドライバのバージョンを取得します。

7.13.3 データ構造体

- flash_ctrl_t
- flash_callback_args_t
- flash_cfg_t
- flash_instance_t

7.13.4 列挙

- flash_result_t
- flash_startup_area_swap_t
- flash_event_t

7.13.5 定義

- #define FLASH_API_VERSION_MAJOR

  初期値 :(1)

  フラッシュ HAL API のバージョン番号 （メジャー バージョン）

- #define FLASH_API_VERSION_MINOR

  初期値 :(1)

  フラッシュ HAL API のバージョン番号 （マイナー バージョン）

7.13.6 API データ

7.13.6.1 flash_result_t

flash_result_t

詳細説明

特定の操作に対する結果タイプ

列挙値

名前	説明
FLASH_RESULT_BLANK	ブランク チェック関数のステータスを返します。
FLASH_RESULT_NOT_BLANK	ブランク チェック関数のステータスを返します。
FLASH_RESULT_BGO_ACTIVE	フラッシュが BGO モードに構成されます。結果はコールバックで返されます。

7.13.6.2 flash_startup_area_swap_t

flash_startup_area_swap_t

詳細説明

startupAreaSelect() によって要求されているスタートアップ領域スワップを指定するためのパラメータ

列挙値

名前	説明
FLASH_STARTUP_AREA_BLOCK1	スタートアップ領域は、ブロック 1 に設定されます。
FLASH_STARTUP_AREA_BLOCK0	スタートアップ領域は、ブロック 0 に設定されます。
FLASH_STARTUP_AREA_BTFLG	スタートアップ領域は、BTFLG の値に基づいて設定されます。

7.13.6.3 flash_event_t

flash_event_t

詳細説明

データ フラッシュ BGO モードで使用された ISR コールバックが返すイベント タイプ

列挙値

名前	説明
FLASH_EVENT_ERASE_COMPLETE	消去動作が正常に完了しました。
FLASH_EVENT_WRITE_COMPLETE	書き込み操作が正常に終了しました。
FLASH_EVENT_BLANK	ブランク チェック動作が正常に完了しました。指定された領域はブランクです。
FLASH_EVENT_NOT_BLANK	ブランク チェック動作が正常に完了しました。指定された領域はブランクではありません。
FLASH_EVENT_ERR_DF_ACCESS	データ フラッシュ動作が失敗しました。これは未消去セクションの書き込み時に発生することがあります。
FLASH_EVENT_ERR_CF_ACCESS	コード フラッシュ動作が失敗しました。これは未消去セクションの書き込み時に発生することがあります。
FLASH_EVENT_ERR_CMD_LOCKED	動作が失敗し、FCU はロック状態です （通常、不正なコマンドを使用した場合に生じます）
FLASH_EVENT_ERR_FAILURE	消去動作またはプログラム動作が失敗しました。

7.13.7 API 構造

7.13.7.1 flash_ctrl_t

flash_ctrl_t

詳細説明

パラメータ flash_cfg_t

変数

- bool opened

7.13.7.2 flash_callback_args_t

flash_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- flash_event_t event
  イベントを使用して、コールバックの原因 （フラッシュ レディまたはエラー）を特定できます。
- void const * p_context
  ユーザー データのプレースホルダー。::flash_cfg_t の flash_api_t::open 関数で設定されます。

7.13.7.3 flash_cfg_t

flash_cfg_t

詳細説明

FLASH 設定

変数

- bool data_flash_bgo
  データ フラッシュで BGO （バックグラウンド操作）で有効な場合は true。
- void(* p_callback)(flash_callback_args_t *p_args)
  フラッシュ割り込み ISR の発生時に提供されるコールバック。
- void const * p_extend
  FLASH ハードウェアに依存する設定。

- void const * p_context

  ユーザー データのプレースホルダー。flash_callback_args_t 内のユーザー コールバックに渡されます。

7.13.7.4 flash_api_t

flash_api_t

詳細説明

FLASH の共有インタフェース定義

7.13.7.5 open

ssp_err_t(* flash_api_t::open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)

詳細説明

FLASH デバイスを開きます。また電力消費を低減できます。

- R_FLASH_LP_Open
- R_FLASH_HP_Open

表 387: パラメータ

名前	方向	説明
p_ctrl	out	FLASH デバイス制御へのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
flash_cfg_t	複数のビットを書き換えることもできます。	FLASH 設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

定義 :

定義 : flash_cfg_t const *const p_cfg

FLASH 設定

- flash_cfg_t::data_flash_bgo

  データ フラッシュで BGO （バックグラウンド操作）で有効な場合は true。

- flash_cfg_t::p_callback

  フラッシュ割り込み ISR の発生時に提供されるコールバック。

- flash_cfg_t::p_extend

  FLASH ハードウェアに依存する設定。

- flash_cfg_t::p_context

  ユーザー データのプレースホルダー。flash_callback_args_t 内のユーザー コールバックに渡されます。

7.13.7.6 write

ssp_err_t(* flash_api_t::write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

詳細説明

FLASH デバイスに書き込みます。また電力消費を低減できます。

- R_FLASH_LP_Write
- R_FLASH_HP_Write

表 388: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス コンテキストの制御。
src_address	複数のビットを書き換えることもできます。	フラッシュに書き込むデータが含まれているバッファのアドレス。
flash_address	複数のビットを書き換えることもできます。	書き込むコード フラッシュまたはデータ フラッシュ。このアドレスは、プログラミング ライン境界上に位置する必要があります。
num_bytes	複数のビットを書き換えることもできます。	書き込むバイト数。この数値は、プログラミング サイズの倍数である必要があります。コード フラッシュの場合、これは FLASH_MIN_PGM_SIZE_CF です。データ フラッシュの場合は、FLASH_MIN_PGM_SIZE_DF です。

I a: プログラミング サイズの倍数でない数値を指定した場合、SF_FLASH_ERR_BYTES が返され、

データは書き込まれません。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

パラメータ src_address

uint32_t

パラメータ flash_address

uint32_t

パラメータ num_bytes

uint32_t

7.13.7.7 read

ssp_err_t(* flash_api_t::read)(flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)

詳細説明

FLASH デバイスから読み取ります。また電力消費を低減できます。

- R_FLASH_LP_Read
- R_FLASH_HP_Read

表 389: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス コンテキストの制御。
p_dest_address	複数のビットを書き換えることもできます。	フラッシュから読み取られたデータを保持するための、呼び出し側の宛先バッファへのポインタ。
flash_address	複数のビットを書き換えることもできます。	読み取りを開始するコード フラッシュまたはデータ フラッシュのアドレス。

表 389: パラメータ (続き)

名前	方向	説明
num_bytes	複数のビットを書き換えることもできます。	読み取るバイト数。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

パラメータ p_dest_address

uint8_t

パラメータ flash_address

uint32_t

パラメータ num_bytes

uint32_t

7.13.7.8 erase

ssp_err_t(* flash_api_t::erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

詳細説明

FLASH デバイスを消去します。また電力消費を低減できます。R_FLASH_LP_EraseR_FLASH_HP_Erase

表 390: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイスの制御。
address	複数のビットを書き換えることもできます。	このアドレスが含まれているブロックが、最初に消去されるブロックです。
num_blocks	複数のビットを書き換えることもできます。	block_erase_address で指定される開始ブロックに続いて消去されるブロック数を指定します。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

パラメータ **address**

uint32_t

パラメータ **num_blocks**

uint32_t

7.13.7.9 blankCheck

ssp_err_t(* flash_api_t::blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)

詳細説明

FLASH デバイスのブランク チェックを実行します。また電力消費を低減できます。

- R_FLASH_LP_BlankCheck
- R_FLASH_HP_BlankCheck

表 391:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス コンテキストの制御。
address	複数のビットを書き換えることもできます。	ブランク チェックを行うフラッシュ領域の開始アドレス。
num_bytes	複数のビットを書き換えることもできます。	チェックが必要なバイト数を指定します。詳細については、特定のハンドラーを参照してください。
p_blank_check_result	out	非 -BGO （ブロッキング）モードの場合、API によってブランク チェック操作の結果を入力されるポインタ。この場合、ブランク チェック操作がここで完了し、結果が返されます。データ フラッシュ BGO モードの場合、ブランク チェック操作は開始するだけで、結果は後に、提供されたコールバック ルーチンが呼び出されたときに取得されます。この場合、p_blank_check_result では FLASH_RESULT_BGO_ACTIVE が返されます。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

パラメータ address

uint32_t

パラメータ num_bytes

uint32_t

パラメータ p_blank_check_result

定義 : `flash_result_t`*const p_blank_check_result

特定の操作に対する結果タイプ

7.13.7.10 close

ssp_err_t(* `flash_api_t`::close)(`flash_ctrl_t` *const p_ctrl)

詳細説明

FLASH デバイスを閉じます。また電力消費を低減できます。

- R_FLASH_LP_Close
- R_FLASH_HP_Close

表 392: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。

定義 : `flash_ctrl_t`

パラメータ flash_cfg_t

7.13.7.11 statusGet

ssp_err_t(* `flash_api_t`::statusGet)(`flash_ctrl_t` *const p_ctrl)

詳細説明

FLASH デバイスのステータスを取得します。また電力消費を低減できます。

- R_FLASH_LP_StatusGet
- R_FLASH_HP_StatusGet

表 393: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

7.13.7.12 accessWindowSet

ssp_err_t(* flash_api_t::accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)

詳細説明

FLASH デバイスのアクセス ウィンドウを設定します。また電力消費を低減できます。

- R_FLASH_LP_AccessWindowSet
- R_FLASH_HP_AccessWindowSet

表 394: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。
start_addr	複数のビットを書き換えることもできます。	コード フラッシュ アクセス ウィンドウの開始ブロックを決定します。
end_addr	複数のビットを書き換えることもできます。	コード フラッシュ アクセス ウィンドウの終了ブロックを決定します。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

パラメータ **start_addr**

uint32_t

パラメータ **end_addr**

uint32_t

7.13.7.13 accessWindowClear

ssp_err_t(* flash_api_t::accessWindowClear)(flash_ctrl_t *const p_ctrl)

詳細説明

FLASH デバイスの既存のコード フラッシュ アクセス ウィンドウをクリアします。また電力消費を低減できます。

- R_FLASH_LP_AccessWindowClear
- R_FLASH_HP_AccessWindowClear

表 395: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。
start_addr	複数のビットを書き換えることもできます。	コード フラッシュ アクセス ウィンドウの開始ブロックを決定します。
end_addr	複数のビットを書き換えることもできます。	コード フラッシュ アクセス ウィンドウの終了ブロックを決定します。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

パラメータ start_addr

パラメータ end_addr

7.13.7.14 reset

ssp_err_t(* flash_api_t::reset)(flash_ctrl_t *const p_ctrl)

詳細説明

FLASH デバイス用の関数をリセットします。また電力消費を低減できます。

- R_FLASH_LP_Reset
- R_FLASH_HP_Reset

表 396: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

7.13.7.15 updateFlashClockFreq

ssp_err_t(* flash_api_t::updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)

詳細説明

フラッシュ クロック周波数 （FCLK）を更新し、タイムアウト値を再計算します。以下として実装されます。

- R_FLASH_LP_UpdateFlashClockFreq
- R_FLASH_HP_UpdateFlashClockFreq

表 397: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。

定義 : flash_ctrl_t

パラメータ flash_cfg_t

7.13.7.16 startupAreaSelect

ssp_err_t(* flash_api_t::startupAreaSelect)(flash_ctrl_t *const p_ctrl,
flash_startup_area_swap_t swap_type, bool is_temporary)

詳細説明

Default （ブロック 0）と Alternate （ブロック 1）のどちらのブロックをスタートアップ領域ブロックとして使用するかを選択します。また電力消費を低減できます。

- R_FLASH_LP_StartUpAreaSelect
- R_FLASH_HP_StartUpAreaSelect

表 398: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	FLASH デバイス制御へのポインタ。
swap_type	複数のビットを書き換えることもできます。	FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG.
is_temporary	複数のビットを書き換えることもできます。	true または false。以下を参照してください。

swap_type | is_temporary | Operation FLASH_STARTUP_AREA_BLOCK0: false 次回のリセットにより、スタートアップ領域がブロック 0 になります。

FLASH_STARTUP_AREA_BLOCK0 | false | 次回のリセット時に、スタートアップ領域がブロック 0 になります。ブロック 0。

FLASH_STARTUP_AREA_BLOCK1: false 次回のリセット時に、スタートアップ領域がブロック 1 になります。

FLASH_STARTUP_AREA_BLOCK1 | true | スタートアップ領域が、即座に、しかし一時的にブロック 1 に切り替わります。ブロック 1。

FLASH_STARTUP_AREA_BTFLG | true | スタートアップ領域が、即座に、しかし一時的に

設定 BTFLG によって指定されたブロックに切り替わります。

定義 : `flash_ctrl_t`

パラメータ flash_cfg_t

パラメータ swap_type

定義 : `flash_startup_area_swap_t`swap_type

startupAreaSelect() によって要求されているスタートアップ領域スワップを指定するためのパラメータ

パラメータ is_temporary

const

7.13.7.17 versionGet

ssp_err_t(* `flash_api_t`::versionGet)(ssp_version_t *p_version)

詳細説明

フラッシュ ドライバのバージョンを取得します。また電力消費を低減できます。

- R_FLASH_LP_VersionGet
- R_FLASH_HP_VersionGet

表 399: パラメータ

名前	方向	説明
p_version	out	バージョンを返します。

パラメータ p_version

7.13.7.18 flash_instance_t

flash_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- flash_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。
- flash_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。
- flash_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.14 FMI インタフェース

オンチップ ファクトリ情報のリード用インタフェース。

7.14.1 概要

FMI （ファクトリ MCU 情報）モジュールは、製品情報レコードを読み取るための関数を提供します。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

FMI インタフェースの説明 : FMI ドライバ

7.14.2 インタフェース API

fmi_api_t

関数名	説明
.productInfoGet	製品情報レコードのアドレスを取得して、呼び出し側のポインタに格納します。
.versionGet	コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

7.14.3 データ構造体

- fmi_header_t
- fmi_product_info_t
- fmi_instance_t

7.14.4 定義

- #define FMI_API_VERSION_MAJOR

  初期値 :(1)

  定義、共通サービス、およびエラー コードを登録します。

- #define FMI_API_VERSION_MINOR

  初期値 :(1)

7.14.5 API 構造

7.14.5.1 fmi_header_t

fmi_header_t

詳細説明

変数

- uint32_t contents
- uint32_t variant
- uint32_t count
- uint32_t minor
- uint32_t major

7.14.5.2 fmi_product_info_t

fmi_product_info_t

詳細説明

変数

- fmi_header_t header
- uint8_t unique_id[16]
- uint8_t product_name[16]
- uint8_t product_marking[16]
- uint32_t mask_revision
- uint32_t pin_count
- uint32_t pkg_type
- uint32_t temp_range
- uint32_t quality_code
- uint32_t reserved

- struct{} struct{}

  このメンバーの定義については、ソース コードを参照してください。
- uint32_t max_freq
- uint32_t reserved1
- struct{} struct{}

  このメンバーの定義については、ソース コードを参照してください。

7.14.5.3 fmi_api_t

fmi_api_t

詳細説明

fmi ドライバ構造体。HAL レイヤーに実装された汎用 fmi 関数は、この API に従います。

7.14.5.4 productInfoGet

(* `fmi_api_t`::productInfoGet)(`fmi_product_info_t` **pp_product_info)

詳細説明

製品情報レコードのアドレスを取得して、呼び出し側のポインタに格納します。また電力消費を低減できます。

- R_FMI_ProductInfoGet

7.14.5.5 versionGet

(* `fmi_api_t`::versionGet)(*const p_version)

詳細説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。また電力消費を低減できます。

- R_FMI_VersionGet

7.14.5.6 fmi_instance_t

fmi_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- fmi_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.15 I²C インタフェース

I²C 通信用のインタフェース。

7.15.1 概要

I²C マスター インタフェースは、I²C HAL ドライバ用に共通の API を提供します。I²C マスター インタフェースは次をサポートします :

- 割り込み駆動型の送信 / 受信処理
- イベントコードを返すことのできるコールバック関数サポート

以下によって実装されます。

- SCI 上の簡易 I2C
- IIC

関連する SSP アーキテクチャのトピック :

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

I²C インタフェースの説明 : I2C ドライバ

7.15.2 インタフェース API

i2c_api_master_t

関数名	説明
.open	I²C ドライバを開き、ハードウェアを初期化します。
.close	ドライバを閉じ、I²C デバイスを解放します。
.read	I²C デバイスで読み取り操作を実行します。
.write	I²C デバイスで書き込み操作を実行します。
.reset	ペリフェラルのリセットを実行します。
.versionGet	バージョン情報を取得して、それを指定されたバージョン構造体に保存します。

7.15.3 データ構造体

- i2c_callback_args_t
- i2c_cfg_t
- i2c_ctrl_t
- i2c_api_master_t
- i2c_master_instance_t

7.15.4 列挙

- i2c_rate_t
- i2c_addr_mode_t
- i2c_event_t

7.15.5 定義

- #define I2C_MASTER_API_VERSION_MAJOR

  初期値 :(1)
- #define I2C_MASTER_API_VERSION_MINOR

  初期値 :(1)

7.15.6 API データ

7.15.6.1 i2c_rate_t

i2c_rate_t

詳細説明

通信速度オプション

列挙値

名前	説明
I2C_RATE_STANDARD	100 kHz
I2C_RATE_FAST	400 kHz

名前	説明
I2C_RATE_FASTPLUS	1 MHz

7.15.6.2 i2c_addr_mode_t

i2c_addr_mode_t

詳細説明

アドレシング モード オプション

列挙値

名前	説明
I2C_ADDR_MODE_7BIT	7 ビット アドレシング モードを使用します。
I2C_ADDR_MODE_10BIT	10 ビット アドレシング モードを使用します。

7.15.6.3 i2c_event_t

i2c_event_t

詳細説明

コールバック イベント

列挙値

名前	説明
I2C_EVENT_ABORTED	送信が中止されました。
I2C_EVENT_RX_COMPLETE	受信操作が正常に完了しました。
I2C_EVENT_TX_COMPLETE	送信操作が正常に完了しました。

7.15.7 API 構造

7.15.7.1 i2c_callback_args_t

i2c_callback_args_t

詳細説明

I²C コールバック パラメータ定義

変数

- void const *const p_context

  ユーザー定義のコンテキストへのポインタ。

- uint32_t const bytes

  バッファ内の受信 / 送信バイト数。

- i2c_event_t const event

  イベント コード。

7.15.7.2 i2c_cfg_t

i2c_cfg_t

詳細説明

I²C 設定ブロック

変数

- uint32_t channel

  uint32_t

  実装によって設定可能な識別子。

- i2c_rate_t rate

  enum i2c_rate_t のデバイスの最大クロック レート。

- uint16_t slave

  スレーブ デバイスのアドレス。

- i2c_addr_mode_t addr_mode

  スレーブ フィールドの解釈方法を示します。

- void(* p_callback)(i2c_callback_args_t *p_args)

  コールバック関数へのポインタ。

  ソフトウェアの動作を制御するパラメータ

- void const * p_context

  ユーザー定義のコンテキストへのポインタ。

- void const * p_extend

  ハードウェアで必要な任意の設定データ。

  実装固有の拡張設定

7.15.7.3 i2c_ctrl_t

i2c_ctrl_t

詳細説明

初期化しないでください I²C 制御構造体。

変数

- i2c_cfg_t info

  I²C デバイスに関する情報。

- uint32_t open

  デバイスが開いているかどうかを示すフラグ。

7.15.7.4 i2c_api_master_t

i2c_api_master_t

詳細説明

マスターとしての I²C アクセスのインタフェース定義

7.15.7.5 open

ssp_err_t(* `i2c_api_master_t`::open)(`i2c_ctrl_t` *const p_ctrl, `i2c_cfg_t` const *const p_cfg)

詳細説明

I²C ドライバを開き、ハードウェアを初期化します。また電力消費を低減できます。

- R_RIIC_MasterOpen
- R_SCI_SIIC_MasterOpen

表 400: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。

表 400: パラメータ (続き)

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。

定義 : i2c_ctrl_t

初期化しないでください I²C 制御構造体。

定義 :

定義 : i2c_cfg_t const *const p_cfg

I²C 設定ブロック

- i2c_cfg_t::channel

  uint32_t

  実装によって設定可能な識別子。

- i2c_cfg_t::i2c_rate_t

  enum i2c_rate_t のデバイスの最大クロック レート。

  列挙値 :

  – I2C_RATE_STANDARD

  – I2C_RATE_FAST

  – I2C_RATE_FASTPLUS

- i2c_cfg_t::slave

  スレーブ デバイスのアドレス。

- i2c_cfg_t::i2c_addr_mode_t

  スレーブ フィールドの解釈方法を示します。

  列挙値 :

  – I2C_ADDR_MODE_7BIT

  – I2C_ADDR_MODE_10BIT

- i2c_cfg_t::p_callback

  コールバック関数へのポインタ。

  ソフトウェアの動作を制御するパラメータ

- i2c_cfg_t::p_context

  ユーザー定義のコンテキストへのポインタ。

- i2c_cfg_t::p_extend

  ハードウェアで必要な任意の設定データ。

  実装固有の拡張設定

7.15.7.6 close

ssp_err_t(* i2c_api_master_t::close)(i2c_ctrl_t *const p_ctrl)

詳細説明

ドライバを閉じ、I2C デバイスを解放します。また電力消費を低減できます。

- R_RIIC_MasterClose
- R_SCI_SIIC_MasterClose

表 401: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロックへのポインタ。

定義 : i2c_ctrl_t

初期化しないでください I2C 制御構造体。

7.15.7.7 read

ssp_err_t(* i2c_api_master_t::read)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

詳細説明

I2C デバイスで読み取り操作を実行します。また電力消費を低減できます。

- R_RIIC_MasterRead
- R_SCI_SIIC_MasterRead

表 402: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロックへのポインタ。

表 402: パラメータ (続き)

名前	方向	説明
p_dest	複数のビットを書き換えることもできます。	読み取ったデータの格納場所へのポインタ。
バイト	複数のビットを書き換えることもできます。	読み取るバイト数。
restart	複数のビットを書き換えることもできます。	読み取り後にリスタート条件を発行するかどうかを指定します。

定義 : `i2c_ctrl_t`

初期化しないでください I²C 制御構造体。

パラメータ **p_dest**

uint8_t

パラメータ **bytes**

uint32_t

パラメータ **restart**

const

7.15.7.8 write

ssp_err_t(* `i2c_api_master_t`::write)(`i2c_ctrl_t` *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

詳細説明

I²C デバイスで書き込み操作を実行します。また電力消費を低減できます。

- R_RIIC_MasterWrite
- R_SCI_SIIC_MasterWrite

表 403: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロックへのポインタ。

表 403: パラメータ (続き)

名前	方向	説明
p_src	複数のビットを書き換えることもできます。	書き込みデータを取得する場所へのポインタ。
バイト	複数のビットを書き換えることもできます。	書き込むバイト数。
restart	複数のビットを書き換えることもできます。	書き込み後にリスタート条件を発行するかどうかを指定します。

定義 : `i2c_ctrl_t`

初期化しないでください I²C 制御構造体。

パラメータ **p_src**

uint8_t

パラメータ **bytes**

uint32_t

パラメータ **restart**

const

7.15.7.9 reset

ssp_err_t(* `i2c_api_master_t`::reset)(`i2c_ctrl_t` *const p_ctrl)

詳細説明

ペリフェラルのリセットを実行します。また電力消費を低減できます。

- R_RIIC_MasterReset
- R_SCI_SIIC_MasterReset

表 404: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロックへのポインタ。

定義 : `i2c_ctrl_t`

初期化しないでください I²C 制御構造体。

7.15.7.10 versionGet

ssp_err_t(* i2c_api_master_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョン情報を取得して、それを指定されたバージョン構造体に保存します。また電力消費を低減できます。

- R_RIIC_MasterVersionGet
- R_SCI_SIIC_MasterVersionGet

表 405: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.15.7.11 i2c_master_instance_t

i2c_master_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- i2c_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- i2c_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- i2c_api_master_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.16 I2S インタフェース

I2S （IC 間サウンド）インタフェースは、I2S オーディオ通信に API および定義を提供します。

7.16.1 概要

I2S （IC 間サウンド）インタフェースは、I2S オーディオ通信に API および定義を提供します。

7.16.2 既知の実装

SSI

7.16.3 関連するモジュール

関連項目 :I2S オーディオ再生フレームワーク

7.16.4 インタフェース API

i2s_api_t

関数名	説明
.open	初期設定。
.stop	通信を停止します。I2S_EVENT_IDLE でコールバックを呼び出した場合、送信が停止します。 I2S_EVENT_RX_EMPTY でコールバックを呼び出した場合、受信が停止します。
.mute	ミュートを有効化 / 無効化します。
.write	I2S データを書き込みます。I2S_EVENT_TX_EMPTY でコールバックを呼び出した場合、すべての送信データがキューイングされます。I2S_EVENT_IDLE でコールバックを呼び出した場合、送信が完了します。
.read	I2S データを読み取ります。I2S_EVENT_RX_EMPTY でコールバックを呼び出した場合、受信が完了します。
.writeRead	I2S データの書き込みと読み取りを同時に行います。 I2S_EVENT_IDLE でコールバックを呼び出した場合、送信と受信が完了します。

関数名	説明
.infoGet	インスタンス固有情報を取得し、指定されたポインタ p_info に格納します。
.close	ドライバを再設定できるようになります。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.16.5 データ構造体

- i2s_callback_args_t
- i2s_ctrl_t
- i2s_info_t
- i2s_cfg_t
- i2s_instance_t

7.16.6 列挙

- i2s_pcm_width_t
- i2s_word_length_t
- i2s_event_t
- i2s_dir_t
- i2s_mute_t
- i2s_ws_continue_t
- i2s_status_t

7.16.7 定義

- #define I2S_API_VERSION_MAJOR

  初期値 :(01)

  定義、共通サービス、およびエラー コードを登録します。
- #define I2S_API_VERSION_MINOR

  初期値 :(1)

7.16.8 API データ

7.16.8.1 i2s_pcm_width_t

i2s_pcm_width_t

詳細説明

オーディオ PCM 幅

列挙値

名前	説明
I2S_PCM_WIDTH_8_BITS	8 ビット PCM を使用。
I2S_PCM_WIDTH_16_BITS	16 ビット PCM を使用。
I2S_PCM_WIDTH_18_BITS	18 ビット PCM を使用。
I2S_PCM_WIDTH_20_BITS	20 ビット PCM を使用。
I2S_PCM_WIDTH_22_BITS	22 ビット PCM を使用。
I2S_PCM_WIDTH_24_BITS	24 ビット PCM を使用。

7.16.8.2 i2s_word_length_t

i2s_word_length_t

詳細説明

オーディオ システムのワード長

列挙値

名前	説明
I2S_WORD_LENGTH_8_BITS	8 ビットのシステム ワード長を使用。
I2S_WORD_LENGTH_16_BITS	16 ビットのシステム ワード長を使用。
I2S_WORD_LENGTH_24_BITS	24 ビットのシステム ワード長を使用。
I2S_WORD_LENGTH_32_BITS	32 ビットのシステム ワード長を使用。

7.16.8.3 i2s_event_t

i2s_event_t

詳細説明

コールバック関数をトリガー可能なイベント

列挙値

名前	説明
I2S_EVENT_IDLE	通信がアイドル状態です。
I2S_EVENT_TX_EMPTY	送信バッファが FIFO トリガ レベルを下回っています。
I2S_EVENT_RX_FULL	受信バッファが FIFO トリガ レベルを上回っています。

7.16.8.4 i2s_dir_t

i2s_dir_t

詳細説明

I2S 通信方向

列挙値

名前	説明
I2S_DIR_TX	送信方向のみ。
I2S_DIR_RX	受信方向のみ。
I2S_DIR_TX_RX	通信および受信方向。

7.16.8.5 i2s_mute_t

i2s_mute_t

詳細説明

オーディオ サンプルをミュートします。

列挙値

名前	説明
I2S_MUTE_ON	ミュートを有効化します。
I2S_MUTE_OFF	ミュートを無効化します。

7.16.8.6 i2s_ws_continue_t

i2s_ws_continue_t

詳細説明

アイドル状態において、WS （ワード選択ライン）の送信を続行するかどうかを示します。

列挙値

名前	説明
I2S_WS_CONTINUE_ON	WS の続行モードを有効化します。
I2S_WS_CONTINUE_OFF	WS の続行モードを無効化します。

7.16.8.7 i2s_status_t

i2s_status_t

詳細説明

infoGet から返される可能性のあるステータス値。

列挙値

名前	説明
I2S_STATUS_IN_USE	I2S が使用中です。
I2S_STATUS_STOPPED	I2S は停止中です。

7.16.9 API 構造

7.16.9.1 i2s_callback_args_t

i2s_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- void const * p_context

  ユーザー データのプレースホルダー。i2s_cfg_t 内の i2s_api_t::open 関数で設定されます。

- i2s_event_t event

  このイベントを使用して、コールバックの原因 （オーバーフローまたはエラー）を特定できます。

7.16.9.2 i2s_ctrl_t

i2s_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- void(* p_callback)(i2s_callback_args_t *p_args)

  I2S ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL を使用します。

- void const * p_context

  ユーザー データのプレースホルダー。i2s_callback_args_t 内のユーザー コールバックに渡されます。

- timer_instance_t const * p_timer

  オーディオ クロックの作成に使用されるタイマ。

- transfer_instance_t const * p_transfer_tx

  書き込み時のハードウェア加速に使用される転送。

- transfer_instance_t const * p_transfer_rx

  読み取り時のハードウェア加速に使用される転送。

- uint32_t const * p_tx_src

  送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファ ポインタ。

- uint32_t tx_src_bytes

  送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファのサイズ。

- uint32_t * p_rx_dest

  受信 ISR からハードウェア FIFO を指定するために使用される宛先バッファ ポインタ。
- uint32_t rx_dest_bytes

  受信 ISR からハードウェア FIFO を指定するために使用される宛先バッファのサイズ。
- uint32_t sampling_freq_hz

  サンプリング周波数 （ヘルツ単位）。
- uint8_t channel

  チャネル番号。
- bool stop_requested_tx

  送信の完了時に I2S を停止します。
- bool stop_requested_rx

  受信の完了時に I2S を停止します。
- bool tx_in_progress

  送信転送が実行中の場合は true。
- bool zeros_written

  送信転送が実行中の場合は true。

7.16.9.3 i2s_info_t

i2s_info_t

詳細説明

I2S インスタンスに関するさまざまな情報を格納するタイマ情報構造体。

変数

- i2s_status_t status
- uint32_t sampling_freq_hz

  サンプリング周波数 （ヘルツ単位）。

7.16.9.4 i2s_cfg_t

i2s_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- uint8_t channel

  ハードウェアのチャネル番号に対応するチャネルを選択します。

- i2s_pcm_width_t pcm_width

  オーディオ PCM のデータ幅。

- i2s_word_length_t word_length

  オーディオのワード長、>= i2s_cfg_t::pcm_width ビットである必要があります。

- i2s_ws_continue_t ws_continue

  アイドル状態において、WS の送信を続行するかどうかを示します。

- uint32_t sampling_freq_hz

  サンプリング周波数 （ヘルツ単位）。

- uint32_t audio_clk_freq_hz

  オーディオのクロック周波数 （ヘルツ単位）。（16 * i2s_cfg_t::sampling_freq_hz * （i2s_cfg_t::word_length <enum_value> + 1）の 1 ～ 128 の倍数である必要があります。

- timer_instance_t const * p_timer

  GPT でオーディオ クロックを作成する場合、ここにタイマ インスタンスをリンクします。使用しない場合は、NULL に設定します。

- transfer_instance_t const * p_transfer_tx

  書き込み時に DTC を使用する場合、ここに DTC インスタンスをリンクします。使用しない場合は、NULL に設定します。

- transfer_instance_t const * p_transfer_rx

  読み取り時に DTC を使用する場合、ここに DTC インスタンスをリンクします。使用しない場合は、NULL に設定します。

- void(* p_callback)(i2s_callback_args_t *p_args)

  I2S ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

- void const * p_context

  ユーザー データのプレースホルダー。i2s_callback_args_t 内のユーザー コールバックに渡されます。

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.16.9.5 i2s_api_t

i2s_api_t

詳細説明

HAL レイヤーに実装された I2S 関数は、この API に従います。

7.16.9.6 open

ssp_err_t(* i2s_api_t::open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_SSI_Open

  I：この関数を呼び出す前に、ペリフェラルのクロックおよび必要なすべての出力ピンを設定する必要があります。

  I：この関数を呼び出した後に再構成を行うには、まず close を呼び出します。

表 406: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : i2s_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : i2s_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- i2s_cfg_t::channel

  ハードウェアのチャネル番号に対応するチャネルを選択します。

- i2s_cfg_t::i2s_pcm_width_t

  オーディオ PCM のデータ幅。

  列挙値 :

  – I2S_PCM_WIDTH_8_BITS
  – I2S_PCM_WIDTH_16_BITS
  – I2S_PCM_WIDTH_18_BITS
  – I2S_PCM_WIDTH_20_BITS
  – I2S_PCM_WIDTH_22_BITS
  – I2S_PCM_WIDTH_24_BITS

- i2s_cfg_t::i2s_word_length_t

  オーディオのワード長、>= i2s_cfg_t::pcm_width ビットである必要があります。

  列挙値 :

  – I2S_WORD_LENGTH_8_BITS
  – I2S_WORD_LENGTH_16_BITS
  – I2S_WORD_LENGTH_24_BITS
  – I2S_WORD_LENGTH_32_BITS

- i2s_cfg_t::i2s_ws_continue_t

  アイドル状態において、WS の送信を続行するかどうかを示します。

  列挙値 :

  – I2S_WS_CONTINUE_ON
  – I2S_WS_CONTINUE_OFF

- i2s_cfg_t::sampling_freq_hz

  サンプリング周波数 （ヘルツ単位)。

- i2s_cfg_t::audio_clk_freq_hz

  オーディオのクロック周波数 （ヘルツ単位)。(16 * i2s_cfg_t::sampling_freq_hz * (i2s_cfg_t::word_length <enum_value> + 1) の 1 ～ 128 の倍数である必要があります。

- i2s_cfg_t::timer_instance_t

  GPT でオーディオ クロックを作成する場合、ここにタイマ インスタンスをリンクします。使用しない場合は、NULL に設定します。

- i2s_cfg_t::transfer_instance_t

  書き込み時に DTC を使用する場合、ここに DTC インスタンスをリンクします。使用しない場合は、NULL に設定します。

- i2s_cfg_t::transfer_instance_t

  読み取り時に DTC を使用する場合、ここに DTC インスタンスをリンクします。使用しない場合は、NULL に設定します。

- i2s_cfg_t::p_callback

  I2S ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

- i2s_cfg_t::p_context

  ユーザー データのプレースホルダー。i2s_callback_args_t 内のユーザー コールバックに渡されます。

- i2s_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.16.9.7 stop

ssp_err_t(* i2s_api_t::stop)(i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)

詳細説明

通信を停止します。I2S_EVENT_IDLE でコールバックを呼び出した場合、送信が停止します。
I2S_EVENT_RX_EMPTY でコールバックを呼び出した場合、受信が停止します。また電力消費を低減できます。

- R_SSI_Stop

表 407: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。
dir	複数のビットを書き換えることもできます。	停止する通信の方向。

定義 : i2s_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ **dir**

7.16.9.8 mute

ssp_err_t(* i2s_api_t::mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

詳細説明

ミュートを有効化 / 無効化します。また電力消費を低減できます。

- R_SSI_Mute

表 408: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。
mute_enable	複数のビットを書き換えることもできます。	ミュートを有効化 / 無効化するかどうかを示します。

定義 : i2s_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ mute_enable

定義 : i2s_mute_tconst mute_enable

オーディオ サンプルをミュートします。

7.16.9.9 write

ssp_err_t(* i2s_api_t::write)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)

詳細説明

I2S データを書き込みます。I2S_EVENT_TX_EMPTY でコールバックを呼び出した場合、すべての送信データがキューイングされます。I2S_EVENT_IDLE でコールバックを呼び出した場合、送信が完了します。また電力消費を低減できます。

- R_SSI_Write

表 409: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。
p_src	複数のビットを書き換えることもできます。	PCM サンプルのバッファ。4 バイトに調整する必要があります。
バイト	複数のビットを書き換えることもできます。	バッファ内のバイト数。8 バイトの倍数を要求することが推奨されます。8 の倍数でない場合、これを 8 の倍数にするため送信にパディング 0 が追加されます。

定義 : `i2s_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_src

uint8_t

パラメータ bytes

uint16_t

7.16.9.10 read

ssp_err_t(* `i2s_api_t`::read)(`i2s_ctrl_t` *const p_ctrl, uint8_t *const p_dest, uint16_t const bytes)

詳細説明

I2S データを読み取ります。I2S_EVENT_RX_EMPTY でコールバックを呼び出した場合、受信が完了します。また電力消費を低減できます。

- R_SSI_Read

表 410: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。

表 410: パラメータ (続き)

名前	方向	説明
p_dest	複数のビットを書き換えることもできます。	PCM サンプルを保存するためのバッファ。4 バイトに調整する必要があります。
バイト	複数のビットを書き換えることもできます。	バッファ内のバイト数。8 バイトの倍数を要求することが推奨されます。8 の倍数でない場合、要求されたバイト数を下回る 8 の倍数で受信が停止します。

定義 : i2s_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_dest

uint8_t

パラメータ bytes

uint16_t

7.16.9.11 writeRead

ssp_err_t(* i2s_api_t::writeRead)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes)

詳細説明

I2S データの書き込みと読み取りを同時に行います。I2S_EVENT_IDLE でコールバックを呼び出した場合、送信と受信が完了します。また電力消費を低減できます。

- R_SSI_WriteRead

表 411: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。
p_src	複数のビットを書き換えることもできます。	PCM サンプルのバッファ。4 バイトに調整する必要があります。

表 411: パラメータ (続き)

名前	方向	説明
p_dest	複数のビットを書き換えることもできます。	PCM サンプルを保存するためのバッファ。4 バイトに調整する必要があります。
バイト	複数のビットを書き換えることもできます。	バッファ内のバイト数。8 バイトの倍数を要求することが推奨されます。8 の倍数でない場合、これを 8 の倍数にするため送信にパディング 0 が追加され、要求されたバイト数を下回る 8 の倍数で受信が停止します。

定義 : `i2s_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_src

uint8_t

パラメータ p_dest

uint8_t

パラメータ bytes

uint16_t

7.16.9.12 infoGet

ssp_err_t(* `i2s_api_t`::infoGet)(`i2s_ctrl_t` *const p_ctrl, `i2s_info_t` *const p_info)

詳細説明

インスタンス固有情報を取得し、指定されたポインタ p_info に格納します。また電力消費を低減できます。

- R_SSI_InfoGet

表 412: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。
p_info	out	このインスタンスに関する一連の情報。

定義 : i2s_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_info

定義 : i2s_info_t*const p_info

I2S インスタンスに関するさまざまな情報を格納するタイマ情報構造体。

- i2s_info_t::status
- i2s_info_t::sampling_freq_hz

  サンプリング周波数 （ヘルツ単位）。

7.16.9.13 close

ssp_err_t(* i2s_api_t::close)(i2s_ctrl_t *const p_ctrl)

詳細説明

ドライバを再設定できるようになります。また電力消費を低減できます。

- R_SSI_Close

表 413: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このインスタンスの open 呼び出しで設定された制御ブロック。

定義 : i2s_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.16.9.14 versionGet

ssp_err_t(* i2s_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_SSI_VersionGet

表 414: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.16.9.15 i2s_instance_t

i2s_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- i2s_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- i2s_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- i2s_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.17 入力キャプチャインタフェース

パルス幅の入力信号をサンプリングするためのインタフェース。

7.17.1 概要

入力キャプチャ インタフェースは、パルス幅 （エッジから反対側のエッジまで）を特定するための入力信号のサンプリングを提供します。各測定結果がキャプチャされると割り込みがトリガされます。

以下によって実装されます。GPT 入力キャプチャ

関連項目：タイマインタフェース

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

7.17.2 インタフェース API

input_capture_api_t

関数名	説明
.open	初期設定。
.disable	入力キャプチャ測定を無効化します。
.enable	入力キャプチャ測定を有効化します。
.infoGet	測定カウンタのステータス （実行中かどうか）を取得します。
.lastCaptureGet	最後にキャプチャされたタイマ / カウンタの値を取得します
.close	入力キャプチャ操作を終了します。ドライバを再設定できるようになります。
.versionGet	この API のバージョンを取得し、p_version がポイントする構造体に格納します。

7.17.3 データ構造体

- input_capture_callback_args_t
- input_capture_capture_t
- input_capture_info_t
- input_capture_ctrl_t
- input_capture_cfg_t
- input_capture_instance_t

7.17.4 列挙

- input_capture_mode_t
- input_capture_signal_edge_t
- input_capture_signal_level_t
- input_capture_repetition_t
- input_capture_event_t
- info_capture_status_t

7.17.5 定義

- #define INPUT_CAPTURE_API_VERSION_MAJOR

  初期値 :(1)
- #define INPUT_CAPTURE_API_VERSION_MINOR

  初期値 :(1)

7.17.6 API データ

7.17.6.1 input_capture_mode_t

input_capture_mode_t

詳細説明

入力キャプチャ動作モード

列挙値

名前	説明
INPUT_CAPTURE_MODE_PULSE_WIDTH	信号パルス幅を測定します。

7.17.6.2 input_capture_signal_edge_t

input_capture_signal_edge_t

詳細説明

入力キャプチャ信号エッジトリガ

列挙値

名前	説明
INPUT_CAPTURE_SIGNAL_EDGE_RISING	上昇エッジでキャプチャが開始されます。
INPUT_CAPTURE_SIGNAL_EDGE_FALLING	下降エッジでキャプチャが開始されます。

7.17.6.3 input_capture_signal_level_t

input_capture_signal_level_t

詳細説明

入力キャプチャ信号レベル、信号の有効化に主に使用されます。

列挙値

名前	説明
INPUT_CAPTURE_SIGNAL_LEVEL_NONE	信号レベルが特定の測定に適用されない場合にこれを使用します。
INPUT_CAPTURE_SIGNAL_LEVEL_LOW	ローレベルでキャプチャが有効になります。
INPUT_CAPTURE_SIGNAL_LEVEL_HIGH	ハイレベルでキャプチャが有効になります。

7.17.6.4 input_capture_repetition_t

input_capture_repetition_t

詳細説明

ワンタイムまたは継続測定を指定します。

列挙値

名前	説明
INPUT_CAPTURE_REPETITION_PERIODIC	明示的に停止または無効にするまで、継続的に測定をキャプチャします。
INPUT_CAPTURE_REPETITION_ONE_SHOT	単一の測定をキャプチャすると、割り込みは無効になります。

7.17.6.5 input_capture_event_t

input_capture_event_t

詳細説明

コールバック関数をトリガー可能なイベント

列挙値

名前	説明
INPUT_CAPTURE_EVENT_MEASUREMENT	キャプチャ測定がキャプチャされました。
INPUT_CAPTURE_EVENT_OVERFLOW	キャプチャ測定がカウンタをオーバーフローしました。

7.17.6.6 info_capture_status_t

info_capture_status_t

詳細説明

入力キャプチャの状態。

列挙値

名前	説明
INPUT_CAPTURE_STATUS_IDLE	入力キャプチャ タイマがアイドル状態です。
INPUT_CAPTURE_STATUS_CAPTURING	キャプチャ測定を実行中です。

7.17.7 API 構造

7.17.7.1 input_capture_callback_args_t

input_capture_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- uint8_t channel

  使用中のチャネル。

- input_capture_event_t event

  割り込みとコールバックを発生させたイベント。

- uint32_t counter

  割り込み時にキャプチャされたタイマの値。

- uint32_t overflows

  この測定中に発生したカウンタ オーバーフローの回数。

- void const * p_context

  input_capture_cfg_t::p_context で設定されたユーザー データのプレースホルダー。

7.17.7.2 input_capture_capture_t

input_capture_capture_t

詳細説明

データをキャプチャします

変数

- uint32_t counter

  割り込み時にキャプチャされたタイマの値。

- uint32_t overflows

  この測定中に発生したカウンタ オーバーフローの回数。

7.17.7.3 input_capture_info_t

input_capture_info_t

詳細説明

ドライバ情報

変数

- info_capture_status_t status

  このキャプチャが実行中かどうかを示します。

7.17.7.4 input_capture_ctrl_t

input_capture_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- uint8_t channel

  使用中のチャネル。

- input_capture_mode_t mode

  実行されている測定モード。

- input_capture_repetition_t repetition

  ユーザーのコールバック関数へのポインタ。

- uint32_t overflows_last

  最後の測定時に発生したオーバーフロー カウント。

- uint32_t overflows_current

  現在の測定中に発生しているオーバーフローの累積カウント。

- void(* p_callback)(input_capture_callback_args_t *p_args)

  ユーザー コールバックへのポインタ。

- void const * p_context

  コールバック関数に渡される、ユーザーのコンテキスト データへのポインタ。

7.17.7.5 input_capture_cfg_t

input_capture_cfg_t

詳細説明

open 関数に渡されるユーザー構成構造体

変数

- uint8_t channel

  使用中のチャネル。

- input_capture_mode_t mode

  実行される測定モード。

- input_capture_signal_edge_t edge

  測定を開始するトリガー エッジ （立ち上がりエッジ、立ち下がりエッジ、両エッジ）。

- input_capture_repetition_t repetition

  ユーザーのコールバック関数へのポインタ。

- bool autostart

  開いた後に、割り込みを有効にするかどうかの指定。

- void const * p_extend

  必須です。ペリフェラルに固有の拡張パラメータへのポインタ。GPT については gpt_input_capture_extend_t を参照してください。

- void(* p_callback)(input_capture_callback_args_t *p_args)

  割り込みを行わない場合は、NULL です。

- void const * p_context

  コールバックに渡される、ユーザーのコンテキスト データへのポインタ。

7.17.7.6 input_capture_api_t

input_capture_api_t

詳細説明

入力キャプチャの API 構造体。この API は、HAL レイヤーに実装された関数によって実装されます。

7.17.7.7 open

ssp_err_t(* input_capture_api_t::open)(input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_GPT_InputCaptureOpen

  I：この関数を呼び出した後に再構成を行うには、まず close を呼び出します。

表 415: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ブロックへのポインタ：メモリは呼び出し側によって割り当てられ、開くことで内容が指定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : input_capture_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : input_capture_cfg_t const *const p_cfg

open 関数に渡されるユーザー構成構造体

- input_capture_cfg_t::channel

  使用中のチャネル。

- input_capture_cfg_t::input_capture_mode_t

  実行される測定モード。

  列挙値 :

  – INPUT_CAPTURE_MODE_PULSE_WIDTH

- input_capture_cfg_t::input_capture_signal_edge_t

  測定を開始するトリガー エッジ （立ち上がりエッジ、立ち下がりエッジ、両エッジ)。

  列挙値 :

  – INPUT_CAPTURE_SIGNAL_EDGE_RISING

  – INPUT_CAPTURE_SIGNAL_EDGE_FALLING

- input_capture_cfg_t::input_capture_repetition_t

  ユーザーのコールバック関数へのポインタ。

  列挙値 :

  – INPUT_CAPTURE_REPETITION_PERIODIC

– INPUT_CAPTURE_REPETITION_ONE_SHOT

- input_capture_cfg_t::autostart

  開いた後に、割り込みを有効にするかどうかの指定。

- input_capture_cfg_t::p_extend

  必須です。ペリフェラルに固有の拡張パラメータへのポインタ。GPT については gpt_input_capture_extend_t を参照してください。

- input_capture_cfg_t::p_callback

  割り込みを行わない場合は、NULL です。

- input_capture_cfg_t::p_context

  コールバックに渡される、ユーザーのコンテキスト データへのポインタ。

7.17.7.8 disable

ssp_err_t(* input_capture_api_t::disable)(input_capture_ctrl_t const *const p_ctrl)

詳細説明

入力キャプチャ測定を無効化します。また電力消費を低減できます。

- R_GPT_InputCaptureDisable

表 416: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しによって初期化された制御ブロックへのポインタ。

定義 : input_capture_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.17.7.9 enable

ssp_err_t(* input_capture_api_t::enable)(input_capture_ctrl_t const *const p_ctrl)

詳細説明

入力キャプチャ測定を有効化します。また電力消費を低減できます。

- R_GPT_InputCaptureEnable

表 417: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しによって初期化された制御ブロックへのポインタ。

I：割り込みは、input_capture_cfg_t::irq_enable で指定されている場合、すでに有効である可能性があります。

定義 : input_capture_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.17.7.10 infoGet

ssp_err_t(* input_capture_api_t::infoGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_info_t *const p_info)

詳細説明

測定カウンタのステータス （実行中かどうか）を取得します。また電力消費を低減できます。

- R_GPT_InputCaptureInfoGet

表 418: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しによって初期化された制御ブロックへのポインタ。
p_info	out	返されたステータスへのポインタ。結果は、input_capture_status_t のいずれかです。

定義 : input_capture_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_info

定義 : input_capture_info_t*const p_info

ドライバ情報

- input_capture_info_t::status

  このキャプチャが実行中かどうかを示します。

7.17.7.11 lastCaptureGet

ssp_err_t(* input_capture_api_t::lastCaptureGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_capture_t *const p_counter)

詳細説明

最後にキャプチャされたタイマ / カウンタの値を取得します。以下として実装されます。

- R_GPT_InputCaptureLastCaptureGet

表 419: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しによって初期化された制御ブロックへのポインタ。
p_counter	out	最後にキャプチャされたカウンタの格納場所へのポインタ。

定義 : input_capture_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_counter

定義 : input_capture_capture_t*const p_counter

データをキャプチャします

- input_capture_capture_t::counter

  割り込み時にキャプチャされたタイマの値。

- input_capture_capture_t::overflows

  この測定中に発生したカウンタ オーバーフローの回数。

7.17.7.12 close

ssp_err_t(* input_capture_api_t::close)(input_capture_ctrl_t *const p_ctrl)

詳細説明

入力キャプチャ操作を終了します。ドライバを再設定できるようになります。また電力消費を低減できます。

- R_GPT_InputCaptureClose

表 420: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しによって初期化された制御ブロックへのポインタ。

定義 : `input_capture_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.17.7.13 versionGet

ssp_err_t(* `input_capture_api_t`::versionGet)(ssp_version_t *const p_version)

詳細説明

この API のバージョンを取得し、p_version がポイントする構造体に格納します。また電力消費を低減できます。

- R_GPT_InputCaptureVersionGet

表 421: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.17.7.14 input_capture_instance_t

input_capture_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- input_capture_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- input_capture_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- input_capture_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.18 I/O ポートインタフェース

I/O ポートにアクセスして I/O 関数を構成するためのインタフェース。

IOPort 共有インタフェースは、ビットおよびポートの両方のレベルでデバイスの I/O ポートにアクセスするための機能を提供します。ポートとピンの方向は変更できます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

IOPort インタフェースの説明 : I/O ポート ドライバ

7.18.1 インタフェース API

ioport_api_t

関数名	説明
.init	複数のピンの設定を初期化します。
.pinCfg	1 本のピンの設定を指定します。
.pinDirectionSet	ピンのピン方向を設定します。
.pinEventInputRead	指定されたピンのイベント入力データを読み取り、そのレベルを返します。
.pinEventOutputWrite	ピンのイベント データを書き込みます。
.pinEthernetModeCfg	イーサネット チャネルの PHY モードを設定します。
.pinRead	ピンのレベルを読み取ります。
.pinWrite	指定されたレベルをピンに書き込みます。
.portDirectionSet	ポート上の 1 つ以上のピンに対し、ピンの方向を設定します。
.portEventInputRead	ポートに関してキャプチャされたイベント データを読み取ります。
.portEventOutputWrite	ポートに関するイベント出力データを書き込みます。

関数名	説明
.portRead	指定されたポート上のピンの状態を読み取ります。
.portWrite	ポート上の複数のピンに書き込みます。
.versionGet	IOPort ドライバのバージョンを返します。

7.18.2 データ構造体

- ioport_pin_cfg_t
- ioport_cfg_t
- ioport_instance_t

7.18.3 列挙

- ioport_level_t
- ioport_direction_t

7.18.4 定義

- #define IOPORT_API_VERSION_MAJOR
  初期値 :(1)
- #define IOPORT_API_VERSION_MINOR
  初期値 :(1)

7.18.5 API データ

7.18.5.1 ioport_level_t

ioport_level_t

詳細説明

個別のピンに対して設定および読み取りが可能なレベル

列挙値

名前	説明
IOPORT_LEVEL_LOW	低。
IOPORT_LEVEL_HIGH	高。

7.18.5.2 ioport_direction_t

ioport_direction_t

詳細説明

個々のピンの方向

列挙値

名前	説明
IOPORT_DIRECTION_INPUT	入力。
IOPORT_DIRECTION_OUTPUT	出力。

7.18.6 API 構造

7.18.6.1 ioport_pin_cfg_t

ioport_pin_cfg_t

詳細説明

ピンの識別子とピン PFS ピン設定値

変数

- uint32_t pin_cfg

  ioport_cfg_options_t パラメータを使用して設定します。
- ioport_port_pin_t pin

  ピンの識別子。

7.18.6.2 ioport_cfg_t

ioport_cfg_t

詳細説明

以下によって PFS レジスタに読み込まれる、複数ピンの設定データ :R_IOPORT_Init

変数

- uint16_t number_of_pins

  設定データが存在するピンの数。

- ioport_pin_cfg_t const * p_pin_cfg_data

  ピンの設定データ。

7.18.6.3 ioport_api_t

ioport_api_t

詳細説明

IOPort ドライバ構造体。HAL レイヤーに実装された IOPort 関数は、この API に従います。

7.18.6.4 init

ssp_err_t(* `ioport_api_t`::init)(const `ioport_cfg_t` *p_cfg)

詳細説明

複数のピンの設定を初期化します。また電力消費を低減できます。

- R_IOPORT_Init

表 422: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	ピン設定データ配列へのポインタ。

定義 :

定義 : `ioport_cfg_t` ioport_cfg_t *p_cfg

以下によって PFS レジスタに読み込まれる、複数ピンの設定データ :R_IOPORT_Init

- `ioport_cfg_t`::number_of_pins

  設定データが存在するピンの数。

- `ioport_cfg_t`::`ioport_pin_cfg_t`

  ピンの設定データ。

7.18.6.5 pinCfg

ssp_err_t(* ioport_api_t::pinCfg)(ioport_port_pin_t pin, uint32_t cfg)

詳細説明

1 本のピンの設定を指定します。また電力消費を低減できます。

- R_IOPORT_PinCfg

表 423: パラメータ

名前	方向	説明
pin	複数のビットを書き換えることもできます。	読み取り対象のピン。
cfg	複数のビットを書き換えることもできます。	ピンの設定オプション。

パラメータ pin

パラメータ cfg

uint32_t

7.18.6.6 pinDirectionSet

ssp_err_t(* ioport_api_t::pinDirectionSet)(ioport_port_pin_t pin, ioport_direction_t direction)

詳細説明

ピンのピン方向を設定します。また電力消費を低減できます。

- R_IOPORT_PinDirectionSet

表 424: パラメータ

名前	方向	説明
pin	複数のビットを書き換えることもできます。	設定対象のピン。
direction	複数のビットを書き換えることもできます。	ピンの方向を指定する ioport_direction_t のメンバー。

パラメータ **pin**

パラメータ **direction**

7.18.6.7 pinEventInputRead

ssp_err_t(* ioport_api_t::pinEventInputRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_event)

詳細説明

指定されたピンのイベント入力データを読み取り、そのレベルを返します。また電力消費を低減できます。

- R_IOPORT_PinEventInputRead

表 425: パラメータ

名前	方向	説明
pin	複数のビットを書き換えることもできます。	読み取り対象のピン。
p_pin_event	複数のビットを書き換えることもできます。	イベント データを返すポインタ。

パラメータ **pin**

パラメータ **p_pin_event**

定義 : ioport_level_t*p_pin_event

個別のピンに対して設定および読み取りが可能なレベル

7.18.6.8 pinEventOutputWrite

ssp_err_t(* ioport_api_t::pinEventOutputWrite)(ioport_port_pin_t pin, ioport_level_t pin_value)

詳細説明

ピンのイベント データを書き込みます。また電力消費を低減できます。

- R_IOPORT_PinEventOutputWrite

表 426: パラメータ

名前	方向	説明
pin	複数のビットを書き換えることもできます。	イベント データの書き込み先のピン。

表 426: パラメータ (続き)

名前	方向	説明
pin_value	複数のビットを書き換えることもできます。	ピンの出力イベントに書き込まれるレベル。

パラメータ pin

パラメータ pin_value

定義 : ioport_level_tpin_value

個別のピンに対して設定および読み取りが可能なレベル

7.18.6.9 pinEthernetModeCfg

ssp_err_t(* ioport_api_t::pinEthernetModeCfg)(ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)

詳細説明

イーサネット チャネルの PHY モードを設定します。また電力消費を低減できます。

- R_IOPORT_EthernetModeCfg

表 427: パラメータ

名前	方向	説明
channel	複数のビットを書き換えることもできます。	チャネル設定を指定するチャネル。
mode	複数のビットを書き換えることもできます。	チャネルに対して指定する PHY モード。

パラメータ channel

パラメータ mode

7.18.6.10 pinRead

ssp_err_t(* ioport_api_t::pinRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_value)

詳細説明

ピンのレベルを読み取ります。また電力消費を低減できます。

- R_IOPORT_PinRead

表 428: パラメータ

名前	方向	説明
pin	複数のビットを書き換えることもできます。	読み取り対象のピン。
p_pin_value	複数のビットを書き換えることもできます。	ピンのレベルを返すポインタ。

パラメータ **pin**

パラメータ **p_pin_value**

定義 : `ioport_level_t`*p_pin_value

個別のピンに対して設定および読み取りが可能なレベル

7.18.6.11 pinWrite

ssp_err_t(* `ioport_api_t`::pinWrite)(ioport_port_pin_t pin, `ioport_level_t` level)

詳細説明

指定されたレベルをピンに書き込みます。また電力消費を低減できます。

- R_IOPORT_PinWrite

表 429: パラメータ

名前	方向	説明
pin	複数のビットを書き換えることもできます。	書き込み先のピン。
level	複数のビットを書き換えることもできます。	ピンに書き込む状態。

パラメータ **pin**

パラメータ **level**

7.18.6.12 portDirectionSet

ssp_err_t(* `ioport_api_t`::portDirectionSet)(ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask)

詳細説明

ポート上の 1 つ以上のピンに対し、ピンの方向を設定します。また電力消費を低減できます。

- R_IOPORT_PortDirectionSet

表 430: パラメータ

名前	方向	説明
port	複数のビットを書き換えることもできます。	設定対象のポート。
direction_values	複数のビットを書き換えることもできます。	ポート上のピンの方向を制御する値（1 - 出力、0 - 入力）。
mask	複数のビットを書き換えることもできます。	ポート上のどのピンを設定するかを制御するためのマスク。

パラメータ port

パラメータ direction_values

パラメータ mask

7.18.6.13 portEventInputRead

ssp_err_t(* ioport_api_t::portEventInputRead)(ioport_port_t port, ioport_size_t *p_event_data)

詳細説明

ポートに関してキャプチャされたイベント データを読み取ります。また電力消費を低減できます。

- R_IOPORT_PortEventInputRead

表 431: パラメータ

名前	方向	説明
port	複数のビットを書き換えることもできます。	読み取り対象のポート。
p_event_data	複数のビットを書き換えることもできます。	イベント データを返すポインタ。

パラメータ **port**

パラメータ **p_event_data**

7.18.6.14 portEventOutputWrite

ssp_err_t(* ioport_api_t::portEventOutputWrite)(ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value)

詳細説明

ポートに関するイベント出力データを書き込みます。また電力消費を低減できます。

- R_IOPORT_PortEventOutputWrite

表 432: パラメータ

名前	方向	説明
port	複数のビットを書き換えることもできます。	イベント データの書き込み先のポート。
event_data	複数のビットを書き換えることもできます。	指定されたポートにイベント データとして書き込まれるデータ。
mask_value	複数のビットを書き換えることもできます。	マスクで 1 に設定されている各ビットは、ポートに書き込まれるイベント データのビットの値に対応します。

パラメータ **port**

パラメータ **event_data**

パラメータ **mask_value**

7.18.6.15 portRead

ssp_err_t(* ioport_api_t::portRead)(ioport_port_t port, ioport_size_t *p_port_value)

詳細説明

指定されたポート上のピンの状態を読み取ります。また電力消費を低減できます。

- R_IOPORT_PortRead

表 433: パラメータ

名前	方向	説明
port	複数のビットを書き換えることもできます。	読み取り対象のポート。
p_port_value	複数のビットを書き換えることもできます。	ポートの値を返すポインタ。

パラメータ **port**

パラメータ **p_port_value**

7.18.6.16 portWrite

ssp_err_t(* ioport_api_t::portWrite)(ioport_port_t port, ioport_size_t value, ioport_size_t mask)

詳細説明

ポート上の複数のピンに書き込みます。また電力消費を低減できます。

- R_IOPORT_PortWrite

表 434: パラメータ

名前	方向	説明
port	複数のビットを書き換えることもできます。	書き込み先のポート。
value	複数のビットを書き換えることもできます。	ポートに書き込まれる値。
mask	複数のビットを書き換えることもできます。	ポート上の書き込み先のピンを選択するためのマスク。

パラメータ **port**

パラメータ **value**

パラメータ **mask**

7.18.6.17 versionGet

ssp_err_t(* ioport_api_t::versionGet)(ssp_version_t *p_data)

詳細説明

IOPort ドライバのバージョンを返します。また電力消費を低減できます。

- R_IOPORT_VersionGet

表 435: パラメータ

名前	方向	説明
p_data	out	バージョン情報を返す先のメモリ アドレス。

パラメータ p_data

定義 : `ssp_version_t` *p_data

- `ssp_version_t`::version_id

  バージョン ID

- `ssp_version_t`::code_version_minor

  コードのマイナー バージョン。

- `ssp_version_t`::code_version_major

  コードのメジャー バージョン。

- `ssp_version_t`::api_version_minor

  API のマイナー バージョン。

- `ssp_version_t`::api_version_major

  API のメジャー バージョン。

- `ssp_version_t`struct{}

  コード バージョンのパラメータ

7.18.6.18 ioport_instance_t

ioport_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- ioport_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- ioport_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.19 JPEG デコードインタフェース

JPEG デコード機能用インタフェース。

7.19.1 概要

JPEG DECODE インタフェースは JPEG デコーダ機能を提供します。アプリケーションで、JPEG イメージをディスプレイ フレーム バッファ用ビットマップ データに変換できるようになります。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

JPEG DECODE インタフェースの説明 : JPEG デコード ドライバ

7.19.2 インタフェース API

jpeg_decode_api_t

関数名	説明
.open	初期設定
.outputBufferSet	JPEG コーデックに対し、出力データを格納するための出力バッファを割り当てます。
.horizontalStrideSet	水平ストライド値を設定します。
.imageSubsampleSet	水平 / 垂直サブサンプル設定値を指定します。
.inputBufferSet	JPEG コーデックに入力データ バッファを割り当てます。
.linesDecodedGet	出力バッファにデコードされたライン数を返します。
.imageSizeGet	デコード操作時に画像サイズを取得します。
.statusGet	JPEG コーデック モジュールの現在のステータスを取得します。
.close	未処理の操作をキャンセルします。

関数名	説明
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。
.pixelFormatGet	入力ピクセル形式を取得します。

7.19.3 データ構造体

- jpeg_decode_callback_args_t
- jpeg_decode_cfg_t
- jpeg_decode_ctrl_t
- jpeg_decode_instance_t

7.19.4 列挙

- jpeg_decode_color_space_t
- jpeg_decode_data_format_t
- jpeg_decode_pixel_format_t
- jpeg_decode_status_t
- jpeg_decode_subsample_t
- jpeg_decode_count_enable_t
- jpeg_decode_resume_mode_t

7.19.5 定義

- #define JPEG_DECODE_API_VERSION_MAJOR

  初期値 :(1)

  定義、共通サービス、およびエラー コードを登録します。このモジュールの構成
- #define JPEG_DECODE_API_VERSION_MINOR

  初期値 :(1)

7.19.6 API データ

7.19.6.1 jpeg_decode_color_space_t

jpeg_decode_color_space_t

詳細説明

画像の色空間の定義

列挙値

名前	説明
JPEG_DECODE_COLOR_SPACE_YCBCR444	カラースペース YCbCr 444。
JPEG_DECODE_COLOR_SPACE_YCBCR422	カラースペース YCbCr 422。
JPEG_DECODE_COLOR_SPACE_YCBCR420	カラースペース YCbCr 420。
JPEG_DECODE_COLOR_SPACE_YCBCR411	カラースペース YCbCr 411。

7.19.6.2 jpeg_decode_data_format_t

jpeg_decode_data_format_t

詳細説明

マルチバイト データ フォーマット

列挙値

名前	説明
JPEG_DECODE_DATA_FORMAT_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) ノーマルバイト順序
JPEG_DECODE_DATA_FORMAT_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) バイト スワップ
JPEG_DECODE_DATA_FORMAT_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) ワード スワップ
JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) ワード バイト スワップ
JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) ロングワード スワップ

名前	説明
JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) ロングワード バイト スワップ
JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) ロングワード ワード スワップ
JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) ロングワード ワード バイト スワップ
JPEG_DECODE_DATA_FORMAT_MAX	

7.19.6.3 jpeg_decode_pixel_format_t

jpeg_decode_pixel_format_t

詳細説明

ピクセル データ フォーマット

列挙値

名前	説明
JPEG_DECODE_PIXEL_FORMAT_ARGB8888	ピクセル データ ARG B888 フォーマット。
JPEG_DECODE_PIXEL_FORMAT_RGB565	ピクセル データ RGB 565 フォーマット。

7.19.6.4 jpeg_decode_status_t

jpeg_decode_status_t

詳細説明

JPEG HLD ドライバの内部ステータス情報

列挙値

名前	説明
JPEG_DECODE_STATUS_FREE	JPEG。
JPEG_DECODE_STATUS_IDLE	JPEG コーデックモジュールが開いており、操作できません。

名前	説明
JPEG_DECODE_STATUS_RUNNING	JPEG コーデックが実行中です。
JPEG_DECODE_STATUS_DONE	JPEG コーデックが操作を正常に完了しました。
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG コーデックは追加の入力データを待つために一時停止しています。
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG コーデックは、ユーザーが指定した行数をデコードした後に一時停止しています
JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG デコーディング操作はイメージ サイズを取得して、一時停止しています。
JPEG_DECODE_STATUS_ERROR	JPEG コーデック モジュールでエラーが発生しました。
JPEG_DECODE_STATUS_HEADER_PROCESSING	JPEG コーデック モジュールでエラーが発生しました。

7.19.6.5 jpeg_decode_subsample_t

jpeg_decode_subsample_t

詳細説明

水平 / 垂直サブサンプル設定のデータ型。この設定は、デコード操作にのみ適用されます。

列挙値

名前	説明
JPEG_DECODE_OUTPUT_NO_SUBSAMPLE	サブサンプルなし。減少なしでイメージがデコードされます。
JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF	出力イメージ サイズは半分に減少します。
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER	出力イメージ サイズは 1/4 に減少します。
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH	出力イメージ サイズは 1/8 に減少します。

7.19.6.6 jpeg_decode_count_enable_t

jpeg_decode_count_enable_t

詳細説明

カウント モード有効をデコードするためのデータ タイプです。

列挙値

名前	説明
JPEG_DECODE_COUNT_DISABLE	カウント モードが無効です。
JPEG_DECODE_COUNT_ENABLE	カウント モードが有効です。

7.19.6.7 jpeg_decode_resume_mode_t

jpeg_decode_resume_mode_t

詳細説明

カウント モード有効をデコードするためのデータ タイプです。

列挙値

名前	説明
JPEG_DECODE_COUNT_MODE_ADDRESS_CONTINUE	データ バッファ アドレスは初期化されません。
JPEG_DECODE_COUNT_MODE_ADDRESS_REINITIALIZE	データ バッファ アドレスは次の場合に初期化されます。

7.19.7 API 構造

7.19.7.1 jpeg_decode_callback_args_t

jpeg_decode_callback_args_t

詳細説明

コールバック ステータス構造体

変数

- jpeg_decode_status_t status

  JPEG のステータス。

- void const * p_context

  ユーザー定義のコンテキストへのポインタ。

7.19.7.2 jpeg_decode_cfg_t

jpeg_decode_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体。

変数

- jpeg_decode_color_space_t color_space

  色空間。

- jpeg_decode_data_format_t input_data_format

  入力データ ストリームのバイト順序。

- jpeg_decode_data_format_t output_data_format

  出力データ ストリームのバイト順序。

- jpeg_decode_pixel_format_t pixel_format

  ピクセル フォーマット。

- uint8_t alpha_value

  デコードされたピクセル データに適用されるアルファ値。これのみです。

- void(* p_callback)(jpeg_decode_callback_args_t *p_args)

  ユーザー定義のコールバック関数。

- void const * p_context

  ユーザー データのプレースホルダー。jpeg_decode_callback_args_t 内のユーザー コールバックに渡されます。

7.19.7.3 jpeg_decode_ctrl_t

jpeg_decode_ctrl_t

詳細説明

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

変数

- jpeg_decode_status_t status

  JPEG コーデック モジュールのステータス。

- ssp_err_t error_code

  JPEG コーデックのエラー コード （ある場合）。

- void(* p_callback)(jpeg_decode_callback_args_t *p_args)

  ユーザー定義のコールバック関数。

- void const * p_extend

  JPEG コーデックのハードウェアに依存する設定 */。

- void const * p_context

  ユーザー データのプレースホルダー。jpeg_decode_callback_args_t 内のユーザー コールバックに渡されます。

- jpeg_decode_pixel_format_t pixel_format

  ピクセル フォーマット。

- uint32_t horizontal_stride

  水平ストライドの設定値。

- uint32_t outbuffer_size

  出力バッファのサイズ

- uint32_t total_lines_decoded

  それまでにデコードされたライン数を追跡します。

7.19.7.4 jpeg_decode_api_t

jpeg_decode_api_t

詳細説明

HAL レイヤーに実装された JPEG 関数は、この API に従います。

7.19.7.5 open

ssp_err_t(* `jpeg_decode_api_t`::open)(`jpeg_decode_ctrl_t` *const p_ctrl, `jpeg_decode_cfg_t` const *const p_cfg)

詳細説明

初期設定。以下として実装されます。

- R_JPEG_Decode_Open

  I：なし

表 436: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

定義 :

定義 : jpeg_decode_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体。

- jpeg_decode_cfg_t::jpeg_decode_color_space_t

  色空間。

  列挙値 :

  – JPEG_DECODE_COLOR_SPACE_YCBCR444

  – JPEG_DECODE_COLOR_SPACE_YCBCR422

  – JPEG_DECODE_COLOR_SPACE_YCBCR420

  – JPEG_DECODE_COLOR_SPACE_YCBCR411

- jpeg_decode_cfg_t::jpeg_decode_data_format_t

  入力データ ストリームのバイト順序。

  列挙値 :

  – JPEG_DECODE_DATA_FORMAT_NORMAL

  – JPEG_DECODE_DATA_FORMAT_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_WORD_SWAP

  – JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP

- JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP
- JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP
- JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP
- JPEG_DECODE_DATA_FORMAT_MAX

- jpeg_decode_cfg_t::jpeg_decode_data_format_t

  出力データ ストリームのバイト順序。

  列挙値 :

  - JPEG_DECODE_DATA_FORMAT_NORMAL
  - JPEG_DECODE_DATA_FORMAT_BYTE_SWAP
  - JPEG_DECODE_DATA_FORMAT_WORD_SWAP
  - JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP
  - JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP
  - JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP
  - JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP
  - JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP
  - JPEG_DECODE_DATA_FORMAT_MAX

- jpeg_decode_cfg_t::jpeg_decode_pixel_format_t

  ピクセル フォーマット。

  列挙値 :

  - JPEG_DECODE_PIXEL_FORMAT_ARGB8888
  - JPEG_DECODE_PIXEL_FORMAT_RGB565

- jpeg_decode_cfg_t::alpha_value

  デコードされたピクセル データに適用されるアルファ値。これのみです。

- jpeg_decode_cfg_t::p_callback

  ユーザー定義のコールバック関数。

- jpeg_decode_cfg_t::p_context

  ユーザー データのプレースホルダー。jpeg_decode_callback_args_t 内のユーザー コールバックに渡されます。

7.19.7.6 outputBufferSet

ssp_err_t(* jpeg_decode_api_t::outputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

詳細説明

JPEG コーデックに対し、出力データを格納するための出力バッファを割り当てます。また電力消費を低減できます。

- R_JPEG_Decode_OutputBufferSet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

I : バッファの開始アドレスは、8 バイトでアラインする必要があります。デコード プロセスの場合、HLD ドライバが目的のスペースに合わせて出力データを調整し、それに従ってデコード後の画像のライン数を自動的に計算します。用意された出力バッファにフレームが入りきらない場合、アプリケーションは Output Full Callback 関数を呼び出して、追加のバッファ空間が必要になった場合に通知を受け取るようにする必要があります。

表 437: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
p_buffer	複数のビットを書き換えることもできます。	出力バッファ空間へのポインタ
buffer_size	複数のビットを書き換えることもできます。	出力バッファのサイズ

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ p_buffer

const

パラメータ buffer_size

uint32_t

7.19.7.7 horizontalStrideSet

ssp_err_t(* jpeg_decode_api_t::horizontalStrideSet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

詳細説明

水平ストライド値を設定します。また電力消費を低減できます。

- R_JPEG_Decode_HorizontalStrideSet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

表 438: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
horizontal_stride	複数のビットを書き換えることもできます。	デコードされた画像データに使用する水平ストライド値。
buffer_size	複数のビットを書き換えることもできます。	出力バッファのサイズ

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ horizontal_stride

uint32_t

パラメータ buffer_size

7.19.7.8 imageSubsampleSet

ssp_err_t(* jpeg_decode_api_t::imageSubsampleSet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

詳細説明

水平 / 垂直サブサンプル設定値を指定します。また電力消費を低減できます。

- R_JPEG_Decode_ImageSubsampleSet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

表 439: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
horizontal_subsample	複数のビットを書き換えることもできます。	水平サブサンプル値
vertical_subsample	複数のビットを書き換えることもできます。	垂直サブサンプル値

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ horizontal_subsample

定義 : jpeg_decode_subsample_thorizontal_subsample

水平 / 垂直サブサンプル設定のデータ型。この設定は、デコード操作にのみ適用されます。

パラメータ vertical_subsample

定義 : jpeg_decode_subsample_tvertical_subsample

水平 / 垂直サブサンプル設定のデータ型。この設定は、デコード操作にのみ適用されます。

7.19.7.9 inputBufferSet

ssp_err_t(* jpeg_decode_api_t::inputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

詳細説明

JPEG コーデックに入力データ バッファを割り当てます。また電力消費を低減できます。

- R_JPEG_Decode_InputBufferSet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

I : バッファの開始アドレスは、8 バイトでアラインする必要があります。

表 440: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
p_buffer	複数のビットを書き換えることもできます。	入力バッファ空間へのポインタ
buffer_size	複数のビットを書き換えることもできます。	入力バッファのサイズ

定義 : `jpeg_decode_ctrl_t`

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ p_buffer

const

パラメータ buffer_size

uint32_t

7.19.7.10 linesDecodedGet

ssp_err_t(* `jpeg_decode_api_t`::linesDecodedGet)(`jpeg_decode_ctrl_t` *const p_ctrl, uint32_t *const p_lines)

詳細説明

出力バッファにデコードされたライン数を返します。また電力消費を低減できます。

- R_JPEG_Decode_LinesDecodedGet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

表 441: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
p_lines	out	デコードされたライン数

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ p_lines

uint32_t

7.19.7.11 imageSizeGet

ssp_err_t(* jpeg_decode_api_t::imageSizeGet)(jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

詳細説明

デコード操作時に画像サイズを取得します。また電力消費を低減できます。

- R_JPEG_Decode_ImageSizeGet

l :JPEG コーデック モジュールが、適切に開かれている必要があります。

l : エンコード操作またはデコード操作がエラーなしで終了した場合、デバイスは HLD ドライバによって自動的に閉じられます。この場合、アプリケーションが、明示的に JPEG デバイスを閉じる必要はありません。

表 442: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。
p_horizontal_size	out	画像の幅 （ピクセル単位)。
p_vertical_size	out	画像の高さ （ピクセル単位)。

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ **p_horizontal_size**

uint16_t

パラメータ **p_vertical_size**

uint16_t

7.19.7.12 statusGet

ssp_err_t(* jpeg_decode_api_t::statusGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)

詳細説明

JPEG コーデック モジュールの現在のステータスを取得します。また電力消費を低減できます。

- R_JPEG_Decode_StatusGet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

表 443: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。

表 443: パラメータ (続き)

名前	方向	説明
p_status	out	JPEG モジュールのステータス

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ p_status

定義 : jpeg_decode_status_t*const p_status

JPEG HLD ドライバの内部ステータス情報

7.19.7.13 close

ssp_err_t(* jpeg_decode_api_t::close)(jpeg_decode_ctrl_t *const p_ctrl)

詳細説明

未処理の操作をキャンセルします。また電力消費を低減できます。

- R_JPEG_Decode_Close

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

I : エンコード操作またはデコード操作がエラーなしで終了した場合、デバイスは HLD ドライバによって自動的に閉じられます。この場合、アプリケーションが、明示的に JPEG デバイスを閉じる必要はありません。

表 444: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの jpeg_decode_api_t::Open 呼び出しで設定された制御ブロック。

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

7.19.7.14 versionGet

ssp_err_t(* jpeg_decode_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_JPEG_Decode_VersionGet

表 445: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.19.7.15 pixelFormatGet

ssp_err_t(* jpeg_decode_api_t::pixelFormatGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

詳細説明

入力ピクセル形式を取得します。また電力消費を低減できます。

- R_JPEG_Decode_PixelFormatGet

I :JPEG コーデック モジュールが、適切に開かれている必要があります。

表 446: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	open 呼び出しで設定された制御ブロック。

表 446: パラメータ (続き)

名前	方向	説明
p_color_space	out	JPEG 入力フォーマット。

定義 : jpeg_decode_ctrl_t

JPEG コーデック モジュール制御ブロック。初期化しないでください。初期化は、jpep_api_t::open の呼び出し時に実行されます。

パラメータ p_color_space

定義 : jpeg_decode_color_space_t*const p_color_space

画像の色空間の定義

7.19.7.16 jpeg_decode_instance_t

jpeg_decode_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- jpeg_decode_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- jpeg_decode_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- jpeg_decode_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.20 Key Matrix インタフェース

Key matrix 関数用のインタフェース。

7.20.1 概要

KEYMATRIX インタフェースは、同時に 1 つ以上のチャネルの立ち上がりエッジまたは立ち下がりエッジでのイベント生成を含む標準の KeyMatrix 機能を提供します。生成されたイベントは、ビット マスク経由でそのインスタンス内でアクティブになっているすべてのチャネルを示します。これにより、立ち上がりエッジと立ち下がりエッジのどちらかでトリガーされたマトリックス設定または 1 対 1 ハードウェア実装でインタフェースを使用できます。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

Key Matrix インタフェースの説明 : Key Matrix ドライバ

7.20.2 インタフェース API

keymatrix_api_t

関数名	説明
.open	初期設定。
.enable	キー割り込みを有効にします
.disable	キー割り込みを無効にします。
.triggerSet	キー割り込みのトリガーを設定します。
.close	ドライバを再設定できるようになります。電力消費を低減できます。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.20.3 データ構造体

- keymatrix_ctrl_t

- keymatrix_callback_args_t
- keymatrix_cfg_t
- keymatrix_instance_t

7.20.4 列挙

- keymatrix_trigger_t

7.20.5 型定義

- keymatrix_channels_t

7.20.6 定義

- #define KEYMATRIX_API_VERSION_MAJOR

  初期値 :(1U)

  KEY MATRIX API バージョン番号 （メジャー）
- #define KEYMATRIX_API_VERSION_MINOR

  初期値 :(0U)

  KEY MATRIX API バージョン番号 （マイナー）

7.20.7 API データ

7.20.7.1 keymatrix_trigger_t

keymatrix_trigger_t

詳細説明

トリガー タイプ：立ち上がり、立ち下がり

列挙値

名前	説明
KEYMATRIX_TRIG_FALLING	立ち下がりエッジ トリガー。
KEYMATRIX_TRIG_RISING	立ち上がりエッジ トリガー。

7.20.7.2 keymatrix_channels_t

typedef uint32_t keymatrix_channels_t

詳細説明

チャネル定義。これは、それぞれがチャネル 0 ～ 7 を表す 0 ～ 7 の各ビットで構成されたビット マスクです。

7.20.8 API 構造

7.20.8.1 keymatrix_ctrl_t

keymatrix_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- keymatrix_channels_t channels

7.20.8.2 keymatrix_callback_args_t

keymatrix_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- void const * p_context

  ユーザー データのホルダー。keymatrix_cfg_t 内の keymatrix_api_t::open 関数で設定されます。

- keymatrix_channels_t channels

  割り込みを発生させた物理ハードウェア チャネルを表すビット ベクター。このビット ベクターは、同時に複数の入力がアクティブになるマトリクス設計との互換性のために使用されます。HAL ドライバがすべてマトリクス モードに対応しているわけではありません。r_kint.h を参照してください。

7.20.8.3 keymatrix_cfg_t

keymatrix_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- keymatrix_channels_t channels

  キー入力チャネル。開くチャネルを指定するためのビット マスクです。

- keymatrix_trigger_t trigger

  キー入力のトリガー設定値。

- bool autostart

  open() の実行中に操作を開始し、割り込みを有効にします。

- void(* p_callback)(keymatrix_callback_args_t *p_args)

  キー割り込み ISR のコールバック。

- void const * p_context

  ユーザー データのホルダー。keymatrix_user_cb_data_t 内のコールバックに渡されます。

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.20.8.4 keymatrix_api_t

keymatrix_api_t

詳細説明

Key Matrix ドライバ構造体。HAL レイヤーに実装された Key Matrix 関数は、この API を使用します。

7.20.8.5 open

ssp_err_t(* keymatrix_api_t::open)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_KINT_KEYMATRIX_Open

表 447: パラメータ

名前	方向	説明
p_ctrl	out	制御ブロックへのポインタ。ユーザーが宣言する必要があります。値はこの関数で設定されます。

表 447: パラメータ (続き)

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : keymatrix_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : keymatrix_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- keymatrix_cfg_t::keymatrix_channels_t

  キー入力チャネル。開くチャネルを指定するためのビット マスクです。
- keymatrix_cfg_t::keymatrix_trigger_t

  キー入力のトリガー設定値。

  列挙値 :
  - KEYMATRIX_TRIG_FALLING
  - KEYMATRIX_TRIG_RISING
- keymatrix_cfg_t::autostart

  open() の実行中に操作を開始し、割り込みを有効にします。
- keymatrix_cfg_t::p_callback

  キー割り込み ISR のコールバック。
- keymatrix_cfg_t::p_context

  ユーザー データのホルダー。keymatrix_user_cb_data_t 内のコールバックに渡されます。
- keymatrix_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.20.8.6 enable

ssp_err_t(* keymatrix_api_t::enable)(keymatrix_ctrl_t *const p_ctrl)

詳細説明

Key 割り込みを有効にします。以下として実装されます。

- R_KINT_KEYMATRIX_Enable

表 448: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このキー割り込みの Open 呼び出しで設定された制御ブロック ポインタ。

定義 : keymatrix_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.20.8.7 disable

ssp_err_t(* keymatrix_api_t::disable)(keymatrix_ctrl_t *const p_ctrl)

詳細説明

キー割り込みを無効にします。また電力消費を低減できます。

- R_KINT_KEYMATRIX_Disable

表 449: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このキー割り込みの Open 呼び出しで設定された制御ブロック ポインタ。

定義 : keymatrix_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.20.8.8 triggerSet

ssp_err_t(* keymatrix_api_t::triggerSet)(keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t const trigger)

詳細説明

キー割り込みのトリガーを設定します。また電力消費を低減できます。

- R_KINT_KEYMATRIX_TriggerSet

表 450: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このキー割り込みの Open 呼び出しで設定された制御ブロック ポインタ。
trigger	複数のビットを書き換えることもできます。	キー割り込みのトリガ ソース。keymatrix_trigger_t の列挙値で定義されます。

定義 : keymatrix_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ trigger

7.20.8.9 close

ssp_err_t(* keymatrix_api_t::close)(keymatrix_ctrl_t *const p_ctrl)

詳細説明

ドライバを再設定できるようになります。電力消費を低減できます。また電力消費を低減できます。

- R_KINT_KEYMATRIX_Close

表 451: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このキー割り込みの Open 呼び出しで設定された制御ブロック ポインタ。

定義 : keymatrix_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.20.8.10 versionGet

ssp_err_t(* keymatrix_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_KINT_VersionGet

表 452: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.20.8.11 keymatrix_instance_t

keymatrix_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- keymatrix_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- keymatrix_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- keymatrix_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.21 低電力モードインタフェース

ローパワー モードにアクセスするためのインタフェースです。

7.21.1 概要

ここでは、LPM （ローパワー モード）ドライバ用 API を定義します。この LPM ドライバには、モジュールの停止、動作モードの選択、ローパワー モードの構成、ローパワー モードへの移行など、電力消費を制御する機能がいくつかあります。LPM ドライバは、LPM ハードウェア周辺機器を使用する MCU 動作モードと MCU ローパワーモードの構成をサポートしています。LPM ドライバは、動作モードの定電圧、低速、中速、高速、および副発振器モードをサポートしています。また、LPM ドライバは、ディープ スタンバイ、スタンバイ、スリープ、およびスヌーズの低電力モードをサポートしています。LPM ドライバは、ディープ スタンバイ モード時、内部の電力供給制御と IO ポートのステータス再設定を通して電力消費の低減をサポートします。さらに、LPM ドライバは、MCU のその他のハードウェア周辺機器の無効化と有効化をサポートしています。
I：すべての MCU で、すべての動作モードが利用可能とは限りません。すべての MCU で、すべてのローパワー モードが利用可能とは限りません。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

LPM インタフェースの説明：ローパワー モードドライバ

7.21.2 インタフェース API

lpm_api_t

関数名	説明
.init	LPM ドライバ モジュールを開き、設定構造体に渡された値に従って、LPM ブロックを初期化します
.mstpcrSet	すべてのモジュール ストップ コントロール レジスタの値を設定します
.mstpcrGet	すべてのモジュール ストップ コントロール レジスタの値を取得します
.moduleStop	モジュールを停止します

関数名	説明
.moduleStart	モジュールを実行します
.operatingPowerModeSet	動作電力モードを設定します
.snoozeEnable	スヌーズ モードを設定し、有効にします
.snoozeDisable	スヌーズ モードを無効にします
.lowPowerCfg	低電力モードを設定します
.wupenSet	ウェイクアップ割り込み有効レジスタ WUPEN の値を設定します
.wupenGet	ウェイクアップ割り込み有効レジスタ WUPEN の値を取得します
.deepStandbyCancelRequestEnable	ディープ スタンバイのキャンセル要求を有効にします
.deepStandbyCancelRequestDisable	ディープ スタンバイのキャンセル要求を無効にします
.lowPowerModeEnter	WFI マクロを使用して、ローパワー モード （スリープ / スタンバイ / ディープ スタンバイ）に移行します。ローパワー モードが解除されると、関数が復帰します。
.versionGet	コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

7.21.3 データ構造体

- lpm_cfg_t
- lpm_instance_t

7.21.4 列挙

- lpm_cancel_request_edge_t
- lpm_deep_standby_t
- lpm_low_power_mode_t
- lpm_output_port_enable_t
- lpm_dpsby_t
- lpm_io_port_t

- lpm_power_supply_t
- lpm_power_save_memory_t
- lpm_code_flash_t
- lpm_snooze_request_t
- lpm_snooze_end_t
- lpm_snooze_rxd0_t
- lpm_snooze_dtc_t
- lpm_operating_power_t
- lpm_subosc_t
- lpm_mstp_t

7.21.5 定義

- #define LPM_API_VERSION_MAJOR

  初期値 :(2)

  定義、共通サービス、およびエラー コードを登録します。
- #define LPM_API_VERSION_MINOR

  初期値 :(1)

7.21.6 API データ

7.21.6.1 lpm_cancel_request_edge_t

lpm_cancel_request_edge_t

詳細説明

ディープ スタンバイの割り込みエッジ

列挙値

名前	説明
LPM_CANCEL_REQUEST_EDGE_FALLING	立ち下がりエッジでキャンセル要求が生成されます。
LPM_CANCEL_REQUEST_EDGE_RISING	立ち上がりエッジでキャンセル要求が生成されます。

7.21.6.2 lpm_deep_standby_t

lpm_deep_standby_t

詳細説明

ディープ スタンバイのピンおよび信号

列挙値

名前	説明
LPM_DEEP_STANDBY_IRQ0_DS	IRQ0-DS ピン。
LPM_DEEP_STANDBY_IRQ1_DS	IRQ1-DS ピン。
LPM_DEEP_STANDBY_IRQ2_DS	IRQ2-DS ピン。
LPM_DEEP_STANDBY_IRQ3_DS	IRQ3-DS ピン。
LPM_DEEP_STANDBY_IRQ4_DS	IRQ4-DS ピン。
LPM_DEEP_STANDBY_IRQ5_DS	IRQ5-DS ピン。
LPM_DEEP_STANDBY_IRQ6_DS	IRQ6-DS ピン。
LPM_DEEP_STANDBY_IRQ7_DS	IRQ7-DS ピン。
LPM_DEEP_STANDBY_IRQ8_DS	IRQ8-DS ピン。
LPM_DEEP_STANDBY_IRQ9_DS	IRQ9-DS ピン。
LPM_DEEP_STANDBY_IRQ10_DS	IRQ10-DS ピン。
LPM_DEEP_STANDBY_IRQ11_DS	IRQ11-DS ピン。
LPM_DEEP_STANDBY_IRQ12_DS	IRQ12-DS ピン。
LPM_DEEP_STANDBY_IRQ13_DS	IRQ13-DS ピン。
LPM_DEEP_STANDBY_IRQ14_DS	IRQ14-DS ピン。
LPM_DEEP_STANDBY_IRQ15_DS	IRQ15-DS ピン。
LPM_DEEP_STANDBY_LVD1	LVD1。
LPM_DEEP_STANDBY_LVD2	LVD2。

名前	説明
LPM_DEEP_STANDBY_RTC_INTERVAL	RTC インターバル割り込み。
LPM_DEEP_STANDBY_RTC_ALARM	RTC アラーム割り込み。
LPM_DEEP_STANDBY_NMI	NMI ピン。
LPM_DEEP_STANDBY_USBFS	USBFS サスペンド / 再開。
LPM_DEEP_STANDBY_USBHS	USBHS サスペンド / 再開。
LPM_DEEP_STANDBY_AGT1	AGT1 アンダーフロー。

7.21.6.3 lpm_low_power_mode_t

lpm_low_power_mode_t

詳細説明

低電力モード

列挙値

名前	説明
LPM_LOW_POWER_MODE_SLEEP	スリープ モード （ARM Cortex スリープ モード）。
LPM_LOW_POWER_MODE_STANDBY	ソフトウェア スタンバイ モード。
LPM_LOW_POWER_MODE_DEEP	ディープ ソフトウェア スタンバイ モード。

7.21.6.4 lpm_output_port_enable_t

lpm_output_port_enable_t

詳細説明

出力ポート イネーブル

列挙値

名前	説明
LPM_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: ソフトウェア スタンバイ モードまたはディープ ソフトウェア スタンバイ モードで、アドレス出力ピン、データ出力ピン、およびその他のバスコントロール信号出力ピンが高インピーダンス状態に設定されています。スヌーズでは、アドレス バス信号とバスコントロール信号のステータスが、ソフトウェア スタンバイ モードに入る前と同じです。
LPM_OUTPUT_PORT_ENABLE_RETAIN	1: ソフトウェア スタンバイ モードで、アドレス出力ピン、データ出力ピン、およびその他のバスコントロール信号出力ピンが出力ステータスを保持しています。

7.21.6.5 lpm_dpsby_t

lpm_dpsby_t

詳細説明

ソフトウェア スタンバイとディープ ソフトウェア スタンバイを切り替えます。

列挙値

名前	説明
LPM_DPSBY_SOFTWARE_STANDBY	スリープ モード （SBYCR.SSBY = 0）/ ソフトウェア スタンバイ モード （SBYCR.SSBY = 1）
LPM_DPSBY_DEEP_SOFTWARE_STANDBY	スリープ モード （SBYCR.SSBY = 0）/ ディープ ソフトウェア スタンバイ モード （SBYCR.SSBY = 1）

7.21.6.6 lpm_io_port_t

lpm_io_port_t

詳細説明

ディープ ソフトウェア スタンバイ モード移行後の I/O ポートの状態

列挙値

名前	説明
LPM_IO_PORT_RESET	ディープ ソフトウェア スタンバイ モードがキャンセルされたときに、I/O ポートがリセット状態になります。
LPM_IO_PORT_NO_CHANGE	ディープ ソフトウェア スタンバイ モードがキャンセルされたとき、I/O ポートはディープ ソフトウェア スタンバイ モードと同じ状態です

7.21.6.7 lpm_power_supply_t

lpm_power_supply_t

詳細説明

電力供給制御

列挙値

名前	説明
LPM_POWER_SUPPLY_DEEPCUT0	スタンバイ RAM、低速オンチップ発振器、AGTn、および USBFS/HS 再開検知ユニットへの電力は、ディープ ソフトウェア スタンバイ モードで供給されます
LPM_POWER_SUPPLY_DEEPCUT1	スタンバイ RAM、低速オンチップ発振器、AGTn、および USBFS/HS 再開検知ユニットへの電力は、ディープ ソフトウェア スタンバイ モードで供給されません
LPM_POWER_SUPPLY_DEEPCUT3	スタンバイ RAM、低速オンチップ発振器、AGTn、および USBFS/HS 再開検知ユニットへの電力は、ディープ ソフトウェア スタンバイ モードで供給されません。さらに、LVD が無効化され、パワーオン リセット回路のローパワー機能が有効化されています

7.21.6.8 lpm_power_save_memory_t

lpm_power_save_memory_t

詳細説明

省電力メモリ制御

列挙値

名前	説明
LPM_POWER_SAVE_MEMORY_ALL_RAM	ソフトウェア スタンバイ モードですべての RAM をオンにします。
LPM_POWER_SAVE_MEMORY_48KB	ソフトウェア スタンバイ モードで 48KB RAM （2000 0000h ～ 2000 BFFFh）をオンにします。

7.21.6.9 lpm_code_flash_t

lpm_code_flash_t

詳細説明

コード フラッシュ操作モード

列挙値

名前	説明
LPM_CODE_FLASH_OPERATES	コード フラッシュ メモリが作動します。
LPM_CODE_FLASH_STOPS	コード フラッシュ メモリが停止します。

7.21.6.10 lpm_snooze_request_t

lpm_snooze_request_t

詳細説明

スヌーズ終了コントロール

列挙値

名前	説明
LPM_SNOOZE_REQUEST_RXD0_FALLING	RXD0 立ち下がりエッジ スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ0	IRQ0 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ1	IRQ1 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ2	IRQ2 ピン スヌーズ要求を有効にします。

名前	説明
LPM_SNOOZE_REQUEST_IRQ3	IRQ3 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ4	IRQ4 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ5	IRQ5 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ6	IRQ6 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ7	IRQ7 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ8	IRQ8 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ9	IRQ9 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ10	IRQ10 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ11	IRQ11 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ12	IRQ12 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ13	IRQ13 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ14	IRQ14 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_IRQ15	IRQ15 ピン スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_KR	KR スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_COMPARATOR_OC0	コンパレーター OC0 スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_COMPARATOR_LP	コンパレーター LP スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_RTC_ALARM	RTC アラーム スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_RTC_PERIOD	RTC 期間スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW	AGT1 アンダープロー スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_AGT1_COMPARE_A	AGT1 比較一致 A スヌーズ要求を有効にします。
LPM_SNOOZE_REQUEST_AGT1_COMPARE_B	AGT1 比較一致 B スヌーズ要求を有効にします。

7.21.6.11 lpm_snooze_end_t

lpm_snooze_end_t

詳細説明

スヌーズ終了コントロール

列挙値

名前	説明
LPM_SNOOZE_END_VIA_WUPEN	スヌーズからノーマル モードに直接移行します。
LPM_SNOOZE_END_AGT1_UNDERFLOW	AGT1 アンダーフロー。
LPM_SNOOZE_END_DTC_TRANS_COMPLETE	最後の DTC 送信完了。
LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	最後以外の DTC 送信完了。
LPM_SNOOZE_END_ADC0_COMPARE_MATCH	AD チャネル 0 比較一致。
LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH	AD チャネル 0 比較不一致。
LPM_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 比較一致。
LPM_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 比較不一致。
LPM_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 アドレス不一致。

7.21.6.12 lpm_snooze_rxd0_t

lpm_snooze_rxd0_t

詳細説明

RXD0 のスヌーズ要求を有効にします。

列挙値

名前	説明
LPM_SNOOZE_RXD0_FALLING_EDGE_IGNORE	ソフトウェア スタンバイ モードで RXD 立ち下がりエッジを無視します。
LPM_SNOOZE_RXD0_FALLING_EDGE_DETECT	スヌーズ モードへの移行要求に応じて、ソフトウェア スタンバイ モードで RXD0 立ち下がりエッジを検出します。調歩同期式モードでない場合は 1 に設定しないでください。

7.21.6.13 lpm_snooze_dtc_t

lpm_snooze_dtc_t

詳細説明

スヌーズ モードで、DTC を有効にします。

列挙値

名前	説明
LPM_SNOOZE_DTC_DISABLE	DTC 操作を無効にします。
LPM_SNOOZE_DTC_ENABLE	DTC 操作を有効にします。

7.21.6.14 lpm_operating_power_t

lpm_operating_power_t

詳細説明

動作電力モード

列挙値

名前	説明
LPM_OPERATING_POWER_HIGH_SPEED_MODE	高速モード。
LPM_OPERATING_POWER_MIDDLE_SPEED_MODE	中速モード。
LPM_OPERATING_POWER_LOW_VOLTAGE_MODE	低電圧モード。
LPM_OPERATING_POWER_LOW_SPEED_MODE	低速モード。

7.21.6.15 lpm_subosc_t

lpm_subosc_t

詳細説明

サブオシレーター速度モード選択

列挙値

名前	説明
LPM_SUBOSC_OTHER	非サブオシレーター速度モード。
LPM_SUBOSC_SELECT	サブ発振器速度モード。

7.21.6.16 lpm_mstp_t

lpm_mstp_t

詳細説明

モジュールのストップ ビット

列挙値

名前	説明
LPM_MSTP_RAM0	RAM0。
LPM_MSTP_RAM1	RAM1。
LPM_MSTP_HSRAM	HSRAM。
LPM_MSTP_ECCRAM	ECCRAM。
LPM_MSTP_STANDBYRAM	STANDBYRAM。
LPM_MSTP_DMAC_DTC	DMAC_DTC。
LPM_MSTP_RCAN1	RCAN1。
LPM_MSTP_RCAN0	RCAN0。
LPM_MSTP_IRDA	IRDA。
LPM_MSTP_QSPI	QSPI。
LPM_MSTP_I2C2	I^2C2。
LPM_MSTP_I2C1	I^2C1。
LPM_MSTP_I2C0	I^2C0。
LPM_MSTP_USBFS	USBFS。

名前	説明
LPM_MSTP_USBHS	USBHS。
LPM_MSTP_EPTPC_PTPEDMAC	EPTPC_PTPEDMAC。
LPM_MSTP_ETHERC1_EDMAC1	ETHERC1_EDMAC1。
LPM_MSTP_ETHERC0_EDMAC0	ETHERC0_EDMAC0。
LPM_MSTP_RSPI1	RSPI1。
LPM_MSTP_RSPI0	RSPI0。
LPM_MSTP_SCI9	SCI9。
LPM_MSTP_SCI8	SCI8。
LPM_MSTP_SCI7	SCI7。
LPM_MSTP_SCI6	SCI6。
LPM_MSTP_SCI5	SCI5。
LPM_MSTP_SCI4	SCI4。
LPM_MSTP_SCI3	SCI3。
LPM_MSTP_SCI2	SCI2。
LPM_MSTP_SCI1	SCI1。
LPM_MSTP_SCI0	SCI0。
LPM_MSTP_CAC	CAC。
LPM_MSTP_CRC	CRC。
LPM_MSTP_PDC	PDC。
LPM_MSTP_CTSU	CTSU。
LPM_MSTP_GLCDC	GLCDC。
LPM_MSTP_JPEG	JPEG。
LPM_MSTP_2GD	2GD。

参考資料

名前	説明
LPM_MSTP_SSI1	SSI1。
LPM_MSTP_SSI0	SSI0。
LPM_MSTP_SRC	SRC。
LPM_MSTP_SDH_MMC1	SDH_MMC1。
LPM_MSTP_SDH_MMC0	SDH_MMC0。
LPM_MSTP_DOC	DOC。
LPM_MSTP_ELC	ELC。
LPM_MSTP_TSIP	TSIP。
LPM_MSTP_AGT1	AGT1。
LPM_MSTP_AGT0	AGT0。
LPM_MSTP_GPT_CH7_0	GPT_CH7_0。
LPM_MSTP_GPT_CH13_8	GPT_CH13_8。
LPM_MSTP_PGI	PGI。
LPM_MSTP_S12AD_UNIT1	S12AD_UNIT1。
LPM_MSTP_S12AD_UNIT0	S12AD_UNIT0。
LPM_MSTP_D_A0	D_A0。
LPM_MSTP_TEMP_SENSE	TEMP_SENSE。
LPM_MSTP_COMP_OC5	COMP_OC5。
LPM_MSTP_COMP_OC4	COMP_OC4。
LPM_MSTP_COMP_OC3	COMP_OC3。
LPM_MSTP_COMP_OC2	COMP_OC2。
LPM_MSTP_COMP_OC1	COMP_OC1。
LPM_MSTP_COMP_OC0	COMP_OC0。

名前	説明
LPM_MSTP_COMP_LP	COMP_LP。
LPM_MSTP_COMP_RD	COMP_RD。
LPM_MSTP_COMP_OPAMP	COMP_OPAMP。

7.21.7 API 構造

7.21.7.1 lpm_cfg_t

lpm_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- lpm_operating_power_t operating_power
  動作電力モード。
- lpm_subosc_t sub_oscillator
  サブオシレーター。
- lpm_code_flash_t code_flash
  コード フラッシュを有効にします。

7.21.7.2 lpm_api_t

lpm_api_t

詳細説明

lpm ドライバ構造体。HAL レイヤーに実装された汎用 lpm 関数は、この API に従います。

7.21.7.3 init

ssp_err_t(* `lpm_api_t`::init)(`lpm_cfg_t` const *const p_cfg)

詳細説明

LPM ドライバ モジュールを開き、設定構造体に渡された値に従って、LPM ブロックを初期化します

表 453: パラメータ

名前	方向	説明
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 :

定義 : lpm_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- lpm_cfg_t::lpm_operating_power_t

  動作電力モード。

  列挙値 :

  – LPM_OPERATING_POWER_HIGH_SPEED_MODE

  – LPM_OPERATING_POWER_MIDDLE_SPEED_MODE

  – LPM_OPERATING_POWER_LOW_VOLTAGE_MODE

  – LPM_OPERATING_POWER_LOW_SPEED_MODE

- lpm_cfg_t::lpm_subosc_t

  サブオシレーター。

  列挙値 :

  – LPM_SUBOSC_OTHER

  – LPM_SUBOSC_SELECT

- lpm_cfg_t::lpm_code_flash_t

  コード フラッシュを有効にします。

  列挙値 :

  – LPM_CODE_FLASH_OPERATES

  – LPM_CODE_FLASH_STOPS

7.21.7.4 mstpcrSet

ssp_err_t(* lpm_api_t::mstpcrSet)(uint32_t mstpcra_value, uint32_t mstpcrb_value, uint32_t mstpcrc_value, uint32_t mstpcrd_value)

詳細説明

すべてのモジュール ストップ コントロール レジスタの値を設定します

表 454: パラメータ

名前	方向	説明
mstpcra_value	複数のビットを書き換えることもできます。	MSTPCRA に設定する値
mstpcrb_value	複数のビットを書き換えることもできます。	MSTPCRB に設定する値
mstpcrc_value	複数のビットを書き換えることもできます。	MSTPCRC に設定する値
mstpcrd_value	複数のビットを書き換えることもできます。	MSTPCRD に設定する値

パラメータ **mstpcra_value**

uint32_t

パラメータ **mstpcrb_value**

uint32_t

パラメータ **mstpcrc_value**

uint32_t

パラメータ **mstpcrd_value**

uint32_t

7.21.7.5 mstpcrGet

ssp_err_t(* lpm_api_t::mstpcrGet)(uint32_t *mstpcra_value, uint32_t *mstpcrb_value, uint32_t *mstpcrc_value, uint32_t *mstpcrd_value)

詳細説明

すべてのモジュール ストップ コントロール レジスタの値を取得します

表 455: パラメータ

名前	方向	説明
mstpcra_value	入力 / 出力	MSTPCRA の値
mstpcrb_value	入力 / 出力	MSTPCRB の値

表 455: パラメータ (続き)

名前	方向	説明
mstpcrc_value	入力 / 出力	MSTPCRC の値
mstpcrd_value	入力 / 出力	MSTPCRD の値

パラメータ **mstpcra_value**

uint32_t

パラメータ **mstpcrb_value**

uint32_t

パラメータ **mstpcrc_value**

uint32_t

パラメータ **mstpcrd_value**

uint32_t

7.21.7.6 moduleStop

ssp_err_t(* lpm_api_t::moduleStop)(lpm_mstp_t module)

詳細説明

モジュールを停止します

表 456: パラメータ

名前	方向	説明
module	複数のビットを書き換えることもできます。	状態を設定するモジュール

パラメータ **module**

定義 : lpm_mstp_tmodule

モジュールのストップ ビット

7.21.7.7 moduleStart

ssp_err_t(* lpm_api_t::moduleStart)(lpm_mstp_t module)

詳細説明

モジュールを実行します

表 457: パラメータ

名前	方向	説明
module	複数のビットを書き換えることもできます。	状態を設定するモジュール

パラメータ module

定義 : lpm_mstp_tmodule

モジュールのストップ ビット

7.21.7.8 operatingPowerModeSet

ssp_err_t(* lpm_api_t::operatingPowerModeSet)(lpm_operating_power_t power_mode, lpm_subosc_t subosc)

詳細説明

動作電力モードを設定します

表 458: パラメータ

名前	方向	説明
power_mode	複数のビットを書き換えることもできます。	チップに設定する電力モード
subosc	複数のビットを書き換えることもできます。	サブオシレーターまたは他のオシレーターを選択します

パラメータ power_mode

定義 : lpm_operating_power_tpower_mode

動作電力モード

パラメータ subosc

7.21.7.9 snoozeEnable

ssp_err_t(* lpm_api_t::snoozeEnable)(lpm_snooze_rxd0_t rdx0_mode, lpm_snooze_dtc_t dtc_mode, lpm_snooze_request_t requests, lpm_snooze_end_t triggers)

詳細説明

スヌーズ モードを設定し、有効にします

表 459: パラメータ

名前	方向	説明
rdx0_mode	複数のビットを書き換えることもできます。	RXD0 の立ち下がり エッジでのみスヌーズ モードに移行します （非同期モードのみ）
dtc_mode	複数のビットを書き換えることもできます。	スヌーズ モードの間、DTC および RAM を使用します
requests	複数のビットを書き換えることもできます。	スヌーズ モードに移行させるイベントを指定します
triggers	複数のビットを書き換えることもできます。	スヌーズ モードを終了させるイベントを指定します

パラメータ rdx0_mode

定義 : lpm_snooze_rxd0_trdx0_mode

RXD0 のスヌーズ要求を有効にします

パラメータ dtc_mode

定義 : lpm_snooze_dtc_tdtc_mode

スヌーズ モードで、DTC を有効にします

パラメータ requests

定義 : lpm_snooze_request_trequests

スヌーズ終了コントロール

パラメータ triggers

定義 : lpm_snooze_end_ttriggers

スヌーズ終了コントロール

7.21.7.10 snoozeDisable

ssp_err_t(* lpm_api_t::snoozeDisable)(void)

詳細説明

スヌーズ モードを無効にします

表 460: 戻り値

名前	説明
SSP_SUCCESS	スヌーズ モードが無効になりました

パラメータ **SSP_SUCCESS**

7.21.7.11 lowPowerCfg

ssp_err_t(* lpm_api_t::lowPowerCfg)(lpm_low_power_mode_t power_mode, lpm_output_port_enable_t output_port_enable, lpm_power_supply_t power_supply, lpm_io_port_t io_port_state)

詳細説明

低電力モードを設定します

表 461: パラメータ

名前	方向	説明
low_power_mode	複数のビットを書き換えることもできます。	移行後の低電力モードを指定します
output_port_enable	複数のビットを書き換えることもできます。	ポートの出力ステータスをスリープまたはスタンバイに維持します
power_supply	複数のビットを書き換えることもできます。	ディープ ソフトウェア スタンバイで、引き続き電力を供給する対象を指定します
io_port_state	複数のビットを書き換えることもできます。	ディープ ソフトウェア スタンバイ モード移行後の I/O ポートの状態

パラメータ **low_power_mode**

パラメータ **output_port_enable**

パラメータ **power_supply**

パラメータ **io_port_state**

定義 : lpm_io_port_tio_port_state

ディープ ソフトウェア スタンバイ モード移行後の I/O ポートの状態

7.21.7.12 wupenSet

ssp_err_t(* lpm_api_t::wupenSet)(uint32_t wupen_value)

詳細説明

ウェイクアップ割り込み有効レジスタ WUPEN の値を設定します

表 462: パラメータ

名前	方向	説明
wupen_value	複数のビットを書き換えることもできます。	WUPEN に設定する値

Parameter wupen_value

uint32_t

7.21.7.13 wupenGet

ssp_err_t(* lpm_api_t::wupenGet)(uint32_t *wupen_value)

詳細説明

ウェイクアップ割り込み有効レジスタ WUPEN の値を取得します

表 463: パラメータ

名前	方向	説明
wupen_value	入力 / 出力	WUPEN の値

Parameter wupen_value

uint32_t

7.21.7.14 deepStandbyCancelRequestEnable

ssp_err_t(* lpm_api_t::deepStandbyCancelRequestEnable)(lpm_deep_standby_t pin_signal, lpm_cancel_request_edge_t rising_falling)

詳細説明

ディープ スタンバイのキャンセル要求を有効にします

表 464: パラメータ

名前	方向	説明
pin_signal	複数のビットを書き換えることもできます。	ディープ スタンバイ モードに関連付けられたピンまたは信号
rising_falling	複数のビットを書き換えることもできます。	キャンセル要求を発生させるエッジ (RTC 割り込みでは無視されます)

パラメータ pin_signal

定義 : lpm_deep_standby_tpin_signal

ディープ スタンバイのピンおよび信号

パラメータ rising_falling

定義 : lpm_cancel_request_edge_trising_falling

ディープ スタンバイの割り込みエッジ

7.21.7.15 deepStandbyCancelRequestDisable

ssp_err_t(* lpm_api_t::deepStandbyCancelRequestDisable)(lpm_deep_standby_t pin_signal)

詳細説明

ディープ スタンバイのキャンセル要求を無効にします

表 465: パラメータ

名前	方向	説明
pin_signal	複数のビットを書き換えることもできます。	ディープ スタンバイ モードに関連付けられたピンまたは信号

パラメータ pin_signal

定義 : lpm_deep_standby_tpin_signal

ディープ スタンバイのピンおよび信号

7.21.7.16 lowPowerModeEnter

ssp_err_t(* lpm_api_t::lowPowerModeEnter)(void)

詳細説明

WFI マクロを使用して、ローパワー モード （スリープ / スタンバイ / ディープ スタンバイ）に移行します。ローパワー モードが解除されると、関数が復帰します。

7.21.7.17 versionGet

ssp_err_t(* lpm_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

表 466: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
SSP_ERR_INVALID_PTR	p_version が NULL です。

パラメータ SSP_SUCCESS

パラメータ SSP_ERR_INVALID_PTR

7.21.7.18 lpm_instance_t

lpm_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- lpm_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- lpm_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.22 低電圧検出ドライバインタフェース

ここでは、LVD （低電圧検出）ドライバ用 API を定義します。

LVD ドライバは、LVD ハードウェア周辺機器を構成するための機能を提供します。

低電圧検出監視を構成 / 有効化するプロセスは、具体的な時間的制約とレジスタの書き込み順序に拘束されます。このような制約があるため、電圧監視を設定 / 有効化するプロセスは、全体を単一の関数で処理するのが最も効率的です。API 関数構成は、タイミングとレジスタの書き込み順序の制約を正しく強制するように、構成を実行し、監視を有効にします。

LVD ドライバは、用意されている構成可能な LVD 監視のすべての設定を構成します。

設定には以下が含まれます :

- 電圧しきい値 : 電圧検出のしきい値を決定します （2.99 ボルト）。
- sample_clock_divisor: LOCO クロックの分周に基づきデジタル フィルタのサンプル クロック レートを決定します。また、MCU で利用可能な場合はデジタル フィルタを有効化 / 無効化します。
- detection_response: 電圧しきい値を超えた場合、どのイベントが発生またはリセット、割り込み、マスク不可能な割り込みの対象となるか、または何のレスポンスも行われないかを決定します
- voltage_slope: 電圧検出トリガの割り込みが発生した場合、電圧を上昇させるか、下降させるかを選択します。
- negation_delay: 電圧イベントのネゲートのタイミングをリセット イベント ベースにするか、あるいは電圧イベント ベースにするか決定します。
- p_callback: 電圧イベントの割り込みが発生した場合に呼び出される、ユーザー定義関数のアドレス。

I : 低電圧監視 0 （LVD0）は実行時には構成できませんが、e^2 studio ISDE の Synergy プロジェクト コンフィギュレータにある [BSP Properties] タブで OFS1 レジスタ値を変更することで構成可能です。

I : スタンバイ モードではデジタル フィルタは使用できません。ソフトウェアのスタンバイまたはディープスタンバイ モードを使用する場合、デジタル フィルタは無効化しておく必要があります。

ドライバ関数の実装の詳細については、「LVD」セクションを参照してください。

7.22.1 インタフェース API

lvd_api_t

関数名	説明
.open	渡された構成構造体に従って、低電圧検出ドライバを初期化します。構成構造体に従い、LVD 周辺機器を有効化します。
.statusGet	監視の現在の状態を取得します （しきい値の超過が検出された、現在電圧が範囲内にある）。LVD 監視の状態のポーリングにいつでも使用できます。lvd_response_t を LVD_RESPONSE_NONE に設定した状態で周辺機器を初期化した場合は、必ず使用する必要があります。
.statusClear	監視のラッチ状態をクリアします。lvd_response_t を LVD_RESPONSE_NONE に設定した状態で周辺機器を初期化した場合は、必ず使用する必要があります。
.close	LVD 周辺機器を無効化します。ドライバ インスタンスを閉じます。
.versionGet	コンパイル時マクロに基づいて、LVD ドライバのバージョンを返します。

7.22.2 データ構造体

- lvd_status_t
- lvd_callback_args_t
- lvd_cfg_t
- lvd_ctrl_t
- lvd_instance_t

7.22.3 列挙

- lvd_threshold_t
- lvd_response_t
- lvd_voltage_slope_t
- lvd_threshold_crossing_t
- lvd_current_state_t

7.22.4 定義

- #define LVD_API_VERSION_MAJOR

  初期値 :(01U)

  定義、共通サービス、およびエラー コードを登録します。

- #define LVD_API_VERSION_MINOR

  初期値 :(00U)

7.22.5 API データ

7.22.5.1 lvd_threshold_t

lvd_threshold_t

詳細説明

電圧検出レベル

列挙値

名前	説明
LVD_THRESHOLD_MONITOR_1_LEVEL_0	4.29V （Vdet1_0）
LVD_THRESHOLD_MONITOR_1_LEVEL_1	4.14V （Vdet1_1）
LVD_THRESHOLD_MONITOR_1_LEVEL_2	4.02V （Vdet1_2）
LVD_THRESHOLD_MONITOR_1_LEVEL_3	3.84V （Vdet1_3）
LVD_THRESHOLD_MONITOR_1_LEVEL_4	3.10V （Vdet1_4）
LVD_THRESHOLD_MONITOR_1_LEVEL_5	3.00V （Vdet1_5）
LVD_THRESHOLD_MONITOR_1_LEVEL_6	2.90V （Vdet1_6）
LVD_THRESHOLD_MONITOR_1_LEVEL_7	2.79V （Vdet1_7）
LVD_THRESHOLD_MONITOR_1_LEVEL_8	2.68V （Vdet1_8）
LVD_THRESHOLD_MONITOR_1_LEVEL_9	2.58V （Vdet1_9）
LVD_THRESHOLD_MONITOR_1_LEVEL_A	2.48V （Vdet1_A）

名前	説明
LVD_THRESHOLD_MONITOR_1_LEVEL_B	2.20V （Vdet1_B）
LVD_THRESHOLD_MONITOR_1_LEVEL_C	1.96V （Vdet1_C）
LVD_THRESHOLD_MONITOR_1_LEVEL_D	1.86V （Vdet1_D）
LVD_THRESHOLD_MONITOR_1_LEVEL_E	1.75V （Vdet1_E）
LVD_THRESHOLD_MONITOR_1_LEVEL_F	1.65V （Vdet1_F）
LVD_THRESHOLD_MONITOR_1_LEVEL_11	2.99V （Vdet1_11）
LVD_THRESHOLD_MONITOR_1_LEVEL_12	2.92V （Vdet1_12）
LVD_THRESHOLD_MONITOR_1_LEVEL_13	2.85V （Vdet1_13）
LVD_THRESHOLD_MONITOR_2_LEVEL_0	4.29V （Vdet2_0）
LVD_THRESHOLD_MONITOR_2_LEVEL_1	4.14V （Vdet2_1）
LVD_THRESHOLD_MONITOR_2_LEVEL_2	4.02V （Vdet2_2）
LVD_THRESHOLD_MONITOR_2_LEVEL_3	3.84V （Vdet2_3）
LVD_THRESHOLD_MONITOR_2_LEVEL_5	2.99V （Vdet2_5）
LVD_THRESHOLD_MONITOR_2_LEVEL_6	2.92V （Vdet2_6）
LVD_THRESHOLD_MONITOR_2_LEVEL_7	2.85V （Vdet2_7）

7.22.5.2 lvd_response_t

lvd_response_t

詳細説明

しきい値の超過イベント、割り込み、リセット、NMI... に対するレスポンス

列挙値

名前	説明
LVD_RESPONSE_NMI	マスク不可能な割り込み。

名前	説明
LVD_RESPONSE_INTERRUPT	マスク可能な割り込み。
LVD_RESPONSE_RESET	リセット。
LVD_RESPONSE_NONE	レスポンスなし、ステータスは statusGet 関数を使って要求する必要があります。

7.22.5.3 lvd_voltage_slope_t

lvd_voltage_slope_t

詳細説明

電圧スロープ、上昇または下降、あるいは両方

列挙値

名前	説明
LVD_VOLTAGE_SLOPE_RISING	VCC >= Vdet2 （上昇）が検出された場合。
LVD_VOLTAGE_SLOPE_FALLING	VCC < Vdet2 （下降）が検出された場合。
LVD_VOLTAGE_SLOPE_BOTH	上昇と下降が検出された場合。

7.22.5.4 lvd_threshold_crossing_t

lvd_threshold_crossing_t

詳細説明

しきい値超過検出 （ラッチ）

列挙値

名前	説明
LVD_THRESHOLD_CROSSING_NOT_DETECTED	しきい値超過が検出されませんでした。
LVD_THRESHOLD_CROSSING_DETECTED	しきい値超過が検出されました。

7.22.5.5 lvd_current_state_t

lvd_current_state_t

詳細説明

VCC の瞬間的なステータス （しきい値を上回るまたは下回る）

列挙値

名前	説明
LVD_CURRENT_STATE_BELOW_THRESHOLD	VCC < のしきい値。
LVD_CURRENT_STATE_ABOVE_THRESHOLD	VCC >= しきい値または監視が無効化されています。

7.22.6 API 構造

7.22.6.1 lvd_status_t

lvd_status_t

詳細説明

電圧監視ステータス構造体、statusGet 関数および p_callback とともに使用し監視の現在の状態を取得します （しきい値の超過が検出された、現在 VCC が範囲内にある）。

変数

- lvd_threshold_crossing_t crossing_detected

  しきい値超過検出 （ラッチ）

- lvd_current_state_t current_state

  監視対象となる電圧の瞬間的なステータス （しきい値を上回るまたは下回る）

7.22.6.2 lvd_callback_args_t

lvd_callback_args_t

詳細説明

LVD コールバック パラメータ定義

変数

- const uint32_t monitor_number

  監視番号。

- lvd_status_t status

  監視のステータス。

- void const * p_context

  ユーザー データのプレースホルダー。

7.22.6.3 lvd_cfg_t

lvd_cfg_t

詳細説明

LVD 構成構造体

変数

- const uint32_t monitor_number

  監視番号 1、2、…

- lvd_threshold_t voltage_threshold

  範囲外の電圧検出しきい値

- lvd_response_t detection_response

  しきい値調査検出に対するレスポンス

- lvd_voltage_slope_t voltage_slope

  電圧の上昇または下降が検出されます

- void(* p_callback)(lvd_callback_args_t *p_args)

  割り込みから呼び出されるユーザー関数

- void const * p_context

  ユーザー データのプレースホルダー。以下に含まれるユーザー コールバックに渡されます :

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータ

7.22.6.4 lvd_ctrl_t

lvd_ctrl_t

詳細説明

LVD 制御構造体

変数

- uint32_t monitor_number

  監視番号 1、2、…

7.22.6.5 lvd_api_t

lvd_api_t

詳細説明

LVD ドライバ API 構造体。HAL レイヤーに実装された LVD ドライバ関数は、この API に従います。

7.22.6.6 open

ssp_err_t(* lvd_api_t::open)(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)

詳細説明

渡された構成構造体に従って、低電圧検出ドライバを初期化します。構成構造体に従い、LVD 周辺機器を有効化します。また電力消費を低減できます。

- R_LVD_Open

表 467: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの監視制御構造体へのポインタ。
p_cfg	複数のビットを書き換えることもできます。	ドライバ インスタンスの構成構造体へのポインタ

定義 : lvd_ctrl_t

LVD 制御構造体

定義 :

定義 : lvd_cfg_t const *const p_cfg

LVD 構成構造体

- lvd_cfg_t::monitor_number

  監視番号 1、2、…

- lvd_cfg_t::lvd_threshold_t

  範囲外の電圧検出しきい値

  列挙値 :

  – LVD_THRESHOLD_MONITOR_1_LEVEL_0
  – LVD_THRESHOLD_MONITOR_1_LEVEL_1
  – LVD_THRESHOLD_MONITOR_1_LEVEL_2
  – LVD_THRESHOLD_MONITOR_1_LEVEL_3
  – LVD_THRESHOLD_MONITOR_1_LEVEL_4
  – LVD_THRESHOLD_MONITOR_1_LEVEL_5
  – LVD_THRESHOLD_MONITOR_1_LEVEL_6
  – LVD_THRESHOLD_MONITOR_1_LEVEL_7
  – LVD_THRESHOLD_MONITOR_1_LEVEL_8
  – LVD_THRESHOLD_MONITOR_1_LEVEL_9
  – LVD_THRESHOLD_MONITOR_1_LEVEL_A
  – LVD_THRESHOLD_MONITOR_1_LEVEL_B
  – LVD_THRESHOLD_MONITOR_1_LEVEL_C
  – LVD_THRESHOLD_MONITOR_1_LEVEL_D
  – LVD_THRESHOLD_MONITOR_1_LEVEL_E
  – LVD_THRESHOLD_MONITOR_1_LEVEL_F
  – LVD_THRESHOLD_MONITOR_1_LEVEL_11
  – LVD_THRESHOLD_MONITOR_1_LEVEL_12
  – LVD_THRESHOLD_MONITOR_1_LEVEL_13
  – LVD_THRESHOLD_MONITOR_2_LEVEL_0
  – LVD_THRESHOLD_MONITOR_2_LEVEL_1
  – LVD_THRESHOLD_MONITOR_2_LEVEL_2
  – LVD_THRESHOLD_MONITOR_2_LEVEL_3
  – LVD_THRESHOLD_MONITOR_2_LEVEL_5
  – LVD_THRESHOLD_MONITOR_2_LEVEL_6
  – LVD_THRESHOLD_MONITOR_2_LEVEL_7

- lvd_cfg_t::lvd_response_t

  しきい値調査検出に対するレスポンス

  列挙値 :

  – LVD_RESPONSE_NMI

  – LVD_RESPONSE_INTERRUPT

  – LVD_RESPONSE_RESET

  – LVD_RESPONSE_NONE

- lvd_cfg_t::lvd_voltage_slope_t

  電圧の上昇または下降が検出されます

  列挙値 :

  – LVD_VOLTAGE_SLOPE_RISING

  – LVD_VOLTAGE_SLOPE_FALLING

  – LVD_VOLTAGE_SLOPE_BOTH

- lvd_cfg_t::p_callback

  割り込みから呼び出されるユーザー関数

- lvd_cfg_t::p_context

  ユーザー データのプレースホルダー。以下に含まれるユーザー コールバックに渡されます :

- lvd_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータ

7.22.6.7 statusGet

ssp_err_t(* lvd_api_t::statusGet)(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)

詳細説明

監視の現在の状態を取得します （しきい値の超過が検出された、現在電圧が範囲内にある）。LVD 監視の状態のポーリングにいつでも使用できます。lvd_response_t を LVD_RESPONSE_NONE に設定した状態で周辺機器を初期化した場合は、必ず使用する必要があります。また電力消費を低減できます。

- R_LVD_StatusGet

表 468: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの制御構造体へのポインタ
p_lvd_status	入力 / 出力	lvd_status_t インスタンスへのポインタ

定義 : lvd_ctrl_t

LVD 制御構造体

パラメータ p_lvd_status

定義 : lvd_status_t*p_lvd_status

電圧監視ステータス構造体、statusGet 関数および p_callback とともに使用し監視の現在の状態を取得します （しきい値の超過が検出された、現在 VCC が範囲内にある）。

- lvd_status_t::crossing_detected

  しきい値超過検出 （ラッチ）

- lvd_status_t::current_state

  監視対象となる電圧の瞬間的なステータス （しきい値を上回るまたは下回る）

7.22.6.8 statusClear

ssp_err_t(* lvd_api_t::statusClear)(lvd_ctrl_t *const p_ctrl)

詳細説明

監視のラッチ状態をクリアします。lvd_response_t を LVD_RESPONSE_NONE に設定した状態で周辺機器を初期化した場合は、必ず使用する必要があります。また電力消費を低減できます。

- R_LVD_StatusClear

表 469: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの制御構造体へのポインタ

定義 : lvd_ctrl_t

LVD 制御構造体

7.22.6.9 close

ssp_err_t(* lvd_api_t::close)(lvd_ctrl_t *const p_ctrl)

詳細説明

LVD 周辺機器を無効化します。ドライバ インスタンスを閉じます。また電力消費を低減できます。

- R_LVD_Close

表 470: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの制御構造体へのポインタ

定義 : lvd_ctrl_t

LVD 制御構造体

7.22.6.10 versionGet

ssp_err_t(* lvd_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

コンパイル時マクロに基づいて、LVD ドライバのバージョンを返します。また電力消費を低減できます。

- R_LVD_VersionGet

表 471: パラメータ

名前	方向	説明
p_version	入力 / 出力	バージョン構造体へのポインタ

パラメータ p_version

7.22.6.11 lvd_instance_t

lvd_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- lvd_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- lvd_cfg_t const * p_cfg

  インタフェース インスタンスの構成構造体へのポインタ。

- lvd_api_t const * p_api

  インタフェース インスタンスの API 構造体へのポインタ。

7.23 PDC インタフェース

PDC 関数用のインタフェース。

7.23.1 概要

PDC インタフェースは、カメラからの画像キャプチャ機能を提供します。キャプチャが完了すると、転送終了割り込みがトリガされます。

7.23.2 既知の実装

PDC 関連する SSP アーキテクチャのトピック：

- SSP インタフェースとは SSP インタフェース
- SSP レイヤーとは SSP 定義レイヤー
- SSP インタフェースおよびモジュールの使用方法 SSP モジュールの使用

7.23.3 インタフェース API

pdc_api_t

関数名	説明
.open	初期設定。
.close	ドライバを閉じ、再構成を許可します。電力消費を低減できます。
.captureStart	キャプチャを開始します。
.stateGet	VSYNC および HSYNC ピンの状態を取得します。
.versionGet	ドライバのバージョンを返します。

7.23.4 データ構造体

- pdc_state_t
- pdc_callback_args_t
- pdc_cfg_t
- pdc_ctrl_t

- pdc_instance_t

7.23.5 列挙

- pdc_clock_division_t
- pdc_endian_t
- pdc_hsync_polarity_t
- pdc_vsync_polarity_t
- pdc_event_t
- pdc_vsync_state_t
- pdc_hsync_state_t

7.23.6 定義

- #define PDC_API_VERSION_MAJOR

  初期値 :(1)

  定義、共通サービス、およびエラー コードを登録します。
- #define PDC_API_VERSION_MINOR

  初期値 :(1)

7.23.7 API データ

7.23.7.1 pdc_clock_division_t

pdc_clock_division_t

詳細説明

PCKO 出力周波数を提供するため PDC クロックに適用されたクロック分周器

列挙値

名前	説明
PDC_CLOCK_DIVISION_2	CLK/2。
PDC_CLOCK_DIVISION_4	CLK/4。
PDC_CLOCK_DIVISION_6	CLK/6。

名前	説明
PDC_CLOCK_DIVISION_8	CLK/8。
PDC_CLOCK_DIVISION_10	CLK/10。
PDC_CLOCK_DIVISION_12	CLK/12。
PDC_CLOCK_DIVISION_14	CLK/14。
PDC_CLOCK_DIVISION_16	CLK/16。

7.23.7.2 pdc_endian_t

pdc_endian_t

詳細説明

キャプチャされたデータのエンディアン

列挙値

名前	説明
PDC_ENDIAN_LITTLE	データがリトル エンディアン フォーマットです。
PDC_ENDIAN_BIG	データがビッグ エンディアン フォーマットです。

7.23.7.3 pdc_hsync_polarity_t

pdc_hsync_polarity_t

詳細説明

入力 HSYNC 信号の極性

列挙値

名前	説明
PDC_HSYNC_POLARITY_HIGH	HSYNC 信号がアクティブ ハイです。
PDC_HSYNC_POLARITY_LOW	HSYNC 信号がアクティブ ローです。

7.23.7.4 pdc_vsync_polarity_t

pdc_vsync_polarity_t

詳細説明

入力 VSYNC 信号の極性

列挙値

名前	説明
PDC_VSYNC_POLARITY_HIGH	VSYNC 信号がアクティブ ハイです。
PDC_VSYNC_POLARITY_LOW	VSYNC 信号がアクティブ ローです。

7.23.7.5 pdc_event_t

pdc_event_t

詳細説明

PDC イベント

列挙値

名前	説明
PDC_EVENT_TRANSFER_COMPLETE	DMAC/DTC によるフレームの転送が完了しました。
PDC_EVENT_RX_DATA_READY	受信データ レディ割り込み。
PDC_EVENT_FRAME_END	フレーム終了割り込み。
PDC_EVENT_ERR_OVERRUN	オーバーラン割り込み。
PDC_EVENT_ERR_UNDERRUN	アンダーラン割り込み。
PDC_EVENT_ERR_V_SET	垂直ライン設定エラー割り込み。
PDC_EVENT_ERR_H_SET	水平バイト数設定エラー割り込み。

7.23.7.6 pdc_vsync_state_t

pdc_vsync_state_t

詳細説明

VSYNC の信号状態

列挙値

名前	説明
PDC_VSYNC_STATE_LOW	VSYNC 信号がローです。
PDC_VSYNC_STATE_HIGH	VSYNC 信号がハイです。

7.23.7.7 pdc_hsync_state_t

pdc_hsync_state_t

詳細説明

HSYNC の信号状態

列挙値

名前	説明
PDC_HSYNC_STATE_LOW	HSYNC 信号がローです。
PDC_HSYNC_STATE_HIGH	HSYNC 信号がハイです。

7.23.8 API 構造

7.23.8.1 pdc_state_t

pdc_state_t

詳細説明

PDC VSYNC/HSYNC の状態

変数

- pdc_vsync_state_t vsync
  VSYNC の信号状態。
- pdc_hsync_state_t hsync
  HSYNC の信号状態。

7.23.8.2 pdc_callback_args_t

pdc_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- pdc_event_t event
  コールバックを生じさせるイベント。
- uint8_t * p_buffer
  キャプチャしたデータを含むバッファへのポインタ。
- void const * p_context
  ユーザー データのプレースホルダー。pdc_cfg_t 内の pdc_api_t::open 関数で設定されます。

7.23.8.3 pdc_cfg_t

pdc_cfg_t

詳細説明

PDC 設定パラメータ。

変数

- uint8_t bytes_per_pixel
  ピクセルごとのバイト数。
- uint16_t x_capture_start_pixel
  キャプチャを開始する水平位置。
- uint16_t x_capture_pixels
  キャプチャする水平ピクセル数。
- uint16_t y_capture_start_pixel
  キャプチャを開始する垂直位置。
- uint16_t y_capture_pixels
  キャプチャする垂直ライン / ピクセルの数。
- pdc_clock_division_t clock_division
  クロック分周器。

- pdc_endian_t endian

  キャプチャするデータのエンディアン。
- pdc_hsync_polarity_t hsync_polarity

  HSYNC 入力の極性。
- pdc_vsync_polarity_t vsync_polarity

  VSYNC 入力の極性。
- uint8_t * p_buffer

  画像書き込み先バッファへのポインタ。
- transfer_instance_t const *const p_lower_lvl_transfer

  PDC が使用する転送インスタンスへのポインタ。
- void(* p_callback)(pdc_callback_args_t *p_args)

  PDC 転送 ISR の発生時に提供されるコールバック。
- void const * p_context

  ユーザー データのプレースホルダー。pdc_callback_args_t 内のユーザー コールバックに渡されます。
- void const * p_extend

7.23.8.4 pdc_ctrl_t

pdc_ctrl_t

詳細説明

PDC 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- bool pdc_open

  open() API が正常に呼び出されたかどうかを示します。
- uint8_t bytes_per_pixel

  ピクセルごとのバイト数。
- uint16_t x_resolution_pixels

  PDC への水平ピクセル入力の合計数。
- uint16_t y_resolution_pixels

  PDC へのライン入力の合計数。
- uint16_t x_capture_start_pixel

  キャプチャを開始する水平位置。

- uint16_t x_capture_pixels

  キャプチャする水平ピクセル数。
- uint16_t y_capture_start_pixel

  キャプチャを開始する垂直位置。
- uint16_t y_capture_pixels

  キャプチャする垂直ライン / ピクセルの数。
- pdc_endian_t endian

  キャプチャするデータのエンディアン。
- pdc_hsync_polarity_t hsync_polarity

  HSYNC 入力の極性。
- pdc_vsync_polarity_t vsync_polarity

  VSYNC 入力の極性。
- uint8_t * p_current_buffer

  現在使用中のバッファへのポインタ。
- bool transfer_in_progress

  PDC 転送がすでに実行中であるかどうかを示します。
- transfer_instance_t const * p_lower_lvl_transfer

  PDC が使用する転送インスタンスへのポインタ。
- transfer_info_t info_transfer

  ローレベル転送インタフェースに対する転送情報構造体。
- void const * p_context

  ユーザー データのプレースホルダー。pdc_callback_args_t 内のユーザー コールバックに渡されます。
- void(* p_callback)(pdc_callback_args_t *p_args)

  PDC 転送 ISR の発生時に提供されるコールバック。

7.23.8.5 pdc_api_t

pdc_api_t

詳細説明

HAL レイヤーに実装された PDC 関数は、この API に従います。

7.23.8.6 open

ssp_err_t(* pdc_api_t::open)(pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_PDC_Open

  I：この関数を呼び出した後に再構成を行うには、まず close を呼び出します。

表 472: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_cfg	複数のビットを書き換えることもできます。	ピン設定構造体へのポインタ。

定義 : pdc_ctrl_t

PDC 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : pdc_cfg_t const *const p_cfg

PDC 設定パラメータ。

- pdc_cfg_t::bytes_per_pixel

  ピクセルごとのバイト数。

- pdc_cfg_t::x_capture_start_pixel

  キャプチャを開始する水平位置。

- pdc_cfg_t::x_capture_pixels

  キャプチャする水平ピクセル数。

- pdc_cfg_t::y_capture_start_pixel

  キャプチャを開始する垂直位置。

- pdc_cfg_t::y_capture_pixels

  キャプチャする垂直ライン / ピクセルの数。

- pdc_cfg_t::pdc_clock_division_t

  クロック分周器。

  列挙値 :

  – PDC_CLOCK_DIVISION_2
  – PDC_CLOCK_DIVISION_4
  – PDC_CLOCK_DIVISION_6
  – PDC_CLOCK_DIVISION_8
  – PDC_CLOCK_DIVISION_10
  – PDC_CLOCK_DIVISION_12
  – PDC_CLOCK_DIVISION_14
  – PDC_CLOCK_DIVISION_16

- pdc_cfg_t::pdc_endian_t

  キャプチャするデータのエンディアン。

  列挙値 :

  – PDC_ENDIAN_LITTLE
  – PDC_ENDIAN_BIG

- pdc_cfg_t::pdc_hsync_polarity_t

  HSYNC 入力の極性。

  列挙値 :

  – PDC_HSYNC_POLARITY_HIGH
  – PDC_HSYNC_POLARITY_LOW

- pdc_cfg_t::pdc_vsync_polarity_t

  VSYNC 入力の極性。

  列挙値 :

  – PDC_VSYNC_POLARITY_HIGH
  – PDC_VSYNC_POLARITY_LOW

- pdc_cfg_t::p_buffer

  画像書き込み先バッファへのポインタ。

- pdc_cfg_t::transfer_instance_t

  PDC が使用する転送インスタンスへのポインタ。

- pdc_cfg_t::p_callback

  PDC 転送 ISR の発生時に提供されるコールバック。

- pdc_cfg_t::p_context

  ユーザー データのプレースホルダー。pdc_callback_args_t 内のユーザー コールバックに渡されます。

- pdc_cfg_t::p_extend

7.23.8.7 close

ssp_err_t(* pdc_api_t::close)(pdc_ctrl_t *const p_ctrl)

詳細説明

ドライバを閉じ、再構成を許可します。電力消費を低減できます。また電力消費を低減できます。

- R_PDC_Close

表 473: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。

定義 : pdc_ctrl_t

PDC 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.23.8.8 captureStart

ssp_err_t(* pdc_api_t::captureStart)(pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)

詳細説明

キャプチャを開始します。また電力消費を低減できます。

- R_PDC_CaptureStart

表 474: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_buffer	複数のビットを書き換えることもできます。	キャプチャした画像データを保存するためのポインタ。

定義 : pdc_ctrl_t

PDC 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_buffer

uint8_t

7.23.8.9 stateGet

ssp_err_t(* pdc_api_t::stateGet)(pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)

詳細説明

VSYNC および HSYNC ピンの状態を取得します。また電力消費を低減できます。

- R_PDC_StateGet

表 475: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_state	複数のビットを書き換えることもできます。	状態データを保存するためのポインタ。

定義 : pdc_ctrl_t

PDC 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_state

定義 : pdc_state_t*p_state

PDC VSYNC/HSYNC の状態

- pdc_state_t::vsync

  VSYNC の信号状態。
- pdc_state_t::hsync

  HSYNC の信号状態。

7.23.8.10 versionGet

ssp_err_t(* pdc_api_t::versionGet)(ssp_version_t *const p_data)

詳細説明

ドライバのバージョンを返します。また電力消費を低減できます。

- R_PDC_VersionGet

表 476: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_data	out	バージョン情報を返す先のメモリ アドレス。

パラメータ p_data

定義 : `ssp_version_t` *const p_data

- `ssp_version_t`::version_id

  バージョン ID
- `ssp_version_t`::code_version_minor

  コードのマイナー バージョン。
- `ssp_version_t`::code_version_major

  コードのメジャー バージョン。
- `ssp_version_t`::api_version_minor

  API のマイナー バージョン。
- `ssp_version_t`::api_version_major

  API のメジャー バージョン。
- `ssp_version_t`struct{}

  コード バージョンのパラメータ

7.23.8.11 pdc_instance_t

pdc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- pdc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- pdc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- pdc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.24 クワッド SPI フラッシュインタフェース

外部 SPI フラッシュ デバイスにアクセスするためのインタフェース。

7.24.1 概要

QSPI モジュールは、QSPI インタフェースに接続されたクワッド SPI フラッシュ デバイス内のセクターを書き込むまたは消去するインタフェースを提供します。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

QSPI インタフェースの説明 : QSPI ドライバ

7.24.2 インタフェース API

qspi_api_t

関数名	説明
.open	QSPI ドライバ モジュールを開きます。
.close	QSPI ドライバ モジュールを閉じます。
.read	フラッシュからデータ ブロックを読み取ります。
.pageProgram	フラッシュ デバイスにページ単位でデータをプログラミングします。
.sectorErase	フラッシュ デバイスのセクターを消去します。
.statusGet	フラッシュ デバイスの書き込みまたは消去のステータスを取得します。
.bankSelect	アクセスするバンクを選択します。
.versionGet	コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

7.24.3 データ構造体

- qspi_cfg_t
- qspi_ctrl_t
- qspi_instance_t

7.24.4 定義

- #define QSPI_API_VERSION_MAJOR

  初期値 :(1)
- #define QSPI_API_VERSION_MINOR

  初期値 :(1)

7.24.5 API 構造

7.24.5.1 qspi_cfg_t

qspi_cfg_t

詳細説明

オープン関数によって使用されるユーザー設定構造体

変数

- void * p_extend

  将来の開発用のプレース ホルダー

7.24.5.2 qspi_ctrl_t

qspi_ctrl_t

詳細説明

パラメータ p_cfg

変数

- uint8_t manufacturer_id

  メーカー ID。
- uint8_t memory_type

  メモリ タイプ。

- uint8_t memory_capacity

  メモリ容量 （MB 単位）

- uint32_t max_eraseable_size

  消去可能な最大セクター （KB 単位）バッファ サイズの判定に使用されます。消去可能な最大セクター （KB 単位）バッファ サイズの判定に使用されます。

- uint32_t num_address_bytes

  アドレスの表現に使用するバイト数。

- uint32_t spi_mode

  0 = 拡張、1 = デュアル、2 = クワッド。

- uint32_t page_size

  プログラミング可能なページのバイト数。

- bool xip_mode

  0 = 読み取りモードで実行、1 = XIP モードで実行

7.24.5.3 qspi_api_t

qspi_api_t

詳細説明

HAL レイヤーに実装された QSPI 関数は、この API に従います。

7.24.5.4 open

(* `qspi_api_t`::open)(`qspi_ctrl_t` *p_ctrl, `qspi_cfg_t` const *const p_cfg)

詳細説明

QSPI ドライバ モジュールを開きます。また電力消費を低減できます。

- R_QSPI_Open

表 477: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ ハンドルへのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : qspi_ctrl_t

パラメータ p_cfg

定義 :

定義 : qspi_cfg_t const *const p_cfg

オープン関数によって使用されるユーザー設定構造体

- qspi_cfg_t::p_extend

  将来の開発用のプレース ホルダー

7.24.5.5 close

(* qspi_api_t::close)(qspi_ctrl_t *p_ctrl)

詳細説明

QSPI ドライバ モジュールを閉じます。また電力消費を低減できます。

- R_QSPI_Close

表 478: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ ハンドルへのポインタ

定義 : qspi_ctrl_t

パラメータ p_cfg

7.24.5.6 read

(* qspi_api_t::read)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)

詳細説明

フラッシュからデータ ブロックを読み取ります。また電力消費を低減できます。

- R_QSPI_Read

表 479: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ ハンドルへのポインタ
p_device_address	複数のビットを書き換えることもできます。	フラッシュ デバイス内の読み取るアドレス空間の位置
p_memory_address	複数のビットを書き換えることもできます。	読み取ったデータを格納するバッファのメモリ アドレス
byte_count	複数のビットを書き換えることもできます。	読み取るバイト数

定義 : qspi_ctrl_t

パラメータ p_cfg

パラメータ p_device_address

uint8_t

パラメータ p_memory_address

uint8_t

パラメータ byte_count

uint32_t

7.24.5.7 pageProgram

(* qspi_api_t::pageProgram)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)

詳細説明

フラッシュ デバイスにページ単位でデータをプログラミングします。また電力消費を低減できます。

- R_QSPI_PageProgram

表 480: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ ハンドルへのポインタ

表 480: パラメータ (続き)

名前	方向	説明
p_device_address	複数のビットを書き換えることもできます。	フラッシュ デバイス内のデータを書き込むアドレス空間の位置
p_memory_address	複数のビットを書き換えることもできます。	フラッシュ デバイスに書き込むデータのメモリ アドレス
byte_count	複数のビットを書き換えることもできます。	書き込むバイト数

定義 : qspi_ctrl_t

パラメータ p_cfg

パラメータ p_device_address

uint8_t

パラメータ p_memory_address

uint8_t

パラメータ byte_count

uint32_t

7.24.5.8 sectorErase

(* qspi_api_t::sectorErase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)

詳細説明

フラッシュ デバイスのセクターを消去します。また電力消費を低減できます。

- R_QSPI_SectorErase

表 481: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ ハンドルへのポインタ
p_device_address	複数のビットを書き換えることもできます。	フラッシュ デバイス内の消去を開始するアドレス空間の位置

定義 : qspi_ctrl_t

パラメータ p_cfg

パラメータ p_device_address

uint8_t

7.24.5.9 statusGet

(* qspi_api_t::statusGet)(qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)

詳細説明

フラッシュ デバイスの書き込みまたは消去のステータスを取得します。また電力消費を低減できます。

- R_QSPI_StatusGet

表 482: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ ハンドルへのポインタ
p_write_in_progress	複数のビットを書き換えることもできます。	書き込みまたは消去のステータス (TRUE の場合は、書き込み中 / 消去の実行中)

定義 : qspi_ctrl_t

パラメータ p_cfg

パラメータ p_write_in_progress

const

7.24.5.10 bankSelect

(* qspi_api_t::bankSelect)(uint32_t bank)

詳細説明

アクセスするバンクを選択します。また電力消費を低減できます。

- R_QSPI_BankSelect

表 483: パラメータ

名前	方向	説明
bank	複数のビットを書き換えることもできます。	バンク番号

パラメータ **bank**

uint32_t

7.24.5.11 versionGet

(* qspi_api_t::versionGet)(*const p_version)

詳細説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。また電力消費を低減できます。

- R_QSPI_VersionGet

表 484: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ **p_version**

7.24.5.12 qspi_instance_t

qspi_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- qspi_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- qspi_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- qspi_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.25 RTC インタフェース

リアルタイム クロックにアクセスするためのインタフェース。

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

RTC の説明 : RTC ドライバ

7.25.1 インタフェース API

rtc_api_t

関数名	説明
.open	RTC ドライバを開きます。
.close	RTC ドライバを閉じます。
.calendarTimeSet	カレンダーの日付を設定します。
.calendarTimeGet	カレンダーの時間を取得します。
.calendarAlarmSet	カレンダーのアラーム時間を設定します。
.calendarAlarmGet	カレンダーのアラーム時間を取得します。
.calendarCounterStart	カレンダー カウンタを開始します。
.calendarCounterStop	カレンダー カウンタを停止します。
.irqEnable	アラーム irq を有効にします。
.irqDisable	アラーム irq を無効にします。
.periodicIrqRateSet	周期 IRQ レートを設定します
.infoGet	現在構成されている RTC のクロック ソースを返します
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.25.2 データ構造体

- rtc_callback_args_t
- rtc_alarm_time_t
- rtc_info_t
- rtc_cfg_t
- rtc_ctrl_t
- rtc_instance_t

7.25.3 列挙

- rtc_event_t
- rtc_clock_source_t
- rtc_error_adjustment_t
- rtc_periodic_irq_select_t

7.25.4 型定義

- rtc_time_t

7.25.5 定義

- #define RTC_API_VERSION_MAJOR

  初期値 :(1)

  時間構造体 tm の使用
- #define RTC_API_VERSION_MINOR

  初期値 :(1)

7.25.6 API データ

7.25.6.1 rtc_event_t

rtc_event_t

詳細説明

コールバック関数をトリガー可能なイベント

列挙値

名前	説明
RTC_EVENT_ALARM_IRQ	リアルタイム クロック アラーム IRQ。
RTC_EVENT_PERIODIC_IRQ	リアルタイム クロック周期 IRQ。
RTC_EVENT_CARRY_IRQ	リアルタイム クロック キャリー IRQ。

7.25.6.2 rtc_clock_source_t

rtc_clock_source_t

詳細説明

RTC ブロックのクロック ソース

列挙値

名前	説明
RTC_CLOCK_SOURCE_SUBCLK	サブ クロック オシレーター。
RTC_CLOCK_SOURCE_LOCO	低電力オンチップ オシレーター。

7.25.6.3 rtc_error_adjustment_t

rtc_error_adjustment_t

詳細説明

時間エラー調整の設定

列挙値

名前	説明
RTC_ERROR_ADJUSTMENT_NONE	調整が実行されません。
RTC_ERROR_ADJUSTMENT_ADD_PRESCALER	プリスケーラーに加算することによって調整が実行されます。

名前	説明
RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER	プリスケーラーから減算することによって調整が実行されます。

7.25.6.4 rtc_periodic_irq_select_t

rtc_periodic_irq_select_t

詳細説明

周期割り込み選択

列挙値

名前	説明
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND	周期 irq が 1/256 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND	周期 irq が 1/128 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND	周期 irq が 1/64 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND	周期 irq が 1/32 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND	周期 irq が 1/16 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND	周期 irq が 1/8 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND	周期 irq が 1/4 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND	周期 irq が 1/2 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_1_SECOND	周期 irq が 1 秒ごとに生成されます。
RTC_PERIODIC_IRQ_SELECT_2_SECONDS	周期 irq が 2 秒ごとに生成されます。

7.25.6.5 rtc_time_t

typedef struct tm rtc_time_t

詳細説明

C の標準ライブラリ <time.h> で定義されている日時構造体

7.25.7 API 構造

7.25.7.1 rtc_callback_args_t

rtc_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- rtc_event_t event

  このイベントを使用して、コールバックの原因 （コンペア マッチまたはエラー）を特定できます。
- void const * p_context

  ユーザー データのプレースホルダー。timer_cfg_t 内の r_timer_t::open 関数で設定されます。

7.25.7.2 rtc_alarm_time_t

rtc_alarm_time_t

詳細説明

アラーム時間の設定値構造体

変数

- rtc_time_t time

  時間構造体。
- bool sec_match

  秒フィールドが一致した場合、アラームを有効にします。
- bool min_match

  分フィールドが一致した場合、アラームを有効にします。
- bool hour_match

  時刻フィールドが一致した場合、アラームを有効にします。
- bool mday_match

  日付フィールドが一致した場合、アラームを有効にします。

- bool mon_match

  月フィールドが一致した場合、アラームを有効にします。

- bool year_match

  年フィールドが一致した場合、アラームを有効にします。

- bool dayofweek_match

  曜日フィールドが一致した場合、アラームを有効にします。

7.25.7.3 rtc_info_t

rtc_info_t

詳細説明

infoGet() によって返される情報用の RTC 情報構造体

変数

- rtc_clock_source_t clock_source

  RTC ブロックのクロック ソース。

7.25.7.4 rtc_cfg_t

rtc_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- rtc_clock_source_t clock_source

  RTC ブロックのクロック ソース。

- uint32_t error_adjustment_value

  エラー調整用のプリスケーラーの値。

- rtc_error_adjustment_t error_adjustment_type

  プリスケーラの値の適用方法。

- void(* p_callback)(rtc_callback_args_t *p_args)

  ISR から呼び出されます。

- void const * p_context

  コールバックに渡されます。

- void const * p_extend

  RTC ハードウェアに依存する設定。

7.25.7.5 rtc_ctrl_t

rtc_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

変数

- rtc_clock_source_t clock_source

  RTC ブロックのクロック ソース。

7.25.7.6 rtc_api_t

rtc_api_t

詳細説明

RTC ドライバ構造体。HAL レイヤーに実装された汎用 RTC 関数は、この API に従います。

7.25.7.7 open

ssp_err_t(* `rtc_api_t`::open)(`rtc_ctrl_t` *const p_ctrl, `rtc_cfg_t` const *const p_cfg)

詳細説明

RTC ドライバを開きます。また電力消費を低減できます。

- R_RTC_Open

表 485: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ

定義 : `rtc_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

定義 :

定義 : rtc_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- rtc_cfg_t::rtc_clock_source_t

  RTC ブロックのクロック ソース。

  列挙値 :

  – RTC_CLOCK_SOURCE_SUBCLK

  – RTC_CLOCK_SOURCE_LOCO

- rtc_cfg_t::error_adjustment_value

  エラー調整用のプリスケーラーの値。

- rtc_cfg_t::rtc_error_adjustment_t

  プリスケーラの値の適用方法。

  列挙値 :

  – RTC_ERROR_ADJUSTMENT_NONE

  – RTC_ERROR_ADJUSTMENT_ADD_PRESCALER

  – RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER

- rtc_cfg_t::p_callback

  ISR から呼び出されます。

- rtc_cfg_t::p_context

  コールバックに渡されます。

- rtc_cfg_t::p_extend

  RTC ハードウェアに依存する設定。

7.25.7.8 close

ssp_err_t(* rtc_api_t::close)(rtc_ctrl_t *const p_ctrl)

詳細説明

RTC ドライバを閉じます。また電力消費を低減できます。

- R_RTC_Close

表 486: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ。

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

7.25.7.9 calendarTimeSet

ssp_err_t(* rtc_api_t::calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time, bool clock_start)

詳細説明

カレンダーの日付を設定します。また電力消費を低減できます。

- R_RTC_CalendarTimeSet

表 487: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ
p_time	複数のビットを書き換えることもできます。	設定する時間が含まれている時間構造体へのポインタ
clock_start	複数のビットを書き換えることもできます。	クロックの設定直後にクロックを開始するかどうかを設定するフラグ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

パラメータ p_time

定義 : rtc_time_t*p_time

C の標準ライブラリ <time.h> で定義されている日時構造体

パラメータ clock_start

const

7.25.7.10 calendarTimeGet

ssp_err_t(* rtc_api_t::calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)

詳細説明

カレンダーの時間を取得します。また電力消費を低減できます。

- R_RTC_CalendarTimeGet

表 488: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ
p_time	out	取得する時間が含まれている時間構造体へのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

パラメータ p_time

定義 : rtc_time_t*p_time

C の標準ライブラリ <time.h> で定義されている日時構造体

7.25.7.11 calendarAlarmSet

ssp_err_t(* rtc_api_t::calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm, bool irq_enable_flag)

詳細説明

カレンダーのアラーム時間を設定します。また電力消費を低減できます。

- R_RTC_CalendarAlarmSet

表 489: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ
p_alarm	複数のビットを書き換えることもできます。	設定するアラーム時間が含まれているアラーム構造体へのポインタ

表 489: パラメータ (続き)

名前	方向	説明
irq_enable_flag	複数のビットを書き換えることもできます。	アラーム irq が設定されている場合は、有効にします

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

パラメータ p_alarm

定義 : rtc_alarm_time_t*p_alarm

アラーム時間の設定値構造体

- rtc_alarm_time_t::time

  時間構造体。
- rtc_alarm_time_t::sec_match

  秒フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::min_match

  分フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::hour_match

  時刻フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::mday_match

  日付フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::mon_match

  月フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::year_match

  年フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::dayofweek_match

  曜日フィールドが一致した場合、アラームを有効にします。

パラメータ irq_enable_flag

const

7.25.7.12 calendarAlarmGet

ssp_err_t(* rtc_api_t::calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm)

詳細説明

カレンダーのアラーム時間を取得します。また電力消費を低減できます。

- R_RTC_CalendarAlarmGet

表 490: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ
p_alarm	out	アラーム時間を設定するアラーム構造体へのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

パラメータ p_alarm

定義 : rtc_alarm_time_t*p_alarm

アラーム時間の設定値構造体

- rtc_alarm_time_t::time

  時間構造体。
- rtc_alarm_time_t::sec_match

  秒フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::min_match

  分フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::hour_match

  時刻フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::mday_match

  日付フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::mon_match

  月フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::year_match

  年フィールドが一致した場合、アラームを有効にします。
- rtc_alarm_time_t::dayofweek_match

  曜日フィールドが一致した場合、アラームを有効にします。

7.25.7.13 calendarCounterStart

ssp_err_t(* rtc_api_t::calendarCounterStart)(rtc_ctrl_t *const p_ctrl)

詳細説明

カレンダー カウンタを開始します。また電力消費を低減できます。

- R_RTC_CalendarCounterStart

表 491: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

7.25.7.14 calendarCounterStop

ssp_err_t(* rtc_api_t::calendarCounterStop)(rtc_ctrl_t *const p_ctrl)

詳細説明

カレンダー カウンタを停止します。また電力消費を低減できます。

- R_RTC_CalendarCounterStop

表 492: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

7.25.7.15 irqEnable

ssp_err_t(* rtc_api_t::irqEnable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)

詳細説明

アラーム irq を有効にします。また電力消費を低減できます。

- R_RTC_IrqEnable

表 493: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

7.25.7.16 irqDisable

ssp_err_t(* rtc_api_t::irqDisable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)

詳細説明

アラーム irq を無効にします。また電力消費を低減できます。

- R_RTC_IrqDisable

表 494: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

7.25.7.17 periodicIrqRateSet

ssp_err_t(* rtc_api_t::periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t rate)

詳細説明

周期 IRQ レートを設定します。以下として実装されます。

- R_RTC_PeriodicIrqRateSet

表 495: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	RTC デバイス ハンドルへのポインタ
rate	複数のビットを書き換えることもできます。	周期割り込みのレート

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

パラメータ rate

定義 : rtc_periodic_irq_select_trate

周期割り込み選択

7.25.7.18 infoGet

ssp_err_t(* rtc_api_t::infoGet)(rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)

詳細説明

現在構成されている RTC のクロック ソースを返します

また電力消費を低減できます。

- R_RTC_InfoGet

表 496: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ハンドル構造体へのポインタ
p_rtc_info	out	ADC 情報構造体へのポインタ

定義 : rtc_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、r_rtc_t::open の呼び出し時に実行されます

パラメータ p_rtc_info

定義 : rtc_info_t*p_rtc_info

infoGet() によって返される情報用の RTC 情報構造体

- rtc_info_t::clock_source

  RTC ブロックのクロック ソース。

7.25.7.19 versionGet

ssp_err_t(* rtc_api_t::versionGet)(ssp_version_t *version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_RTC_VersionGet

表 497: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン

パラメータ p_version

7.25.7.20 rtc_instance_t

rtc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- rtc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- rtc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- rtc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.26 SDMMC インタフェース

SDMMC カードにアクセスするためのインタフェース。

7.26.1 概要

r_sdmmc インタフェースは、標準の SD、MMC、および eMMC メディア機能を提供します。FileX、sf_el_fx、sf_block_media_sdmmc、および r_sdmmc モジュールを使用して、完全なファイル システムを実装できます。SDMMC インタフェースは次のように実装されます。

- SDMMC

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

SDMMC の説明 : SD/MMC ドライバおよび SDIO ドライバ

7.26.2 インタフェース API

sdmmc_api_t

関数名	説明
.open	読み書きおよび制御のためにデバイス チャネルを開きます。最初の呼び出しで、ドライバとハードウェアを初期化します。
.close	開いている SDMMC チャネルを閉じます。開いている最後のチャネルである場合は、ハードウェアをオフにします。
.read	SDMMC チャネルからデータを読み取ります。
.write	SDMMC チャネルにデータを書き込みます。
.control	この制御関数は、SD/MMC ポートとの間で、制御コマンドを送信し情報を受信します。
.readIo	SDMMC チャネルから I/O データを読み取ります。
.writeIo	SDMMC チャネルに I/O データを書き込みます。

関数名	説明
.readIoExt	SDMMC チャネルから I/O 拡張データを読み取ります。
.writeIoExt	SDMMC チャネルに I/O 拡張データを書き込みます。
.ioIntEnable	SDMMC チャネルに対して、SDIO 割り込みを有効化します。
.versionGet	SDMMC ドライバのバージョンを返します。
.infoGet	SDMMC チャネルの情報を取得します。
.erase	SDMMC セクターを消去します。

7.26.3 データ構造体

- sdmmc_hw_t
- sdmmc_info_t
- sdmmc_io_mode_t
- sdmmc_callback_args_t
- sdmmc_cfg_t
- sdmmc_ctrl_t
- sdmmc_instance_t

7.26.4 列挙

- sdmmc_ready_status_t
- sdmmc_card_type_t
- sdmmc_media_type_t
- sdmmc_bus_width_t
- sdmmc_io_command_t
- sdmmc_io_transfer_mode_t
- sdmmc_io_address_mode_t
- sdmmc_io_write_mode_t
- sdmmc_event_t

7.26.5 定義

- #define SDMMC_API_VERSION_MAJOR

  初期値 :((uint8_t)1)
- #define SDMMC_API_VERSION_MINOR

  初期値 :((uint8_t)0)
- #define SDMMC_MAX_BLOCK_SIZE

  初期値 :(512)

7.26.6 API データ

7.26.6.1 sdmmc_ready_status_t

sdmmc_ready_status_t

詳細説明

SDMMC ステータス

列挙値

名前	説明
SDMMC_STATUS_CARD_NOT_READY	SD カードまたは eMMC デバイスが初期化されていません。
SDMMC_STATUS_CARD_READY	SD カードまたは eMMC デバイスが初期化済みで、いつでもアクセスできます。

7.26.6.2 sdmmc_card_type_t

sdmmc_card_type_t

詳細説明

SDMMC メディアは SD プロトコルまたは MMC プロトコルを使用します。

列挙値

名前	説明
SDMMC_CARD_TYPE_MMC	メディアは MMC カードまたは eMMC デバイスです。

名前	説明
SDMMC_CARD_TYPE_SD	メディアは SD カードです。

7.26.6.3 sdmmc_media_type_t

sdmmc_media_type_t

詳細説明

SDMMC メディアは、組み込まれている場合と取り付け / 取り外しが可能な場合があります。

列挙値

名前	説明
SDMMC_MEDIA_TYPE_EMBEDDED	メディアは、組み込みカード、つまり、eMMC デバイスです。
SDMMC_MEDIA_TYPE_CARD	メディアは、プラグ可能なカードです。

7.26.6.4 sdmmc_bus_width_t

sdmmc_bus_width_t

詳細説明

SDMMC データ バスは 1、4、または 8 ビット幅です。

列挙値

名前	説明
SDMMC_BUS_WIDTH_1_BIT	データ バスは 1 ビット幅です。
SDMMC_BUS_WIDTH_4_BITS	データ バスは 4 ビット幅です。
SDMMC_BUS_WIDTH_8_BITS	データ バスは 8 ビット幅です。

7.26.6.5 sdmmc_io_command_t

sdmmc_io_command_t

詳細説明

SDIO コマンド

列挙値

名前	説明
SDMMC_IO_COMMAND_DIRECT_IO	
SDMMC_IO_COMMAND_DIRECT_IO_EXT	

7.26.6.6 sdmmc_io_transfer_mode_t

sdmmc_io_transfer_mode_t

列挙値

名前	説明
SDMMC_IO_MODE_TRANSFER_BYTE	
SDMMC_IO_MODE_TRANSFER_BLOCK	

7.26.6.7 sdmmc_io_address_mode_t

sdmmc_io_address_mode_t

列挙値

名前	説明
SDMMC_IO_ADDRESS_MODE_FIXED	
SDMMC_IO_ADDRESS_MODE_INCREMENT	

7.26.6.8 sdmmc_io_write_mode_t

sdmmc_io_write_mode_t

列挙値

名前	説明
SDMMC_IO_WRITE_MODE_NO_READ	
SDMMC_IO_WRITE_READ_AFTER_WRITE	

7.26.6.9 sdmmc_event_t

sdmmc_event_t

詳細説明

コールバック関数をトリガー可能なイベント

列挙値

名前	説明
SDMMC_EVENT_CARD_REMOVED	カード取り外しイベント。
SDMMC_EVENT_CARD_INSERTED	カード挿入イベント。
SDMMC_EVENT_ACCESS	アクセス イベント。
SDMMC_EVENT_SDIO	IO イベント。
SDMMC_EVENT_TRANSFER_COMPLETE	読み取りまたは書き込みが完了しました。
SDMMC_EVENT_TRANSFER_ERROR	読み取りまたは書き込みに失敗しました。
SDMMC_EVENT_NONE	イベントなし。

7.26.7 API 構造

7.26.7.1 sdmmc_hw_t

sdmmc_hw_t

詳細説明

ハードウェアごとに定義されているチャネル、メディア タイプ、バス幅。

変数

- uint8_t channel

  SD/MMC ホスト インタフェースのチャネル。

- sdmmc_media_type_t media_type

  組み込みカードまたはプラグ可能カード。

- sdmmc_bus_width_t bus_width

  デバイスのバス幅は、1、4、8 ビット幅のいずれかです。

7.26.7.2 sdmmc_info_t

sdmmc_info_t

詳細説明

メディア デバイスから取得したステータスなどの情報。

変数

- sdmmc_card_type_t card_type

  SD または MMC。

- bool ready

  準備完了または未完了

- bool hc

  true = カードが大容量カード

- bool sdio

  true = SDIO が存在

- bool write_protected

  カードが書き込み保護されています。

- bool transfer_in_progress

  DMA または DTC 転送ステータス。

- uint8_t csd_version

  CSD のバージョン。

- uint8_t device_type

  速度とデータ レート。

- sdmmc_bus_width_t bus_width

  現在のメディア バス幅。

- uint8_t hs_timing

  高速ステータス。
- uint32_t sdhi_rca

  相対カード アドレス。
- uint32_t max_clock_rate

  メディア カードの最大クロック レート。
- uint32_t clock_rate

  現在のクロック レート
- uint32_t sector_size

  セクター サイズ
- uint32_t sector_count

  セクター数
- uint32_t erase_sector_count

  一度に消去する最大セクター数。

7.26.7.3 sdmmc_io_mode_t

sdmmc_io_mode_t

詳細説明

変数

- sdmmc_io_command_t command

  SDIO コマンド。
- sdmmc_io_transfer_mode_t transfer_mode

  SDIO 転送タイプ。
- sdmmc_io_address_mode_t address_mode

  SDIO アドレス モード。
- sdmmc_io_write_mode_t write_mode

  SDIO 書き込みモード。

7.26.7.4 sdmmc_callback_args_t

sdmmc_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- sdmmc_event_t event

  このイベントは、コールバックの原因を特定するために使用できます。

- void const * p_context

  ユーザー データのプレースホルダー。

7.26.7.5 sdmmc_cfg_t

sdmmc_cfg_t

詳細説明

SDMMC の設定

変数

- sdmmc_hw_t hw

  ハードウェアごとに定義されているチャネル、メディア タイプ、バス幅。

- transfer_instance_t const * p_lower_lvl_transfer

  DMA または DTC を使用してデータを移動するための転送インスタンス。

- void(* p_callback)(sdmmc_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void const * p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。

- void const * p_extend

  SDMMC ハードウェアに依存する設定。

7.26.7.6 sdmmc_ctrl_t

sdmmc_ctrl_t

詳細説明

パラメータ p_cfg

変数

- sdmmc_hw_t hw

  ハードウェアごとに定義されているチャネル、メディア タイプ、バス幅。

- transfer_instance_t const * p_lower_lvl_transfer

  DMA または DTC を使用してデータを移動するための転送インスタンス。

- sdmmc_info_t status

  メディア ステータス情報。

- bool transfer_in_progress

  DMA または DTC 転送ステータス。

- void(* p_callback)(sdmmc_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void const * p_context

  ハイ レベルのデバイス コンテキストへのポインタ。

7.26.7.7 sdmmc_api_t

sdmmc_api_t

詳細説明

HAL レイヤー API に実装された SDMMC 関数。

7.26.7.8 open

ssp_err_t(* sdmmc_api_t::open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)

詳細説明

読み書きおよび制御のためにデバイス チャネルを開きます。最初の呼び出しで、ドライバとハードウェアを初期化します。また電力消費を低減できます。R_SDMMC_Open

表 498: パラメータ

名前	方向	説明
p_ctrl	out	SDMMC チャネルのハンドル（SDMMC チャネル デバイス コンテキストへのポインタ）。
p_cfg	複数のビットを書き換えることもできます。	チャネルの SDMMC 設定構造体へのポインタ。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

定義 :

定義 : sdmmc_cfg_t const *const p_cfg

SDMMC の設定

- sdmmc_cfg_t::sdmmc_hw_t

  ハードウェアごとに定義されているチャネル、メディア タイプ、バス幅。

- sdmmc_cfg_t::transfer_instance_t

  DMA または DTC を使用してデータを移動するための転送インスタンス。

- sdmmc_cfg_t::p_callback

  コールバック関数へのポインタ。

- sdmmc_cfg_t::p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。

- sdmmc_cfg_t::p_extend

  SDMMC ハードウェアに依存する設定。

7.26.7.9 close

ssp_err_t(* sdmmc_api_t::close)(sdmmc_ctrl_t *const p_ctrl)

詳細説明

開いている SDMMC チャネルを閉じます。開いている最後のチャネルである場合は、ハードウェアをオフにします。また電力消費を低減できます。R_SDMMC_Close

表 499: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SDMMC チャネルのハンドル（SDMMC チャネル デバイス コンテキストへのポインタ）。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

7.26.7.10 read

ssp_err_t(* sdmmc_api_t::read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

詳細説明

SDMMC チャネルからデータを読み取ります。また電力消費を低減できます。R_SDMMC_Read

表 500: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SDMMC チャネルのハンドル（SDMMC チャネル デバイス コンテキストへのポインタ）。
p_dest	out	データ読み出し先のデータ バッファへのポインタ。
start_sector	複数のビットを書き換えることもできます。	読み取る開始セクターのアドレス。
sector_count	複数のビットを書き換えることもできます。	読み取るセクター数。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ p_dest

uint8_t

パラメータ start_sector

uint32_t

パラメータ sector_count

uint32_t

7.26.7.11 write

ssp_err_t(* sdmmc_api_t::write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)

詳細説明

SDMMC チャネルにデータを書き込みます。また電力消費を低減できます。R_SDMMC_Write

表 501: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）。
p_source	複数のビットを書き換えることもできます。	データ書き込み元のデータ バッファへのポインタ。
start_sector	複数のビットを書き換えることもできます。	書き込み先の開始セクターのアドレス。
sector_count	複数のビットを書き換えることもできます。	書き込むセクター数。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ p_source

uint8_t

パラメータ start_sector

uint32_t

パラメータ sector_count

uint32_t

7.26.7.12 control

ssp_err_t(* sdmmc_api_t::control)(sdmmc_ctrl_t *const p_ctrl, ssp_command_t const command, void *p_data)

詳細説明

この制御関数は、SD/MMC ポートとの間で、制御コマンドを送信し情報を受信します。また電力消費を低減できます。R_SDMMC_Control

表 502: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）。

表 502: パラメータ (続き)

名前	方向	説明
command	複数のビットを書き換えることもできます。	実行するコマンド。
p_data	入力 / 出力	データ入出力のための void ポインタ。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ command

パラメータ p_data

const

7.26.7.13 readIo

ssp_err_t(* sdmmc_api_t::readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)

詳細説明

SDMMC チャネルから I/O データを読み取ります。また電力消費を低減できます。R_SDMMC_ReadIo

表 503: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SDMMC チャネルのハンドル (SDMMC チャネル デバイス コンテキストへのポインタ)。
p_data	out	データ読み出し先のデータ バッファへのポインタ。
function	複数のビットを書き換えることもできます。	SDIO 関数番号。
address	複数のビットを書き換えることもできます。	SDIO レジスタアドレス。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ **p_data**

uint8_t

パラメータ関数

uint32_t

パラメータ **address**

uint32_t

7.26.7.14 writeIo

ssp_err_t(* sdmmc_api_t::writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)

詳細説明

SDMMC チャネルに I/O データを書き込みます。また電力消費を低減できます。R_SDMMC_WriteIo

表 504:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）。
p_data	複数のビットを書き換えることもできます。	データ書き込み元のデータ バッファへのポインタ。
function	複数のビットを書き換えることもできます。	SDIO 関数番号。
address	複数のビットを書き換えることもできます。	SDIO レジスタアドレス。
read_after_write	複数のビットを書き換えることもできます。	書き込みの後に読み込みを行う場合は、true に設定します。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ **p_data**

uint8_t

パラメータ関数

uint32_t

パラメータ **address**

uint32_t

Parameter read_after_write

定義 : sdmmc_io_write_mode_tconst read_after_write

7.26.7.15 readIoExt

ssp_err_t(* sdmmc_api_t::readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

詳細説明

SDMMC チャネルから I/O 拡張データを読み取ります。また電力消費を低減できます。
R_SDMMC_ReadIoExt

表 505: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SDMMC チャネルのハンドル（SDMMC チャネル デバイス コンテキストへのポインタ）。
p_dest	out	データ読み出し先のデータ バッファへのポインタ。
function	複数のビットを書き換えることもできます。	SDIO 関数番号。
address	複数のビットを書き換えることもできます。	SDIO レジスタアドレス。
count	複数のビットを書き換えることもできます。	読み込むバイト数またはブロック数。
transfer_mode	複数のビットを書き換えることもできます。	バイト モード = 0、ブロック モード = 1。
address_mode	複数のビットを書き換えることもできます。	0 = 固定アドレス、1 = インクリメント アドレス。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ **p_dest**

uint8_t

パラメータ関数

uint32_t

パラメータ **address**

uint32_t

パラメータ カウント

uint32_t

パラメータ **transfer_mode**

パラメータ **address_mode**

7.26.7.16 writeIoExt

ssp_err_t(* sdmmc_api_t::writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

詳細説明

SDMMC チャネルに I/O 拡張データを書き込みます。また電力消費を低減できます。R_SDMMC_WriteIoExt

表 506: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）。
p_source	複数のビットを書き換えることもできます。	データ書き込み元のデータ バッファへのポインタ。
関数番号	複数のビットを書き換えることもできます。	SDIO 関数番号。
address	複数のビットを書き換えることもできます。	SDIO レジスタアドレス。
count	複数のビットを書き換えることもできます。	書き込むバイト数またはブロック数。

表 506: パラメータ (続き)

名前	方向	説明
transfer_mode	複数のビットを書き換えることもできます。	バイト モード = 0、ブロック モード = 1。
address_mode	複数のビットを書き換えることもできます。	0 = 固定アドレス、1 = インクリメント アドレス。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ p_source

uint8_t

パラメータ関数番号

パラメータ address

uint32_t

パラメータ カウント

uint32_t

パラメータ transfer_mode

パラメータ address_mode

7.26.7.17 IoIntEnable

ssp_err_t(* sdmmc_api_t::IoIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)

詳細説明

SDMMC チャネルに対して、SDIO 割り込みを有効化します。R_SDMMC_IoIntEnable として実装

表 507: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）。
enable	複数のビットを書き換えることもできます。	割り込みを有効化 = true、割り込みを無効化 = false.

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ enable

const

7.26.7.18 versionGet

ssp_err_t(* sdmmc_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

SDMMC ドライバのバージョンを返します。また電力消費を低減できます。R_SDMMC_VersionGet

表 508: パラメータ

名前	方向	説明
p_version	out	バージョン情報を返す先へのポインタ。

パラメータ p_version

7.26.7.19 infoGet

ssp_err_t(* sdmmc_api_t::infoGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)

詳細説明

SDMMC チャネルの情報を取得します。また電力消費を低減できます。R_SDMMC_InfoGet

表 509: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ）
p_info	out	カード固有のデータを返す先のメモリ アドレス。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ p_info

定義 : sdmmc_info_t*const p_info

メディア デバイスから取得したステータスなどの情報。

- sdmmc_info_t::card_type

  SD または MMC。

- sdmmc_info_t::ready

  準備完了または未完了

- sdmmc_info_t::hc

  true = カードが大容量カード

- sdmmc_info_t::sdio

  true = SDIO が存在

- sdmmc_info_t::write_protected

  カードが書き込み保護されています。

- sdmmc_info_t::transfer_in_progress

  DMA または DTC 転送ステータス。

- sdmmc_info_t::csd_version

  CSD のバージョン。

- sdmmc_info_t::device_type

  速度とデータ レート。

- sdmmc_info_t::bus_width

  現在のメディア バス幅。

- sdmmc_info_t::hs_timing

  高速ステータス。

- sdmmc_info_t::sdhi_rca

  相対カード アドレス。

- sdmmc_info_t::max_clock_rate

  メディア カードの最大クロック レート。

- sdmmc_info_t::clock_rate

  現在のクロック レート

- sdmmc_info_t::sector_size

  セクター サイズ

- sdmmc_info_t::sector_count

  セクター数

- sdmmc_info_t::erase_sector_count

  一度に消去する最大セクター数。

7.26.7.20 erase

ssp_err_t(* sdmmc_api_t::erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

詳細説明

SDMMC セクターを消去します。R_SDMMC_Erase として実装されます。

表 510: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルのハンドル （チャネル制御ブロックへのポインタ)。
start_sector	複数のビットを書き換えることもできます。	消去を開始するセクター。
sector_count	複数のビットを書き換えることもできます。	消去するセクター数。

定義 : sdmmc_ctrl_t

パラメータ p_cfg

パラメータ start_sector

uint32_t

パラメータ sector_count

uint32_t

7.26.7.21 sdmmc_instance_t

sdmmc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- sdmmc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- sdmmc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- sdmmc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.27 SLCDC インタフェース

セグメント LCD コントローラのインタフェース。

7.27.1 概要

このドライバはセグメント LCD コントローラ （SLCDC）を使用してデータをセグメント LCD に表示します。

以下によって実装されます。SLCDC

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

SLCDC インタフェースの説明：セグメント LCD ドライバ

7.27.2 インタフェース API

slcdc_api_t

関数名	説明
.open	SLCD デバイスを開きます。
.write	SLCD セグメントにデータを書き込みます。初期表示データを指定します。8 時分割以外の場合、A パターン領域に書き込む際は下位 4 ビット、B パターン領域に書き込む際は上位 4 ビットに値を格納します。表示データは、表示データ レジスタに格納されます。
.modify	SLCD セグメントのデータを書き換えます。LCD の表示データがビット単位で書き換えられます。指定されたビット以外の値はそのまま保持されます。書き換えるデータを指定します
.start	SLCD での表示を有効にします。指定されたデータが LCD に表示されます。データを表示する前に、セグメントにデータを書き込んでおく必要があります。
.stop	SLCD での表示を無効化します。これにより、SLCD でのデータの表示が停止します。

関数名	説明
.contrastIncrease	表示コントラストを上げます。コントラストを 1 単位ずつ上昇させます。この関数は、駆動電圧生成回路に内部昇圧方式を使用している場合に選択できます
.contrastDecrease	表示コントラストを下げます。コントラストを 1 単位ずつ低下させます。この関数は、駆動電圧生成回路に内部昇圧方式を使用している場合に選択できます
.setdisplayArea	LCD の表示領域を設定します。この関数では、指定に従って、表示領域が A パターン領域か B パターン領域のいずれかに設定されます。この関数を使用して、A パターン領域と B パターン領域のデータを交互に表示する点滅表示を設定できます。点滅を使用する場合、この関数を実行する前に RTC を操作する必要があります。RTC を設定するには、次の手順を実行します。1）RTC を開きます。2）周期的割り込み要求を 1/2 秒に設定します。3）RTC カウンタを開始します。4）IRQ、RTC_EVENT_PERIODIC_IRQ を設定します。詳細な手順については、『ユーザーズ マニュアル : ハードウェア』を参照してください。
.close	表示デバイスを閉じます。
.versionGet	バージョンを取得します。

7.27.3 データ構造体

- slcdc_cfg_t
- slcdc_ctrl_t
- slcdc_instance_t

7.27.4 列挙

- slcdc_display_state_t
- slcdc_bias_method_t
- slcdc_time_slice_t
- slcdc_wave_form_t
- slcdc_drive_volt_gen_t
- slcdc_display_area_control_blink_t

- slcdc_display_area_t
- slcdc_display_on_off_t
- slcdc_display_enable_disable_t
- slcdc_display_clock_t
- slcdc_clk_div_t

7.27.5 型定義

- slcdc_size_t

7.27.6 定義

- #define SLCDC_API_VERSION_MAJOR

  初期値 :(1)

  定義、共通サービス、およびエラー コードを登録します。
- #define SLCDC_API_VERSION_MINOR

  初期値 :(1)
- #define SLCDC_VOL_MIN

  初期値 :(4)
- #define SLCDC_VOL_MAX

  初期値 :(19)
- #define SLCDC_VOL_MAX_4BIAS

  初期値 :(10)
- #define SLCDC_CFG_REF_VCC

  初期値 :(12)
- #define SLCDC_BOOST_COUNTER

  初期値 :(5) /* The number of times of performing waiting for 100 ms */
- #define SLCDC_BOOST_WAIT

  初期値 :(uint32_t)(100000) /* Waiting for 100ms */
- #define SLCDC_SETUP_WAIT

  初期値 :(5) /* Waiting for 5ms */
- #define MAX_NUM_SEG

  初期値 :51

7.27.7 API データ

7.27.7.1 slcdc_display_state_t

slcdc_display_state_t

詳細説明

表示インタフェース操作の状態

列挙値

名前	説明
SLCDC_DISPLAY_STATE_CLOSED	表示が閉じています。
SLCDC_DISPLAY_STATE_OPENED	表示が開いています。

7.27.7.2 slcdc_bias_method_t

slcdc_bias_method_t

詳細説明

LCD 表示のバイアス法

列挙値

名前	説明
SLCDC_BIAS_2	1/2 バイアス法
SLCDC_BIAS_3	1/3 バイアス法
SLCDC_BIAS_4	1/4 バイアス法

7.27.7.3 slcdc_time_slice_t

slcdc_time_slice_t

詳細説明

LCD 表示の時分割数

列挙値

名前	説明
SLCDC_STATIC	スタティック。
SLCDC_SLICE_2	2 時分割
SLCDC_SLICE_3	3 時分割
SLCDC_SLICE_4	4 時分割
SLCDC_SLICE_8	8 時分割

7.27.7.4 slcdc_wave_form_t

slcdc_wave_form_t

詳細説明

LCD 表示波形選択

列挙値

名前	説明
SLCDC_WAVE_A	A 波形。
SLCDC_WAVE_B	B 波形。

7.27.7.5 slcdc_drive_volt_gen_t

slcdc_drive_volt_gen_t

詳細説明

LCD 駆動電圧生成回路選択

列挙値

名前	説明
SLCDC_VOLT_EXTERNAL	外部抵抗分割方式。
SLCDC_VOLT_INTERNAL	内部昇圧方式。

名前	説明
SLCDC_VOLT_CAPACITOR	容量分割方式。

7.27.7.6 slcdc_display_area_control_blink_t

slcdc_display_area_control_blink_t

詳細説明

表示データ領域の制御

列挙値

名前	説明
SLCDC_NOT_BLINKING	A パターン領域および B パターン領域のデータを交互に表示します。
SLCDC_BLINKING	A パターン領域または B パターン領域のデータを表示します。

7.27.7.7 slcdc_display_area_t

slcdc_display_area_t

詳細説明

表示領域データ

列挙値

名前	説明
SLCDC_DISP_A	A パターン領域のデータを表示します。
SLCDC_DISP_B	B パターン領域のデータを表示します。
SLCDC_DISP_BLINK	

7.27.7.8 slcdc_display_on_off_t

slcdc_display_on_off_t

詳細説明

LCD 表示をオン / オフします。

列挙値

名前	説明
SLCDC_DISP_OFF	表示オフ。
SLCDC_DISP_ON	表示オン。

7.27.7.9 slcdc_display_enable_disable_t

slcdc_display_enable_disable_t

詳細説明

LCD 表示出力有効化

列挙値

名前	説明
SLCDC_DISP_DISABLE	セグメント / 共通ピンの出力グラウンド レベル。
SLCDC_DISP_ENABLE	出力有効化。

7.27.7.10 slcdc_display_clock_t

slcdc_display_clock_t

詳細説明

LCD 表示クロック選択

列挙値

名前	説明
SLCDC_CLOCK_LOCO	クロック ソース LOCO を表示します。
SLCDC_CLOCK_SOSC	クロック ソース SOSC を表示します。
SLCDC_CLOCK_MOSC	クロック ソース MOSC を表示します。

名前	説明
SLCDC_CLOCK_HOCO	クロック ソース HOCO を表示します。

7.27.7.11 slcdc_clk_div_t

slcdc_clk_div_t

詳細説明

LCD クロック設定値

列挙値

名前	説明
SLCDC_CLK_DIVISOR_LOCO_4	LOCO クロック /4。
SLCDC_CLK_DIVISOR_LOCO_8	LOCO クロック /8。
SLCDC_CLK_DIVISOR_LOCO_16	LOCO クロック /16。
SLCDC_CLK_DIVISOR_LOCO_32	LOCO クロック /32。
SLCDC_CLK_DIVISOR_LOCO_64	LOCO クロック /64。
SLCDC_CLK_DIVISOR_LOCO_128	LOCO クロック /128。
SLCDC_CLK_DIVISOR_LOCO_256	LOCO クロック /256。
SLCDC_CLK_DIVISOR_LOCO_512	LOCO クロック /512。
SLCDC_CLK_DIVISOR_LOCO_1024	LOCO クロック /1024。
SLCDC_CLK_DIVISOR_HOCO_256	HOCO クロック /256。
SLCDC_CLK_DIVISOR_HOCO_512	HOCO クロック /512。
SLCDC_CLK_DIVISOR_HOCO_1024	HOCO クロック /1024。
SLCDC_CLK_DIVISOR_HOCO_2048	HOCO クロック /2048。
SLCDC_CLK_DIVISOR_HOCO_4096	HOCO クロック /4096。
SLCDC_CLK_DIVISOR_HOCO_8192	HOCO クロック /8192。

名前	説明
SLCDC_CLK_DIVISOR_HOCO_16384	HOCO クロック /16384。
SLCDC_CLK_DIVISOR_HOCO_32768	HOCO クロック /32768。
SLCDC_CLK_DIVISOR_HOCO_65536	HOCO クロック /65536。
SLCDC_CLK_DIVISOR_HOCO_131072	HOCO クロック /131072。
SLCDC_CLK_DIVISOR_HOCO_262144	HOCO クロック /262144。
SLCDC_CLK_DIVISOR_HOCO_524288	HOCO クロック /524288。

7.27.7.12 slcdc_size_t

typedef uint8_t slcdc_size_t

詳細説明

slcdc のサイズ定義

7.27.8 API 構造

7.27.8.1 slcdc_cfg_t

slcdc_cfg_t

詳細説明

SLCDC 設定ブロック

変数

- slcdc_display_clock_t slcdc_clock

  LCD クロック ソース （LCDSCKSEL）
- slcdc_clk_div_t slcdc_clock_setting

  LCD クロック設定値 （LCDC0）
- slcdc_bias_method_t bias_method

  LCD の表示バイアス法選択 （LBAS ビット）
- slcdc_time_slice_t time_slice

  LCD 表示の時分割数選択 （LDTY ビット）

- slcdc_wave_form_t wave_form

  LCD 表示波形選択 （LWAVE ビット）

- slcdc_drive_volt_gen_t drive_volt_gen

  LCD 駆動電圧生成回路選択 （MDSTET ビット）

7.27.8.2 slcdc_ctrl_t

slcdc_ctrl_t

詳細説明

パラメータ start_segment

変数

- slcdc_display_state_t state

  SLCD モジュールのステータス。

- slcdc_cfg_t info

  SLCDC 設定情報。

- void const * p_context

  ハイレベルのデバイス コンテキストへのポインタ。

7.27.8.3 slcdc_api_t

slcdc_api_t

詳細説明

HAL レイヤーに実装された SLCDC 関数は、この API に従います。

7.27.8.4 open

ssp_err_t(* `slcdc_api_t`::open)(`slcdc_ctrl_t` *const p_ctrl, `slcdc_cfg_t` const *const p_cfg)

詳細説明

SLCD デバイスを開きます。また電力消費を低減できます。

- R_SLCDC_Open

表 511: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	表示インタフェース制御ブロックへのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	表示設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : slcdc_ctrl_t

パラメータ start_segment

定義 :

定義 : slcdc_cfg_t const *const p_cfg

SLCDC 設定ブロック

- slcdc_cfg_t::slcdc_display_clock_t

  LCD クロック ソース （LCDSCKSEL）

  列挙値 :

  – SLCDC_CLOCK_LOCO

  – SLCDC_CLOCK_SOSC

  – SLCDC_CLOCK_MOSC

  – SLCDC_CLOCK_HOCO

- slcdc_cfg_t::slcdc_clk_div_t

  LCD クロック設定値 （LCDC0）

  列挙値 :

  – SLCDC_CLK_DIVISOR_LOCO_4

  – SLCDC_CLK_DIVISOR_LOCO_8

  – SLCDC_CLK_DIVISOR_LOCO_16

  – SLCDC_CLK_DIVISOR_LOCO_32

  – SLCDC_CLK_DIVISOR_LOCO_64

  – SLCDC_CLK_DIVISOR_LOCO_128

- SLCDC_CLK_DIVISOR_LOCO_256
- SLCDC_CLK_DIVISOR_LOCO_512
- SLCDC_CLK_DIVISOR_LOCO_1024
- SLCDC_CLK_DIVISOR_HOCO_256
- SLCDC_CLK_DIVISOR_HOCO_512
- SLCDC_CLK_DIVISOR_HOCO_1024
- SLCDC_CLK_DIVISOR_HOCO_2048
- SLCDC_CLK_DIVISOR_HOCO_4096
- SLCDC_CLK_DIVISOR_HOCO_8192
- SLCDC_CLK_DIVISOR_HOCO_16384
- SLCDC_CLK_DIVISOR_HOCO_32768
- SLCDC_CLK_DIVISOR_HOCO_65536
- SLCDC_CLK_DIVISOR_HOCO_131072
- SLCDC_CLK_DIVISOR_HOCO_262144
- SLCDC_CLK_DIVISOR_HOCO_524288

• slcdc_cfg_t::slcdc_bias_method_t

LCD の表示バイアス法選択 （LBAS ビット）

列挙値 :

- SLCDC_BIAS_2
- SLCDC_BIAS_3
- SLCDC_BIAS_4

• slcdc_cfg_t::slcdc_time_slice_t

LCD 表示の時分割数選択 （LDTY ビット）

列挙値 :

- SLCDC_STATIC
- SLCDC_SLICE_2
- SLCDC_SLICE_3
- SLCDC_SLICE_4
- SLCDC_SLICE_8

- slcdc_cfg_t::slcdc_wave_form_t

  LCD 表示波形選択 （LWAVE ビット）

  列挙値 :

  – SLCDC_WAVE_A

  – SLCDC_WAVE_B

- slcdc_cfg_t::slcdc_drive_volt_gen_t

  LCD 駆動電圧生成回路選択 （MDSTET ビット）

  列挙値 :

  – SLCDC_VOLT_EXTERNAL

  – SLCDC_VOLT_INTERNAL

  – SLCDC_VOLT_CAPACITOR

7.27.8.5 write

ssp_err_t(* slcdc_api_t::write)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)

詳細説明

SLCD セグメントにデータを書き込みます。初期表示データを指定します。8 時分割以外の場合、A パターン領域に書き込む際は下位 4 ビット、B パターン領域に書き込む際は上位 4 ビットに値を格納します。表示データは、表示データ レジスタに格納されます。また電力消費を低減できます。

- R_SLCDC_Write

表 512: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
start_segment	複数のビットを書き換えることもできます。	書き込み先の開始セグメント番号を指定します。
p_data	複数のビットを書き換えることもできます。	指定したセグメントに書き込む表示データへのポインタ
segment_count	複数のビットを書き換えることもできます。	書き込むセグメント数

定義 : slcdc_ctrl_t

パラメータ start_segment

定義 :

定義 : slcdc_size_tconst start_segment

slcdc のサイズ定義

パラメータ p_data

定義 : slcdc_size_tconst *const p_data

slcdc のサイズ定義

パラメータ segment_count

定義 : slcdc_size_tconst segment_count

slcdc のサイズ定義

7.27.8.6 modify

ssp_err_t(* slcdc_api_t::modify)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment, slcdc_size_t const data_mask, slcdc_size_t const data)

詳細説明

SLCD セグメントのデータを書き換えます。LCD の表示データがビット単位で書き換えられます。指定されたビット以外の値はそのまま保持されます。書き換えるデータを指定します。以下として実装されます。

- R_SLCDC_Modify

表 513: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
seg	複数のビットを書き換えることもできます。	書き込み先のセグメントを指定します。
data_mask	複数のビットを書き換えることもできます。	表示されているデータをマスクします。書き換えるビットを 0、その他のビットを 1 に設定します。複数のビットを書き換えることもできます。data_mask の設定値は、Bit 3 - 0xf7、Bit 2 - 0xfb、Bit 1 - 0xfd、Bit 0 - 0xfe です

表 513: パラメータ (続き)

名前	方向	説明
data	複数のビットを書き換えることもできます。	指定したセグメントに書き込む表示データを設定します。

定義 : slcdc_ctrl_t

パラメータ start_segment

パラメータ seg

定義 : slcdc_size_tconst segment

slcdc のサイズ定義

パラメータ data_mask

定義 : slcdc_size_tconst data_mask

slcdc のサイズ定義

パラメータ data

定義 : slcdc_size_tconst data

slcdc のサイズ定義

7.27.8.7 start

ssp_err_t(* slcdc_api_t::start)(slcdc_ctrl_t *const p_ctrl)

詳細説明

SLCD での表示を有効にします。指定されたデータが LCD に表示されます。データを表示する前に、セグメントにデータを書き込んでおく必要があります。また電力消費を低減できます。

- R_SLCDC_Start

表 514: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : slcdc_ctrl_t

パラメータ start_segment

7.27.8.8 stop

ssp_err_t(* slcdc_api_t::stop)(slcdc_ctrl_t *const p_ctrl)

詳細説明

SLCD での表示を無効化します。これにより、SLCD でのデータの表示が停止します。また電力消費を低減できます。

- R_SLCDC_Stop

表 515: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : slcdc_ctrl_t

パラメータ start_segment

7.27.8.9 contrastIncrease

ssp_err_t(* slcdc_api_t::contrastIncrease)(slcdc_ctrl_t *const p_ctrl)

詳細説明

表示コントラストを上げます。コントラストを 1 単位ずつ上昇させます。この関数は、駆動電圧生成回路に内部昇圧方式を使用している場合に選択できます。以下として実装されます。

- R_SLCDC_ContrastIncrease

表 516: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : slcdc_ctrl_t

パラメータ start_segment

7.27.8.10 contrastDecrease

ssp_err_t(* slcdc_api_t::contrastDecrease)(slcdc_ctrl_t *const p_ctrl)

詳細説明

表示コントラストを下げます。コントラストを 1 単位ずつ低下させます。この関数は、駆動電圧生成回路に内部昇圧方式を使用している場合に選択できます。以下として実装されます。

- R_SLCDC_ContrastDecrease

表 517: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : slcdc_ctrl_t

パラメータ start_segment

7.27.8.11 setdisplayArea

ssp_err_t(* slcdc_api_t::setdisplayArea)(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)

詳細説明

LCD の表示領域を設定します。この関数では、指定に従って、表示領域が A パターン領域か B パターン領域のいずれかに設定されます。この関数を使用して、A パターン領域と B パターン領域のデータを交互に表示する点滅表示を設定できます。

点滅を使用する場合、この関数を実行する前に RTC を操作する必要があります。RTC を設定するには、次の手順を実行します。1）RTC を開きます。2）周期的割り込み要求を 1/2 秒に設定します。3）RTC カウンタを開始します。4）IRQ、RTC_EVENT_PERIODIC_IRQ を設定します。詳細な手順については、『ユーザーズマニュアル : ハードウェア』を参照してください。

また電力消費を低減できます。

- R_SLCDC_SetdisplayArea()

表 518: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。
display_area	複数のビットを書き換えることもできます。	使用する表示領域。

定義 : slcdc_ctrl_t

パラメータ start_segment

パラメータ display_area

7.27.8.12 close

ssp_err_t(* slcdc_api_t::close)(slcdc_ctrl_t *const p_ctrl)

詳細説明

表示デバイスを閉じます。また電力消費を低減できます。

- R_SLCDC_Close

表 519: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	表示インタフェース制御ブロックへのポインタ。

定義 : slcdc_ctrl_t

パラメータ start_segment

7.27.8.13 versionGet

ssp_err_t(* slcdc_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

バージョンを取得します。また電力消費を低減できます。

- R_SLCDC_VersionGet

表 520: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン情報を格納するメモリへのポインタ。

パラメータ p_version

7.27.8.14 slcdc_instance_t

slcdc_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- slcdc_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- slcdc_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- slcdc_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.28 SPI インタフェース

SPI 通信用のインタフェース。

7.28.1 概要

この SPI インタフェースは、SPI バス API へのアクセスを提供します。このインタフェースは、SCI 上の簡易 SPI HAL レイヤー ドライバ モジュールを実装します。

以下によって実装されます。

- SPI
- SCI 上の簡易 SPI

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

SPI インタフェースの説明 : SPI ドライバ

7.28.2 インタフェース API

spi_api_t

関数名	説明
.open	SPI 通信モードのチャネルを初期化します。
.read	SPI デバイスからデータを受信します。
.write	SPI デバイスにデータを送信します。
.writeRead	SPI デバイスとの間で、データの送信と受信を同時に実行します （全二重通信）。
.close	ハンドルで指定された SPI チャネルへの電源を遮断し、関連付けられた割り込みを無効にします。
.versionGet	基礎となるドライバのバージョン情報を取得します。

7.28.3 データ構造体

- spi_callback_args_t
- spi_cfg_t
- spi_ctrl_t
- spi_instance_t

7.28.4 列挙

- spi_bit_width_t
- spi_mode_t
- spi_clk_phase_t
- spi_clk_polarity_t
- spi_mode_fault_t
- spi_bit_order_t
- spi_event_t

7.28.5 定義

- #define SPI_API_VERSION_MAJOR

  初期値 :(1)
- #define SPI_API_VERSION_MINOR

  初期値 :(1)

7.28.6 API データ

7.28.6.1 spi_bit_width_t

spi_bit_width_t

詳細説明

データ ビット幅

列挙値

名前	説明
SPI_BIT_WIDTH_8_BITS	データ バス幅は 8 ビット バイトです。
SPI_BIT_WIDTH_16_BITS	データ ビット幅は 16 ビット ワードです。
SPI_BIT_WIDTH_32_BITS	データ ビット幅は 32 ビット ロング ワードです。

7.28.6.2 spi_mode_t

spi_mode_t

詳細説明

マスターまたはスレーブのいずれかの動作モード

列挙値

名前	説明
SPI_MODE_MASTER	チャネルは SPI マスターとして動作します。
SPI_MODE_SLAVE	チャネルは SPI スレーブとして動作します。

7.28.6.3 spi_clk_phase_t

spi_clk_phase_t

詳細説明

クロック フェーズ

列挙値

名前	説明
SPI_CLK_PHASE_EDGE_ODD	0: 奇数エッジのデータ サンプリング、偶数エッジのデータ変化量
SPI_CLK_PHASE_EDGE_EVEN	1: 奇数エッジのデータ変化量、偶数エッジのデータ サンプリング

7.28.6.4 spi_clk_polarity_t

spi_clk_polarity_t

詳細説明

クロック極性

列挙値

名前	説明
SPI_CLK_POLARITY_LOW	0: アイドル時のクロック極性はロー
SPI_CLK_POLARITY_HIGH	1: アイドル時のクロック極性はハイ

7.28.6.5 spi_mode_fault_t

spi_mode_fault_t

詳細説明

モード障害エラー フラグ。このエラーは、デバイスがマスターとしてセットアップされているが、SSLA ラインがマスターによって制御されていない発生します。これは通常、接続されているデバイスもマスターとして動作している場合に起こります。同様の状況は、スレーブとして設定されている場合にも発生します。

列挙値

名前	説明
SPI_MODE_FAULT_ERROR_ENABLE	モード障害エラー フラグがオン。
SPI_MODE_FAULT_ERROR_DISABLE	モード障害エラー フラグがオフ。

7.28.6.6 spi_bit_order_t

spi_bit_order_t

詳細説明

ビット順序

列挙値

名前	説明
SPI_BIT_ORDER_MSB_FIRST	転送時に MSB から送信。
SPI_BIT_ORDER_LSB_FIRST	転送時に LSB から送信。

7.28.6.7 spi_event_t

spi_event_t

詳細説明

SPI イベント

列挙値

名前	説明
SPI_EVENT_TRANSFER_COMPLETE	データ転送が完了しました。
SPI_EVENT_TRANSFER_ABORTED	データ転送が中止されました。
SPI_EVENT_ERR_MODE_FAULT	モード障害エラー。
SPI_EVENT_ERR_READ_OVERFLOW	読み取りオーバーフロー エラー。
SPI_EVENT_ERR_PARITY	パリティ エラー。
SPI_EVENT_ERR_OVERRUN	オーバーラン エラー。
SPI_EVENT_ERR_FRAMING	フレーミング エラー。
SPI_EVENT_ERR_MODE_UNDERRUN	アンダーラン エラー。

7.28.7 API 構造

7.28.7.1 spi_callback_args_t

spi_callback_args_t

詳細説明

共通コールバック パラメータ定義

変数

- uint32_t channel

  デバイス チャネル番号。

- spi_event_t event

  イベント コード。

- void const * p_context

  コールバック時にユーザーに提供されるコンテキスト。

7.28.7.2 spi_cfg_t

spi_cfg_t

詳細説明

SPI インタフェース設定

変数

- uint32_t channel

  使用するチャネル番号。

- spi_mode_t operating_mode

  マスターまたはスレーブのいずれかの動作モードを選択します。

- spi_clk_phase_t clk_phase

  奇数または偶数クロック エッジのいずれかのデータ サンプリングを選択します。

- spi_clk_polarity_t clk_polarity

  アイドル時のクロック レベル。

- spi_mode_fault_t mode_fault

  モード障害エラー （マスターまたはスレーブ）のフラグ。

- spi_bit_order_t bit_order

  MSB ファーストと LSB ファーストのいずれかの送信順序を選択します。

- uint32_t bitrate

  ビット / 秒。

- transfer_instance_t const * p_transfer_tx

  SPI DTC/DMA 書き込み転送を使用する場合、ここに DTC/DMA インスタンスをリンクします。使用しない場合は、NULL に設定します。

- transfer_instance_t const * p_transfer_rx

  SPI DTC/DMA リード転送を使用する場合、ここに DTC/DMA インスタンスをリンクします。使用しない場合は、NULL に設定します。

- void(* p_callback)(spi_callback_args_t *p_args)

  ユーザー コールバック関数へのポインタ。

- void const * p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。

- void const * p_extend

  SPI ハードウェアに依存する拡張設定。

7.28.7.3 spi_ctrl_t

spi_ctrl_t

詳細説明

SPI インタフェース制御ブロック

変数

- uint8_t channel

  使用するチャネル番号。

- uint8_t current_slave

  現在割り当てられているスレーブ数。

- bool channel_opened

  ペリフェラルが初期化されたことを示す内部フラグ。

- transfer_instance_t const * p_transfer_tx

  SPI DTC/DMA 書き込み転送を使用します。

- transfer_instance_t const * p_transfer_rx

  SPI DTC/DMA リード転送を使用します。

- void(* p_callback)(spi_callback_args_t *p_args)

  ユーザー コールバック関数へのポインタ。

- void const * p_context

  ハイレベルのデバイス コンテキストへのポインタ。

7.28.7.4 spi_api_t

spi_api_t

詳細説明

SPI の共有インタフェース定義

7.28.7.5 open

ssp_err_t(* spi_api_t::open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

詳細説明

SPI 通信モードのチャネルを初期化します。また電力消費を低減できます。

- R_RSPI_Open
- R_SCI_SSPI_Open

表 521: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御ブロック用のユーザー定義のストレージへのポインタ。
p_cfg	複数のビットを書き換えることもできます。	SPI 設定構造体へのポインタ。

定義 : spi_ctrl_t

SPI インタフェース制御ブロック

定義 :

定義 : spi_cfg_t const *const p_cfg

SPI インタフェース設定

- spi_cfg_t::channel

  使用するチャネル番号。
- spi_cfg_t::spi_mode_t

  マスターまたはスレーブのいずれかの動作モードを選択します。

  列挙値 :
  - SPI_MODE_MASTER
  - SPI_MODE_SLAVE
- spi_cfg_t::spi_clk_phase_t

  奇数または偶数クロック エッジのいずれかのデータ サンプリングを選択します。

  列挙値 :

  - SPI_CLK_PHASE_EDGE_ODD
  - SPI_CLK_PHASE_EDGE_EVEN
- spi_cfg_t::spi_clk_polarity_t

  アイドル時のクロック レベル。

  列挙値 :
  - SPI_CLK_POLARITY_LOW
  - SPI_CLK_POLARITY_HIGH
- spi_cfg_t::spi_mode_fault_t

  モード障害エラー （マスターまたはスレーブ）のフラグ。

  列挙値 :
  - SPI_MODE_FAULT_ERROR_ENABLE
  - SPI_MODE_FAULT_ERROR_DISABLE
- spi_cfg_t::spi_bit_order_t

  MSB ファーストと LSB ファーストのいずれかの送信順序を選択します。

  列挙値 :
  - SPI_BIT_ORDER_MSB_FIRST
  - SPI_BIT_ORDER_LSB_FIRST
- spi_cfg_t::bitrate

  ビット / 秒。
- spi_cfg_t::transfer_instance_t

  SPI DTC/DMA 書き込み転送を使用する場合、ここに DTC/DMA インスタンスをリンクします。使用しない場合は、NULL に設定します。
- spi_cfg_t::transfer_instance_t

  SPI DTC/DMA リード転送を使用する場合、ここに DTC/DMA インスタンスをリンクします。使用しない場合は、NULL に設定します。
- spi_cfg_t::p_callback

  ユーザー コールバック関数へのポインタ。
- spi_cfg_t::p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。
- spi_cfg_t::p_extend

  SPI ハードウェアに依存する拡張設定。

7.28.7.6 read

ssp_err_t(* spi_api_t::read)(spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

詳細説明

SPI デバイスからデータを受信します。また電力消費を低減できます。

- R_RSPI_Read
- R_SCI_SSPI_Read

表 522: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの制御ブロックへのポインタ。
length	複数のビットを書き換えることもできます。	転送するデータの単位数 （単位サイズは bit_width によって指定)。
bit_width	複数のビットを書き換えることもできます。	転送データ ビット幅。
p_dest	out	SPI デバイスから受け取ったデータをコピーする宛先バッファへのポインタ。要求したデータ量を保持するための容量を確保することは、呼び出し側の責任です。

定義 : spi_ctrl_t

SPI インタフェース制御ブロック

パラメータ length

uint32_t

パラメータ bit_width

パラメータ p_dest

const

7.28.7.7 write

ssp_err_t(* spi_api_t::write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)

詳細説明

SPI デバイスにデータを送信します。また電力消費を低減できます。

- R_RSPI_Write
- R_SCI_SSPI_Write

表 523: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	SPI デバイス宛てにデータを送信するソース データ バッファへのポインタ。引数に NULL を指定することはできません。
length	複数のビットを書き換えることもできます。	転送するデータの単位数 （単位サイズは bit_width によって指定）。
bit_width	複数のビットを書き換えることもできます。	転送データ ビット幅。

定義 : `spi_ctrl_t`

SPI インタフェース制御ブロック

パラメータ p_src

const

パラメータ length

uint32_t

パラメータ bit_width

7.28.7.8 writeRead

ssp_err_t(* `spi_api_t`::writeRead)(`spi_ctrl_t` *const p_ctrl, void const *p_src, void const *p_dest, uint32_t const length, `spi_bit_width_t` const bit_width)

詳細説明

SPI デバイスとの間で、データの送信と受信を同時に実行します （全二重通信）。また電力消費を低減できます。

- R_RSPI_WriteRead

- R_SCI_SSPI_WriteRead

表 524: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	SPI デバイス宛てにデータを送信するソース データ バッファへのポインタ。引数に NULL を指定することはできません。
p_dest	out	SPI デバイスから受け取ったデータをコピーする宛先バッファへのポインタ。要求したデータ量を保持するための容量を確保することは、呼び出し側の責任です。引数に NULL を指定することはできません。
length	複数のビットを書き換えることもできます。	転送するデータの単位数 （単位サイズは bit_width によって指定)。
bit_width	複数のビットを書き換えることもできます。	転送データ ビット幅。

定義 : spi_ctrl_t

SPI インタフェース制御ブロック

パラメータ p_src

const

パラメータ p_dest

const

パラメータ length

uint32_t

パラメータ bit_width

7.28.7.9 close

ssp_err_t(* spi_api_t::close)(spi_ctrl_t *const p_ctrl)

詳細説明

ハンドルで指定された SPI チャネルへの電源を遮断し、関連付けられた割り込みを無効にします。また電力消費を低減できます。

- R_RSPI_Close
- R_SCI_SSPI_Close

表 525: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの制御ブロックへのポインタ。

定義 : `spi_ctrl_t`

SPI インタフェース制御ブロック

7.28.7.10 versionGet

ssp_err_t(* `spi_api_t`::versionGet)(ssp_version_t *p_version)

詳細説明

基礎となるドライバのバージョン情報を取得します。また電力消費を低減できます。

- R_RSPI_VersionGet
- R_SCI_SSPI_VersionGet

表 526: パラメータ

名前	方向	説明
p_version	out	バージョン番号を返すメモリ位置へのポインタ

パラメータ p_version

7.28.7.11 spi_instance_t

spi_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- spi_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- spi_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- spi_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.29 タイマインタフェース

タイマ機能用インタフェース。

7.29.1 概要

汎用タイマ インタフェースは、周期モード、ワンショット モード、およびフリーラン タイマ モードを含む標準のタイマ機能を提供します。タイマ サイクル （オーバーフローまたはアンダーフロー）が終了するごとに、割り込みをトリガーすることができます。

インスタンスが出力コンペア モードをサポートしている場合は、r_<instance>.h で定義された拡張構成 timer_on_<instance>_cfg_t で提供されます。

以下によって実装されます。

- GPT
- AGT

関連項目：入力キャプチャインタフェース

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

タイマ インタフェースの説明：タイマ ドライバ

7.29.2 インタフェース API

timer_api_t

関数名	説明
.open	初期設定。
.stop	カウンタを停止します。
.start	カウンタを開始します。
.reset	カウンタを初期値にリセットします。
.counterGet	現在のカウンタ値を取得し、指定されたポインタ p_value に格納します。
.periodSet	タイマが期限切れになるまでの時間を設定します。

関数名	説明
.dutyCycleSet	デューティ サイクルが期限切れになるまでの時間を設定します。
.infoGet	タイマが期限切れになるまでの時間をクロック カウント単位で取得し、指定されたポインタ p_period_counts に格納します。
.close	ドライバを再設定できるようになります。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.29.3 データ構造体

- timer_callback_args_t
- timer_ctrl_t
- timer_info_t
- timer_cfg_t
- timer_instance_t

7.29.4 列挙

- timer_event_t
- timer_status_t
- timer_mode_t
- timer_unit_t
- timer_pwm_unit_t
- timer_direction_t

7.29.5 型定義

- timer_size_t
- timer_period_t

7.29.6 定義

- #define TIMER_API_VERSION_MAJOR

  初期値 :((uint8_t)1)

- #define TIMER_API_VERSION_MINOR

  初期値 :((uint8_t)0)

7.29.7 API データ

7.29.7.1 timer_event_t

timer_event_t

詳細説明

コールバック関数をトリガー可能なイベント

列挙値

名前	説明
TIMER_EVENT_EXPIRED	要求されたタイマの遅延時間が期限切れになったか、タイマがラップ アラウンドしました。

7.29.7.2 timer_status_t

timer_status_t

詳細説明

infoGet から返される可能性のあるステータス値

列挙値

名前	説明
TIMER_STATUS_COUNTING	タイマは実行中です。
TIMER_STATUS_STOPPED	タイマは停止中です。

7.29.7.3 timer_mode_t

timer_mode_t

詳細説明

タイマ動作モード

列挙値

名前	説明
TIMER_MODE_PERIODIC	タイマは、遅延期間の後に再開始します。
TIMER_MODE_ONE_SHOT	タイマは、遅延期間の後に停止します。
TIMER_MODE_PWM	タイマは PWM 出力を生成します。

7.29.7.4 timer_unit_t

timer_unit_t

詳細説明

タイマ期間値の単位

列挙値

名前	説明
TIMER_UNIT_PERIOD_RAW_COUNTS	クロック カウント単位の期間。
TIMER_UNIT_PERIOD_NSEC	ナノ秒単位の期間。
TIMER_UNIT_PERIOD_USEC	マイクロ秒単位の期間。
TIMER_UNIT_PERIOD_MSEC	ミリ秒単位の期間。
TIMER_UNIT_PERIOD_SEC	秒単位の期間。
TIMER_UNIT_FREQUENCY_HZ	Hz 単位の周波数。
TIMER_UNIT_FREQUENCY_KHZ	kHz 単位の周波数。

7.29.7.5 timer_pwm_unit_t

timer_pwm_unit_t

詳細説明

タイマのデューティ サイクル値の単位

列挙値

名前	説明
TIMER_PWM_UNIT_RAW_COUNTS	クロック カウント単位の期間。
TIMER_PWM_UNIT_PERCENT	ディーティ サイクルに使用されるパーセント単位。
TIMER_PWM_UNIT_PERCENT_X_1000	特別な分解能用の 1000 倍したパーセント。

7.29.7.6 timer_direction_t

timer_direction_t

詳細説明

タイマカウントの方向

列挙値

名前	説明
TIMER_DIRECTION_DOWN	タイマカウントは増加します。
TIMER_DIRECTION_UP	タイマカウントは減少します。

7.29.7.7 timer_size_t

typedef uint32_t timer_size_t

詳細説明

サポートされている最大タイマ サイズ。現在、最大で 32 ビット タイマがサポートされています。16 ビット タイマを使用する場合、timer_size_t パラメータの下位 16 ビットのみを使用できます。16 ビットより大きい値を渡した場合、エラーが発生します。

7.29.7.8 timer_period_t

typedef timer_size_t timer_period_t

詳細説明

非推奨の機能です。期間には timer_size_t の使用をおすすめします。

7.29.8 API 構造

7.29.8.1 timer_callback_args_t

timer_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- void const * p_context

  ユーザー データのプレースホルダー。timer_cfg_t 内の timer_api_t::open 関数で設定されます。

- timer_event_t event

  このイベントを使用して、コールバックの原因 （オーバーフローまたはエラー）を特定できます。

7.29.8.2 timer_ctrl_t

timer_ctrl_t

詳細説明

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- void(* p_callback)(timer_callback_args_t *p_args)

  タイマ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL を使用します。

- void const * p_context

  ユーザー データのプレースホルダー。timer_callback_args_t 内のユーザー コールバックに渡されます。

- uint8_t channel

  チャネル番号。

7.29.8.3 timer_info_t

timer_info_t

詳細説明

タイマ リソースに関するさまざまな情報を格納するタイマ情報構造体

変数

- timer_direction_t count_direction

  タイマ リソースのクロックのカウント方向。

- uint32_t clock_frequency

  タイマ リソースのクロック周波数。

- timer_size_t period_counts

  タイマが期限切れになるまでの時間 （単位はクロック カウント）

- timer_status_t status

7.29.8.4 timer_cfg_t

timer_cfg_t

詳細説明

オープン関数で使用されるユーザー設定構造体

変数

- timer_mode_t mode

  timer_mode_t から列挙値を選択します。

- uint32_t period

  タイマが期限切れになるタイミングを定義します。フリーラン カウンタの場合は、TIMER_MAX_CLOCK に設定し、単位 TIMER_UNIT_PERIOD_RAW_COUNTS を使用してください

- timer_unit_t unit

  timer_cfg_t::period の単位。

- timer_size_t duty_cycle

  timer_cfg_t::duty_cycle_unit 単位のデューティ サイクル。

- timer_pwm_unit_t duty_cycle_unit

  timer_cfg_t::duty_cycle の単位。

- uint8_t channel

  ハードウェアのチャネル番号に対応するチャネルを選択します。

- bool autostart

  Open 呼び出しの間に開始するかどうかを指定します。True の場合は、Open 呼び出しの間に開始します。False の場合は、Open 呼び出しの間に開始しません。

- void(* p_callback)(timer_callback_args_t *p_args)

  タイマ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

- void const * p_context

  ユーザー データのプレースホルダー。timer_callback_args_t 内のユーザー コールバックに渡されます。

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.29.8.5 timer_api_t

timer_api_t

詳細説明

タイマ API 構造体。HAL レイヤーに実装された汎用タイマ関数は、この API に従います。

7.29.8.6 open

ssp_err_t(* `timer_api_t`::open)(`timer_ctrl_t` *const p_ctrl, `timer_cfg_t` const *const p_cfg)

詳細説明

初期設定。また電力消費を低減できます。

- R_GPT_TimerOpen
- R_AGT_TimerOpen

I：この関数を呼び出す前に、ペリフェラルのクロックおよび必要なすべての出力ピンを設定する必要があります。

I：この関数を呼び出した後に再構成を行うには、まず close を呼び出します。

表 527: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : timer_cfg_t const *const p_cfg

オープン関数で使用されるユーザー設定構造体

- timer_cfg_t::timer_mode_t

  timer_mode_t から列挙値を選択します。

  列挙値 :

  – TIMER_MODE_PERIODIC

  – TIMER_MODE_ONE_SHOT

  – TIMER_MODE_PWM

- timer_cfg_t::period

  タイマが期限切れになるタイミングを定義します。フリーラン カウンタの場合は、TIMER_MAX_CLOCK に設定し、単位 TIMER_UNIT_PERIOD_RAW_COUNTS を使用してください

- timer_cfg_t::timer_unit_t

  timer_cfg_t::period の単位。

  列挙値 :

  – TIMER_UNIT_PERIOD_RAW_COUNTS

  – TIMER_UNIT_PERIOD_NSEC

  – TIMER_UNIT_PERIOD_USEC

  – TIMER_UNIT_PERIOD_MSEC

  – TIMER_UNIT_PERIOD_SEC

  – TIMER_UNIT_FREQUENCY_HZ

– TIMER_UNIT_FREQUENCY_KHZ

- timer_cfg_t::timer_size_t

  timer_cfg_t::duty_cycle_unit 単位のデューティ サイクル。

- timer_cfg_t::timer_pwm_unit_t

  timer_cfg_t::duty_cycle の単位。

  列挙値 :

  – TIMER_PWM_UNIT_RAW_COUNTS

  – TIMER_PWM_UNIT_PERCENT

  – TIMER_PWM_UNIT_PERCENT_X_1000

- timer_cfg_t::channel

  ハードウェアのチャネル番号に対応するチャネルを選択します。

- timer_cfg_t::autostart

  Open 呼び出しの間に開始するかどうかを指定します。True の場合は、Open 呼び出しの間に開始します。False の場合は、Open 呼び出しの間に開始しません。

- timer_cfg_t::p_callback

  タイマ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

- timer_cfg_t::p_context

  ユーザー データのプレースホルダー。timer_callback_args_t 内のユーザー コールバックに渡されます。

- timer_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.29.8.7 stop

ssp_err_t(* timer_api_t::stop)(timer_ctrl_t *const p_ctrl)

詳細説明

カウンタを停止します。また電力消費を低減できます。

- R_GPT_Stop
- R_AGT_Stop

表 528: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.29.8.8 start

ssp_err_t(* timer_api_t::start)(timer_ctrl_t *const p_ctrl)

詳細説明

カウンタを開始します。また電力消費を低減できます。

- R_GPT_Start
- R_AGT_Start

l :autostart が true の場合、open 関数でカウンタを開始することもできます。

表 529: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.29.8.9 reset

ssp_err_t(* timer_api_t::reset)(timer_ctrl_t *const p_ctrl)

詳細説明

カウンタを初期値にリセットします。また電力消費を低減できます。

- R_GPT_Reset
- R_AGT_Reset

表 530:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : `timer_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.29.8.10 counterGet

ssp_err_t(* `timer_api_t`::counterGet)(`timer_ctrl_t` *const p_ctrl, `timer_size_t` *const p_value)

詳細説明

現在のカウンタ値を取得し、指定されたポインタ p_value に格納します。また電力消費を低減できます。

- R_GPT_CounterGet
- R_AGT_CounterGet

表 531:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。
p_value	out	現在のカウンタ値を格納するためのポインタ。

定義 : `timer_ctrl_t`

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_value

定義 : `timer_size_t`*const p_value

サポートされている最大タイマ サイズ。現在、最大で 32 ビット タイマがサポートされています。16 ビット タイマを使用する場合、timer_size_t パラメータの下位 16 ビットのみを使用できます。16 ビットより大きい値を渡した場合、エラーが発生します。

7.29.8.11 periodSet

ssp_err_t(* timer_api_t::periodSet)(timer_ctrl_t *const p_ctrl, timer_size_t const period, timer_unit_t const unit)

詳細説明

タイマが期限切れになるまでの時間を設定します。また電力消費を低減できます。

- R_GPT_PeriodSet
- R_AGT_PeriodSet

I：タイマが期限切れになったときに CPU 割り込みが発生するかどうかは、open のタイマの構成に依存します。

表 532: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。
period	複数のビットを書き換えることもできます。	タイマが時間切れになるまでの時間。
unit	複数のビットを書き換えることもできます。	period パラメータの単位。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ period

定義 : timer_size_tconst period

サポートされている最大タイマ サイズ。現在、最大で 32 ビット タイマがサポートされています。16 ビット タイマを使用する場合、timer_size_t パラメータの下位 16 ビットのみを使用できます。16 ビットより大きい値を渡した場合、エラーが発生します。

パラメータ **unit**

7.29.8.12 dutyCycleSet

ssp_err_t(* timer_api_t::dutyCycleSet)(timer_ctrl_t *const p_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)

詳細説明

デューティ サイクルが期限切れになるまでの時間を設定します。

I：この関数を使用する前に、pwm_api_t::open を呼び出してタイマを設定してください。タイマが期限切れになったときに CPU 割り込みが発生するかどうかは、pwm_api_t::open のタイマの設定に依存します。

表 533:　パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの pwm_api_t::open 呼び出しで設定された制御ブロック。
period	複数のビットを書き換えることもできます。	タイマが時間切れになるまでの時間。
unit	複数のビットを書き換えることもできます。	period パラメータの単位。
pin	複数のビットを書き換えることもできます。	更新する出力ピン。ピン番号を入力してください。ピンが文字で識別される場合は、A は 0、B は 1、C は 2 のように入力してください。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ **period**

パラメータ **unit**

パラメータ **pin**

uint8_t

7.29.8.13 infoGet

ssp_err_t(* timer_api_t::infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

詳細説明

タイマが期限切れになるまでの時間をクロック カウント単位で取得し、指定されたポインタ p_period_counts に格納します。また電力消費を低減できます。

- R_GPT_InfoGet
- R_AGT_InfoGet

表 534: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。
p_info	out	このタイマに関する一連の情報。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_info

定義 : timer_info_t*const p_info

タイマ リソースに関するさまざまな情報を格納するタイマ情報構造体

- timer_info_t::count_direction

  タイマ リソースのクロックのカウント方向。
- timer_info_t::clock_frequency

  タイマ リソースのクロック周波数。
- timer_info_t::period_counts

  タイマが期限切れになるまでの時間 （単位はクロック カウント）。
- timer_info_t::status

7.29.8.14 close

ssp_err_t(* timer_api_t::close)(timer_ctrl_t *const p_ctrl)

詳細説明

ドライバを再設定できるようになります。また電力消費を低減できます。

- R_GPT_Close
- R_AGT_Close

表 535: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	このタイマの open 呼び出しで設定された制御ブロック。

定義 : timer_ctrl_t

チャネル制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.29.8.15 versionGet

ssp_err_t(* timer_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_GPT_VersionGet
- R_AGT_VersionGet

表 536: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ p_version

7.29.8.16 timer_instance_t

timer_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- timer_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- timer_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- timer_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.30 転送インタフェース

データ転送関数用インタフェース。

7.30.1 概要

転送インタフェースは、バックグラウンドでのデータ転送をサポートします （CPU による介入不要）。

転送インタフェースは、以下のように実装できます。

- DTC
- DMAC

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

転送インタフェースの説明 : 転送ドライバ

7.30.2 インタフェース API

transfer_api_t

関数名	説明
.open	初期設定。auto_enable が true で、p_src、p_dest、length がすべて有効な場合は、転送が有効になります。enable または reset を使用して、転送を有効にすることもできます。
.reset	他のすべての設定値を維持したまま、ソース アドレス ポインタ、宛先アドレス ポインタ、長さを再設定できます。p_src、p_dest、length がすべて有効な場合は、転送が有効になります。
.enable	転送を有効にします。転送は、アクティベーション ソース イベントの後 （またはアクティベーション ソースとして ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 が選択されている場合は、start の呼び出し時）に実行されます。

関数名	説明
.disable	転送を無効にします。転送は、transfer_info_t::activation ソース イベントが発生しても （あるいは ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 に transfer_info_t::activation_source が選択されていて、start が呼び出されても）実行されません。
.start	転送をソフトウェア内で開始します。
.stop	転送をソフトウェア内で停止します。現在実行中の転送が完了した後、後続の転送が停止されます。
.infoGet	この転送に関する情報を提供します。
.close	これにより、以下を使用して転送を再設定できるようになります: これにより、open を使用して転送を再設定できるようになります。
.versionGet	バージョンを取得し、指定されたポインタ p_version に格納します。

7.30.3 データ構造体

- transfer_ctrl_t
- transfer_properties_t
- transfer_info_t
- transfer_callback_args_t
- transfer_cfg_t
- transfer_instance_t

7.30.4 列挙

- transfer_mode_t
- transfer_size_t
- transfer_addr_mode_t
- transfer_repeat_area_t
- transfer_chain_mode_t
- transfer_irq_t

- transfer_start_mode_t

7.30.5 定義

- #define TRANSFER_API_VERSION_MAJOR

  初期値 :(1)
- #define TRANSFER_API_VERSION_MINOR

  初期値 :(1)

7.30.6 API データ

7.30.6.1 transfer_mode_t

transfer_mode_t

詳細説明

転送モードは、転送要求が発生したときの動作を記述します。

列挙値

名前	説明
TRANSFER_MODE_NORMAL	ノーマル モードでは、転送要求が発生するたびに、ソース ポインタから宛先ポインタに transfer_size_t が送信されます。転送長がデクリメントされ、ソース ポインタとアドレス ポインタが transfer_addr_mode_t に従って更新されます。転送長が 0 に達した後は、転送要求があってもそれ以上の転送は行われません。
TRANSFER_MODE_REPEAT	リピート モードはノーマル モードとほぼ同じですが、ノーマル モードと違って、転送長が 0 に達すると、リピート領域へのポインタと転送長が初期値にリセットされます。DMAC を使用する場合、転送は transfer_info_t::num_blocks の回数のみリピートされます。転送のリピート回数が transfer_info_t::num_blocks に達した後は、転送要求があってもそれ以上の転送は行われません。DTC を使用する場合、転送のリピートは継続して行われます （リピート転送回数に制限はありません)。

名前	説明
TRANSFER_MODE_BLOCK	ブロック モードでは、転送要求が発生するたびに、transfer_size_t の transfer_info_t::length の転送が実行されます。各転送が終了するたびに、ソース ポインタと宛先ポインタが transfer_addr_mode_t に従って更新されます。ブロック転送の完了後、transfer_info_t::num_blocks がデクリメントされます。transfer_info_t::num_blocks が 0 に達した後は、転送要求があってもそれ以上の転送は行われません。

7.30.6.2 transfer_size_t

transfer_size_t

詳細説明

転送サイズは、個別の転送のサイズを指定します。合計転送長 = transfer_size_t * transfer_length_t

列挙値

名前	説明
TRANSFER_SIZE_1_BYTE	各転送は、8 ビット値を転送します。
TRANSFER_SIZE_2_BYTE	アドレス ポインタは、各転送後にインクリメントされます。
TRANSFER_SIZE_4_BYTE	アドレス ポインタは、各転送後にインクリメントされます。

7.30.6.3 transfer_addr_mode_t

transfer_addr_mode_t

詳細説明

アドレス モードは、各転送後にポインタを変更するかどうか（インクリメントまたはデクリメント）を指定します。

列挙値

名前	説明
TRANSFER_ADDR_MODE_FIXED	アドレス ポインタは、各転送後も固定されます。

名前	説明
TRANSFER_ADDR_MODE_INCREMENTED	アドレス ポインタは、各転送後、関連付けられた transfer_size_t だけインクリメントされます。
TRANSFER_ADDR_MODE_DECREMENTED	アドレス ポインタは、各転送後、関連付けられた transfer_size_t だけデクリメントされます。

7.30.6.4 transfer_repeat_area_t

transfer_repeat_area_t

詳細説明

繰り返し領域のオプション （ソースまたは宛先）。TRANSFER_MODE_REPEAT の場合、選択されたポインタは、length 転送の終了後、元の値に戻ります。TRANSFER_MODE_BLOCK の場合、選択されたポインタは、各転送の終了後、元の値に戻ります。

列挙値

名前	説明
TRANSFER_REPEAT_AREA_DESTINATION	TRANSFER_MODE_REPEAT または TRANSFER_MODE_BLOCK で繰り返される宛先領域。
TRANSFER_REPEAT_AREA_SOURCE	TRANSFER_MODE_REPEAT または TRANSFER_MODE_BLOCK で繰り返されるソース領域。

7.30.6.5 transfer_chain_mode_t

transfer_chain_mode_t

詳細説明

チェーン転送モードのオプション。
l :DTC の場合のみ適用されます。

列挙値

名前	説明
TRANSFER_CHAIN_MODE_DISABLED	チェーン モードを使用しません。

名前	説明
TRANSFER_CHAIN_MODE_EACH	この transfer_info_t からの単一の転送の後、次の転送に切り替えます。
TRANSFER_CHAIN_MODE_END	この transfer_info_t で定義されたすべての転送が完了した後、次の転送にチェーンします。

7.30.6.6 transfer_irq_t

transfer_irq_t

詳細説明

割り込みオプション

列挙値

名前	説明
TRANSFER_IRQ_END	割り込みは、最後の転送が終了した後でのみ発生します。この転送が後続の転送にチェーンする場合、後続のチェーンされた転送がすべて完了した後に割り込みが発生します。DTC は、アクティベーション ソースの割り込みをトリガーします。DTC で TRANSFER_IRQ_END を選択した場合、転送が完了するまでアクティベーション ソース割り込みは発生しません。
TRANSFER_IRQ_EACH	割り込みは、各転送後に発生します。一部の HAL ドライバでは使用できません。詳細については、HAL ドライバを参照してください。この設定では、1 つの転送が終了した後、次のアクティベーション ソースまで、チェーンされた後続の転送は実行されません。

7.30.6.7 transfer_start_mode_t

transfer_start_mode_t

詳細説明

ソフトウェア スタートによって一度限りの転送を行うか、繰り返し転送を行うかを選択します。

列挙値

名前	説明
TRANSFER_START_MODE_SINGLE	ソフトウェア スタートにより、一度限りの転送がトリガーされます。
TRANSFER_START_MODE_REPEAT	ソフトウェア スタートにより、すべてのデータが転送されるまで転送が継続します。

7.30.7 API 構造

7.30.7.1 transfer_ctrl_t

transfer_ctrl_t

詳細説明

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

変数

- uint32_t id

  ドライバ ID。

- elc_event_t trigger

  転送アクティベーション イベント。transfer_api_t::infoGet によって返されたイベントと照合されます。

- IRQn_Type irq

  一部の HAL ドライバではサポートされていません。

- uint8_t channel

  一部の HAL ドライバではサポートされていません。

7.30.7.2 transfer_properties_t

transfer_properties_t

詳細説明

ドライバ固有の情報。

変数

- uint32_t transfer_length_max

  転送の最大数。

- uint16_t transfer_length_remaining

  残りの転送数。

- bool in_progress

  この転送が実行中かどうかを示します。

7.30.7.3 transfer_info_t

transfer_info_t

詳細説明

この構造体は、転送のプロパティを指定します。
l a:DTC を使用する際、この構造体は、DTC が要求する記述子ブロック レジスタに対応します。以下のコンポーネントは、ドライバによって変更される可能性があります :

l a:p_src、p_dest、num_blocks、length。各転送には固有の が必要です。

l a: 転送ごとに、以下を一意に指定する必要があります .DTC を使用する際、この構造体を一時的な場所に配置する必要があります。この構造体のインスタンスは、それを使用する転送が終了するまでスコープ内に維持されます。

l :DTC を使用する場合、可能であれば、この構造体のインスタンスをメモリの保護されたセクションに配置してください。

変数

- uint32_t __pad0__
- uint32_t __pad1__
- transfer_addr_mode_t dest_addr_mode

  各転送が終了した後の、宛先ポインタの扱いを選択します。

- transfer_repeat_area_t repeat_area

  ソース領域または宛先領域を繰り返す場合に選択します。TRANSFER_MODE_NORMAL では使用されません。

- transfer_irq_t irq

  1 つの転送が完了するごとに、または予定したすべての転送が完了した後、割り込みを行う場合に選択します。

- transfer_chain_mode_t chain_mode

  チェーン転送が完了するタイミングを選択します。

- uint32_t __pad2__

- transfer_addr_mode_t src_addr_mode

  各転送が終了した後の、ソース ポインタの扱いを選択します。

- transfer_size_t size

  transfer_info_t::length で、一度に転送するバイト数を選択します。

- transfer_mode_t mode

  ソース ポインタ。

- void const *volatile p_src

  ソース ポインタ。

- void *volatile p_dest

  宛先ポインタ。

- uint16_t num_blocks

  TRANSFER_MODE_BLOCK （DTC および DMAC）と TRANSFER_MODE_REPEAT （DMAC のみ）を使用する場合の転送ブロック数。他のモードでは使用されません。

- uint16_t length

  各転送の長さ。詳細については、HAL ドライバを参照してください。

7.30.7.4 transfer_callback_args_t

transfer_callback_args_t

詳細説明

コールバック関数のパラメータ データ。

変数

- void const * p_context

  ユーザー データのプレースホルダー。transfer_cfg_t 内の r_transfer_t::open 関数で設定されます。

7.30.7.5 transfer_cfg_t

transfer_cfg_t

詳細説明

open で設定されたドライバ構成。.p_extend 以外のすべての要素を指定して、初期化する必要があります。

変数

- transfer_info_t * p_info

  転送設定オプションへのポインタ。チェーン転送を使用する場合 （DTC のみ）、チェーンする転送の配列を作成し、その配列へのポインタを指定して、転送を順に実行することができます。

- elc_event_t activation_source

  転送をトリガーするイベントを選択します。また ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 を選択して、ソフトウェア アクティベーションを指定します。DTC を使用する場合、これらのイベントをそれぞれ一度限りしか使用できないことがあります。これらのイベントのいずれかを選択すると、DMAC の内部ソフトウェア開始機能が使用されます。

- bool auto_enable

  オープン後に転送を有効にするかどうかを選択します。

- void(* p_callback)(transfer_callback_args_t *p_args)

  転送終了割り込みのコールバック。CPU 割り込みを行わない場合は、NULL に設定します。DTC ではサポートされていません。DTC 転送は、アクティベーション ソースに関連付けられた割り込みをトリガーします。

- void const * p_context

  ユーザー データのプレースホルダー。transfer_callback_args_t のユーザー p_callback に渡されます。

- void const * p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.30.7.6 transfer_api_t

transfer_api_t

詳細説明

HAL レイヤーに実装された転送関数は、この API に従います。

7.30.7.7 open

ssp_err_t(* `transfer_api_t`::open)(`transfer_ctrl_t` *const p_ctrl, `transfer_cfg_t` const *const p_cfg)

詳細説明

初期設定。auto_enable が true で、p_src、p_dest、length がすべて有効な場合は、転送が有効になります。enable または reset を使用して、転送を有効にすることもできます。また電力消費を低減できます。

- R_DTC_Open
- R_DMAC_Open

表 537: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	制御ブロックへのポインタ。ユーザーが宣言する必要があります。要素はここで設定されます。
p_cfg	複数のビットを書き換えることもできます。	設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : `transfer_ctrl_t`

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

定義 :

定義 : `transfer_cfg_t` const *const p_cfg

open で設定されたドライバ構成。.p_extend 以外のすべての要素を指定して、初期化する必要があります。

- `transfer_cfg_t`::`transfer_info_t`

  転送設定オプションへのポインタ。チェーン転送を使用する場合 （DTC のみ）、チェーンする転送の配列を作成し、その配列へのポインタを指定して、転送を順に実行することができます。

- `transfer_cfg_t`::`elc_event_t`

  転送をトリガーするイベントを選択します。また ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 を選択して、ソフトウェア アクティベーションを指定します。DTC を使用する場合、これらのイベントをそれぞれ一度限りしか使用できないことがあります。これらのイベントのいずれかを選択すると、DMAC の内部ソフトウェア開始機能が使用されます。

- `transfer_cfg_t`::auto_enable

  オープン後に転送を有効にするかどうかを選択します。

- `transfer_cfg_t`::p_callback

  転送終了割り込みのコールバック。CPU 割り込みを行わない場合は、NULL に設定します。DTC ではサポートされていません。DTC 転送は、アクティベーション ソースに関連付けられた割り込みをトリガーします。

- transfer_cfg_t::p_context

  ユーザー データのプレースホルダー。transfer_callback_args_t のユーザー p_callback に渡されます。

- transfer_cfg_t::p_extend

  ハードウェア固有の設定値に対応するための拡張パラメータです。

7.30.7.8 reset

ssp_err_t(* transfer_api_t::reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)

詳細説明

他のすべての設定値を維持したまま、ソース アドレス ポインタ、宛先アドレス ポインタ、長さを再設定できます。p_src、p_dest、length がすべて有効な場合は、転送が有効になります。また電力消費を低減できます。

- R_DTC_Reset
- R_DMAC_Reset

表 538: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。
p_src	複数のビットを書き換えることもできます。	ソースへのポインタ。ソース ポインタを変更しない場合は、NULL に設定します。
p_dest	複数のビットを書き換えることもできます。	宛先へのポインタ。宛先ポインタを変更しない場合は、NULL に設定します。
num_transfers	複数のビットを書き換えることもできます。	ノーマル モードでは転送長、ブロック モードではブロック数です。DMAC のみの場合、リピート モードのリピート数 （初期設定では num_blocks に保存されています）をリセットします。DTC のリピート モードでは使用されません。

定義 : transfer_ctrl_t

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

パラメータ **p_src**

const

パラメータ **p_dest**

const

パラメータ **num_transfers**

uint16_t

7.30.7.9 enable

ssp_err_t(* transfer_api_t::enable)(transfer_ctrl_t *const p_ctrl)

詳細説明

転送を有効にします。転送は、アクティベーション ソース イベントの後 （またはアクティベーション ソースとして ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 が選択されている場合は、start の呼び出し時）に実行されます。また電力消費を低減できます。

- R_DMAC_Enable
- R_DTC_Enable

表 539: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。

定義 : transfer_ctrl_t

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

7.30.7.10 disable

ssp_err_t(* transfer_api_t::disable)(transfer_ctrl_t *const p_ctrl)

詳細説明

転送を無効にします。転送は、transfer_info_t::activation ソース イベントが発生しても （あるいは ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 に transfer_info_t::activation_source が選択されていて、start が呼び出されても）実行されません。

I : 実行中の転送は、そのまま完了されます。後続の転送要求は実行されません。

また電力消費を低減できます。

- R_DMAC_Disable
- R_DTC_Disable

表 540: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。

定義 : `transfer_ctrl_t`

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

7.30.7.11 start

ssp_err_t(* `transfer_api_t`::start)(`transfer_ctrl_t` *const p_ctrl, `transfer_start_mode_t` mode)

詳細説明

転送をソフトウェア内で開始します。

l a:ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 に transfer_info_t::activation_source が選択されている場合のみ機能します。

l :DTC は、TRANSFER_START_MODE_SINGLE のみをサポートしています。
TRANSFER_START_MODE_REPEAT はサポートされていません。

また電力消費を低減できます。

- R_DMAC_Start
- R_DTC_Start

表 541: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。
mode	複数のビットを書き換えることもできます。	transfer_start_mode_t からモードを選択します。

定義 : `transfer_ctrl_t`

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

パラメータ **mode**

7.30.7.12 stop

ssp_err_t(* `transfer_api_t`::stop)(`transfer_ctrl_t` *const p_ctrl)

詳細説明

転送をソフトウェア内で停止します。現在実行中の転送が完了した後、後続の転送が停止されます。

l :DTC ではサポートされていません。

l :TRANSFER_START_MODE_REPEAT を使用して転送を開始された場合にのみ適用されます。

l a:ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 に transfer_info_t::activation_source が選択されている場合のみ機能します。

また電力消費を低減できます。

- R_DMAC_Stop

表 542: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。

定義 : transfer_ctrl_t

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

7.30.7.13 infoGet

ssp_err_t(* transfer_api_t::infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_info)

詳細説明

この転送に関する情報を提供します。また電力消費を低減できます。

- R_DTC_InfoGet
- R_DMAC_InfoGet

表 543: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。
p_info	out	ドライバ固有の情報。

定義 : transfer_ctrl_t

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

パラメータ p_info

定義 : transfer_properties_t*const p_info

ドライバ固有の情報。

- transfer_properties_t::transfer_length_max

  転送の最大数。
- transfer_properties_t::transfer_length_remaining

  残りの転送数。

- transfer_properties_t::in_progress

  この転送が実行中かどうかを示します。

7.30.7.14 close

ssp_err_t(* transfer_api_t::close)(transfer_ctrl_t *const p_ctrl)

詳細説明

これにより、以下を使用して転送を再設定できるようになります : これにより、open を使用して転送を再設定できるようになります。また電力消費を低減できます。

- R_DTC_Close
- R_DMAC_Close

表 544: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	この転送の open 呼び出しで設定された制御ブロック。

定義 : transfer_ctrl_t

初期化しないでください。初期化しないでください。この構造体は、open で初期化されます。

7.30.7.15 versionGet

ssp_err_t(* transfer_api_t::versionGet)(ssp_version_t *const p_version)

詳細説明

バージョンを取得し、指定されたポインタ p_version に格納します。また電力消費を低減できます。

- R_DTC_VersionGet
- R_DMAC_VersionGet

表 545: パラメータ

名前	方向	説明
p_version	out	使用されているコードおよび API のバージョン。

パラメータ **p_version**

7.30.7.16 transfer_instance_t

transfer_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- transfer_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- transfer_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- transfer_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.31 UART インタフェース

UART 通信用のインタフェース。

7.31.1 概要

UART インタフェースは、UART HAL ドライバ用の共通 API を提供します。UART インタフェースは、以下の機能を実装します。

- 全二重 UART 通信
- 汎用 UART パラメータ設定値
- 割り込み駆動型の送信 / 受信処理
- イベント コードを返すコールバック関数
- ランタイムのボー レートの変更
- トランザクション中のハードウェア リソース ロック
- CTS/RTS ハードウェア フロー制御のサポート （関連付けられた IOPORT ピンを使用）
- 循環バッファのサポート
- ランタイムの送信 / 受信循環バッファのフラッシュ

以下によって実装されます。

- SCI 上の UART

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

UART インタフェースの説明 : UART ドライバ

7.31.2 インタフェース API

uart_api_t

関数名	説明
.open	UART デバイスを開きます。

関数名	説明
.read	UART デバイスから読み取ります。p_transfer_rx が NULL の場合、読み取られたバイトは、p_callback で指定されるコールバック関数に渡されます。p_transfer_rx が NULL でない場合、読み取られたバイトは、リード入力バッファに直接書き込まれます。
.write	UART デバイスに書き込みます。
.baudSet	ボー レートを変更します。
.infoGet	ドライバ固有の情報を取得します。
.close	UART デバイスを閉じます。
.versionGet	バージョンを取得します。

7.31.3 データ構造体

- uart_info_t
- uart_callback_args_t
- uart_cfg_t
- uart_ctrl_t
- uart_instance_t

7.31.4 列挙

- uart_event_t
- uart_data_bits_t
- uart_parity_t
- uart_stop_bits_t

7.31.5 定義

- #define UART_API_VERSION_MAJOR

  初期値 :(1)
- #define UART_API_VERSION_MINOR

  初期値 :(1)

7.31.6 API データ

7.31.6.1 uart_event_t

uart_event_t

詳細説明

UART イベントコード

列挙値

名前	説明
UART_EVENT_RX_COMPLETE	完全なイベントを受信します。
UART_EVENT_TX_COMPLETE	完全なイベントを送信します。
UART_EVENT_ERR_PARITY	パリティ エラー イベント。
UART_EVENT_ERR_FRAMING	モード障害エラー イベント。
UART_EVENT_BREAK_DETECT	ブレーク検出エラー イベント。
UART_EVENT_ERR_OVERFLOW	FIFO オーバーフロー エラー イベント。
UART_EVENT_ERR_RXBUF_OVERFLOW	受信バッファ オーバーフロー エラー イベント。
UART_EVENT_RX_CHAR	受信された文字。

7.31.6.2 uart_data_bits_t

uart_data_bits_t

詳細説明

UART データのビット長定義

列挙値

名前	説明
UART_DATA_BITS_8	データ ビットは 8 ビット。
UART_DATA_BITS_7	データ ビットは 7 ビット。

名前	説明
UART_DATA_BITS_9	データ ビットは 9 ビット。

7.31.6.3 uart_parity_t

uart_parity_t

詳細説明

UART パリティ定義

列挙値

名前	説明
UART_PARITY_OFF	パリティなし。
UART_PARITY_ODD	奇数パリティ。
UART_PARITY_EVEN	偶数パリティ。

7.31.6.4 uart_stop_bits_t

uart_stop_bits_t

詳細説明

UART ストップ ビット定義

列挙値

名前	説明
UART_STOP_BITS_1	ストップ ビットは 1 ビット。
UART_STOP_BITS_2	ストップ ビットは 2 ビット。

7.31.7 API 構造

7.31.7.1 uart_info_t

uart_info_t

詳細説明

UART ドライバ固有の情報

変数

- uint32_t write_bytes_max

  一度に書き込むことができる最大バイト数。uart_cfg_t::p_transfer_tx が NULL でない場合のみ適用されます。

- uint32_t read_bytes_max

  一度に読み取ることができる最大バイト数。uart_cfg_t::p_transfer_rx が NULL でない場合のみ適用されます。

7.31.7.2 uart_callback_args_t

uart_callback_args_t

詳細説明

UART コールバック パラメータ定義

変数

- uint32_t channel

  デバイス チャネル番号。

- uart_event_t event

  イベント コード。

- uint32_t data

  汎用のデータ ストレージ。

- void const * p_context

  コールバック時にユーザーに提供されるコンテキスト。

7.31.7.3 uart_cfg_t

uart_cfg_t

詳細説明

UART 設定

変数

- uint32_t channel

  ハードウェアのチャネル番号に対応するチャネルを選択します。

- uint32_t baud_rate

  ボー レート （9600、19200、115200）

- uart_data_bits_t data_bits

  データのビット長 （8、7、9 のいずれか）

- uart_parity_t parity

  パリティ タイプ （パリティなし、奇数、偶数のいずれか）

- uart_stop_bits_t stop_bits

  ストップ ビット長 （1 または 2）

- bool ctsrts_en

  CTS/RTS ハードウェア フロー制御有効化。

- transfer_instance_t const * p_transfer_rx

  複数バイトを割り込みなく受信するために使用されるオプションの転送インスタンス。使用しない場合は、NULL に設定します。NULL の場合、読み取り API で許可されるバイト数は、一度に 1 バイトのみです。

- transfer_instance_t const * p_transfer_tx

  複数バイトを割り込みなく送信するために使用されるオプションの転送インスタンス。使用しない場合は、NULL に設定します。NULL の場合、書き込み API で許可されるバイト数は、一度に 1 バイトのみです。

- void(* p_callback)(uart_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void const * p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。

- void const * p_extend

  UART ハードウェアに依存する設定。

7.31.7.4 uart_ctrl_t

uart_ctrl_t

詳細説明

パラメータ uart_cfg_t

変数

- uint32_t channel

  チャネル番号。

- transfer_instance_t const * p_transfer_rx

  複数バイトを割り込みなく送信または受信するために使用されるオプションの転送インスタンス。

- transfer_instance_t const * p_transfer_tx

  複数バイトを割り込みなく送信または受信するために使用されるオプションの転送インスタンス。

- uint8_t const * p_tx_src

  送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファ ポインタ。

- uint32_t tx_src_bytes

  送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファ ポインタのサイズ。

- bool rx_transfer_in_progress

  この受信転送が実行中かどうかを示します。

- void(* p_callback)(uart_callback_args_t *p_args)

  コールバック関数へのポインタ。

- void const * p_context

  ハイレベルのデバイス コンテキストへのポインタ。

7.31.7.5 uart_api_t

uart_api_t

詳細説明

UART の共有インタフェース定義

7.31.7.6 open

ssp_err_t(* uart_api_t::open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)

詳細説明

UART デバイスを開きます。また電力消費を低減できます。

- R_SCI_UartOpen

表 546: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	UART 制御ブロックへのポインタ。ユーザーが宣言する必要があります。値はここで設定されます。

表 546: パラメータ (続き)

名前	方向	説明
uart_cfg_t	複数のビットを書き換えることもできます。	UART 設定構造体へのポインタ。この構造体のすべての要素は、ユーザーが設定する必要があります。

定義 : uart_ctrl_t

パラメータ uart_cfg_t

定義 :

定義 : uart_cfg_t const *const p_cfg

UART 設定

- uart_cfg_t::channel

  ハードウェアのチャネル番号に対応するチャネルを選択します。

- uart_cfg_t::baud_rate

  ボー レート （9600、19200、115200）

- uart_cfg_t::uart_data_bits_t

  データのビット長 （8、7、9 のいずれか）

  列挙値 :

  – UART_DATA_BITS_8

  – UART_DATA_BITS_7

  – UART_DATA_BITS_9

- uart_cfg_t::uart_parity_t

  パリティ タイプ （パリティなし、奇数、偶数のいずれか）

  列挙値 :

  – UART_PARITY_OFF

  – UART_PARITY_ODD

  – UART_PARITY_EVEN

- uart_cfg_t::uart_stop_bits_t

  ストップ ビット長 （1 または 2）

  列挙値 :

  – UART_STOP_BITS_1

  – UART_STOP_BITS_2

- uart_cfg_t::ctsrts_en

  CTS/RTS ハードウェア フロー制御有効化。

- uart_cfg_t::transfer_instance_t

  複数バイトを割り込みなく受信するために使用されるオプションの転送インスタンス。使用しない場合は、NULL に設定します。NULL の場合、読み取り API で許可されるバイト数は、一度に 1 バイトのみです。

- uart_cfg_t::transfer_instance_t

  複数バイトを割り込みなく送信するために使用されるオプションの転送インスタンス。使用しない場合は、NULL に設定します。NULL の場合、書き込み API で許可されるバイト数は、一度に 1 バイトのみです。

- uart_cfg_t::p_callback

  コールバック関数へのポインタ。

- uart_cfg_t::p_context

  コールバック関数に渡されるユーザー定義のコンテキスト。

- uart_cfg_t::p_extend

  UART ハードウェアに依存する設定。

7.31.7.7 read

ssp_err_t(* uart_api_t::read)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest, uint32_t const bytes)

詳細説明

UART デバイスから読み取ります。p_transfer_rx が NULL の場合、読み取られたバイトは、p_callback で指定されるコールバック関数に渡されます。p_transfer_rx が NULL でない場合、読み取られたバイトは、リード入力バッファに直接書き込まれます。また電力消費を低減できます。

- R_SCI_UartRead

表 547: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	チャネルの UART 制御ブロックへのポインタ。
p_dest	複数のビットを書き換えることもできます。	データの読み取り元の宛先アドレス。

表 547: パラメータ (続き)

名前	方向	説明
バイト	複数のビットを書き換えることもできます。	読み取りデータ長。p_transfer_rx が NULL でない場合のみ使用されます。NULL の場合、読み取られたすべてのバイトは、p_callback で設定されたコールバックを通じて提供されます。

定義 : uart_ctrl_t

パラメータ uart_cfg_t

パラメータ p_dest

uint8_t

パラメータ bytes

uint32_t

7.31.7.8 write

ssp_err_t(* uart_api_t::write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

詳細説明

UART デバイスに書き込みます。また電力消費を低減できます。

- R_SCI_UartWrite

表 548: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART 制御ブロックへのポインタ。
p_src	複数のビットを書き換えることもできます。	データを書き込むソース アドレス。
バイト	複数のビットを書き換えることもできます。	書き込みデータ長。

定義 : uart_ctrl_t

パラメータ uart_cfg_t

パラメータ **p_src**

uint8_t

パラメータ **bytes**

uint32_t

7.31.7.9 baudSet

ssp_err_t(* uart_api_t::baudSet)(uart_ctrl_t const *const p_ctrl, uint32_t const baudrate)

詳細説明

ボー レートを変更します。また電力消費を低減できます。

- R_SCI_UartBaudSet

表 549: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART 制御ブロックへのポインタ。
baudrate	複数のビットを書き換えることもできます。	ボー レート （単位 :

定義 : uart_ctrl_t

パラメータ uart_cfg_t

パラメータ **baudrate**

uint32_t

7.31.7.10 infoGet

ssp_err_t(* uart_api_t::infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

詳細説明

ドライバ固有の情報を取得します。また電力消費を低減できます。

- R_SCI_UartInfoGet

表 550: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART 制御ブロックへのポインタ。
baudrate	複数のビットを書き換えることもできます。	ボー レート （単位 :

定義 : uart_ctrl_t

パラメータ uart_cfg_t

パラメータ **baudrate**

7.31.7.11 close

ssp_err_t(* uart_api_t::close)(uart_ctrl_t *const p_ctrl)

詳細説明

UART デバイスを閉じます。また電力消費を低減できます。

- R_SCI_UartClose

表 551: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	UART 制御ブロックへのポインタ。

定義 : uart_ctrl_t

パラメータ uart_cfg_t

7.31.7.12 versionGet

ssp_err_t(* uart_api_t::versionGet)(ssp_version_t *p_version)

詳細説明

バージョンを取得します。また電力消費を低減できます。

- R_SCI_UartVersionGet

表 552: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン情報を格納するメモリへのポインタ。

パラメータ p_version

7.31.7.13 uart_instance_t

uart_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- uart_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- uart_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- uart_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

7.32 WDT インタフェース

ウォッチドッグ タイマ関数用のインタフェース。

7.32.1 概要

ウォッチドッグ タイマ（WDT）ペリフェラルの WDT インタフェースは、デバイスのリセットや割り込みの生成などのウォッチドッグ機能を提供します。

およびスレッド監視フレームワークインタフェースもあわせて参照してください。

ウォッチドッグ タイマ インタフェースは、以下のように実装できます。

- WDT
- IWDT

関連する SSP アーキテクチャのトピック：

- SSP インタフェース
- SSP 定義レイヤー
- SSP モジュールの使用

WDT インタフェースの説明：ウォッチドッグドライバ

7.32.2 インタフェース API

wdt_api_t

関数名	説明
.cfgGet	WDT をレジスタ スタート モードに初期化します。NMI 出力を使用したオートスタート モードの場合は、NMI コールバックが登録されます。
.open	WDT をレジスタ スタート モードに初期化します。NMI 出力を使用したオートスタート モードの場合は、NMI コールバックが登録されます。
.refresh	ウォッチドッグ タイマをリフレッシュします。
.statusGet	WDT のステータスを読み取ります。
.statusClear	WDT のステータス フラグをクリアします。
.counterGet	現在の WDT カウンタ値を読み取ります。

関数名	説明
.timeoutGet	ウォッチドッグのタイムアウト値を読み取ります。
.versionGet	IOPort ドライバのバージョンを返します。

7.32.3 データ構造体

- wdt_callback_args_t
- wdt_timeout_values_t
- wdt_ctrl_t
- wdt_cfg_t
- wdt_instance_t

7.32.4 列挙

- wdt_timeout_t
- wdt_clock_division_t
- wdt_window_start_t
- wdt_window_end_t
- wdt_reset_control_t
- wdt_stop_control_t
- wdt_status_t
- wdt_start_mode_t

7.32.5 定義

- #define WDT_API_VERSION_MAJOR

  初期値 :(1)
- #define WDT_API_VERSION_MINOR

  初期値 :(1)

7.32.6 API データ

7.32.6.1 wdt_timeout_t

wdt_timeout_t

詳細説明

WDT タイムアウト期間

列挙値

名前	説明
WDT_TIMEOUT_128	128 クロック サイクル
WDT_TIMEOUT_512	512 クロック サイクル
WDT_TIMEOUT_1024	1024 クロック サイクル
WDT_TIMEOUT_2048	2048 クロック サイクル
WDT_TIMEOUT_4096	4096 クロック サイクル
WDT_TIMEOUT_8192	8192 クロック サイクル
WDT_TIMEOUT_16384	16384 クロック サイクル

7.32.6.2 wdt_clock_division_t

wdt_clock_division_t

詳細説明

WDT クロック分割比

列挙値

名前	説明
WDT_CLOCK_DIVISION_1	CLK/1。
WDT_CLOCK_DIVISION_4	CLK/4。
WDT_CLOCK_DIVISION_16	CLK/16。

名前	説明
WDT_CLOCK_DIVISION_32	CLK/32。
WDT_CLOCK_DIVISION_64	CLK/64。
WDT_CLOCK_DIVISION_128	CLK/128。
WDT_CLOCK_DIVISION_256	CLK/256。
WDT_CLOCK_DIVISION_512	CLK/512。
WDT_CLOCK_DIVISION_2048	CLK/2048。
WDT_CLOCK_DIVISION_8192	CLK/8192。

7.32.6.3 wdt_window_start_t

wdt_window_start_t

詳細説明

WDT が許可期間ウィンドウの開始位置をリフレッシュします。

列挙値

名前	説明
WDT_WINDOW_START_25	開始位置 = 25%。
WDT_WINDOW_START_50	開始位置 = 50%。
WDT_WINDOW_START_75	開始位置 = 75%。
WDT_WINDOW_START_100	開始位置 = 100%。

7.32.6.4 wdt_window_end_t

wdt_window_end_t

詳細説明

WDT が許可期間ウィンドウの終了位置をリフレッシュします。

列挙値

名前	説明
WDT_WINDOW_END_75	終了位置 = 75%。
WDT_WINDOW_END_50	終了位置 = 50%。
WDT_WINDOW_END_25	終了位置 = 25%。
WDT_WINDOW_END_0	終了位置 = 0%。

7.32.6.5 wdt_reset_control_t

wdt_reset_control_t

詳細説明

WDT カウンタ アンダーフローとリフレッシュ エラー制御。

列挙値

名前	説明
WDT_RESET_CONTROL_NMI	カウンタ アンダーフローの場合は NMI 要求。
WDT_RESET_CONTROL_RESET	カウンタ アンダーフローの場合は、リセット要求。

7.32.6.6 wdt_stop_control_t

wdt_stop_control_t

詳細説明

スリープ モードでの WDT カウンタの動作。

列挙値

名前	説明
WDT_STOP_CONTROL_DISABLE	デバイスがスリープ モードになってもカウントは停止しません。
WDT_STOP_CONTROL_ENABLE	デバイスがスリープ モードになると、カウントは自動的に停止します。

7.32.6.7 wdt_status_t

wdt_status_t

詳細説明

WDT のステータス

列挙値

名前	説明
WDT_STATUS_NO_ERROR	ステータス フラグが設定されていません。
WDT_STATUS_UNDERFLOW_ERROR	アンダーフロー フラグが設定されています。
WDT_STATUS_REFRESH_ERROR	リフレッシュ エラー フラグが設定されています。許可されている範囲外でリフレッシュされました。
WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR	アンダーフロー フラグとリフレッシュ エラー フラグが設定されています。

7.32.6.8 wdt_start_mode_t

wdt_start_mode_t

詳細説明

WDT の開始モード。WDT が適切に設定されているかどうかを確認するために使用します。

列挙値

名前	説明
WDT_START_MODE_REGISTER	WDT は、WDT レジスタを使用して設定されます。
WDT_START_MODE_AUTO	WDT は、OFS0 ハードウェア レジスタを使用して設定されます。
WDT_START_MODE_DISABLED	WDT は無効です。

7.32.7 API 構造

7.32.7.1 wdt_callback_args_t

wdt_callback_args_t

詳細説明

コールバック関数のパラメータ データ

変数

- void const * p_context

  ユーザー データのプレースホルダー。wdt_cfg_t 内の wdt_api_t::open function 関数で設定されます。

7.32.7.2 wdt_timeout_values_t

wdt_timeout_values_t

詳細説明

WDT タイムアウト データ。WDT クロックの周波数とタイムアウト期間を返すために使用されます

変数

- uint32_t clock_frequency_hz

  分周器後のウォッチドッグ クロックの周波数。

- uint32_t timeout_clocks

  ウォッチドッグ クロック ティック単位のタイムアウト期間。

7.32.7.3 wdt_ctrl_t

wdt_ctrl_t

詳細説明

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

変数

- bool wdt_open

  open() API が正常に呼び出されたかどうかを示します。

- void const * p_context

  ユーザー データのプレースホルダー。wdt_callback_args_t 内のユーザー コールバックに渡されます。

- void(* p_callback)(wdt_callback_args_t *p_args)

  WDT NMI ISR の発生時に提供されるコールバック。

7.32.7.4 wdt_cfg_t

wdt_cfg_t

詳細説明

WDT 設定パラメータ。

変数

- wdt_start_mode_t start_mode

  WDT に対して指定するスタート モード。

- bool autostart

  true の場合、WDT がこの設定の一部として開始されます （レジスタ スタート モード)。false の場合、リフレッシュ API を呼び出して、手動で WDT を開始する必要があります。

- wdt_timeout_t timeout

  タイムアウト期間。

- wdt_clock_division_t clock_division

  クロック分周器。

- wdt_window_start_t window_start

  許可ウィンドウの開始位置をリフレッシュします。

- wdt_window_end_t window_end

  許可ウィンドウの終了位置をリフレッシュします。

- wdt_reset_control_t reset_control

  NMI を選択するか、アンダーフロー時にリセットを生成します。

- wdt_stop_control_t stop_control

  カウンタがスリープ モードで動作するかどうかを選択します。

- void(* p_callback)(wdt_callback_args_t *p_args)

  WDT NMI ISR の発生時に提供されるコールバック。

- void const * p_context

  ユーザー データのプレースホルダー。wdt_callback_args_t 内のユーザー コールバックに渡されます。

- void const * p_extend

  ユーザー拡張のプレースホルダー。

7.32.7.5 wdt_api_t

wdt_api_t

詳細説明

HAL レイヤーに実装された WDT 関数は、この API に従います。

7.32.7.6 cfgGet

ssp_err_t(* wdt_api_t::cfgGet)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)

詳細説明

WDT をレジスタ スタート モードに初期化します。NMI 出力を使用したオートスタート モードの場合は、NMI コールバックが登録されます。また電力消費を低減できます。

- R_WDT_CfgGet
- R_IWDT_CfgGet

表 553: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_cfg	out	WDT 設定を読み取るためのピン設定構造体へのポインタ。

定義 : wdt_ctrl_t

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : wdt_cfg_t *const p_cfg

WDT 設定パラメータ。

- wdt_cfg_t::wdt_start_mode_t

  WDT に対して指定するスタート モード。

  列挙値 :

  – WDT_START_MODE_REGISTER

  – WDT_START_MODE_AUTO

  – WDT_START_MODE_DISABLED

- wdt_cfg_t::autostart

  true の場合、WDT がこの設定の一部として開始されます （レジスタ スタート モード)。false の場合、リフレッシュ API を呼び出して、手動で WDT を開始する必要があります。

- wdt_cfg_t::wdt_timeout_t

  タイムアウト期間。

  列挙値 :

  – WDT_TIMEOUT_128
  – WDT_TIMEOUT_512
  – WDT_TIMEOUT_1024
  – WDT_TIMEOUT_2048
  – WDT_TIMEOUT_4096
  – WDT_TIMEOUT_8192
  – WDT_TIMEOUT_16384

- wdt_cfg_t::wdt_clock_division_t

  クロック分周器。

  列挙値 :

  – WDT_CLOCK_DIVISION_1
  – WDT_CLOCK_DIVISION_4
  – WDT_CLOCK_DIVISION_16
  – WDT_CLOCK_DIVISION_32
  – WDT_CLOCK_DIVISION_64
  – WDT_CLOCK_DIVISION_128
  – WDT_CLOCK_DIVISION_256
  – WDT_CLOCK_DIVISION_512
  – WDT_CLOCK_DIVISION_2048
  – WDT_CLOCK_DIVISION_8192

- wdt_cfg_t::wdt_window_start_t

  許可ウィンドウの開始位置をリフレッシュします。

  列挙値 :

  – WDT_WINDOW_START_25
  – WDT_WINDOW_START_50

  - WDT_WINDOW_START_75
  - WDT_WINDOW_START_100
- wdt_cfg_t::wdt_window_end_t

  許可ウィンドウの終了位置をリフレッシュします。

  列挙値 :
  - WDT_WINDOW_END_75
  - WDT_WINDOW_END_50
  - WDT_WINDOW_END_25
  - WDT_WINDOW_END_0
- wdt_cfg_t::wdt_reset_control_t

  NMI を選択するか、アンダーフロー時にリセットを生成します。

  列挙値 :
  - WDT_RESET_CONTROL_NMI
  - WDT_RESET_CONTROL_RESET
- wdt_cfg_t::wdt_stop_control_t

  カウンタがスリープ モードで動作するかどうかを選択します。

  列挙値 :
  - WDT_STOP_CONTROL_DISABLE
  - WDT_STOP_CONTROL_ENABLE
- wdt_cfg_t::p_callback

  WDT NMI ISR の発生時に提供されるコールバック。
- wdt_cfg_t::p_context

  ユーザー データのプレースホルダー。wdt_callback_args_t 内のユーザー コールバックに渡されます。
- wdt_cfg_t::p_extend

  ユーザー拡張のプレースホルダー。

7.32.7.7 open

ssp_err_t(* wdt_api_t::open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

詳細説明

WDT をレジスタ スタート モードに初期化します。NMI 出力を使用したオートスタート モードの場合は、NMI コールバックが登録されます。また電力消費を低減できます。

- R_WDT_Open
- R_IWDT_Open

表 554: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_cfg	複数のビットを書き換えることもできます。	ピン設定構造体へのポインタ。

定義 : wdt_ctrl_t

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

定義 :

定義 : wdt_cfg_t const *const p_cfg

WDT 設定パラメータ。

- wdt_cfg_t::wdt_start_mode_t

  WDT に対して指定するスタート モード。

  列挙値 :

  – WDT_START_MODE_REGISTER
  – WDT_START_MODE_AUTO
  – WDT_START_MODE_DISABLED

- wdt_cfg_t::autostart

  true の場合、WDT がこの設定の一部として開始されます（レジスタ スタート モード)。false の場合、リフレッシュ API を呼び出して、手動で WDT を開始する必要があります。

- wdt_cfg_t::wdt_timeout_t

  タイムアウト期間。

  列挙値 :

  – WDT_TIMEOUT_128
  – WDT_TIMEOUT_512
  – WDT_TIMEOUT_1024
  – WDT_TIMEOUT_2048
  – WDT_TIMEOUT_4096

  - WDT_TIMEOUT_8192
  - WDT_TIMEOUT_16384
- wdt_cfg_t::wdt_clock_division_t

  クロック分周器。

  列挙値 :
  - WDT_CLOCK_DIVISION_1
  - WDT_CLOCK_DIVISION_4
  - WDT_CLOCK_DIVISION_16
  - WDT_CLOCK_DIVISION_32
  - WDT_CLOCK_DIVISION_64
  - WDT_CLOCK_DIVISION_128
  - WDT_CLOCK_DIVISION_256
  - WDT_CLOCK_DIVISION_512
  - WDT_CLOCK_DIVISION_2048
  - WDT_CLOCK_DIVISION_8192
- wdt_cfg_t::wdt_window_start_t

  許可ウィンドウの開始位置をリフレッシュします。

  列挙値 :
  - WDT_WINDOW_START_25
  - WDT_WINDOW_START_50
  - WDT_WINDOW_START_75
  - WDT_WINDOW_START_100
- wdt_cfg_t::wdt_window_end_t

  許可ウィンドウの終了位置をリフレッシュします。

  列挙値 :
  - WDT_WINDOW_END_75
  - WDT_WINDOW_END_50
  - WDT_WINDOW_END_25
  - WDT_WINDOW_END_0

- wdt_cfg_t::wdt_reset_control_t

  NMI を選択するか、アンダーフロー時にリセットを生成します。

  列挙値 :

  – WDT_RESET_CONTROL_NMI

  – WDT_RESET_CONTROL_RESET

- wdt_cfg_t::wdt_stop_control_t

  カウンタがスリープ モードで動作するかどうかを選択します。

  列挙値 :

  – WDT_STOP_CONTROL_DISABLE

  – WDT_STOP_CONTROL_ENABLE

- wdt_cfg_t::p_callback

  WDT NMI ISR の発生時に提供されるコールバック。

- wdt_cfg_t::p_context

  ユーザー データのプレースホルダー。wdt_callback_args_t 内のユーザー コールバックに渡されます。

- wdt_cfg_t::p_extend

  ユーザー拡張のプレースホルダー。

7.32.7.8 refresh

ssp_err_t(* wdt_api_t::refresh)(wdt_ctrl_t *const p_ctrl)

詳細説明

ウォッチドッグ タイマをリフレッシュします。また電力消費を低減できます。

- R_WDT_Refresh
- R_IWDT_Refresh

I :WDT がオートスタート モードの場合は、この関数を使用する前に、OFS0 レジスタを設定する必要があります。

I a: この関数をレジスタ スタート モードで呼び出した後に R_WDT_Open を呼び出すと、WDT がデフォルトの状態で起動し、その後設定を変更することはできません。

表 555: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。

定義 : wdt_ctrl_t

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

7.32.7.9 statusGet

ssp_err_t(* wdt_api_t::statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)

詳細説明

WDT のステータスを読み取ります。また電力消費を低減できます。

- R_WDT_StatusGet
- R_IWDT_StatusGet

表 556: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_status	out	ステータス情報を返すための変数へのポインタ。

定義 : wdt_ctrl_t

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ p_status

定義 : wdt_status_t*const p_status

WDT のステータス

7.32.7.10 statusClear

ssp_err_t(* wdt_api_t::statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)

詳細説明

WDT のステータス フラグをクリアします。また電力消費を低減できます。

- R_WDT_StatusClear
- R_IWDT_StatusClear

表 557: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
status	複数のビットを書き換えることもできます。	クリアするステータス状態。

定義 : `wdt_ctrl_t`

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ status

7.32.7.11 counterGet

ssp_err_t(* `wdt_api_t`::counterGet)(`wdt_ctrl_t` *const p_ctrl, uint32_t *const p_count)

詳細説明

現在の WDT カウンタ値を読み取ります。また電力消費を低減できます。

- R_WDT_CounterGet
- R_IWDT_CounterGet

表 558: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_count	out	現在の WDT カウンタ値を返すための変数へのポインタ。

定義 : `wdt_ctrl_t`

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ **p_count**

uint32_t

7.32.7.12 timeoutGet

ssp_err_t(* wdt_api_t::timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)

詳細説明

ウォッチドッグのタイムアウト値を読み取ります。また電力消費を低減できます。

- R_WDT_TimeoutGet
- R_IWDT_TimeoutGet

表 559: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_timeout	out	タイムアウト値を返す構造体へのポインタ。

定義 : wdt_ctrl_t

WDT 制御ブロック。初期化しないでください。初期化は、open の呼び出し時に実行されます。

パラメータ **p_timeout**

定義 : wdt_timeout_values_t*const p_timeout

WDT タイムアウト データ。WDT クロックの周波数とタイムアウト期間を返すために使用されます

- wdt_timeout_values_t::clock_frequency_hz

  分周器後のウォッチドッグ クロックの周波数。
- wdt_timeout_values_t::timeout_clocks

  ウォッチドッグ クロック ティック単位のタイムアウト期間。

7.32.7.13 versionGet

ssp_err_t(* wdt_api_t::versionGet)(ssp_version_t *const p_data)

詳細説明

IOPort ドライバのバージョンを返します。また電力消費を低減できます。

- R_WDT_VersionGet
- R_IWDT_VersionGet

表 560: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	制御構造体へのポインタ。
p_data	out	バージョン情報を返す先のメモリ アドレス。

パラメータ p_data

定義 : `ssp_version_t` *const p_data

- `ssp_version_t`::version_id
  バージョン ID
- `ssp_version_t`::code_version_minor
  コードのマイナー バージョン。
- `ssp_version_t`::code_version_major
  コードのメジャー バージョン。
- `ssp_version_t`::api_version_minor
  API のマイナー バージョン。
- `ssp_version_t`::api_version_major
  API のメジャー バージョン。
- `ssp_version_t`struct{}
  コード バージョンのパラメータ

7.32.7.14 wdt_instance_t

wdt_instance_t

詳細説明

この構造体には、このインタフェースのインスタンスを使用するために必要なものがすべて含まれています。

変数

- wdt_ctrl_t * p_ctrl

  このインスタンスの制御構造体へのポインタ。

- wdt_cfg_t const * p_cfg

  イベント クラスのインスタンス範囲の始点。

- wdt_api_t const * p_api

  イベント クラスのインスタンス範囲の終点。

第 8 章 API リファレンス : HAL レイヤー

ハードウェア抽象化レイヤーは Renesas 周辺機器にドライバを提供します。HAL ドライバは通常、インタフェースを実装し、ハードウェア固有の API を追加提供します。

- ADC
- AGT
- CAC
- CAN
- CGC
- CRC
- CTSU
- DAC
- DMAC
- DOC
- DTC
- ELC
- 高性能フラッシュ
- ローパワー フラッシュ
- FMI
- GLCDC
- GPT
- GPT 入力キャプチャ
- ICU
- IOPORT
- IWDT
- JPEG CODEC
- キー割り込み
- LPM
- LVD

参考資料

- PDC
- QSPI
- IIC
- SPI
- RTC
- SCI 共通
- SCI 上の簡易 I2C
- SCI 上の簡易 SPI
- SCI 上の UART
- SDMMC
- SLCDC
- SSI
- WDT
- SCE モジュール

8.1 ADC

14 ビット A/D コンバータ （ADC14）と 12 ビット A/D コンバータ （ADC12）用ドライバ。

このモジュールは、ADC14 と ADC12 周辺機器をサポートします。次のインタフェースを実装します。

- ADC インタフェース

8.1.1 Functions

- r_adc_open_cfg_check_unique
- R_ADC_Open
- R_ADC_SetSampleStateCount
- R_ADC_ScanConfigure
- R_ADC_InfoGet
- R_ADC_ScanStart
- R_ADC_ScanStop
- R_ADC_CheckScanDone
- R_ADC_Read
- R_ADC_Close
- R_ADC_VersionGet

8.1.2 定義

- #define ADC_SAMPLE_STATE_COUNT_MIN

  初期値 :(5U)

  サンプル状態を変更するために使用可能な標準値

- #define ADC_SAMPLE_STATE_HOLD_COUNT_MIN

  初期値 :(4U)

  サンプルに使用可能で、チャネル 0 ～ 2 のカウントを保存可能な標準値。最小サンプルおよびホールド状態

- #define ADC_SAMPLE_STATE_HOLD_COUNT_DEFAULT

  初期値 :(24U)

  デフォルト サンプルおよびホールド状態

- #define ADC_MASK_CHANNEL_0

  初期値 :(1<<0U)

  ADC スキャン構成 adc_channel_cfg_t::scan_mask および add_mask、sample_hold_mask の場合は、OR 演算子を使用して必要なチャネルとセンサー用のマスクを組み合わせます。
- #define ADC_MASK_CHANNEL_1

  初期値 :(1<<1U)
- #define ADC_MASK_CHANNEL_2

  初期値 :(1<<2U)
- #define ADC_MASK_CHANNEL_3

  初期値 :(1<<3U)
- #define ADC_MASK_CHANNEL_4

  初期値 :(1<<4U)
- #define ADC_MASK_CHANNEL_5

  初期値 :(1<<5U)
- #define ADC_MASK_CHANNEL_6

  初期値 :(1<<6U)
- #define ADC_MASK_CHANNEL_7

  初期値 :(1<<7U)
- #define ADC_MASK_CHANNEL_8

  初期値 :(1<<8U)
- #define ADC_MASK_CHANNEL_9

  初期値 :(1<<9U)
- #define ADC_MASK_CHANNEL_10

  初期値 :(1<<10U)
- #define ADC_MASK_CHANNEL_11

  初期値 :(1<<11U)
- #define ADC_MASK_CHANNEL_12

  初期値 :(1<<12U)
- #define ADC_MASK_CHANNEL_13

  初期値 :(1<<13U)
- #define ADC_MASK_CHANNEL_14

  初期値 :(1<<14U)

- #define ADC_MASK_CHANNEL_15
  初期値 :(1<<15U)
- #define ADC_MASK_CHANNEL_16
  初期値 :(1<<16U)
- #define ADC_MASK_CHANNEL_17
  初期値 :(1<<17U)
- #define ADC_MASK_CHANNEL_18
  初期値 :(1<<18U)
- #define ADC_MASK_CHANNEL_19
  初期値 :(1<<19U)
- #define ADC_MASK_CHANNEL_20
  初期値 :(1<<20U)
- #define ADC_MASK_CHANNEL_21
  初期値 :(1<<21U)
- #define ADC_MASK_CHANNEL_22
  初期値 :(1<<22U)
- #define ADC_MASK_CHANNEL_23
  初期値 :(1<<23U)
- #define ADC_MASK_CHANNEL_24
  初期値 :(1<<24U)
- #define ADC_MASK_CHANNEL_25
  初期値 :(1<<25U)
- #define ADC_MASK_CHANNEL_26
  初期値 :(1<<26U)
- #define ADC_MASK_CHANNEL_27
  初期値 :(1<<27U)
- #define ADC_MASK_TEMPERATURE
  初期値 :(1<<28UL)
- #define ADC_MASK_VOLT
  初期値 :(1<<29UL)
- #define ADC_MASK_SENSORS
  初期値 :(ADC_MASK_TEMPERATURE | ADC_MASK_VOLT)

- #define ADC_MASK_GROUP_B_OFF

  初期値 :(0UL)

- #define ADC_MASK_ADD_OFF

  初期値 :(0UL)

- #define ADC_MASK_SAMPLE_HOLD_OFF

  初期値 :(0U)

- #define ADC_SAMPLE_HOLD_CHANNELS

  初期値 :(0x07U)

  サンプルおよびホールド チャネル マスク。サンプルおよびホールドは、チャネル 0、1、および 2 に対してのみ使用できます

- #define ADC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.1.3 r_adc_open_cfg_check_unique

ssp_err_t r_adc_open_cfg_check_unique (uint16_t unit , adc_mode_t const mode , adc_cfg_t const *const p_cfg)

8.1.3.1 概要説明

r_adc_open_cfg_check_unique

8.1.3.2 詳細説明

この関数は ADC オープン構成引数を確認します

表 561: 戻り値

名前	説明
SSP_ERR_INVALID_POINTER	パラメータの値が無効です
SSP_SUCCESS	正常に呼び出しました。

この関数は MCU に固有の ADC オープン構成引数を確認します

表 562: 戻り値

名前	説明
SSP_ERR_INVALID_POINTER	パラメータの値が無効です
SSP_SUCCESS	正常に呼び出しました。

8.1.4 R_ADC_Open

`ssp_err_t` R_ADC_Open (`adc_ctrl_t` * p_ctrl , `adc_cfg_t` const *const p_cfg)

8.1.4.1 概要説明

Open 関数は、A/D ペリフェラルに電源を供給して、ペリフェラル全体の動作モード、トリガー ソース、割り込み優先順位、および設定をセットします。BSP_IRQ_Cfg.h 内の割り込み優先順位が 0 以外の場合は、この関数がスキャンの完了時に割り込みレベルでユーザーに通知するコールバック関数ポインタを取得します。

8.1.4.2 詳細説明

表 563: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	パラメータ p_ctrl または p_cfg が NULL です。
チャネルは現在動作中です。	周辺機器が別のモードで実行しています。先に R_ADC_Close を行ってください。
SSP_ERR_INVALID_POINTER	p_cfg 構造体のモードまたは要素が無効な値になっているか、モードに基づく異常になっています。
チャネルは現在動作中です。	ユニットがすでに初期化されています。

8.1.4.3 関数のステップ

- パラメータ チェックを実行します。
- このユニットが初期化されていないことを確認します。
- NVIC またはペリフェラル内で保留中の割り込み要求をクリアします。

- 設定に基づいてハードウェアを初期化します。
- コールバック関数ポインタを保存します。
- ユニット番号を制御構造体に保存します。
- ユーザー コンテキストを制御構造体に保存します
- モードを制御構造体に保存します。
- 標準モード / グループ A トリガーを内部制御ブロックに保存します。
- コンテンツを保存します
- ドライバをこのユニットに対してオープンにマークします。
- エラー コードを返します。

8.1.5 R_ADC_SetSampleStateCount

`ssp_err_t` R_ADC_SetSampleStateCount (`adc_ctrl_t` * p_ctrl , `adc_sample_state_t` * p_sample)

8.1.5.1 概要説明

個別のチャネルのサンプル状態カウントをセットします。これは、特殊なユース ケースの場合にのみセットする必要があります。通常は、リセット以降のデフォルト値を使用します。

8.1.5.2 詳細説明

表 564: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	パラメータ p_ctrl または p_sample が NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。

8.1.5.3 関数のステップ

- パラメータ チェックを実行します。
- 指定されたレジスタのサンプル状態カウントをセットします。
- エラー コードを返します。

8.1.6 R_ADC_ScanConfigure

```
ssp_err_t R_ADC_ScanConfigure ( adc_ctrl_t * p_ctrl , adc_channel_cfg_t const
*const p_channel_cfg )
```

8.1.6.1 概要説明

ADC スキャン パラメータを設定します。チャネル固有の設定値がこの関数でセットされます。

8.1.6.2 詳細説明

表 565: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	パラメータ p_ctrl または p_ch_cfg が NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。

I：グループ モード プライオリティ構成が ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN に設定されている場合、グループ B はスキャンを継続するため、グループ B 割り込みは無効化され、アプリケーションはコールバックが提供された場合でもグループ B スキャン完了のコールバックを受信しません。グループ A スキャン完了のコールバックについては、コールバックが提供された場合、アプリケーションはこれを受信することができます。

8.1.6.3 関数のステップ

- パラメータ チェックを実行します。
- 設定に基づいてハードウェアを設定します。
- スキャン マスクをローカルに保存します。これは infoGet 関数に必要です
- エラー コードを返します。

8.1.7 R_ADC_InfoGet

```
ssp_err_t  R_ADC_InfoGet ( adc_ctrl_t  * p_ctrl , adc_info_t  * p_adc_info )
```

8.1.7.1 概要説明

この関数は構成チャネルの結果を読み取り ELC イベント名を返すために、最も小さい番号の構成されたチャネルのアドレスと読み取る合計バイト数を返します。

8.1.7.2 詳細説明

表 566: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。
チャネルは現在動作中です。	スキャンの実行がまだ進行中です。

I :: 現在この関数呼び出しはグループ モード動作をサポートしていません。

8.1.7.3 関数のステップ

- パラメータ チェックを実行します。
- 現在のユニット用のベース レジスタへのポインタを取得します
- 制御構造体からアクティブ チャネルのスキャン マスクを取得します
- 構成する最も低いチャネルを決定します
- 構成する最も高いチャネルを決定します
- マスク カウントを設定し、32 ビット マスクの最高ビットで開始できるようにします
- マスク結果を初期化します

- 最も高いチャネルと最も低いチャネルを含むその間のすべてのチャネルを読み取るために、読み取るデータのサイズを決定します。
- ELC リストの周辺機能を指定します

8.1.8 R_ADC_ScanStart

```
ssp_err_t  R_ADC_ScanStart ( adc_ctrl_t  * p_ctrl )
```

8.1.8.1 概要説明

この関数は、Open() 呼び出しでトリガーがどのように設定されたかに応じて、ソフトウェア スキャンを開始するか、スキャン用のハードウェア トリガーを有効にします。ユニットがハードウェア トリガー用に設定されている場合は、この関数がトリガー信号 （ハードウェアまたはソフトウェア）を ADC ユニットに伝送するだけです。この関数は、トリガー自体の生成を制御することはできません。ユニットがソフトウェア トリガー用に設定されている場合は、この関数がソフトウェア トリガー スキャンを開始します。

8.1.8.2 詳細説明

表 567: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。
チャネルは現在動作中です。	スキャンの実行がまだ進行中です。

8.1.8.3 関数のステップ

- パラメータ チェックを実行します。
- 標準 / グループ A トリガーがソフトウェアにセットされていない場合は、ユニットがハードウェア トリガー用に設定されます。
- それ以外の場合は、ソフトウェア トリガーを有効にします。
- 進行中のスキャンが存在するかどうかをチェックして、存在しなければスキャンを開始します。
- エラー コードを返します。

8.1.9 R_ADC_ScanStop

```
ssp_err_t  R_ADC_ScanStop ( adc_ctrl_t  * p_ctrl )
```

8.1.9.1 概要説明

この関数は、Open() 関数でユニットが設定されたトリガーのタイプに基づいて、ソフトウェア スキャンを停止するか、ハードウェア トリガー （内部または外部）によってトリガーされるユニットを無効にします。この関数経由でハードウェア トリガー スキャンを停止しても進行中のスキャンは中断されませんが、次のスキャンの発生は抑えられます。ソフトウェア トリガー スキャンを停止すると、進行中のスキャンが中断されます。

8.1.9.2 詳細説明

表 568: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。

I：ソフトウェア スキャンを停止すると、スキャンの現在の状態に関係なく、スキャンが即座に停止されます。ハードウェア スキャンを停止すると、トリガーが無効にされ、以降のスキャンの開始が阻止されますが、現在のスキャンには影響しません。

8.1.9.3 関数のステップ

- パラメータ チェックを実行します。
- トリガーがソフトウェア スキャンでない場合は、ハードウェア トリガーを禁止します。
- それ以外の場合は、ソフトウェア トリガーを無効にします。
- エラー コードを返します。

8.1.10 R_ADC_CheckScanDone

```
ssp_err_t R_ADC_CheckScanDone ( adc_ctrl_t * p_ctrl )
```

8.1.10.1 概要説明

この関数は、開始されたスキャン プロセスのステータスを返します。

8.1.10.2 詳細説明

表 569: 戻り値

名前	説明
SSP_SUCCESS	成功。スキャンが完了しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。
チャネルは現在動作中です。	スキャンの実行がまだ進行中です。

l：ペリフェラルがシングル スキャン モードで設定された場合は、この関数の戻り値がスキャン ステータスの表示になります。ただし、ペリフェラルがグループ モードで設定された場合は、この関数の戻り値がグループ A とグループ B のどちらかのスキャン状態の表示になります。これは、ADST ビットがスキャンの実行中にセットされ、スキャンの終了後にクリアされるためです。この関数は、通常、ソフトウェア トリガーがシングル スキャン モードで使用されている場合にのみ使用する必要があります。

8.1.10.3 関数のステップ

- パラメータ チェックを実行します。
- ADST ビットのステータスを読み取ります。
- エラー コードを返します。

8.1.11 R_ADC_Read

ssp_err_t R_ADC_Read (adc_ctrl_t * p_ctrl , adc_register_t const reg_id , adc_data_size_t *const p_data)

8.1.11.1 概要説明

この関数は、シングル チャネルまたはセンサー レジスタから変換結果を読み取ります。

8.1.11.2 詳細説明

表 570: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
Null ポインタが指定されました。	パラメータ p_data が NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。

8.1.11.3 関数のステップ

- パラメータ チェックを実行します。
- 適切なベース アドレスへのポインタを取得します。パラメータ チェックが無効化されている場合、この動作は繰り返されます。
- 要求された ADC 変換レジスタからデータを読み取ってそれを返します。
- エラー コードを返します。

8.1.12 R_ADC_Close

ssp_err_t R_ADC_Close (adc_ctrl_t * p_ctrl)

8.1.12.1 概要説明

この関数は、進行中のスキャンを停止して、割り込みを無効にし、A/D ペリフェラルへの電源を遮断します。

8.1.12.2 詳細説明

表 571: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	ユニットが開いていません。
SSP_ERR_INVALID_POINTER	パラメータの値が無効です。

8.1.12.3 関数のステップ

- パラメータ チェックを実行します。
- 特定のユニットのハードウェア停止を実行します。
- ドライバをクローズとしてマークします。
- ロックを解除します。
- エラー コードを返します。

8.1.13 R_ADC_VersionGet

```
ssp_err_t  R_ADC_VersionGet ( ssp_version_t  *const p_version )
```

8.1.13.1 概要説明

API バージョン番号を取得します。

8.1.13.2 詳細説明

表 572: 戻り値

名前	説明
SSP_SUCCESS	正常な復帰。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.1.13.3 関数のステップ

- バージョン番号を返します。

8.2 AGT

調歩同期式汎用タイマ （AGT）用ドライバ。

8.2.1 概要

タイマインタフェース を拡張します。

AGT タイマ モードにアクセスして構成するための HAL ハイレベル ドライバ。

AGT タイマ関数は、タイマによってタイマ サービスを提供するために使用されます。

8.2.2 Functions

- R_AGT_TimerOpen
- R_AGT_Close
- R_AGT_CounterGet
- R_AGT_PeriodSet
- R_AGT_DutyCycleSet
- R_AGT_Reset
- R_AGT_Start
- R_AGT_InfoGet
- R_AGT_Stop
- R_AGT_VersionGet

8.2.3 R_AGT_TimerOpen

```
ssp_err_t R_AGT_TimerOpen ( timer_ctrl_t *const p_ctrl , timer_cfg_t const *const p_cfg )
```

8.2.3.1 概要説明

AGT チャネルをタイマとして開き、ユーザーズマニュアルに記載されている必要な初期化を処理します。
open を実装します。

8.2.3.2 詳細説明

タイマ オープン関数は、タイマ設定構造体で指定されたパラメータを使用してタイマ モードのシングル AGT チャネルを設定します。以降の AGT タイマ API で使用する制御ブロックもセットアップします。

この関数は、他の AGT API 関数を呼び出す前に一度だけ呼び出す必要があります。チャネルが開かれたら、関連する Close 関数を呼び出す前に同じチャネルに対して Open 関数を繰り返し呼び出さないようにする必要があります。

AGT ハードウェアはネイティブでワンショット機能をサポートしません。そのため、ワンショット機能は AGT HAL レイヤーに実装されます。ワンショット タイマとして設定されたタイマは、最初のタイマ期限切れで停止します。

汎用タイマの AGT 実装は、timer_on_agt_cfg_t 拡張パラメータを受け入れることができます。AGT の場合は、拡張機能がタイマ ソースとして使用されるクロックを指定します。拡張機能パラメータが指定されていない場合は、デフォルト クロック PCLKB が使用されます。クロック分周器は、ソース クロック周波数と呼び出し側によって指定されたタイマ期間に基づいて選択されます。

表 573: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われ、タイマが開始しました。
p_ctrl の可能性があります。	p_cfg と p_ctrl のどちらかのパラメータが NULL か、設定チャネル ID が AGT_MAX_CH を超えているか、設定モードが無効です。
チャネルは現在動作中です。	指定したチャネルはすでに開かれています。
SSP_ERR_IRQ_BSP_DISABLED	要求された割り込みは BSP では有効化されていません。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.2.3.3 関数のステップ

- まだ使用されていないチャネルを確認します。
- AGT チャネルの電源をオンにします。
- カウンタが停止するまで待機します。
- AGTO 出力をクリアします。
- TEDGSEL ビットを標準出力にクリアします。
- ユーザーがコールバック関数を指定した場合は IRQ を有効にします。タイマがワンショット タイマの場合は、ドライバが 1 周期後にタイマをオフにできます。

- 保留中の割り込み要求をクリアし、割り込みを有効にします。
- AGT タイマ設定構造体が存在する場合は、カウンタ ソースと出力ピン設定を選択します。カウンタ ソースの有効値は次のとおりです。
- AGT_TCK_PCLKB
- AGT_TCK_LOCO
- AGT_TCK_FSUBAGTO ピンと AGTIO ピンは別々に出力に対して有効にできます。ユーザーが指定した AGTO および AGTIO ピン出力に基づいて、AGT チャネルが設定されるモードについて以下で説明します。
- AGT0 と AGTIO の両方が無効になっている場合は、AGT が TOE ビットがクリアされた状態でタイマ モードで開かれます。AGT0 が有効になっており、AGTIO が無効になっている場合は、AGT が TOE ビットがセットされた状態でタイマ モードで開かれます。AGT0 が無効になっており、AGTIO が有効になっている場合は、AGT が TOE ビットがクリアされた状態でパルス出力モードで開かれます。AGT0 と AGTIO の両方が有効になっている場合は、AGT が TOE ビットがセットされた状態でパルス出力モードで開かれます。
- AGT タイマ設定構造体が存在しない場合は、デフォルト クロック AGT_TCK_PCLKB をタイマ モードのクロック ソースとして使用します。
- agt_mode 値に基づいて AGT モードをセットします。
- 期間が有効なことを確認してから、期間をセットします。
- チャネルを「使用中」としてマークし、制御ブロックを初期化します。
- ユーザーから要求されたら、タイマを開始します。
- すべてが完了しました。

8.2.4 R_AGT_Close

```
ssp_err_t  R_AGT_Close ( timer_ctrl_t  *const p_ctrl )
```

8.2.4.1 詳細説明

カウンタを停止して、割り込みを無効にし、出力ピンを無効にして、内部ドライバ データをクリアします。

表 574: 戻り値

名前	説明
SSP_SUCCESS	AGT タイマ チャネルが正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	AGT チャネルが開かれていません。

8.2.4.2 関数のステップ

- チャネルが開いていることを確認します。開いていない場合は、エラー コード SSP_ERR_NOT_OPEN で戻ります。
- ドライバ内部フラグをクリアします。
- デバイスをクリーンアップします。カウンタを停止して、割り込みを無効にし、他のチャネルが使用中でなければ電源をオフにします。
- TOE （出力イネーブル）ビットをクリアします。
- TEDGSEL ビットをクリアします。
- チャネルをロック解除します。

8.2.5 R_AGT_CounterGet

```
ssp_err_t  R_AGT_CounterGet ( timer_ctrl_t  *const p_ctrl , timer_size_t  *const p_value )
```

8.2.5.1 詳細説明

指定された p_value ポインタ内のカウンタ値を取得して保存します。timer_api_t::CounterGet を実装します。

表 575: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が読み取られ、p_value は有効です。
p_ctrl の可能性があります。	p_ctrl または p_value パラメータが null でした。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.2.5.2 関数のステップ

- カウンタ値を読み取ります。

8.2.6 R_AGT_PeriodSet

```
ssp_err_t  R_AGT_PeriodSet ( timer_ctrl_t  *const p_ctrl , timer_size_t  const period ,
 timer_unit_t  const unit )
```

8.2.6.1 詳細説明

指定された期間値をセットします。periodSet を実装します。

表 576: 戻り値

名前	説明
SSP_SUCCESS	周期値が正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
SSP_ERR_INVALID_ARG	次のいずれかが無効です。 • p_period->unit: timer_size_t::unit のいずれかのオプションにする必要があります。 • p_period->value: Open 呼び出しで指定されたクロック ソースによりサポートされる期間でなければなりません。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.2.6.2 関数のステップ

- 期間が有効なことを確認してから、期間をセットします。

8.2.7 R_AGT_DutyCycleSet

```
ssp_err_t R_AGT_DutyCycleSet ( timer_ctrl_t *const p_ctrl , timer_size_t const duty_cycle ,
 timer_pwm_unit_t const unit , uint8_t const pin )
```

8.2.7.1 詳細説明

デューティ サイクルの設定はこのドライバではサポートされていません。periodSet を実装します。

表 577: 戻り値

名前	説明
データ構造体が割り当てられませんでした。	サポートされていないドライバ機能です。

8.2.8 R_AGT_Reset

```
ssp_err_t R_AGT_Reset ( timer_ctrl_t *const p_ctrl )
```

8.2.8.1 詳細説明

カウンタ値を元々セットされていた期間にリセットします。reset を実装します。

表 578: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.2.8.2 関数のステップ

- すでに実行中の場合は、再起動のために実行ステータスを保存します。
- カウンタを元々送信された期間にリセットします。
- AGT チャネルが稼働中であれば再起動します。

8.2.9 R_AGT_Start

```
ssp_err_t R_AGT_Start ( timer_ctrl_t *const p_ctrl )
```

8.2.9.1 詳細説明

タイマを開始します。start を実装します。

表 579: 戻り値

名前	説明
SSP_SUCCESS	タイマが正常に開始されました。
p_ctrl の可能性があります。	p_ctrl パラメータが null です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.2.9.2 関数のステップ

- タイマを開始します。

8.2.10 R_AGT_InfoGet

`ssp_err_t` R_AGT_InfoGet (`timer_ctrl_t` *const p_ctrl , `timer_info_t` *const p_info)

8.2.10.1 概要説明

タイマ情報を取得し、指定されたポインタ p_info に格納します。infoGet を実装します。

8.2.10.2 詳細説明

表 580: 戻り値

名前	説明
SSP_SUCCESS	期間、ステータス、カウント方向、周波数の値を呼び出し側の構造体に正常に書き込みました。
p_ctrl の可能性があります。	p_ctrl または p_period_counts パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。
SSP_ERR_INVALID_HW_CONDITION	無効なハードウェア設定が検出されました。

8.2.10.3 関数のステップ

- 期間を取得して保存します
- クロック周波数を取得して保存します
- AGT はカウント ダウン方向のみをサポートします。

8.2.11 R_AGT_Stop

`ssp_err_t` R_AGT_Stop (`timer_ctrl_t` *const p_ctrl)

8.2.11.1 詳細説明

ハンドル（制御ブロック）によって指定された AGT チャネルを停止します。この API は stop を実装します。この API は、チャネルをリセットしたり、その電源をオフにしたりしません。

表 581: 戻り値

名前	説明
SSP_SUCCESS	タイマが正常に停止されました。
SSP_ERR_INVALID_PTR	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.2.12 R_AGT_VersionGet

`ssp_err_t` R_AGT_VersionGet (`ssp_version_t` *const p_version)

8.2.12.1 詳細説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

表 582: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.2.13 拡張

8.2.13.1 timer_on_agt_cfg_t

timer_on_agt_cfg_t

詳細説明

AGT 拡張データ構造体。

変数

- agt_count_source_t count_source

  AGT チャネルのクロック ソース。設定可能な値 : AGT_CLOCK_PCLKB、AGT_CLOCK_LOCO、AGT_CLOCK_FSUB。

- bool agto_output_enabled

  AGTO ピンで、出力コンペアが有効 （true、false)

- bool agtio_output_enabled

  AGTIO ピンで、出力コンペアが有効 （true、false)

- bool output_inverted

  出力の反転 （true、false)

8.2.14 モジュール

- ビルドタイム構成

8.2.14.1 ビルドタイム構成

定義

- #define AGT_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.3 CAC

クロック周波数精度測定回路 （CAC）用ドライバ。

8.3.1 概要

このモジュールは、CAC ペリフェラルをサポートします。

8.3.2 Functions

- R_CAC_Open
- R_CAC_Close
- R_CAC_StopMeasurement
- R_CAC_StartMeasurement
- R_CAC_Reset
- R_CAC_Read
- R_CAC_VersionGet

8.3.3 定義

- #define CAC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define CAC_CODE_VERSION_MINOR

  初期値 :(1)

8.3.4 R_CAC_Open

```
ssp_err_t R_CAC_Open ( cac_ctrl_t *const p_ctrl , cac_cfg_t const *const p_cfg )
```

8.3.4.1 概要説明

CAC 周辺機器を初期化します。

8.3.4.2 詳細説明

Open 関数は CAC 周辺機器に電源を供給し、中断優先度を確認 / 設定し、提供されたユーザ構成設定に基づいて CAC を構成します。ユーザ定義済みのコールバック関数が構成で提供されている場合、CAC 中断は有効になり、ユーザコールバック関数はそれに応じて呼び出されます。Implements r_cac_t::open.

表 583: 戻り値

名前	説明
SSP_SUCCESS	CAC が使用可能で、測定に使用できます。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	1 つまたは複数の構成オプションが無効です。
SSP_ERR_INVALID_ARGUMENT	CAC 周辺機器のハードウェアロックがすでに取られています。
SSP_ERR_INVALID_CAC_REF_CLOCK	測定クロックレートが参照クロックレートよりも小さくなっています。

I：単一の CAC 周辺機器しかありません。再入不可です。

8.3.4.3 関数のステップ

- g_module_name[] と g_cac_version にアクセスできるのは ASSERT マクロのみなので、それらにアクセスできないことを示す警告がコンパイラ ツールチェーンから発行されることがあります。以下のコードはこの警告を抑制するとともに、これらのデータ構造体が最適化によって除去されることを防ぎます。
- パラメータ チェックが無効になっている場合、警告を排除します。
- 割り込みが有効になっている場合は、それを選択されたチャネルに対してセットアップします。
- CAC のハードウェアロックを取ります。
- 周辺機器の電源を入れます
- 構成に従って CAC を設定します。
- 正常に開けなかった場合はハードウェアロックを返します。
- CAC のハードウェア ロックを返します。

8.3.5 R_CAC_Close

```
ssp_err_t  R_CAC_Close ( cac_ctrl_t  *const p_ctrl )
```

8.3.5.1 概要説明

Open() またはその後の CAC 動作で割り当てられたすべてのリソースを解放します。r_cac_t::close を実装します。

8.3.5.2 詳細説明

表 584: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
タッチ パネルが設定されていません。	R_CAC_Open 正常に呼び出しされませんでした。

8.3.5.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- CAC のハードウェア ロックを返します。
- NVIC での割り込みを無効化します。
- CAC 割り込みを無効化します。
- 測定を停止します。
- 割り込み要求ビットをクリアします。
- 周辺機能の電源をオフにします。

8.3.6 R_CAC_StopMeasurement

```
ssp_err_t  R_CAC_StopMeasurement ( cac_ctrl_t  *const p_ctrl )
```

8.3.6.1 概要説明

CAC 測定プロセスを停止します。r_cac_t::stopMeasurement を実装します。

8.3.6.2 詳細説明

表 585: 戻り値

名前	説明
SSP_SUCCESS	CAC 測定が停止されました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
タッチ パネルが設定されていません。	R_CAC_Open 正常に呼び出しされませんでした。

8.3.6.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- 測定を停止します。

8.3.7 R_CAC_StartMeasurement

`ssp_err_t` R_CAC_StartMeasurement (`cac_ctrl_t` *const p_ctrl)

8.3.7.1 概要説明

CAC 測定プロセスを開始します。r_cac_t::startMeasurement を実装します。

8.3.7.2 詳細説明

表 586: 戻り値

名前	説明
SSP_SUCCESS	CAC 測定が開始されました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
タッチ パネルが設定されていません。	R_CAC_Open 正常に呼び出しされませんでした。

8.3.7.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- 測定を開始します。

8.3.8 R_CAC_Reset

ssp_err_t R_CAC_Reset (cac_ctrl_t *const p_ctrl)

8.3.8.1 概要説明

オーバフロー、測定終了、および周波数エラーの中断フラグをリセットします。r_cac_t::reset を実装します。

8.3.8.2 詳細説明

表 587: 戻り値

名前	説明
SSP_SUCCESS	CAC リセットが完了しました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
タッチ パネルが設定されていません。	R_CAC_Open 正常に呼び出しされませんでした。

8.3.8.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- CAC をリセットします。

8.3.9 R_CAC_Read

ssp_err_t R_CAC_Read (cac_ctrl_t *const p_ctrl , uint8_t *const p_status , uint16_t *const p_counter)

8.3.9.1 概要説明

CAC のステータスとカウンタ レジスタを読み取って返します。r_cac_t::read を実装します。

8.3.9.2 詳細説明

表 588: 戻り値

名前	説明
SSP_SUCCESS	CAC が正常に読み取られました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
タッチ パネルが設定されていません。	R_CAC_Open 正常に呼び出しされませんでした。

8.3.9.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.3.10 R_CAC_VersionGet

`ssp_err_t` R_CAC_VersionGet (`ssp_version_t` *const p_version)

8.3.10.1 概要説明

API およびコードバージョン情報を取得します。

8.3.10.2 詳細説明

表 589: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報が返されました。

I：この関数は再入可能です。

8.3.11 モジュール

- ビルドタイム構成

8.3.11.1 ビルドタイム構成

定義

- #define CAC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.4 CAN

CAN （コントローラ エリア ネットワーク）用のドライバ。

このモジュールは、コントローラ エリア ネットワーク周辺機器をサポートします。次のインタフェースを実装します。

- CAN インタフェース

8.4.1 Functions

- R_CAN_Open
- R_CAN_Close
- R_CAN_Read
- R_CAN_Write
- R_CAN_Control
- R_CAN_InfoGet
- R_CAN_VersionGet

8.4.2 定義

- #define CAN_CODE_VERSION_MAJOR

  初期値 :(1UL)
- #define CAN_CODE_VERSION_MINOR

  初期値 :(0UL)

8.4.3 R_CAN_Open

```
ssp_err_t R_CAN_Open ( can_ctrl_t *const p_ctrl , can_cfg_t const *const p_cfg )
```

8.4.3.1 概要説明

動作用の CAN チャネルを開いて構成します。

8.4.3.2 詳細説明

表 590: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開かれました
SSP_ERR_INVALID_POINTER	無効なチャネルが引数として渡されました。
SSP_ERR_INVALID_ARGUMENT	ロックはすでに別のユーザーが所有しています。
SSP_ERR_CAN_MODE_SWITCH_FAILED	チャネルがモードの切り替えに失敗しました。
SSP_ERR_CAN_INIT_FAILED	チャネルが初期化に失敗しました。
p_ctrl の可能性があります。	Null ポインタが示されました。

8.4.3.3 関数のステップ

- チャネル ロックの取得を試みます。
- チャネルはすでに開いているためエラーを返します
- 無効なチャネル番号をチェックします
- モジュール停止状態を終了します。
- スリープ モードを終了します。これにより、停止モードが OPERATE_CANMODE に変わります。
- リセット モードに変わります。
- BOM: IEC11898-1 に基づくバス オフ復帰モード
- MBM: 通常のメールボックス モードを選択します。
- 開始時点では、デフォルトですべてのメールボックスに対しマスクが無効になっています。これは、通常動作のテスト後変更されます。
- リセット -> 停止モード
- 停止モードでメールボックスをクリアします。
- 停止モードでのエラー表示モードを設定します。
- 停止 -> 通常動作モード
- タイム スタンプ カウンタのリセット。CAN 動作モードで TSRC ビットを 1 に設定します。
- これまでのエラーをチェックして報告し、クリアします。

- チャネル、コールバック関数、コンテキスト、ID モード、メールボックス カウント、メッセージ モード、OP モード、Opened ステータスを設定します。
- 内部制御に対するインスタンス ポインタを設定します。
- CAN 割り込みを構成します。
- エラーが発生した場合、チャネルのロックを解除します。
- モジュール停止状態に入ります。
- エラーを返す前に処理します。
- エラーまたはアサーションを記録します。

8.4.4 R_CAN_Close

```
ssp_err_t  R_CAN_Close ( can_ctrl_t  *const p_ctrl )
```

8.4.4.1 概要説明

CAN チャネルを閉じます。

8.4.4.2 詳細説明

関数 R_CAN_Open の終点

表 591:　戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました。
タッチ パネルが設定されていません。	ポートが開いていません。
p_ctrl の可能性があります。	Null ポインタが示されました。

8.4.4.3 関数のステップ

- 内部制御ポインタを取得します。
- チャネルが開いていない場合、エラーを返します
- モジュール停止状態に入ります
- チャネルをロック解除します。
- エラーを返す前に処理します。

- エラーまたはアサーションを記録します。

8.4.5 R_CAN_Read

`ssp_err_t` R_CAN_Read (`can_ctrl_t` *const p_ctrl , uint32_t mailbox , `can_frame_t` *const p_frame)

8.4.5.1 概要説明

CAN チャネルからデータを読み取ります。チャネル メールボックスから読み取った最大 8 バイトのデータを返します。

8.4.5.2 詳細説明

表 592: 戻り値

名前	説明
SSP_SUCCESS	データが正常に読み取られました。
SSP_ERR_CAN_DATA_UNAVAILABLE	利用可能なデータがありません。
SSP_ERR_CAN_TRANSMIT_MAILBOX	メールボックスが受信用に設定されていません。
p_ctrl の可能性があります。	Null ポインタが示されました。

8.4.5.3 関数のステップ

- 受信データをチェックします
- 無効なバイト カウント、このメールボックスで利用可能な受信データがないことを報告します。
- フレーム データを取得します。
- エラーを返す前に処理します。
- エラーまたはアサーションを記録します。

8.4.6 R_CAN_Write

`ssp_err_t` R_CAN_Write (`can_ctrl_t` *const p_ctrl , uint32_t mailbox , `can_frame_t` *const p_frame)

8.4.6.1 概要説明

CAN チャネルにデータを書き込みます。チャネル メールボックスに最大 8 バイトのデータを書き込みます。

8.4.6.2 詳細説明

表 593: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。
SSP_ERR_CAN_TRANSMIT_NOT_READY	送信を実行中です。現時点ではデータを書き込むことができません。
SSP_ERR_CAN_RECEIVE_MAILBOX	メールボックスが受信用に設定されておらず、送信できません。
SSP_ERR_INVALID_POINTER	データ長またはフレーム タイプが無効です。
p_ctrl の可能性があります。	Null ポインタが示されました

8.4.6.3 関数のステップ

- 送信準備完了フラグをチェックします。
- 送信 ID を設定します
- 送信フレームを送信します。
- エラーを返す前に処理します。
- エラーまたはアサーションを記録します。

8.4.7 R_CAN_Control

```
ssp_err_t R_CAN_Control ( can_ctrl_t *const p_ctrl , can_command_t const command , void * p_data )
```

8.4.7.1 概要説明

拡張機能の制御には CAN 制御が用いられます。

8.4.7.2 詳細説明

表 594: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。
SSP_ERR_INVALID_POINTER	無効なコマンドです。
p_ctrl の可能性があります。	Null ポインタが示されました

8.4.7.3 関数のステップ

- 内部制御ポインタを取得します。
- 動作モードを変更します。
- 診断目的でモードを保存します。
- エラーを返す前に処理します。
- エラーまたはアサーションを記録します。

8.4.8 R_CAN_InfoGet

```
ssp_err_t  R_CAN_InfoGet ( can_ctrl_t  *const p_ctrl ,  can_info_t *const p_info )
```

8.4.8.1 概要説明

チャネルの CAN 状態およびステータス情報を取得します。

8.4.8.2 詳細説明

表 595: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。
SSP_ERR_CAN_DATA_UNAVAILABLE	チャネルが情報のリターンに失敗しました。
p_ctrl の可能性があります。	Null ポインタが示されました

8.4.8.3 関数のステップ

- 内部制御ポインタを取得します。
- チャネルのステータスを取得します。
- エラーを返す前に処理します。
- エラーまたはアサーションを記録します。

8.4.9 R_CAN_VersionGet

```
ssp_err_t  R_CAN_VersionGet ( ssp_version_t  *const p_version )
```

8.4.9.1 概要説明

CAN モジュール コードおよび API バージョンを取得します。

8.4.9.2 詳細説明

表 596:　戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。
p_ctrl の可能性があります。	Null ポインタが示されました

I：この関数は再入可能です。

8.4.9.3 関数のステップ

- エラーを返す前に処理します。
- エラーまたはアサーションを記録します。

8.4.10 拡張

8.4.10.1 can_extended_cfg_t

can_extended_cfg_t

詳細説明

p_extend が指す CAN クロック構成およびメールボックス マスク。

変数

- can_clock_source_t clock_source

  CAN クロックのソース。

- uint32_t * p_mailbox_mask

  メールボックス マスク、4 つすべてのメールボックスのうちの 1 つ。

8.4.11 モジュール

- ビルドタイム構成

8.4.11.1 ビルドタイム構成

定義

- #define CAN_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.5 CGC

クロック発生回路のドライバ。

クロック発生回路 API。

このモジュールは、クロック発生回路をサポートします。次のインタフェースを実装します。

- CGC インタフェース

8.5.1 Functions

- R_CGC_Init
- R_CGC_ClockStart
- R_CGC_ClockStop
- R_CGC_SystemClockSet
- R_CGC_SystemClockGet
- R_CGC_SystemClockFreqGet
- R_CGC_ClockCheck
- R_CGC_OscStopDetect
- R_CGC_OscStopStatusClear
- R_CGC_BusClockOutCfg
- R_CGC_BusClockOutEnable
- R_CGC_BusClockOutDisable
- R_CGC_ClockOutCfg
- R_CGC_ClockOutEnable
- R_CGC_ClockOutDisable
- R_CGC_LCDClockCfg
- R_CGC_LCDClockEnable
- R_CGC_LCDClockDisable
- R_CGC_SDRAMClockOutEnable
- R_CGC_SDRAMClockOutDisable
- R_CGC_USBClockCfg
- R_CGC_SystickUpdate

- R_CGC_VersionGet

8.5.2 定義

- #define CGC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define CGC_CODE_VERSION_MINOR

  初期値 :(1)

8.5.3 R_CGC_Init

```
ssp_err_t R_CGC_Init ( void )
```

8.5.3.1 概要説明

CGC API を初期化します。

8.5.3.2 詳細説明

クロック ジェネレーター モジュールに対して以下を構成します

- サブクロックドライブ容量（コンパイル時間を構成可能 : CGC_CFG_SUBCLOCK_DRIVE）
- サブクロックの初期設定

他の CGC 関数を使用したり、クロック ソースを MOCO から変更したりするには、この関数をあらかじめスタートアップ時に一度実行する必要があります。

表 597: 戻り値

名前	説明
SSP_SUCCESS	クロックが正常に初期化されました。
SSP_ERR_HARDWARE_TIMEOUT	ハードウェアがタイムアウトしました。

8.5.4 R_CGC_ClockStart

```
ssp_err_t R_CGC_ClockStart ( cgc_clock_t clock_source , cgc_clock_cfg_t * p_clock_cfg )
```

8.5.4.1 概要説明

指定されたクロックが現在アクティブでない場合は開始します。

8.5.4.2 詳細説明

メインのクロック発振器を開始した場合、次を構成します。

- メインクロックドライブ容量（外部クロック周波数に基づいて構成）
- メインクロック安定性待機時間（コンパイル時間を構成可能 : CGC_CFG_MAIN_OSC_WAIT）

表 598: 戻り値

名前	説明
SSP_SUCCESS	クロックが正常に初期化されました。
SSP_ERR_ILL_PARAM	無効な引数が使用されました。
SSP_ERR_MAIN_OCO_INACTIVE	メイン OCO がオフ / 不安定な状態で PLL 初期化が試みられました。
SSP_ERR_CLOCK_ACTIVE	アクティブなクロックソースが変更のために指定されました。これは、PLL がアクティブな場合に変更できない PLL ディバイダ / マルチプライヤに特に適用されます。変更前にまず停止する必要があります。
SSP_ERR_NOT_STABILIZED	クロックソースをオフにした後に安定していません。
SSP_ERR_CLKOUT_EXCEEDED	メインオシレータは 8 または 16 MHz です。
SSP_ERR_NULL_PTR	PLL が clock_source. 引数の場合に構成として NULL が受け渡されます。

8.5.4.3 関数のステップ

- 要求された Iclk 周波数を取得します。
- HOCO を開始する前に、高速モードに切り替える必要があるかどうかを確認します。
- HOCO を開始する前に、高速モードに切り替える必要があるかどうかを確認します。
- PLL を開始する前に、高速モードに切り替える必要があるかどうかを確認します。

8.5.5 R_CGC_ClockStop

```
ssp_err_t R_CGC_ClockStop ( cgc_clock_t clock_source )
```

8.5.5.1 概要説明

指定されたクロックがアクティブで、システム クロックとして構成されていない場合、このクロックを停止します。

8.5.5.2 詳細説明

表 599: 戻り値

名前	説明
SSP_SUCCESS	クロックが正常に停止されました。
SSP_ERR_CLOCK_ACTIVE	現在のシステムクロックソースが停止のために指定されました。これは許可されていません。
SSP_ERR_OSC_DET_ENABLED	オシレーション停止検出が有効な場合、MOCO は停止できません。
SSP_ERR_NOT_STABILIZED	開始後にクロックが安定していません。停止する前に、有限の安定性時間がクロック開始後に経過する必要があります。
SSP_ERR_INVALID_POINTER	無効な引数が使用されました。

8.5.6 R_CGC_SystemClockSet

```
ssp_err_t  R_CGC_SystemClockSet ( cgc_clock_t   clock_source , cgc_system_clock_cfg_t
* p_clock_cfg )
```

8.5.6.1 概要説明

指定されたクロックをシステム クロックとして設定し、ICLK、PCLKA、PCLKB、PCLKC、PCLKD、および FCLK の内部分周器を構成します。

8.5.6.2 詳細説明

この関数は、指定されたクロック ソースと分周器の値が、動作モードでサポートされている範囲にあるかどうかを確認しません。設定したクロック ソースおよび分周器の値が、現在の動作モードでサポートされている範囲にない場合、動作は未定義です。

LOCO、MOCO またはサブクロックをシステム クロックとして選択した場合、この関数はそれらの安定性を確認せずにシステム クロックとして設定します。LOCO、MOCO、またはサブクロックをシステム クロックとして使用する場合は、事前にそれらの安定性を十分に確認してください。

また、この関数では、MCU に対する RAM と ROM の待機状態が設定されます。S7G2 では、ROMWT レジスタによって ROM の待機状態が制御されます。S3A7 では、MEMWAIT レジスタによって、ROM の待機状態が制御されます。

表 600: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。
SSP_ERR_CLOCK_INACTIVE	指定したクロックソースは非アクティブです。
SSP_ERR_NULL_PTR	p_clock_cfg パラメータは NULL です。
SSP_ERR_STABILIZED	クロックソースは安定していません。
SSP_ERR_INVALID_POINTER	無効な引数が使用されました。ICLK は最速クロックとして設定されていません。

8.5.6.3 関数のステップ

- S124 に対するコンパイラ「不使用」警告を回避します
- ROM および RAM 待機状態レジスタを正しく設定するには、現在の （S3A7 のみ）および要求されている iclk 周波数を確認する必要があります。
- < 1 つの RAM 待機サイクル
- < RAM 待機サイクルなし
- 待機サイクルを 0 に変更するには、現在の周波数が 32 MHz 未満、MCU が高速モードに設定されている必要があります。

8.5.7 R_CGC_SystemClockGet

```
ssp_err_t R_CGC_SystemClockGet ( cgc_clock_t * clock_source , cgc_system_clock_cfg_t
* p_set_clock_cfg )
```

8.5.7.1 概要説明

現在のシステム クロック ソースと構成を返します。

8.5.7.2 詳細説明

表 601: 戻り値

名前	説明
SSP_SUCCESS	パラメータが正常に返されました。

8.5.8 R_CGC_SystemClockFreqGet

ssp_err_t R_CGC_SystemClockFreqGet (cgc_system_clocks_t clock , uint32_t * p_freq_hz)

8.5.8.1 概要説明

要求された内部クロック周波数を Hz 単位で返します。

8.5.8.2 詳細説明

表 602: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。
SSP_ERR_INVALID_POINTER	無効なクロックが指定されました。

8.5.9 R_CGC_ClockCheck

ssp_err_t R_CGC_ClockCheck (cgc_clock_t clock_source)

8.5.9.1 概要説明

指定されたクロックの安定性を確認します。

8.5.9.2 詳細説明

表 603: 戻り値

名前	説明
SSP_ERR_STABILIZED	クロックが安定しました。
SSP_ERR_NOT_STABILIZED	クロックが安定していません。
SSP_ERR_CLOCK_ACTIVE	クロックはアクティブですが、安定性を確認できません。
SSP_ERR_CLOCK_INACTIVE	クロックがオンになっていません。
SSP_ERR_INVALID_POINTER	無効なパラメータが受け渡されました。

8.5.10 R_CGC_OscStopDetect

```
ssp_err_t R_CGC_OscStopDetect ( cgc_callback_args_t void(*)( *p_args) p_callback ,
 bool enable )
```

8.5.10.1 概要説明

メイン クロックの発振停止検出機能を有効化または無効化します。システム クロックがメイン クロックである場合、停止が検出されると、MCU によってシステム クロックが自動的に MOCO に切り替えられます。システム クロックが PLL である場合、クロック ソースは変更されず、PLL のフリー ランニング周波数がシステム クロック周波数となります。

8.5.10.2 詳細説明

表 604: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。
SSP_ERR_OSC_STOP_DETECTED	オシレーション停止検出ステータスフラグが設定されます。この状態では、オシレーション停止検出関数を無効にすることはできません。
SSP_ERR_NULL_PTR	2 つ目の引数が "true" の場合にコールバック関数に Null ポインタが受け渡されます。

8.5.11 R_CGC_OscStopStatusClear

ssp_err_t R_CGC_OscStopStatusClear (void)

8.5.11.1 概要説明

発振停止検出ステータス レジスタをクリアします。

8.5.11.2 詳細説明

停止したクロックを再開した場合、このレジスタは自動的にクリアされません。この関数は、ステータス レジスタがクリアされるまで約 3 ICLK サイクルをブロックします。

表 605: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	メイン Osc または PLL がシステムクロックとして設定されている場合、オシレーション検出ステータスフラグはクリアできません。システムクロックを変更してから、このビットのクリアを試みてください。

8.5.12 R_CGC_BusClockOutCfg

ssp_err_t R_CGC_BusClockOutCfg (cgc_bclockout_dividers_t divider)

8.5.12.1 概要説明

BCLKOUT のセカンダリ分周器を構成します。プライマリ分周器は、bsp クロック構成および R_CGC_SystemClockSet 関数を使用して設定されます。

8.5.12.2 詳細説明

表 606: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.13 R_CGC_BusClockOutEnable

ssp_err_t R_CGC_BusClockOutEnable (void)

8.5.13.1 概要説明

BCLKOUT 出力を有効化します。

8.5.13.2 詳細説明

表 607: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.14 R_CGC_BusClockOutDisable

ssp_err_t R_CGC_BusClockOutDisable (void)

8.5.14.1 概要説明

BCLKOUT 出力を無効化します。

8.5.14.2 詳細説明

表 608: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.15 R_CGC_ClockOutCfg

ssp_err_t R_CGC_ClockOutCfg (cgc_clock_t clock , cgc_clockout_dividers_t divider)

8.5.15.1 概要説明

CLKOUT の分周器を構成します。

8.5.15.2 詳細説明

表 609: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.16 R_CGC_ClockOutEnable

```
ssp_err_t R_CGC_ClockOutEnable ( void )
```

8.5.16.1 概要説明

CLKOUT 出力を有効化します。

8.5.16.2 詳細説明

表 610: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.17 R_CGC_ClockOutDisable

```
ssp_err_t R_CGC_ClockOutDisable ( void )
```

8.5.17.1 概要説明

CLKOUT 出力を無効化します。

8.5.17.2 詳細説明

表 611: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.18 R_CGC_LCDClockCfg

ssp_err_t R_CGC_LCDClockCfg (cgc_clock_t clock)

8.5.18.1 概要説明

セグメント LCDCLK のソースを構成します。

8.5.18.2 詳細説明

表 612: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.19 R_CGC_LCDClockEnable

ssp_err_t R_CGC_LCDClockEnable (void)

8.5.19.1 概要説明

セグメント LCDCLK 出力を有効化します。

8.5.19.2 詳細説明

表 613: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.20 R_CGC_LCDClockDisable

ssp_err_t R_CGC_LCDClockDisable (void)

8.5.20.1 概要説明

セグメント LCDCLK 出力を無効化します。

8.5.20.2 詳細説明

表 614: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.21 R_CGC_SDRAMClockOutEnable

ssp_err_t R_CGC_SDRAMClockOutEnable (void)

8.5.21.1 概要説明

SDCLK 出力を有効化します。

8.5.21.2 詳細説明

表 615: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.22 R_CGC_SDRAMClockOutDisable

ssp_err_t R_CGC_SDRAMClockOutDisable (void)

8.5.22.1 概要説明

SDCLK 出力を無効化します。

8.5.22.2 詳細説明

表 616: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.23 R_CGC_USBClockCfg

ssp_err_t R_CGC_USBClockCfg (cgc_usb_clock_div_t divider)

8.5.23.1 概要説明

UCLK の分周器を構成します。

8.5.23.2 詳細説明

表 617: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。
SSP_ERR_INVALID_POINTER	無効なディバイダが指定されました。

8.5.24 R_CGC_SystickUpdate

ssp_err_t R_CGC_SystickUpdate (uint32_t period_count , cgc_systick_period_units_t units)

8.5.24.1 概要説明

指定された期間と現在のシステム クロック周波数に基づいて systick を再構成します。

8.5.24.2 詳細説明

表 618: パラメータ

名前	方向	説明
period_count	複数のビットを書き換えることもできます。	systick 期間の長さ。
units	複数のビットを書き換えることもできます。	指定された期間の単位。

表 619: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。
SSP_ERR_INVALID_POINTER	無効な期間が指定されました。
SSP_ERR_ABORTED	Systick タイマの更新に失敗しました。

8.5.25 R_CGC_VersionGet

```
ssp_err_t R_CGC_VersionGet ( ssp_version_t *const p_version )
```

8.5.25.1 概要説明

ドライバのバージョンを返します。

8.5.25.2 詳細説明

表 620: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に実行されました。

8.5.26 モジュール

- ビルドタイム構成

8.5.26.1 ビルドタイム構成

定義

- #define CGC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータチェックのコードを含めるかどうかを指定します
  BSP_CFG_PARAM_CHECKING_ENABLE の設定はシステムデフォルト設定を使用します。1 の設定にはパラメータチェックが含まれます。0 はパラメータチェックをコンパイルアウトします。

- #define CGC_CFG_MAIN_OSC_CLOCK_SOURCE

  初期値 :(0)

  レゾネータまたはクリスタルが使用される場合は CGC_CFG_MAIN_OSC_CLOCK_SOURCE を 0 に設定します。外部オシレータを使用する場合は CGC_CFG_MAIN_OSC_CLOCK_SOURCE を 1 に設定します。

- #define CGC_CFG_MAIN_OSC_WAIT

  初期値 :(0x09)

  CGC_CFG_MAIN_OSC_WAIT を下記のいずれかの値に設定します。この遅延は CGC_CFG_MAIN_OSC_CLOCK_SOURCE が 0 に設定されてレゾネータ / クリスタルが使用されていることを示した場合にのみ構成されます。この遅延は CGC_CFG_MAIN_OSC_CLOCK_SOURCE が 0 に設定されてレゾネータ / クリスタルが使用されていることを示した場合にのみ構成されます。メインクロックオシレーション安定性時間は、オシレータの製造元が推奨する安定性時間以上に設定します。0 0 0 0 0: 待ち時間 = 3 サイクル 0 0 0 0 1: 待ち時間 = 35 サイクル 0 0 0 1 0: 待ち時間 = 67 サイクル 0 0 0 1 1: 待ち時間 = 131 サイクル 0 0 1 0 0: 待ち時間 = 259 サイクル 0 0 1 0 1: 待ち時間 = 547 サイクル 0 0 1 1 0: 待ち時間 = 1059 サイクル 0 0 1 1 1: 待ち時間 = 2147 サイクル 0 1 0 0 0: 待ち時間 = 4291 サイクル 0 1 0 0 1: 待ち時間 = 8163 サイクル デフォルトは最大時間に設定されています。

- #define CGC_CFG_SUBCLOCK_DRIVE

  初期値 :(0x00)

  CGC_CFG_SUBCLOCK_DRIVE はクリスタルパラメータに基づいて以下のいずれかの値に設定します。0: Standard (12.5 pf)1: Middle (4.4 pf)

8.6 CRC

CRC カルキュレータ （CRC）のドライバ。

このモジュールは CRC カルキュレータ （CRC）をサポートします。次のインタフェースを実装します。

- CRC インタフェース

8.6.1 Functions

- R_CRC_Open
- R_CRC_Close
- R_CRC_CalculatedValueGet
- R_CRC_SnoopEnable
- R_CRC_SnoopDisable
- R_CRC_SnoopCfg
- R_CRC_Calculate
- R_CRC_VersionGet

8.6.2 定義

- #define CRC_CODE_VERSION_MAJOR

  初期値 :(1)

- #define CRC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.6.3 R_CRC_Open

```
ssp_err_t R_CRC_Open ( crc_ctrl_t *const p_ctrl , crc_cfg_t const *const p_cfg )
```

8.6.3.1 概要説明

CRC ドライバ モジュールを開きます。

8.6.3.2 詳細説明

次を実装します :open

CRC ドライバ モジュールを開き、渡された構成構造体に従ってブロックを初期化します。

表 621: 戻り値

名前	説明
SSP_SUCCESS	構成が正常に行われました。
p_ctrl の可能性があります。	p_ctrl or p_cfg が NULL です。

8.6.4 R_CRC_Close

`ssp_err_t` R_CRC_Close (`crc_ctrl_t` *const p_ctrl)

8.6.4.1 概要説明

CRC モジュール ドライバを閉じます。

8.6.4.2 詳細説明

次を実装します :close

表 622: 戻り値

名前	説明
SSP_SUCCESS	構成が正常に行われました。
p_ctrl の可能性があります。	p_ctrl が NULL です。

8.6.5 R_CRC_CalculatedValueGet

`ssp_err_t` R_CRC_CalculatedValueGet (`crc_ctrl_t` *const p_ctrl , uint32_t * calculatedValue)

8.6.5.1 概要説明

現在の計算値を返します。

8.6.5.2 詳細説明

次を実装します :crcResultGet

CRC 計算は実行値に対して行われます。この関数は現在の計算値を返します。

表 623: 戻り値

名前	説明
SSP_SUCCESS	計算値が正常に返されました。
SSP_ERR_INVALID_ARG	bitOrder または多項式が有効値ではありませんでした。

8.6.6 R_CRC_SnoopEnable

```
ssp_err_t  R_CRC_SnoopEnable ( crc_ctrl_t  *const p_ctrl ,  uint32_t crc_seed )
```

8.6.6.1 概要説明

スヌーピングを有効にします。

8.6.6.2 詳細説明

次を実装します :snoopEnable

表 624: 戻り値

名前	説明
SSP_SUCCESS	スヌープが有効です。

8.6.7 R_CRC_SnoopDisable

```
ssp_err_t  R_CRC_SnoopDisable ( crc_ctrl_t  *const p_ctrl )
```

8.6.7.1 概要説明

スヌーピングを無効にします。

8.6.7.2 詳細説明

次を実装します :snoopDisable

表 625: 戻り値

名前	説明
SSP_SUCCESS	スヌープが無効です。

8.6.7.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.6.8 R_CRC_SnoopCfg

```
ssp_err_t  R_CRC_SnoopCfg ( crc_ctrl_t  *const p_ctrl , crc_snoop_cfg_t  *const p_snoop_cfg )
```

8.6.8.1 概要説明

スヌープ チャネルと方向を設定します。

8.6.8.2 詳細説明

次を実装します :snoopCfg

CRC カルキュレータは、先頭の 10 個の SCI チャネルのいずれかを経由する読み取りおよび書き込みに対して使用できます。たとえば、" チャネル 0 を経由する送信 " を対象として設定すると、SCI チャネル 0 から書き出されるすべてのバイトが、CRC カルキュレータに直接、明示的に書き込まれた値と同様に、CRC カルキュレータに送信されます。

表 626: 戻り値

名前	説明
SSP_SUCCESS	スヌープの構成が正常に行われました。
p_ctrl の可能性があります。	p_snoop_cfg が NULL です。
SSP_ERR_INVALID_ARG	チャネルが範囲外か、方向が無効な値でした。

8.6.8.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.6.9 R_CRC_Calculate

`ssp_err_t` R_CRC_Calculate (`crc_ctrl_t` *const p_ctrl , void * inputBuffer , uint32_t length , uint32_t crc_seed , uint32_t * calculatedValue)

8.6.9.1 概要説明

8 ビット データのブロックに対して CRC 計算を実行します。

8.6.9.2 詳細説明

次を実装します :calculate

この関数は、8 ビット値の配列に対して CRC 計算を実行し、8 ビットの計算値を返します

表 627: 戻り値

名前	説明
SSP_SUCCESS	正常に計算されました。
p_ctrl の可能性があります。	p_ctrl、inputBuffer、または calculatedValue のいずれかが NULL です。

8.6.10 R_CRC_VersionGet

`ssp_err_t` R_CRC_VersionGet (`ssp_version_t` *const p_version)

8.6.10.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

8.6.10.2 詳細説明

次を実装します :versionGet

表 628: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version が NULL です。

8.7 CTSU

静電容量方式タッチ感応ユニット （CTSU）のドライバ。

CTSU を制御するドライバ。CTSU を起動し、チャネルがタッチされているかどうかを判定するために必要なパラメータを維持する関数を備えています。

次のインタフェースを実装します。

- CTSU インタフェース r_ctsu_api.h で定義されます

8.7.1 Functions

- R_CTSU_VersionGet
- R_CTSU_Open
- R_CTSU_Start_Scan
- R_CTSU_Update_Parameters
- R_CTSU_Process
- R_CTSU_Read_Results
- R_CTSU_Control
- R_CTSU_Close

8.7.2 定義

- #define CTSU_CODE_VERSION_MAJOR

  初期値 :(1)
- #define CTSU_CODE_VERSION_MINOR

  初期値 :(1)
- #define CTSU_MAX_CHANNELS

  初期値 :(36)

8.7.3 R_CTSU_VersionGet

```
ssp_err_t R_CTSU_VersionGet ( ssp_version_t *const p_version )
```

8.7.4 R_CTSU_Open

```
ssp_err_t R_CTSU_Open ( ctsu_ctrl_t * p_ctrl , ctsu_cfg_t * p_cfg )
```

8.7.4.1 概要説明

初期オートチューニング （オプション）とベースライン初期化 （オプション）を使用して、CTSU を初期化し、タッチ判定のソフトウェア ドライバ パラメータを初期化します。

8.7.4.2 詳細説明

表 629: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。CTSU が正常に初期化されました。
p_ctrl の可能性があります。	無効な動作モード （SELF または MUTUAL キャパシタンス）または無効なソフト オプション、無効なクローズ オプション
Null ポインタが指定されました。	NULL ポインタが必要なパラメータに引数として受け渡されました。または、メモリ割り当てに失敗しました。
SSP_ERR_INVALID_POINTER	不正なパラメータが見つかりました。
NULL ポインタが必要なパラメータに引数として受け渡されました。	フィルタの深さが大きすぎるか、あるいはボードまたは現在の状態に対し調整が有効でないため、初期自動調整に失敗しました。

I：データ処理関数は、上級ユーザー向けです。初心者ユーザーは、ダミー引数 （NULL 以外）を指定してください。内部のメンバーは、NULL のままにします。

I :CFG_AUTO_TUNE が有効な場合、Open を呼び出すためには、グローバル割り込み動作が有効化されている必要があります。

8.7.4.3 関数のステップ

- 簡単なチェックを実行します

- CTSU を再構成できません。処理前にクローズを呼び出すことでアクティブな構成をアンロードする必要があります。
- CTSU の電源をオンにします
- 構成設定値をブラインド コピーして、CTSU を初期化します
- 除外するチャネルを検出します
- アクティブ チャネルの数は、除外されたチャネル数より大きくなければなりません
- 以下のループで使用される変数
- 使用するバッファを初期化します
- プライマリ バッファにデータを格納します
- 動作関数を選択します
- 割り込み動作を有効にします
- オプションのアクションを実行します
- R_Touch が使用できることを表示します
- どの CTSU 構成が開かれているかを保存します
- オープン時に使用された CTSU オプションを保存します
- パラメータの更新時に使用する CTSU オプションを保存します
- 渡される通知関数の引数を保存します

8.7.5 R_CTSU_Start_Scan

```
ssp_err_t R_CTSU_Start_Scan ( ctsu_ctrl_t * p_ctrl )
```

8.7.5.1 概要説明

CTSU ハードウェア スキャンを開始します。この関数が呼び出されるレートは基本的にスキャン レートです。

8.7.5.2 詳細説明

表 630: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。CTSU スキャンが正常に初期化されました。

表 630: 戻り値 (続き)

名前	説明
チャネルは現在動作中です。	スキャンを進行中

8.7.5.3 関数のステップ

- CTSU ハードウェアで、新しいスキャンを開始する準備が整いました。
- 新しいスキャンを開始します。
- ハードウェアはビジーです

8.7.6 R_CTSU_Update_Parameters

```
ssp_err_t R_CTSU_Update_Parameters ( ctsu_ctrl_t * p_ctrl )
```

8.7.6.1 概要説明

スキャンされたデータの結果の処理を完了するたびに呼び出される関数。

8.7.6.2 詳細説明

この関数は、次のタスクを実行します。

0. CTSU 動作内のエラーを確認します。
1. CTSU からの未加工値出力に対してフィルタを実行します。
2. チャネルがタッチされているかどうかを判定し、バイナリおよび
   sensor_baseline と現在のセンサー値の間の差を更新します。
3. 更新が完了したら、この関数からユーザーが指定したコールバックが
   CTSU_EVENT_PARAMETERS_UPDATED イベントとともに呼び出されます。
4. (オプション) (単一またはすべての) チャネルがタッチされていない場合、オートチューニングを実行します。
5. (オプション) (単一またはすべての) チャネルがタッチされていない場合、ドリフト補正を実行します。
   ドリフト補正 := N 回のリードの後に、センサー ベースラインを移動します。

表 631: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。
Null ポインタが指定されました。	NULL ポインタが必要なパラメータに引数として受け渡されました。または、メモリ割り当てに失敗しました。

表 631: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_POINTER	不正なパラメータが見つかりました。
NULL ポインタが必要なパラメータに引数として受け渡されました。	CTSU で ICOMP エラーが発生したため、自動調整に失敗しました。失敗したポイントを見つけ、各チャネルの構成パラメータをチェックしてください。

8.7.6.3 関数のステップ

- CTSU のエラー状態を確認します
- 測定された各チャネルを更新します
- 以下のループに対する変数を初期化します
- チャネルは除外されます。バイパスの計算。
- どのチャネルがタッチされているかを確認します。
- 以下のループに対する変数を初期化します
- チャネルは除外されます。バイパスの計算。
- 通知を提供するコールバックを実行します
- ドリフト補正を実行します

8.7.7 R_CTSU_Process

```
ssp_err_t  R_CTSU_Process ( ctsu_ctrl_t  * p_ctrl , ctsu_process_option_t   opts )
```

8.7.7.1 概要説明

ユーザーがメイン ループから呼び出す必要がある関数。この関数は R_CTSU_Update_Parameters 関数によって非推奨とされており、インタフェース API のユーザーには表示されません。

8.7.7.2 詳細説明

この関数は、次のタスクを実行します。

0. CTSU 動作内のエラーを確認します。
1. CTSU からの未加工値出力に対してフィルタを実行します。
2. チャネルがタッチされているかどうかを判定し、バイナリおよび
   sensor_baseline と現在のセンサー値の間の差を更新します。
3. (オプション) (単一またはすべての) チャネルがタッチされていない場合、オートチューニングを実行します。
4. (オプション) (単一またはすべての) チャネルがタッチされていない場合、ドリフト補正を実行します。
   ドリフト補正 := N 回のリードの後に、センサー ベースラインを移動します。
5. (オプション) 新しいスキャンを開始します。

表 632: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。
Null ポインタが指定されました。	NULL ポインタが必要なパラメータに引数として受け渡されました。または、メモリ割り当てに失敗しました。
SSP_ERR_INVALID_POINTER	不正なパラメータが見つかりました。
NULL ポインタが必要なパラメータに引数として受け渡されました。	CTSU で ICOMP エラーが発生したため、自動調整に失敗しました。失敗したポイントを見つけ、各チャネルの構成パラメータをチェックしてください。

8.7.7.3 関数のステップ

- CTSU のエラー状態を確認します
- 測定された各チャネルを更新します
- 以下のループに対する変数を初期化します
- 以下のループに対する変数を初期化します
- どのチャネルがタッチされているかを確認します。
- ボタンを更新します
- スライダを更新します
- ドリフト補正を実行します
- CTSU を実行する準備ができました
- 新しいスキャンを開始します。
- 状態マシンのコピーを保存します

8.7.8 R_CTSU_Read_Results

ssp_err_t R_CTSU_Read_Results (ctsu_ctrl_t * p_ctrl , void * p_dest , ctsu_read_t opts , ctsu_channel_pair_t const * channels , const uint16_t count)

8.7.8.1 概要説明

スキャン実行後に、ユーザーが結果を表示するための関数。

8.7.8.2 詳細説明

表 633: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。
提供されたコマンド オプションが無効です。	提供されたコマンド オプションが無効です。
Null ポインタが指定されました。	NULL ポインタが必要なパラメータに引数として受け渡されました。または、メモリ割り当てに失敗しました。
SSP_ERR_INVALID_POINTER	不正なパラメータが見つかりました。
NULL ポインタが必要なパラメータに引数として受け渡されました。	CTSU で ICOMP エラーが発生したため、自動調整に失敗しました。失敗したポイントを見つけ、各チャネルの構成パラメータをチェックしてください。

8.7.8.3 関数のステップ

- 要求されたオプションに関する結果を取得します
- arr[3] と arr[2] をそれぞれ上位バイト、下位バイトとして組み立てます
- arr[3] と arr[2] をそれぞれ上位バイト、下位バイトとして組み立てます
- arr[3] と arr[2] をそれぞれ上位バイト、下位バイトとして組み立てます
- arr[3] と arr[2] をそれぞれ上位バイト、下位バイトとして組み立てます
- arr[3] と arr[2] をそれぞれ上位バイト、下位バイトとして組み立てます
- arr[3] と arr[2] をそれぞれ上位バイト、下位バイトとして組み立てます

8.7.9 R_CTSU_Control

ssp_err_t R_CTSU_Control (ctsu_cmd_t cmd , void * p_data)

8.7.9.1 概要説明

ユーザーが R_Touch レイヤーと CTSU 動作をクエリ操作できます。この関数はインタフェース API で使用することはできません。

8.7.9.2 詳細説明

表 634: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。
提供されたコマンド オプションが無効です。	提供されたコマンド オプションが無効です。
チャネルは現在動作中です。	CTSU スキャンを実行中です。

8.7.9.3 関数のステップ

- 制御要求を実行します

8.7.10 R_CTSU_Close

```
ssp_err_t R_CTSU_Close ( ctsu_ctrl_t * p_ctrl , ctsu_close_option_t opts )
```

8.7.10.1 概要説明

この関数は、CTSU ドライバのクローズ、CTSU 周辺機器のクロッキングの停止、すべての内部構造体のリセット、そして DTC 転送インスタンスのクローズに使用されます。また、この関数を使って構成を保存することで、再オープン時のウェイクアップを迅速に行うことができます。

8.7.10.2 詳細説明

表 635: 戻り値

名前	説明
SSP_SUCCESS	エラーはありません。

8.7.11 モジュール

- ビルドタイム構成

8.7.11.1 ビルドタイム構成

定義

- #define CTSU_CFG_PARAM_CHECKING_ENABLE

  初期値 :(#define BSP_CFG_PARAM_CHECKING_ENABLE)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define CTSU_CFG_MAX_ACTIVE_CHANNEL_COMBOS

  初期値 :(12)

  任意の構成でアクティブにしたい最大アクティブ チャネル数を定義します。セルフ キャパシタンス モードでは、これは RECEIVE として常に使用されるピンの最大数です （参照 : num_rx）。相互キャパシタンスでは、TRANSMIT ピンの最大数と RECEIVE ピンの最大数の乗算です （参照 : num_rx*num_tx）。

- #define CTSU_CFG_FILTER_DEPTH

  初期値 :(4)

  l-point 移動平均フィルタのウィンドウ サイズを定義します。フィルタ出力 = ((pow(2, CTSU_CFG_FILTER_DEPTH)-1)*previous output + input)/ pow(2, CTSU_CFG_FILTER_DEPTH); これは、ctsu_functions_t で用いられるデフォルトのフィルタ関数で用いられます。フィルタの深さが大きい場合、初期自動調整 （有効化されている場合）には時間がかかる場合があります

- #define CFG_AUTO_TUNE

  初期値 :(1)

  自動調整を有効化 / 無効化します。自動調整は、チャネルがタッチされていないときに CTSU によるセンサー カウントが参照カウント出力とほぼ同じであることを保証します。これにより、感度の最大化、PCB レイアウトに関する従来の制約やオーバーレイの厚さ / マテリアルへに関する制限の大幅な緩和、製造プロセスにおける較正の必要性排除が実現します。

- #define CTSU_DRIFTCOMP_WHEN_ALL_INACTIVE

  初期値 :(1)

  すべてのチャネルがタッチされていない場合にのみ、自動調整とドリフト補正を実行します。この機能を使うと、どのチャネルもタッチされていない場合にのみ、自動調整および / またはドリフト補正が実行されます。

- #define CFG_DRIFT_COMPENSATION

  初期値 :(1)

  ドリフト補正を使用するかどうかを定義します。ドリフト補正はセンサー ベースラインを調節します。これは気温や湿度、ほこり / 汚れの蓄積などが原因でセンサー値が安定しない場合に役立ちます。

- #define CALIB_SCAN_COUNT

  初期値 :(4)

  初期センサー ベースラインを計算するときに考慮に入れるスキャン数を定義します。

- #define DC_TIMING_STEADY_STATE

  初期値 :(500)

  ドリフト補正は N 回ごとに起こります。N は以下で定義します。

- #define DC_TIMING_INITIAL_RATE

  初期値 :(5)

- #define DC_TIMING_BUTTON_RELEASE_RATE

  初期値 :(500)

- #define DRIFT_COMP_BASIC

  初期値 :(0)

- #define DRIFT_COMP_ALT_1

  初期値 :(1)

- #define DRIFT_COMP_ALT_2

  初期値 :(2)

- #define DRIFT_COMP_ALT_3

  初期値 :(3)

- #define DRIFT_COMP_METHOD

  初期値 :(DRIFT_COMP_ALT_1)

- #define AT_TIMING

  初期値 :(`#define DC_TIMING_STEADY_STATE` >> 1)

  自動調整は N スキャンごとに実行します。N の値は、AT_TIMING を使って定義します。自動調整は、ドリフト補正よりも頻繁に行われる必要があります。ドリフト補正を使用する場合、自動調整レートは定常状態のドリフト補正レートの 2 倍に設定されます。

- #define AT_THRESHOLD_PERCENT

  初期値 :(2)

  ランタイム オフセット調整が始まるしきい値に関する偏差の量を定義します。

- #define AT_MAX_DURATION

  初期値 :(`#define CTSU_CFG_MAX_ACTIVE_CHANNEL_COMBOS` * 50 * `#define CTSU_CFG_FILTER_DEPTH`)

  初期自動調整にはかなりの時間がかかります。以下で定義したスキャン数を超える場合、ユーザーは中止することができます。

- #define USING_DTC_FOR_CTSU

  初期値 :(1)

  DTC 転送を CPU ISR0 の代わりにデータの移動に使用するかどうかを定義します。- Not used 1 - DTC ブロック転送をロング アドレス モードで使用する （推奨）

8.8 DAC

12 ビット D/C コンバータ （DAC12）のドライバ。

8.8.1 概要

このモジュールは、次のインタフェースを実装します : DAC インタフェース。

8.8.2 Functions

- R_DAC_Open
- R_DAC_Close
- R_DAC_Write
- R_DAC_Start
- R_DAC_Stop
- R_DAC_VersionGet

8.8.3 定義

- #define DAC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define DAC_CODE_VERSION_MINOR

  初期値 :(1)

8.8.4 R_DAC_Open

```
ssp_err_t R_DAC_Open ( dac_ctrl_t * p_ctrl , dac_cfg_t const *const p_cfg )
```

8.8.4.1 概要説明

ユーザーズマニュアルで説明されている必要な初期化を実行します。open を実装します。この関数では、1 つの DAC チャネルが構成され、チャネルが開始されるほか、DAC API 書き込み関数とクローズ関数のためのハンドルが提供されます。他の DAC API 関数を呼び出す前に、一度呼び出す必要があります。チャネルを開いた後オープン関数を呼び出すには、まずクローズ関数を呼び出す必要があります。

8.8.4.2 詳細説明

表 636: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開かれました。
p_ctrl の可能性があります。	p_ctrl または p_cfg のいずれかのパラメータが NULL の可能性があります。p_cfg で要求したチャネル ID は r_bsp_config.h で選択したデバイスで使用できない可能性があります。p_cfg の data_format 値が範囲外です。p_cfg の ad_da_synchronized 値が範囲外です。
SSP_ERR_INVALID_ARGUMENT	DAC リソースはロックされています。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.8.4.3 関数のステップ

- 入力パラメータを検証します。
- DAC ハードウェア リソースをロックします。
- DAC デバイスの電源をオンにします。
- 制御ブロックを初期化します
- チャネルを停止します。
- データ フォーマットを左揃えまたは右揃えに構成します。
- D/A-A/D 同期開始制御レジスタ （DAADSCR）を構成します。
- チャネルの出力アンプ構成を設定します。
- 基準電圧を設定します。
- チャネル状態情報を初期化します。
- すべてが完了しました。復帰します。

8.8.5 R_DAC_Close

```
ssp_err_t  R_DAC_Close ( dac_ctrl_t  * p_ctrl )
```

8.8.5.1 概要説明

D/A 変換を停止し、出力を停止して、DAC チャネルを閉じます。

8.8.5.2 詳細説明

表 637:　戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました。
p_ctrl の可能性があります。	p_ctrl が NULL です。
タッチ パネルが設定されていません。	p_ctrl に関連づけられたチャネルが開かれていません。

8.8.5.3 関数のステップ

- チャネルが開かれていることを確認します。
- チャネルが有効な場合は、D/A 変換を無効化し、出力を停止します。
- チャネルを停止します
- チャネル状態情報を更新します。
- DAC デバイスの電源をオフにします。
- DAC ハードウェア リソースをロック解除します

8.8.6 R_DAC_Write

```
ssp_err_t  R_DAC_Write ( dac_ctrl_t  * p_ctrl , dac_size_t  value )
```

8.8.6.1 概要説明

D/A コンバータにデータを書き込み、出力が有効でない場合は有効にします。

8.8.6.2 詳細説明

表 638: 戻り値

名前	説明
SSP_SUCCESS	データが D/A コンバータに正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl が NULL です。
タッチ パネルが設定されていません。	p_ctrl に関連づけられたチャネルが開かれていません。

I：データがチャネルに正常に書き込まれた後、書き込み関数が自動で D/A 変換を開始します。

8.8.6.3 関数のステップ

- チャネルが開かれていることを確認します。
- D/A コンバータに値を書き込みます。
- コンバータがアイドル状態である場合は開始します。
- チャネルを開始します

8.8.7 R_DAC_Start

```
ssp_err_t  R_DAC_Start ( dac_ctrl_t  * p_ctrl )
```

8.8.7.1 概要説明

D/A 変換出力が開始されていない場合は開始します。

8.8.7.2 詳細説明

表 639: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開始されました。

表 639: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	p_ctrl が NULL です。
タッチ パネルが設定されていません。	p_ctrl に関連づけられたチャネルが開かれていません。

8.8.7.3 関数のステップ

- チャネルが開かれていることを確認します。
- チャネル出力がすでに有効である場合は、何も処理を行いません。
- 出力を有効化します。
- 内部状態を更新します。

8.8.8 R_DAC_Stop

```
ssp_err_t  R_DAC_Stop ( dac_ctrl_t  * p_ctrl )
```

8.8.8.1 概要説明

D/A 変換を停止し、出力信号を無効化します。

8.8.8.2 詳細説明

表 640: 戻り値

名前	説明
SSP_SUCCESS	コントロールが正常に停止されました。
p_ctrl の可能性があります。	p_ctrl が NULL です。
タッチ パネルが設定されていません。	p_ctrl に関連づけられたチャネルが開かれていません。

8.8.8.3 関数のステップ

- チャネルが開かれていることを確認します。
- チャネル出力がすでに無効である場合は、何も処理を行いません。
- 出力を無効化します。

- 内部状態をマークします。

8.8.9 R_DAC_VersionGet

```
ssp_err_t  R_DAC_VersionGet ( ssp_version_t  * p_version )
```

8.8.9.1 概要説明

バージョンを取得し、指定されたポインタ p_version に格納します。

8.8.9.2 詳細説明

表 641: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報を正常に取得しました。

8.8.10 モジュール

- ビルドタイム構成

8.8.10.1 ビルドタイム構成

定義

- #define DAC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define  BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.9 DMAC

DMA コントローラ （DMAC）。

8.9.1 概要

転送インタフェース を拡張します。

ダイレクト メモリ アクセス コントローラ （DMA）では、割り込みの代わり、または割り込みに加えて、データ転送を行うことができます。ソフトウェア スタートを使用したデータ転送もサポートしています。

I :TRANSFER_MODE_BLOCK と TRANSFER_MODE_REPEAT では、転送長が 1024 （10 ビット）に制限されています。

I : このドライバは、transfer_irq_t の TRANSFER_IRQ_END のみをサポートします。

8.9.2 Functions

- R_DMAC_Open
- R_DMAC_Reset
- R_DMAC_Start
- R_DMAC_Stop
- R_DMAC_Enable
- R_DMAC_Disable
- R_DMAC_InfoGet
- R_DMAC_Close
- R_DMAC_VersionGet

8.9.3 定義

- #define DMAC_CODE_VERSION_MAJOR

  初期値 :(1)

- #define DMAC_CODE_VERSION_MINOR

  初期値 :(1)

- #define DMAC_REPEAT_BLOCK_MAX_LENGTH

  初期値 :(0x400)

  長さはリピートおよびブロック モードで 1024 に制限されています

- #define DMAC_NORMAL_MAX_LENGTH

  初期値 :(0x400)

  長さは通常ブロックモードで 1024 の移動に制限されています

8.9.4 R_DMAC_Open

`ssp_err_t` R_DMAC_Open (`transfer_ctrl_t` *const p_ctrl , `transfer_cfg_t` const *const p_cfg)

8.9.4.1 概要説明

転送を初期化し、ICU で転送を有効にします。open を実装します。

8.9.4.2 詳細説明

表 642: 戻り値

名前	説明
SSP_SUCCESS	正常に開きました。移動が構成され、トリガーが発生すると移動を開始します。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DMAC_Open を呼び出してコントロールブロックを初期化します。
内部エラーが発生しました。	すべての DMAC チャネルが使用中です。

8.9.4.3 関数のステップ

- プライオリティの最も高い未使用チャネルを選択し、BSP ハードウェア ロックを取得します。すべてのチャネルが使用されている場合はエラーを返します。
- ユーザーズマニュアル『NoSecurity_r01uh0488ej0040_sc32.pdf』の 16.3.7 章にあるフローチャート「Activating the DMAC （DMAC のアクティブ化)」に従って DMAC を構成します。

- p_callback が選択されている状態で、割り込みを有効にし、p_callback を配列に格納して、ISR からアクセスできるようにします。
-

  I : 転送エスケープ割り込みはサポートされていません。

  内部変数を更新します。
- auto_enable が true であって、ソフトウェア スタートが使用されている場合は、転送と ELC イベントを有効化します。

8.9.5 R_DMAC_Reset

`ssp_err_t` R_DMAC_Reset (`transfer_ctrl_t` *const p_ctrl , void const *volatile p_src , void *volatile p_dest , uint16_t const num_transfers)

8.9.5.1 概要説明

転送ソース、転送先、転送回数をリセットします。

8.9.5.2 詳細説明

表 643: 戻り値

名前	説明
SSP_SUCCESS	移動が正常にリセットされました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DMAC_Open を呼び出してコントロールブロックを初期化します。

8.9.5.3 関数のステップ

- このアクティベーション ソースで転送を有効化します。

8.9.6 R_DMAC_Start

`ssp_err_t` R_DMAC_Start (`transfer_ctrl_t` *const p_ctrl , `transfer_start_mode_t` mode)

8.9.6.1 概要説明

転送を開始します。start を実装します。

8.9.6.2 詳細説明

表 644: 戻り値

名前	説明
SSP_SUCCESS	移動の開始が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DMAC_Open を呼び出してコントロールブロックを初期化します。
データ構造体が割り当てられませんでした。	ハンドルはソフトウェアアクティベーションのために構成されていません。

8.9.6.3 関数のステップ

- オートクリア ビットとソフトウェア スタート ビットを設定します。
- チャネルに割り込みアクティベーション ソースが構成された場合、ソフトウェア スタートはサポートされません。

8.9.7 R_DMAC_Stop

```
ssp_err_t  R_DMAC_Stop ( transfer_ctrl_t  *const p_ctrl )
```

8.9.7.1 概要説明

転送を停止します。stop を実装します。

8.9.7.2 詳細説明

表 645: 戻り値

名前	説明
SSP_SUCCESS	移動の停止が正常に書き込まれました。

表 645: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DMAC_Open を呼び出してコントロールブロックを初期化します。

8.9.7.3 関数のステップ

- オート クリア ビットをリセットし、ソフトウェア転送要求をクリアします。

8.9.8 R_DMAC_Enable

```
ssp_err_t  R_DMAC_Enable ( transfer_ctrl_t  *const p_ctrl )
```

8.9.8.1 概要説明

転送を有効にします。enable を実装します。

8.9.8.2 詳細説明

表 646: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DTC_Open を呼び出してコントロールブロックを初期化します。

8.9.8.3 関数のステップ

- 転送を有効にします。

8.9.9 R_DMAC_Disable

```
ssp_err_t  R_DMAC_Disable ( transfer_ctrl_t  *const p_ctrl )
```

8.9.9.1 概要説明

転送を無効にします。disable を実装します。

8.9.9.2 詳細説明

表 647: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DTC_Open を呼び出してコントロールブロックを初期化します。

8.9.9.3 関数のステップ

- 転送を無効にします。

8.9.10 R_DMAC_InfoGet

ssp_err_t R_DMAC_InfoGet (transfer_ctrl_t *const p_ctrl , transfer_properties_t *const p_info)

8.9.10.1 概要説明

指定されたポインタにドライバ固有の情報を設定します。infoGet を実装します。

8.9.10.2 詳細説明

表 648: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。

8.9.10.3 関数のステップ

- 転送がアクティブの場合、転送を p_in_progress に格納します。
- 最大転送長を格納します。
- 残りの転送長を保存します。

8.9.11 R_DMAC_Close

```
ssp_err_t  R_DMAC_Close ( transfer_ctrl_t  *const p_ctrl )
```

8.9.11.1 概要説明

ICU で転送を無効化した後、DTC ベクター テーブルで転送データをクリアします。close を実装します。

8.9.11.2 詳細説明

表 649:　戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DMAC_Open を呼び出してコントロールブロックを初期化します。

8.9.11.3 関数のステップ

- DMAC 転送を無効化し、DMAC 割り込みを無効化した後、ICU レジスタで DMAC トリガを削除します。
- 内部ドライバ データのクリアをクリアします
- このチャネルの BSP ハードウェア ロックを解放します
- ID をクリアして、制御ブロックを再利用できるようにします。

8.9.12 R_DMAC_VersionGet

```
ssp_err_t  R_DMAC_VersionGet ( ssp_version_t  *const p_version )
```

8.9.12.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンをセットします。

8.9.12.2 詳細説明

表 650: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	入力パラメータが無効です。

8.9.13 モジュール

- ビルドタイム構成

8.9.13.1 ビルドタイム構成

定義

- #define DMAC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.10 DOC

データ操作回路 （DOC）のドライバ。

8.10.1 概要

このモジュールは、データ操作回路 （DOC）を使用して DOC インタフェース を実装します。

8.10.2 Functions

- R_DOC_Open
- R_DOC_Close
- R_DOC_StatusGet
- R_DOC_StatusClear
- R_DOC_Write
- R_DOC_InputRegisterWrite
- R_DOC_VersionGet

8.10.3 定義

- #define DOC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define DOC_CODE_VERSION_MINOR

  初期値 :(1)

8.10.4 R_DOC_Open

```
ssp_err_t R_DOC_Open ( doc_ctrl_t *const p_ctrl , doc_cfg_t const *const p_cfg )
```

8.10.4.1 概要説明

比較、加算、または減算モードで、データ操作回路 （DOC）を構成します。open を実装します。

8.10.4.2 詳細説明

構成構造体でコールバックが NULL でない場合、DOC IRQ が有効化されます。

表 651: 戻り値

名前	説明
SSP_SUCCESS	DOC が正常に構成されました。
チャネルは現在動作中です。	モジュールはすでに開いています。
p_ctrl の可能性があります。	1 つまたはそれ以上のポインタが NULL をポイントしています。
SSP_ERR_INVALID_POINTER	ISR が有効ではありません。bsp_irq_cfg.h で ISR を有効にします。
SSP_ERR_INVALID_ARGUMENT	DOC リソースがロックされています。

I：この関数は再入可能です。

8.10.4.3 関数のステップ

- DOC ハードウェア リソースをロックします
- DOCR レジスタ経由で DOC を構成します。
- コールバック パラメータが NULL でない場合は、IRQ を構成します
- ドライバをオープンとしてマークします。

8.10.5 R_DOC_Close

```
ssp_err_t  R_DOC_Close ( doc_ctrl_t  *const p_ctrl )
```

8.10.5.1 概要説明

モジュール ドライバを閉じます。モジュール停止モードを有効にします。close を実装します。

8.10.5.2 詳細説明

DOC を閉じるには、それがあらかじめオープン API で開かれている必要があります。オープン時には、タイプ doc_ctrl_t の制御構造体がオープン API に渡されます。クローズ API には、これと同じ制御構造体を渡す必要があります。

表 652: 戻り値

名前	説明
SSP_SUCCESS	モジュールが正常に閉じられました。
タッチ パネルが設定されていません。	ドライバが開かれていません。
p_ctrl の可能性があります。	ポインタが NULL をポイントしています。

I：この関数は再入可能です。

I：この関数は NVIC での DOC 割り込みを無効化します。

8.10.5.3 関数のステップ

- IRQ が有効な場合、NVIC で無効化します。
- DOPCF を DOCR でクリアします
- ドライバをクローズ済みとしてマークします。
- DOC ハードウェア リソースをロック解除します

8.10.6 R_DOC_StatusGet

`ssp_err_t` R_DOC_StatusGet (`doc_ctrl_t` *const p_ctrl , `doc_status_t` * p_status)

8.10.6.1 概要説明

比較 / 加算 / 減算ステータスを返します。statusGet を実装します。

8.10.6.2 詳細説明

ステータスがデータ操作回路フラグ （DOPCF）から読み取られます。

表 653: 戻り値

名前	説明
SSP_SUCCESS	ステータスが正常に読み取られました。
タッチ パネルが設定されていません。	ドライバが開かれていません。
p_ctrl の可能性があります。	1 つまたはそれ以上のポインタが NULL をポイントしています。

I：この関数は再入可能です。

8.10.6.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.10.7 R_DOC_StatusClear

```
ssp_err_t  R_DOC_StatusClear ( doc_ctrl_t  *const p_ctrl )
```

8.10.7.1 概要説明

ハードウェア レイヤーで、DOPCF ステータス フラグをクリアします。このフラグは DOC 動作の結果を示します。statusClear を実装します。

8.10.7.2 詳細説明

表 654: 戻り値

名前	説明
SSP_SUCCESS	中断が正常にクリアされました。
タッチ パネルが設定されていません。	ドライバが開かれていません。
p_ctrl の可能性があります。	ポインタが NULL をポイントしています。

I：この関数は再入可能です。

8.10.7.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.10.8 R_DOC_Write

`ssp_err_t` R_DOC_Write (`doc_ctrl_t` *const p_ctrl , `doc_data_t` *const p_data)

8.10.8.1 概要説明

DODIR レジスタと DODSR レジスタに書き込みます。write を実装します。

8.10.8.2 詳細説明

比較モードでは、16 ビット参照データが DODSR レジスタに書き込まれ、比較のためのデータが DODIR に書き込まれます。加算モードでは、初期データが DODSR に書き込まれ、加算されるデータが DODIR に書き込まれます。減算モードでは、初期データが DODSR に書き込まれ、減算されるデータが DODIR に書き込まれます。

DODSR と DODIR の両方を変える場合は、DODSR を先に更新します。

表 655: 戻り値

名前	説明
SSP_SUCCESS	レジスタに値が正常に書き込まれました。
タッチ パネルが設定されていません。	ドライバが開かれていません。
p_ctrl の可能性があります。	1 つまたはそれ以上のポインタが NULL をポイントしています。

I：この関数は再入可能です。

8.10.8.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.10.9 R_DOC_InputRegisterWrite

`ssp_err_t` R_DOC_InputRegisterWrite (`doc_ctrl_t` *const p_ctrl , `doc_size_t` data)

8.10.9.1 概要説明

DODIR レジスタに書き込みます。inputRegisterWrite を実装します。

8.10.9.2 詳細説明

DODIR レジスタのみに書き込みます。

表 656: 戻り値

名前	説明
SSP_SUCCESS	DODIR レジスタに値が正常に書き込まれました。
タッチ パネルが設定されていません。	ドライバが開かれていません。
p_ctrl の可能性があります。	1 つまたはそれ以上のポインタが NULL をポイントしています。

I：この関数は再入可能です。

8.10.9.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.10.10 R_DOC_VersionGet

`ssp_err_t` R_DOC_VersionGet (`ssp_version_t` *const p_version)

8.10.10.1 概要説明

DOC HAL ドライバのバージョンを返します。versionGet を実装します。

8.10.10.2 詳細説明

表 657: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報を正常に読み取りました。
p_ctrl の可能性があります。	ポインタが NULL をポイントしています。

I：この関数は再入可能です。

8.10.11 モジュール

- ビルドタイム構成

8.10.11.1 ビルドタイム構成

定義

- #define DOC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h.1 からのシステムデフォルト設定を使用します : パラメータのチェック .0 が含まれます : パラメータチェックをコンパイルアウトします。

8.11 DTC

データトランスファコントローラ （DTC）のドライバ。

8.11.1 概要

転送インタフェース を拡張します。

データトランスファコントローラでは、割り込みの代わり、または割り込みに加えて、データ転送を行うことができます。ソフトウェア スタートを使用したデータ転送はサポートされていません。

I :TRANSFER_MODE_BLOCK と TRANSFER_MODE_REPEAT では、転送長が 256 （8 ビット）に制限されています。

8.11.2 Functions

- R_DTC_Open
- R_DTC_Reset
- R_DTC_Start
- R_DTC_Stop
- R_DTC_Enable
- R_DTC_Disable
- R_DTC_InfoGet
- R_DTC_Close
- R_DTC_VersionGet

8.11.3 定義

- #define DTC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define DTC_CODE_VERSION_MINOR

  初期値 :(1)

- #define DTC_REPEAT_BLOCK_MAX_LENGTH

  初期値 :(0x100)

  長さは繰り返しおよびブロックモードで 256 の移動に制限されています
- #define DTC_NORMAL_MAX_LENGTH

  初期値 :(0x400)

  長さは通常ブロックモードで 1024 の移動に制限されています

8.11.4 R_DTC_Open

`ssp_err_t` R_DTC_Open (`transfer_ctrl_t` *const p_ctrl , `transfer_cfg_t` const *const p_cfg)

8.11.4.1 概要説明

ベクター テーブルで転送データを設定し、ICU で転送を有効化します。open を実装します。

8.11.4.2 詳細説明

表 658: 戻り値

名前	説明
SSP_SUCCESS	正常に開きました。移動が構成され、トリガーが発生すると移動を開始します。
p_ctrl の可能性があります。	入力パラメータが無効です。
チャネルは現在動作中です。	DTC の BSP ハードウェアロックが使用できないか、DTC ベクター表の IRQ のインデックスがすでに構成されています。
SSP_ERR_IRQ_BSP_DISABLED	アクティベーションソースに関連付けられた IRQ が BSP で有効にされていません。
SSP_ERR_NOT_ENABLED	自動有効がリクエストされましたが、無効な入力パラメータによって有効化に失敗しました。

8.11.4.3 関数のステップ

- DTC では、一度だけ初期化が必要です。これは、この関数の最初の呼び出し時にのみ処理されます。この初期化では以下の処理が行われます。

1) DTC のために BSP ハードウェアロックを取得して、このセクションスレッドを安全に保ち、別のドライバが DTC ブロックをロックした場合にこのロックの使用を防ぎます。
2) DTC ブロックの電源をオンにします。
3) ベクター表を NULL ポインタに初期化します。
4) ベクター表ベースアドレスを設定します。
5) DTC 移動を有効にします。

- 繰り返しモードおよびブロック モードの場合、初期長からリロード長にデータをコピーします。
- DTC 転送を構成します。詳細については、ハードウェアのマニュアルを参照してください。
- 内部変数を更新します。
- auto_enable が true であって、ソフトウェア スタートが使用されている場合は、転送と ELC イベントを有効化します。
- すべての設定が完了した後、リードスキップを有効化します。

8.11.5 R_DTC_Reset

```
ssp_err_t  R_DTC_Reset ( transfer_ctrl_t  *const p_ctrl ,  void const *volatile p_src ,  void *volatile p_dest ,  uint16_t const num_transfers )
```

8.11.5.1 概要説明

転送ソース、転送先、転送回数をリセットします。reset を実装します。

8.11.5.2 詳細説明

表 659:　戻り値

名前	説明
SSP_SUCCESS	移動が正常にリセットされました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DTC_Open を呼び出してコントロールブロックを初期化します。

表 659: 戻り値 (続き)

名前	説明
SSP_ERR_NOT_ENABLED	無効な入力パラメータにより、有効化に失敗しました。 • 移動ソースは NULL にできません。 • 移動先は NULL にできません。 • 移動の長さは 0 にできません。

8.11.5.3 関数のステップ

- このアクティベーション ソースで転送を無効化します。
- 設定を変更する前に、リードスキップを無効化します。これは後で有効化します。
- 入力パラメータに基づいて転送をリセットします。
- このアクティベーション ソースで転送を有効化します。
- すべての設定が完了した後、リードスキップを有効化します。

8.11.6 R_DTC_Start

`ssp_err_t` R_DTC_Start (`transfer_ctrl_t` *const p_ctrl , `transfer_start_mode_t` mode)

8.11.6.1 概要説明

転送を開始します。start を実装します。

8.11.6.2 詳細説明

表 660: 戻り値

名前	説明
SSP_SUCCESS	移動が正常に開始されました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DMAC_Open を呼び出してコントロールブロックを初期化します。

表 660: 戻り値 (続き)

名前	説明
データ構造体が割り当てられませんでした。	次のいずれかが無効です。 • ハンドルはソフトウェアアクティベーションのために構成されていません。 • モードが TRANSFER_START_MODE_SINGLE に設定されていません。 • DTC_SOFTWARE_START_ENABLE が ssp_cfg/driver/r_dtc_cfg.h で 0 （無効）に設定されています。

8.11.7 R_DTC_Stop

`ssp_err_t` R_DTC_Stop (`transfer_ctrl_t` *const p_ctrl)

8.11.7.1 概要説明

サポートされていないストップ関数用のプレースホルダー。stop を実装します。

8.11.7.2 詳細説明

表 661: 戻り値

名前	説明
データ構造体が割り当てられませんでした。	DTC ソフトウェアスタートはサポートされていません。

8.11.8 R_DTC_Enable

`ssp_err_t` R_DTC_Enable (`transfer_ctrl_t` *const p_ctrl)

8.11.8.1 概要説明

転送と ELC イベントがソフトウェア スタートのために使用されている場合は、それらを有効にします。enable を実装します。

8.11.8.2 詳細説明

表 662: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DTC_Open を呼び出してコントロールブロックを初期化します。

8.11.8.3 関数のステップ

- 転送を有効にします。

8.11.9 R_DTC_Disable

```
ssp_err_t  R_DTC_Disable ( transfer_ctrl_t  *const p_ctrl )
```

8.11.9.1 概要説明

転送を無効にします。disable を実装します。

8.11.9.2 詳細説明

表 663: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DTC_Open を呼び出してコントロールブロックを初期化します。

8.11.9.3 関数のステップ

- 転送を無効にします。

8.11.10 R_DTC_InfoGet

ssp_err_t R_DTC_InfoGet (transfer_ctrl_t *const p_ctrl , transfer_properties_t *const p_info)

8.11.10.1 概要説明

ドライバ固有の情報を設定します。infoGet を実装します。

8.11.10.2 詳細説明

表 664: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	入力パラメータが無効です。

8.11.10.3 関数のステップ

- 転送がアクティブの場合、転送を p_in_progress に格納します。
- アクティベーション ソースの転送情報は、DTC ベクター テーブルから取得されます。
- リピート転送の場合、上位バイトをマスクします。
- 最大転送長を格納します。

8.11.11 R_DTC_Close

ssp_err_t R_DTC_Close (transfer_ctrl_t *const p_ctrl)

8.11.11.1 概要説明

ICU で転送を無効化した後、DTC ベクター テーブルで転送データをクリアします。close を実装します。

8.11.11.2 詳細説明

表 665: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	入力パラメータが無効です。
タッチ パネルが設定されていません。	ハンドルが初期化されていません。R_DTC_Open を呼び出してコントロールブロックを初期化します。

8.11.11.3 関数のステップ

- ICU で DTC 有効化ビットをクリアします。
- ベクター テーブルでポインタをクリアします。

8.11.12 R_DTC_VersionGet

```
ssp_err_t R_DTC_VersionGet ( ssp_version_t *const p_version )
```

8.11.12.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンをセットします。versionGet を実装します。

8.11.12.2 詳細説明

表 666: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	入力パラメータが無効です。

8.11.13 拡張

8.11.13.1 dtc_reg_t

dtc_reg_t

詳細説明

転送インタフェースの DTC 拡張機能。DTC レジスタ。transfer_info_t と同じですが、レジスタ名を使用します。型キャスト transfer_info_t へのサービスとして提供されます。

変数

- uint32_t __pad0__
- uint8_t MRB

  モード レジスタ B
- uint8_t __pad0__
- uint8_t DM

  転送宛先アドレス モード。
- uint8_t DTS

  DTC 転送モード選択。
- uint8_t DISEL

  DTC 割り込み選択。
- uint8_t CHNS

  DTC チェーン転送選択。
- uint8_t CHNE

  DTC チェーン転送の有効化。
- struct{} MRB_b

  MRB ビット */

  このメンバーの定義については、ソース コードを参照してください。
- uint8_t MRA

  モード レジスタ A
- uint8_t SM

  転送ソース アドレス モード。
- uint8_t SZ

  DTC データ転送サイズ。
- uint8_t MD

  DTC 転送モード選択。
- struct{} MRA_b

  MRA ビット */

  このメンバーの定義については、ソース コードを参照してください。

- struct{} struct{}

  モード レジスタ */

  このメンバーの定義については、ソース コードを参照してください。
- void *volatile SAR

  ソース アドレス レジスタ。
- void *volatile DAR

  宛先アドレス レジスタ
- uint16_t CRB

  転送カウント レジスタ B
- uint16_t CRA

  転送カウント レジスタ A
- uint8_t CRAL

  転送カウンタ A 下位レジスタ。
- uint8_t CRAH

  転送カウンタ A 上位レジスタ。
- struct{} CRA_b

  ビット */

  このメンバーの定義については、ソース コードを参照してください。
- struct{} struct{}

  転送カウント レジスタ */

  このメンバーの定義については、ソース コードを参照してください。

8.11.14 モジュール

- ビルドタイム構成

8.11.14.1 ビルドタイム構成

定義

- #define DTC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define DTC_CFG_SOFTWARE_START_ENABLE

  初期値 :(0)

  ソフトウェアスタートにコードを含めるかどうかを指定します。有効な設定には :1 が含まれます。ソフトウェアスタートのコードが含まれます。ELC モジュール .0 からの ELC ソフトウェアスタート関数が必要です。ソフトウェアスタートのコードをコンパイルします

8.12 ELC

イベント リンク コントローラ （ELC）のドライバ。

このモジュールは、イベント リンク コントローラ （ELC）をサポートします。次のインタフェースを実装します : ELC インタフェース

8.12.1 Functions

- R_ELC_Init
- R_ELC_SoftwareEventGenerate
- R_ELC_LinkSet
- R_ELC_LinkBreak
- R_ELC_Enable
- R_ELC_Disable
- R_ELC_VersionGet

8.12.2 定義

- #define ELC_CODE_VERSION_MAJOR

  初期値 :(1)

- #define ELC_CODE_VERSION_MINOR

  初期値 :(1)

- #define ELC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.12.3 R_ELC_Init

```
ssp_err_t R_ELC_Init ( elc_cfg_t const *const p_cfg )
```

8.12.3.1 概要説明

イベント リンク コントローラのすべてのリンクを初期化します。

8.12.3.2 詳細説明

次を実装します :init

この関数に渡される構成構造体には、ELC に含まれるすべてのイベント ソースのリンクが格納されており、すべてのリンクが一度に設定されます。個別のリンクを設定するには、R_ELC_LinkSet を使用します

表 667: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
SSP_ERR_INVALID_PTR	p_config が NULL でした

8.12.3.3 関数のステップ

- ELC の電源をオンにします
- イベント リンク コントローラの動作を有効にします

8.12.4 R_ELC_SoftwareEventGenerate

```
ssp_err_t  R_ELC_SoftwareEventGenerate ( elc_software_event_t  event_number )
```

8.12.4.1 概要説明

イベント リンク コントローラでソフトウェア イベントを生成します。

8.12.4.2 詳細説明

次を実装します :softwareEventGenerate

表 668: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。

8.12.5 R_ELC_LinkSet

```
ssp_err_t  R_ELC_LinkSet ( elc_peripheral_t  peripheral , elc_event_t  signal )
```

8.12.5.1 概要説明

1 つのイベント リンクを作成します。

8.12.5.2 詳細説明

次を実装します :linkSet

表 669: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。

8.12.6 R_ELC_LinkBreak

`ssp_err_t` R_ELC_LinkBreak (`elc_peripheral_t` peripheral)

8.12.6.1 概要説明

イベント リンクを切断します。

8.12.6.2 詳細説明

次を実装します :linkBreak

表 670: 戻り値

名前	説明
SSP_SUCCESS	イベントリンクの解除

8.12.7 R_ELC_Enable

`ssp_err_t` R_ELC_Enable (void)

8.12.7.1 概要説明

イベント リンク コントローラの動作を有効にします。

8.12.7.2 詳細説明

次を実装します :enable

表 671: 戻り値

名前	説明
SSP_SUCCESS	ELC が有効です。

8.12.8 R_ELC_Disable

`ssp_err_t` R_ELC_Disable (void)

8.12.8.1 概要説明

イベント リンク コントローラの動作を無効にします。

8.12.8.2 詳細説明

次を実装します :disable

表 672: 戻り値

名前	説明
SSP_SUCCESS	ELC が無効にされました。

8.12.9 R_ELC_VersionGet

`ssp_err_t` R_ELC_VersionGet (`ssp_version_t` *const p_version)

8.12.9.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

8.12.9.2 詳細説明

次を実装します :versionGet

表 673: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
SSP_ERR_INVALID_PTR	p_version が NULL です。

8.13 高性能フラッシュ

高性能フラッシュ メモリ （S7G2 のみ）のドライバ。

このモジュールは、高性能フラッシュ周辺機能のフラッシュ インタフェースをサポートします。

8.13.1 Functions

- R_FLASH_HP_Open
- R_FLASH_HP_Write
- R_FLASH_HP_Read
- R_FLASH_HP_Erase
- R_FLASH_HP_BlankCheck
- R_FLASH_HP_StatusGet
- R_FLASH_HP_AccessWindowSet
- R_FLASH_HP_AccessWindowClear
- R_FLASH_HP_Reset
- R_FLASH_HP_StartUpAreaSelect
- R_FLASH_HP_UpdateFlashClockFreq
- R_FLASH_HP_Close
- R_FLASH_HP_VersionGet

8.13.2 定義

- #define FLASH_HP_CODE_VERSION_MAJOR

  初期値 :(1)
- #define FLASH_HP_CODE_VERSION_MINOR

  初期値 :(1)
- #define PLACE_IN_RAM_SECTION

  初期値 :

8.13.3 R_FLASH_HP_Open

```
ssp_err_t R_FLASH_HP_Open ( flash_ctrl_t *const p_ctrl , flash_cfg_t const *const p_cfg )
```

8.13.3.1 概要説明

フラッシュ周辺機能を初期化します。open を実装します。

8.13.3.2 詳細説明

このオープン関数はフラッシュを初期化します。まず FCU ファームウェアが FCURAM にコピーされ、現在の FCLK 周波数に基づいて FCU クロックが設定されます。また、コード フラッシュ プログラミングが有効な場合は、コード フラッシュ プログラミング用の API コードが RAM にコピーされます。

この関数は、他のフラッシュ API 関数を呼び出す前に一度呼び出す必要があります。ユーザー定義のコールバック関数を指定した場合は、割り込みソースを記述するイベント タイプを使用して、フラッシュ レディ割り込みとエラー割り込みがユーザーのコールバック ルーチンを呼び出すように構成されます。

I：提供された p_cfg->callback フィールドでコールバック関数を提供すると、データ フラッシュが非ブロック （BGO）モードで実行されるようにフラッシュを自動で構成します。

Open() が正常に完了すると、フラッシュが他のフラッシュ コマンドを処理できるようになります。

表 674: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われ、タイマが開始しました。
チャネルは現在動作中です。	フラッシュ周辺機器は以前に開始されたトランザクションでビジーです。
SSP_FLASH_ERR_	失敗 プログラミング / 消去モードの正常な入力に失敗しました。
SSP_ERR_TIMEOUT	タイムアウトし、FCU が準備完了になるのを待っています。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
SSP_ERR_IRQ_BSP_DISABLED	コーラーは BGO をリクエストしていますが、フラッシュ中断が有効にされていません。

8.13.3.3 関数のステップ

- g_flash_hp_version にアクセスできるのは ASSERT マクロのみなので、それにアクセスできないことを示す警告がコンパイラ ツールチェーンから発行されることがあります。以下のコードはこの警告を排除し、データ構造体が最適化により除去されるのを防ぎます。

- パラメータ チェックが無効になっている場合、警告を排除します。
- 初期化されていない場合、または動作の実行中でなく、再初期化する場合は、初期化を許可します
- API が開かれていないことを確認します
- ユーザーが提供した設定値に基づいて、パラメータ構造体を設定します
- FCLK を確認して、タイムアウト値を計算します
- 呼び出し側の情報に基づいて、フラッシュ割り込みコールバックをセットアップします。BSP で両方のフラッシュ割り込みが有効でない場合は、SSP_ERR_IRQ_BSP_DISABLED を返します
- フラッシュ割り込みが無効であることを確認します。フラッシュ割り込みは BGO モードでのみ使用されます
- オープン処理が正常に完了しなかった場合は、ハードウェア ロックを返します
- これにより、API が開きます

8.13.4 R_FLASH_HP_Write

```
ssp_err_t  R_FLASH_HP_Write ( flash_ctrl_t  *const p_ctrl ,  uint32_t const src_address ,
 uint32_t flash_address ,  uint32_t const num_bytes )
```

8.13.4.1 概要説明

指定されたコードまたはデータ フラッシュ メモリ領域に書き込みます。write を実装します。

8.13.4.2 詳細説明

表 675:　戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。BGO が有効な場合、操作が正常に開始されたことを意味します。
チャネルは現在動作中です。	フラッシュ周辺機器は以前に開始されたトランザクションでビジーです。
SSP_FLASH_ERR_FAILURE	操作が失敗しました。宛先アドレスがアクセスウィンドウコントロールの下にある可能性があります。
SSP_ERR_TIMEOUT	タイムアウトし、FCU 操作が完了するのを待っています。

表 675: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_SIZE	提供されたバイト数が、プログラミングサイズの倍数でないか、最大範囲を超えています。
SSP_ERR_INVALID_ADDRESS	無効なアドレスが入力されたか、アドレスがプログラミング境界にありません。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

8.13.4.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- フラッシュ状態を更新し、それぞれのコードまたはデータ フラッシュ P/E モードに移行します

8.13.5 R_FLASH_HP_Read

`ssp_err_t` R_FLASH_HP_Read (`flash_ctrl_t` *const p_ctrl , uint8_t * p_dest_address , uint32_t const flash_address , uint32_t const num_bytes)

8.13.5.1 概要説明

指定されたデータまたはコード フラッシュ メモリアドレスから、要求されたバイト数を読み取ります。read を実装します。

8.13.5.2 詳細説明

I：この関数は、完全なインタフェースを維持する目的でのみ提供されています。フラッシュ メモリを直接読み取ることができます （推奨されています)。

表 676: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。
SSP_ERR_INVALID_ADDRESS	無効なフラッシュアドレスが提供されました
p_ctrl の可能性があります。	p_ctrl または p_dest_address で NULL が提供されました。

8.13.5.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.13.6 R_FLASH_HP_Erase

ssp_err_t R_FLASH_HP_Erase (flash_ctrl_t *const p_ctrl , uint32_t const address , uint32_t const num_blocks)

8.13.6.1 概要説明

指定されたコードまたはデータ フラッシュ ブロックを消去します。block_erase_address により erase を実装します。

8.13.6.2 詳細説明

表 677: 戻り値

名前	説明
SSP_SUCCESS	正常に開きました。
SSP_ERR_INVALID_BLOCKS	無効な数のブロックが指定されました。
SSP_ERR_INVALID_ADDRESS	無効なアドレスが指定されました。
チャネルは現在動作中です。	他のフラッシュ操作が実行中か、API が初期化されていません。
SSP_FLASH_ERR_FAILURE	他のフラッシュ操作が失敗しました。フラッシュコントロールユニットがリセットされました。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

8.13.6.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- フラッシュ状態を更新し、それぞれのコードまたはデータ フラッシュ P/E モードに移行します

8.13.7 R_FLASH_HP_BlankCheck

ssp_err_t R_FLASH_HP_BlankCheck (flash_ctrl_t *const p_ctrl , uint32_t const address , uint32_t num_bytes , flash_result_t * p_blank_check_result)

8.13.7.1 概要説明

指定されたアドレス領域でブランク チェックを実行します。blankCheck を実装します。

8.13.7.2 詳細説明

表 678: 戻り値

名前	説明
SSP_SUCCESS	バランクチェック操作が完了して結果が p_blank_check_result にあるか、ブランクチェックが開始され実行中です（BGO モード）。
SSP_FLASH_ERR_FAILURE	何らかの理由で操作が失敗しました。
SSP_ERR_INVALID_ADDRESS	無効なデータ フラッシュ アドレスが入力されました。
SSP_ERR_INVALID_SIZE	'num_bytes' が大きすぎるか、CF/DF 境界サイズに調整されていません。
チャネルは現在動作中です。	他のフラッシュ操作が実行中か、API が初期化されていません。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

8.13.7.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- コード フラッシュのブランク チェックでは、FCU 動作は必要ありません。指定されたアドレス領域で、単純に 0xFF 以外をチェックできるからです。
- ブランクでないことが確認できるまで、ブランクであると仮定します

8.13.8 R_FLASH_HP_StatusGet

```
ssp_err_t  R_FLASH_HP_StatusGet ( flash_ctrl_t  *const p_ctrl )
```

8.13.8.1 概要説明

フラッシュ周辺機器に対し、ステータスを照会します。statusGet を実装します。

8.13.8.2 詳細説明

表 679: 戻り値

名前	説明
SSP_SUCCESS	フラッシュ周辺機器が使用可能です。
チャネルは現在動作中です。	フラッシュ周辺機器は以前の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

8.13.8.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.13.9 R_FLASH_HP_AccessWindowSet

ssp_err_t R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const p_ctrl , uint32_t const start_addr , uint32_t const end_addr)

8.13.9.1 概要説明

指定されたスタートアドレスとエンドアドレスを使用して、コード フラッシュ メモリのアクセス ウィンドウを構成します。アクセス ウィンドウは、コード フラッシュ内でプログラミング / 消去が有効な連続した領域を定義します。この領域はブロック境界上にあります。start_addr が格納されているブロックが先頭のブロックです。end_addr が格納されているブロックが最後のブロックです。これにより、アクセスウィンドウは、最初のブロック > 最後のブロック（包含的）となります。この範囲以外のコード フラッシュは、書き込み保護されます。

8.13.9.2 詳細説明

I：スタートアドレスとエンドアドレスを同じ値に設定した場合、アクセス ウィンドウが事実上削除されます。これにより、R_FLASH_HP_AccessWindowClear を実行した場合と同じ機能が得られます。

accessWindowSet を実装します。

表 680: 戻り値

名前	説明
SSP_SUCCESS	アクセスウィンドウが正常に構成されました。
SSP_ERR_INVALID_ADDRESS	start_addr および end_addr の無効な設定です。
チャネルは現在動作中です。	フラッシュ周辺機器は以前の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
SSP_ERR_INVALID_POINTER	コードフラッシュプログラミングが有効になっていません。

8.13.9.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- コード フラッシュ コードがコンパイルされていない場合に作成される警告を削除します。
- < _LP API と一貫性を保つために、コードフラッシュが有効にされていない場合はエラーを返します

8.13.10 R_FLASH_HP_AccessWindowClear

```
ssp_err_t  R_FLASH_HP_AccessWindowClear ( flash_ctrl_t  *const p_ctrl )
```

8.13.10.1 概要説明

コード フラッシュで現在構成されているすべてのアクセス ウィンドウを削除します。この呼び出しの後、すべてのコードフラッシュが書き込み可能となります。accessWindowClear を実装します。

8.13.10.2 詳細説明

表 681: 戻り値

名前	説明
SSP_SUCCESS	アクセスウィンドウが正常に削除されました。
チャネルは現在動作中です。	フラッシュ周辺機器は以前の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

表 681: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_POINTER	コードフラッシュプログラミングが有効になっていません。

8.13.10.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- < _LP API と一貫性を保つために、コードフラッシュが有効にされていない場合はエラーを返します

8.13.11 R_FLASH_HP_Reset

```
ssp_err_t R_FLASH_HP_Reset ( flash_ctrl_t *const p_ctrl )
```

8.13.11.1 概要説明

フラッシュ周辺機能をリセットします。reset を実装します。想定では、リセットによって既存の動作がすべて停止するため、リセットの実行前にフラッシュ ソフトウェア ロックの取得が試みられることはありません。

8.13.11.2 詳細説明

表 682: 戻り値

名前	説明
SSP_SUCCESS	フラッシュ回路が正常にリセットされました。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

8.13.11.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.13.12 R_FLASH_HP_StartUpAreaSelect

```
ssp_err_t R_FLASH_HP_StartUpAreaSelect ( flash_ctrl_t *const p_ctrl ,
 flash_startup_area_swap_t swap_type , bool is_temporary )
```

8.13.12.1 概要説明

Default （ブロック 0）と Alternate （ブロック 1）のどちらのブロックをスタートアップ領域ブロックとして使用するかを選択します。提供されているパラメータは、どのブロックがアクティブ スタートアップ ブロックになり、そのアクションが次のリセットの後にすぐに （しかし一時的に）実行されるのか、恒常的に実行されるのかを決定します。一時的な切り替えのメリットは限定的のようです。ブロック 0 が書き込み保護となるようなアクセス ウィンドウがある場合、アクセス ウィンドウに触らずに、一時的な切り替えを行い、ブロックを更新して、元に切り替えることもできます。startupAreaSelect を実装します。

8.13.12.2 詳細説明

表 683: 戻り値

名前	説明
SSP_SUCCESS	スタートアップエリアが正常に切り替えられました。
SSP_ERR_INVALID_ARGUMENT	フラッシュ周辺機器がすでに初期化され、使用されています。
チャネルは現在動作中です。	フラッシュ周辺機器は以前の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

8.13.12.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- これはすでに標準のリセット動作です。操作は不要です
- フラッシュ状態を更新し、構成セットを呼び出すかどうかに応じて、それぞれのコードまたはデータ フラッシュ P/E モードに移行します

8.13.13 R_FLASH_HP_UpdateFlashClockFreq

```
ssp_err_t  R_FLASH_HP_UpdateFlashClockFreq ( flash_ctrl_t  *const p_ctrl )
```

8.13.13.1 概要説明

すでに開いているフラッシュ API に対し、Open() 実行後に FCLK が変更されたことを示します。たとえば、アプリケーションがシステム クロックを変更したために、FCLK も変更された場合などです。FCLK の変更後にこの関数を呼び出さなかった場合、フラッシュ マクロが損傷する可能性があります。この関数では、R_CGC_SystemClockFreqGet を使用して FCLK の周波数を取得します。updateFlashClockFreq を実装します。

8.13.13.2 詳細説明

表 684: 戻り値

名前	説明
SSP_SUCCESS	スタートアップエリアが正常に切り替えられました。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.13.13.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- 現在のフラッシュ状態を取得します
- API はすでにこの状態にあります
- < FCLK を確認して、タイムアウト値を計算します

8.13.14 R_FLASH_HP_Close

```
ssp_err_t  R_FLASH_HP_Close ( flash_ctrl_t  *const p_ctrl )
```

8.13.14.1 概要説明

Open() またはその後のフラッシュ動作で割り当てられたすべてのリソースを解放します。close を実装します。

8.13.14.2 詳細説明

表 685: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。

8.13.14.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- ロックを解除します。

8.13.15 R_FLASH_HP_VersionGet

```
ssp_err_t  R_FLASH_HP_VersionGet ( ssp_version_t  *const p_version )
```

8.13.15.1 概要説明

この関数は、フラッシュ HAL ドライバ バージョンを取得します。

8.13.15.2 詳細説明

表 686:　戻り値

名前	説明
SSP_SUCCESS	- 操作が正常に実行されました

I：この関数は再入可能です。

8.14 ローパワー フラッシュ

ローパワー フラッシュ メモリ （S3A7 および S124）のドライバ。

このモジュールは、ローパワー フラッシュ周辺機能のフラッシュ インタフェースをサポートします。

8.14.1 Functions

- R_FLASH_LP_Open
- R_FLASH_LP_Write
- R_FLASH_LP_Read
- R_FLASH_LP_Erase
- R_FLASH_LP_BlankCheck
- R_FLASH_LP_StatusGet
- R_FLASH_LP_AccessWindowSet
- R_FLASH_LP_AccessWindowClear
- R_FLASH_LP_Reset
- R_FLASH_LP_StartUpAreaSelect
- R_FLASH_LP_UpdateFlashClockFreq
- R_FLASH_LP_Close
- R_FLASH_LP_VersionGet

8.14.2 定義

- #define FLASH_LP_CODE_VERSION_MAJOR

  初期値 :(1)
- #define FLASH_LP_CODE_VERSION_MINOR

  初期値 :(1)
- #define PLACE_IN_RAM_SECTION

  初期値 :
- #define FLASH_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)
- #define FLASH_CFG_PARAM_CODE_FLASH_PROGRAMMING_ENABLE

  初期値 :(0)

8.14.3 R_FLASH_LP_Open

```
ssp_err_t  R_FLASH_LP_Open ( flash_ctrl_t  *const p_ctrl , flash_cfg_t  const *const p_cfg )
```

8.14.3.1 概要説明

ローパワー フラッシュ周辺機能を初期化します。open を実装します。

8.14.3.2 詳細説明

このオープン関数はフラッシュを初期化します。まず FCU ファームウェアが FCURAM にコピーされ、現在の FCLK 周波数に基づいて FCU クロックが設定されます。また、コード フラッシュ プログラミングが有効な場合は、コード フラッシュ プログラミング用の API コードが RAM にコピーされます。

この関数は、他のフラッシュ API 関数を呼び出す前に一度呼び出す必要があります。ユーザー定義のコールバック関数を指定した場合は、データ フラッシュ動作の割り込みソースを記述するイベント タイプを使用して、フラッシュ レディ割り込みがユーザーのコールバック ルーチンを呼び出すように構成されます。この呼び出しが正常に完了すると、g_flash_api_open が true になります。

I：提供された p_cfg->callback フィールドでコールバック関数を提供すると、データ フラッシュが非ブロック （BGO）モードで実行されるようにフラッシュを自動で構成します。

Open() が正常に完了すると、フラッシュが他のフラッシュ コマンドを処理できるようになります。

表 687: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われ、タイマが開始しました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。
SSP_ERR_INVALID_POINTER	flash_setup() から CGC 呼び出しに無効な引数が提供されました。発生しないはずです ...
SSP_ERR_INVALID_ARGUMENT	低出力フラッシュ周辺機器のハードウェアロックがすでに取られています。
SSP_ERR_IRQ_BSP_DISABLED	コーラーは BGO をリクエストしていますが、フラッシュ中断が有効にされていません。

8.14.3.3 関数のステップ

- g_flash_lp_version にアクセスできるのは ASSERT マクロのみなので、それにアクセスできないことを示す警告がコンパイラ ツールチェーンから発行されることがあります。以下のコードはこの警告を排除し、データ構造体が最適化により除去されるのを防ぎます。
- パラメータ チェックが無効になっている場合、警告を排除します。
- 初期化されていない場合、または動作の実行中でなく、再初期化する場合は、初期化を許可します
- API が開かれていないことを確認します
- ユーザーが提供した設定値に基づいて、パラメータ構造体を設定します
- 呼び出し側の情報に基づいて、フラッシュ割り込みコールバックをセットアップします。BSP でフラッシュ割り込みが有効でない場合は、SSP_ERR_IRQ_BSP_DISABLED を返します
- フラッシュ割り込みが無効であることを確認します。フラッシュ割り込みは BGO モードでのみ使用されます
- FCLK を確認して、タイムアウト値を計算します
- 他の動作を実行できるように、状態を解放します。
- オープン処理が正常に完了しなかった場合は、ハードウェア ロックを返します
- フラッシュのハードウェア ロックを返します
- これにより、API が開きます

8.14.4 R_FLASH_LP_Write

```
ssp_err_t  R_FLASH_LP_Write ( flash_ctrl_t  *const p_ctrl ,  uint32_t const src_address ,
 uint32_t flash_address ,  uint32_t const num_bytes )
```

8.14.4.1 概要説明

指定されたコードまたはデータ フラッシュ メモリ領域に書き込みます。write を実装します。

8.14.4.2 詳細説明

表 688:　戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。BGO が有効な場合、操作が正常に開始されたことを意味します。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。

表 688: 戻り値 (続き)

名前	説明
SSP_FLASH_ERR_FAILURE	操作が失敗しました。宛先アドレスがアクセスウィンドウコントロールの下にある可能性があります。
SSP_ERR_TIMEOUT	タイムアウトし、操作が完了するのを待っています。
SSP_ERR_INVALID_SIZE	提供されたバイト数が、プログラミングサイズの倍数でないか、最大範囲を超えています。
SSP_ERR_INVALID_ADDRESS	無効なアドレスが入力されたか、アドレスがプログラミング境界にありません。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.4.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- このアドレスに関するブロック情報を取得します
- フラッシュ状態を更新し、それぞれのコードまたはデータ フラッシュ P/E モードに移行します
- < MF3 DF では常に 1 です
- データを書き込みます
- BGO モードでない場合、現在の動作はこれで完了です。PE モードを終了して、ステータスを返します
- エラーが発生した場合、フラッシュ シーケンサをリセットします。これにより、エラーフラグがクリアされ、P/E モードが終了します

8.14.5 R_FLASH_LP_Read

`ssp_err_t` R_FLASH_LP_Read (`flash_ctrl_t` *const p_ctrl , uint8_t * p_dest_address , uint32_t const flash_address , uint32_t const num_bytes)

8.14.5.1 概要説明

指定されたデータまたはコード フラッシュ メモリアドレスから、要求されたバイト数を読み取ります。read を実装します。

8.14.5.2 詳細説明

I：この関数は、完全なインタフェースを維持する目的でのみ提供されています。フラッシュ メモリを直接読み取ることができます （推奨されています)。

表 689: 戻り値

名前	説明
SSP_SUCCESS	操作が正常に行われました。
SSP_ERR_INVALID_ADDRESS	無効なフラッシュアドレスが提供されました
p_ctrl の可能性があります。	p_ctrl または p_dest_address で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.5.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.14.6 R_FLASH_LP_Erase

`ssp_err_t` R_FLASH_LP_Erase (`flash_ctrl_t` *const p_ctrl , uint32_t const address , uint32_t const num_blocks)

8.14.6.1 概要説明

指定されたコードまたはデータ フラッシュ ブロックを消去します。erase を実装します。

8.14.6.2 詳細説明

表 690: 戻り値

名前	説明
SSP_SUCCESS	正常に開きました。

表 690: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_BLOCKS	無効な数のブロックが指定されました。
SSP_ERR_INVALID_ADDRESS	無効なアドレスが指定されました。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
SSP_ERR_WRITE_FAILED	消去が失敗しました。宛先アドレスがアクセスウィンドウコントロールの下にあるか、フラッシュがロックされている（FSPR ビット）可能性があります。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.6.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- このアドレスに関するブロック情報を取得します
- フラッシュ状態を更新し、それぞれのコードまたはデータ フラッシュ P/E モードに移行します
- 続行するかどうかを確認します
- これがコード フラッシュの消去要求かどうかを確認します
- これはデータ フラッシュの消去要求です
- ブロックを消去し、このブロックに関する詳細をこの関数に提供します
- BGO モードでない場合、現在の動作はこれで完了です。PE モードを終了して、ステータスを返します
- エラーが発生した場合、フラッシュ シーケンサをリセットします。これにより、エラーフラグがクリアされ、P/E モードが終了します

8.14.7 R_FLASH_LP_BlankCheck

```
ssp_err_t  R_FLASH_LP_BlankCheck ( flash_ctrl_t  *const p_ctrl ,  uint32_t const address ,
 uint32_t num_bytes , flash_result_t  * p_blank_check_result )
```

8.14.7.1 概要説明

指定されたアドレス領域でブランク チェックを実行します。blankCheck を実装します。データ フラッシュのブランク チェックを行うバイト数は、1 から FLASH_DATA_BLANK_CHECK_MAX の間で設定します。

コード フラッシュのブランク チェックを行うバイト数は、1 から FLASH_CODE_BLANK_CHECK_MAX の間で設定します。それぞれをフラッシュします。

8.14.7.2 詳細説明

表 691: 戻り値

名前	説明
SSP_SUCCESS	バランクチェック操作が完了して結果が p_blank_check_result にあるか、ブランクチェックが開始され実行中です（BGO モード）。
SSP_ERR_INVALID_ADDRESS	無効なデータフラッシュアドレスが入力されました
SSP_ERR_INVALID_SIZE	'num_bytes' が大きすぎるか、CF/DF 境界サイズに調整されていません。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.7.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- このアドレスに関するブロック情報を取得します
- フラッシュ状態を更新し、それぞれのコードまたはデータ フラッシュ P/E モードに移行します
- 続行するかどうかを確認します
- これがコード フラッシュのブランク チェック要求かどうかを確認します
- BGO モードでない場合、現在の動作はこれで完了です。PE モードを終了して、ステータスを返します
- エラーが発生した場合、FCU をリセットします。これにより、エラーフラグがクリアされて、P/E モードが終了します

8.14.8 R_FLASH_LP_StatusGet

```
ssp_err_t R_FLASH_LP_StatusGet ( flash_ctrl_t *const p_ctrl )
```

8.14.8.1 概要説明

フラッシュに対して、そのステータスを照会します。statusGet を実装します。

8.14.8.2 詳細説明

表 692: 戻り値

名前	説明
SSP_SUCCESS	フラッシュの準備ができ、コマンドを受け取ることができます。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.8.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- フラッシュのステータスを返します

8.14.9 R_FLASH_LP_AccessWindowSet

```
ssp_err_t  R_FLASH_LP_AccessWindowSet ( flash_ctrl_t  *const p_ctrl ,  uint32_t const start_addr ,
 uint32_t const end_addr )
```

8.14.9.1 概要説明

指定されたスタートアドレスとエンドアドレスを使用して、コード フラッシュ メモリのアクセス ウィンドウを構成します。アクセス ウィンドウは、コード フラッシュ内でプログラミング / 消去が有効な連続した領域を定義します。この領域はブロック境界上にあります。start_addr が格納されているブロックが先頭のブロックです。end_addr が格納されているブロックが最後のブロックです。これにより、アクセスウィンドウは、最初のブロック > 最後のブロック（包含的）となります。この範囲以外のコード フラッシュは、書き込み保護されます。

8.14.9.2 詳細説明

I：スタートアドレスとエンドアドレスを同じ値に設定した場合、アクセス ウィンドウが事実上削除されます。これにより、R_FLASH_LP_AccessWindowClear を実行した場合と同じ機能が得られます。

accessWindowSet を実装します。

表 693: 戻り値

名前	説明
SSP_SUCCESS	アクセスウィンドウが正常に構成されました。
SSP_ERR_INVALID_ADDRESS	start_addr および end_addr の無効な設定です。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
SSP_ERR_WRITE_FAILED	操作が失敗しました。フラッシュはロックされている可能性があります（FSPR ビット）。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.9.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- end_addr は、そのアドレス未満を示すため、そのアドレスが格納されたブロックは含まれません。したがって、
- 最後のブロックをアクセス ウィンドウに含めるには、FLASH_CF_BLOCK_END+1 に設定する必要があります
- フラッシュ状態を更新し、コード フラッシュ P/E モードに移行します
- リードモードに戻ります

8.14.10 R_FLASH_LP_AccessWindowClear

```
ssp_err_t  R_FLASH_LP_AccessWindowClear ( flash_ctrl_t  *const p_ctrl )
```

8.14.10.1 概要説明

コード フラッシュで現在構成されているすべてのアクセス ウィンドウを削除します。この呼び出しの後、すべてのコードフラッシュが書き込み可能となります。accessWindowClear を実装します。

8.14.10.2 詳細説明

表 694: 戻り値

名前	説明
SSP_SUCCESS	アクセスウィンドウが正常に削除されました。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
SSP_ERR_WRITE_FAILED	操作が失敗しました。フラッシュはロックされている可能性があります（FSPR ビット）。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.10.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- フラッシュ状態を更新し、コード フラッシュ P/E モードに移行します
- 続行するかどうかを確認します
- リードモードに戻ります
- エラーが発生した場合、フラッシュをリセットします。これにより、エラーフラグがクリアされ、P/E モードが終了します

8.14.11 R_FLASH_LP_Reset

```
ssp_err_t  R_FLASH_LP_Reset ( flash_ctrl_t  *const p_ctrl )
```

8.14.11.1 概要説明

フラッシュ周辺機能をリセットします。reset を実装します。想定では、リセットによって既存の動作がすべて停止するため、リセットの実行前にフラッシュ ソフトウェア ロックの取得が試みられることはありません。

8.14.11.2 詳細説明

表 695: 戻り値

名前	説明
SSP_SUCCESS	フラッシュ回路が正常にリセットされました。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.11.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。

8.14.12 R_FLASH_LP_StartUpAreaSelect

```
ssp_err_t R_FLASH_LP_StartUpAreaSelect ( flash_ctrl_t *const p_ctrl ,
 flash_startup_area_swap_t swap_type , bool is_temporary )
```

8.14.12.1 概要説明

Default （ブロック 0）と Alternate （ブロック 1）のどちらのブロックをスタートアップ領域ブロックとして使用するかを選択します。提供されているパラメータは、どのブロックがアクティブ スタートアップ ブロックになり、そのアクションが次のリセットの後にすぐに （しかし一時的に）実行されるのか、恒常的に実行されるのかを決定します。一時的な切り替えのメリットは限定的のようです。ブロック 0 が書き込み保護となるようなアクセス ウィンドウがある場合、アクセス ウィンドウに触らずに、一時的な切り替えを行い、ブロックを更新して、元に切り替えることもできます。startupAreaSelect を実装します。

8.14.12.2 詳細説明

表 696: 戻り値

名前	説明
SSP_SUCCESS	スタートアップエリアが正常に切り替えられました。
SSP_ERR_INVALID_ARGUMENT	フラッシュ周辺機器がすでに初期化され、使用されています。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。

表 696: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.12.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- これはすでに標準のリセット動作です。操作は不要です
- フラッシュ状態を更新し、コード フラッシュ P/E モードに移行します
- 正常に完了したことを確認します
- リードモードに戻ります
- エラーが発生した場合、フラッシュをリセットします。これにより、エラーフラグがクリアされ、P/E モードが終了します

8.14.13 R_FLASH_LP_UpdateFlashClockFreq

```
ssp_err_t  R_FLASH_LP_UpdateFlashClockFreq ( flash_ctrl_t  *const p_ctrl )
```

8.14.13.1 概要説明

すでに開いているフラッシュ API に対し、Open() 実行後に FCLK が変更されたことを示します。たとえば、アプリケーションがシステム クロックを変更したために、FCLK も変更された場合などです。FCLK の変更後にこの関数を呼び出さなかった場合、フラッシュ マクロが損傷する可能性があります。この関数では、R_CGC_SystemClockFreqGet を使用して FCLK の周波数を取得します。r_flash_t::updateFlashClockFreq を実装します。

8.14.13.2 詳細説明

表 697: 戻り値

名前	説明
SSP_SUCCESS	スタートアップエリアが正常に切り替えられました。
チャネルは現在動作中です。	フラッシュは実行中の操作でビジーです。
p_ctrl の可能性があります。	p_ctrl で NULL が提供されました。

表 697: 戻り値 (続き)

名前	説明
タッチ パネルが設定されていません。	フラッシュ API がまだ開かれていません。

8.14.13.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- 現在のフラッシュ状態を取得します
- API はすでにこの状態にあります
- < FCLK を確認して、タイムアウト値を計算します

8.14.14 R_FLASH_LP_Close

`ssp_err_t` R_FLASH_LP_Close (`flash_ctrl_t` *const p_ctrl)

8.14.14.1 概要説明

Open() またはその後のフラッシュ動作で割り当てられたすべてのリソースを解放します。close を実装します。

8.14.14.2 詳細説明

表 698: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl または p_cfg で NULL が提供されました。

8.14.14.3 関数のステップ

- パラメータ チェックが無効になっている場合、警告を排除します。
- フラッシュのハードウェア ロックを返します
- これにより、API が閉じられます
- ロックを解除します。

8.14.15 R_FLASH_LP_VersionGet

ssp_err_t R_FLASH_LP_VersionGet (ssp_version_t *const p_version)

8.14.15.1 概要説明

フラッシュ HAL ドライバのバージョンを取得します。

8.14.15.2 詳細説明

表 699: 戻り値

名前	説明
SSP_SUCCESS	- 操作が正常に実行されました

I：この関数は再入可能です。

8.15 FMI

ファクトリ MCU 情報 （FMI）にアクセスするためのドライバ。

ファクトリ MCU 情報のフラッシュ メモリを読み取ります。

8.15.1 Functions

- R_FMI_ProductInfoGet
- R_FMI_VersionGet

8.15.2 定義

- #define INLINE_ATTRIBUTE

  初期値 :
- #define FMI_CODE_VERSION_MAJOR

  初期値 :(1)
- #define FMI_CODE_VERSION_MINOR

  初期値 :(0)

8.15.3 R_FMI_ProductInfoGet

```
ssp_err_t R_FMI_ProductInfoGet ( fmi_product_info_t ** pp_product_info )
```

8.15.3.1 概要説明

ファクトリ MCU 情報の先頭のレコードへのポインタを取得します。

8.15.3.2 詳細説明

表 700: 戻り値

名前	説明
SSP_SUCCESS	呼び出し側のポインタを製品情報レコードに設定します。
p_ctrl の可能性があります。	呼び出し側のポインタが NULL です。

8.15.4 R_FMI_VersionGet

```
ssp_err_t  R_FMI_VersionGet ( ssp_version_t  *const p_version )
```

8.15.4.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

8.15.4.2 詳細説明

表 701:　戻り値

名前	説明
SSP_SUCCESS	呼び出し側の構造体を書き込みました。
p_ctrl の可能性があります。	呼び出し側のポインタが NULL です。

8.16 GLCDC

グラフィック LCD コントローラ （GLCDC）のドライバ。

8.16.1 概要

ディスプレイインタフェース を実装します。このモジュールは、グラフィック LCD コントローラ（GLCDC）をサポートします。表示インタフェースを実装し、GLCDC ピンに接続された LCD パネルを制御します。

8.16.2 Functions

- R_GLCD_Open
- R_GLCD_Close
- R_GLCD_Start
- R_GLCD_Stop
- R_GLCD_LayerChange
- R_GLCD_ColorCorrection
- R_GLCD_ClutUpdate
- R_GLCD_StatusGet
- R_GLCD_VersionGet

8.16.3 R_GLCD_Open

```
ssp_err_t R_GLCD_Open ( display_ctrl_t *const p_ctrl , display_cfg_t const *const p_cfg )
```

8.16.3.1 詳細説明

表 702: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。

表 702: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_ARGUMENT	GLCDC リソースがロックされています。
SSP_ERR_CLOCK_GENERATION	ドットクロックはクロックソースから生成できません。
SSP_ERR_INVALID_TIMING_SETTING	無効なパネルタイミングパラメータ。
SSP_ERR_INVALID_LAYER_SETTING	無効なレイヤー設定が見つかりました。
SSP_ERR_INVALID_LAYER_FORMAT	無効なフォーマットが指定されました。
SSP_ERR_INVALID_GAMMA_SETTING	無効なガンマ修正設定が見つかりました。

I：この API を呼び出す前に、グラフィック LCD コントローラ （GLCDC）で PCLKA を指定し、GLCDC ピンを IOPORT に設定する必要があります。

8.16.3.2 関数のステップ

- GLCD リソースをロックします。
- GLCD モジュールで周辺クロックを指定します
- GLCD を SW リセット ステータスから解放します。
- ドット クロック周波数を設定します
- パネル信号タイミングを設定します
- 背景画面を構成します
- バック ポーチ位置を制御ブロックに格納します （後でレイヤー ブレンディング位置を定義するために必要です）
- グラフィックス プレーン レイヤーを構成します
- 出力制御ブロックを構成します
- 色補正設定 （明るさとガンマ補正）を構成します
- GLCD ドライバの状態を変更します
- コールバック関数を保存します
- ユーザー定義のコンテキストを保存します

- 表示インタフェース コンテキストを GLCD HAL 制御ブロックに保存します
- ライン検出割り込みを実行するライン番号を設定します
- GLCD 割り込みを有効にします

8.16.4 R_GLCD_Close

```
ssp_err_t  R_GLCD_Close ( display_ctrl_t  *const p_ctrl )
```

8.16.4.1 詳細説明

表 703:　戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に閉じられました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
タッチ パネルが設定されていません。	ドライバ状態が DISPLAY_STATE_CLOSED と等しくないときに、関数呼び出しが実行されます
SSP_ERR_INVALID_UPDATE_TIMING	GLCD が内部でレジスタ値を更新しているときに関数呼び出しが実行されます。

I：この API は、ドライバが DISPLAY_STATE_CLOSED 状態以外のときに呼び出すことができます。背景画面生成ブロックのレジスタ更新動作が保留されている場合、エラーが返されます。

8.16.4.2 関数のステップ

- GLCD 割り込みを無効化します
- GLCD ハードウェアをリセットします
- GLCD モジュールへの周辺クロックを停止します
- GLCD リソースをロック解除します

8.16.5 R_GLCD_Start

ssp_err_t R_GLCD_Start (display_ctrl_t *const p_ctrl)

8.16.5.1 詳細説明

表 704: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開始されました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
未サポートまたは不正確なモードです。	ドライバ状態が DISPLAY_STATE_OPENED でないときに、関数呼び出しが実行されます。

I：この API は、ドライバが DISPLAY_STATE_OPENED ステータス以外のときに呼び出すことができます。

8.16.5.2 関数のステップ

- 垂直 / 水平同期信号および画面データの出力を開始します。

8.16.6 R_GLCD_Stop

ssp_err_t R_GLCD_Stop (display_ctrl_t *const p_ctrl)

8.16.6.1 詳細説明

表 705: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に停止されました
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です

表 705: 戻り値 (続き)

名前	説明
未サポートまたは不正確なモードです。	ドライバ状態が DISPLAY_STATE_DISPLAYING でないときに、関数呼び出しが実行されます。
SSP_ERR_INVALID_UPDATE_TIMING	GLCD が内部でレジスタ値を更新している間に関数呼び出しが実行されます。

I：この API は、ドライバが DISPLAY_STATE_DISPLAYING 状態のときに呼び出すことができます。背景画面生成ブロック、グラフィックス データ I/F ブロック、または出力制御ブロックのレジスタ更新動作が保留されている場合、エラーが返されます。

8.16.6.2 関数のステップ

- 垂直 / 水平同期信号および画面データの出力を停止します。

8.16.7 R_GLCD_LayerChange

```
ssp_err_t R_GLCD_LayerChange ( display_ctrl_t const *const p_ctrl , display_runtime_cfg_t const *const p_cfg , display_frame_layer_t frame )
```

8.16.7.1 詳細説明

表 706: 戻り値

名前	説明
未サポートまたは不正確なモードです。	ドライバ状態が DISPLAY_STATE_DISPLAYING でないときに、関数呼び出しが実行されます。
SSP_ERR_INVALID_POINTER	引数に無効なパラメータが見つかりました。
SSP_ERR_INVALID_UPDATE_TIMING	GLCD が内部でレジスタ値を更新している間に関数呼び出しが実行されます。

I：この API は、ドライバが DISPLAY_STATE_DISPLAYING 状態のときに呼び出すことができます。背景画面生成ブロックまたはグラフィックス データ I/F ブロックのレジスタ更新動作が保留されてい

る場合、エラーが返されます。

8.16.7.2 関数のステップ

- グラフィックス プレーン レイヤーを構成します
- 次の Vsync アサーション時に、グラフィックス モジュール レジスタ値を GLCD 内部動作に反映します

8.16.8 R_GLCD_ColorCorrection

```
ssp_err_t  R_GLCD_ColorCorrection ( display_ctrl_t  const *const p_ctrl ,
 display_correction_t  const *const p_correction )
```

8.16.8.1 詳細説明

表 707: 戻り値

名前	説明
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
未サポートまたは不正確なモードです。	ドライバ状態が DISPLAY_STATE_DISPLAYING DISPLAY_STATE_DISPLAYING でないときに、関数呼び出しが実行されます。
SSP_ERR_INVALID_UPDATE_TIMING	GLCDC が内部でレジスタを更新している間に関数呼び出しが実行されます。

I：この API は、ドライバが DISPLAY_STATE_DISPLAYING 状態のときに呼び出すことができます。背景画面生成ブロックまたは出力制御ブロックのレジスタ更新動作が保留されている場合、エラーが返されます。

8.16.8.2 関数のステップ

- 明るさおよびコントラスト補正レジスタ設定を構成します。

- 出力ブロック レジスタ設定を更新します。

8.16.9 R_GLCD_ClutUpdate

`ssp_err_t` R_GLCD_ClutUpdate (`display_ctrl_t` const *const p_ctrl , `display_clut_cfg_t` const *const p_clut_cfg , `display_frame_layer_t` frame)

8.16.9.1 詳細説明

表 708: 戻り値

名前	説明
p_ctrl の可能性があります。	コントロールブロックまたは CLUT ソースデータが NULL です。
SSP_ERR_INVALID_CLUT_ACCESS	不正な CLUT エントリまたはサイズが指定されました。

I：この API はいつでも呼び出せます。

8.16.9.2 関数のステップ

- 現在使用されている CLUT テーブルを確認します
- ソース メモリにある新しい CLUT データを GLCD モジュール内の CLUT SRAM にコピーします
- GLCD モジュールによって、次フレームから新しい CLUT テーブル データを読み取ります

8.16.10 R_GLCD_StatusGet

`ssp_err_t` R_GLCD_StatusGet (`display_ctrl_t` const *const p_ctrl , `display_status_t` *const p_status)

8.16.10.1 詳細説明

表 709: 戻り値

名前	説明
SSP_SUCCESS	ステータスが正常に取得されました。
SSP_ASSERT	Null ポインタが提供されました。

I :GLCD ハードウェアは、LayerChange() 呼び出し後、最初の Vsync でフェード処理を開始します。このハードウェアの動作のため、この API が LayerChange() 呼び出し後、最初の Vsync の前に呼び出された場合、フェード ステータスとして DISPLAY_FADE_STATUS_FADING_UNDERWAY が返されない可能性があります。この場合、API は DISPLAY_FADE_STATUS_NOT_UNDERWAY ではなく、DISPLAY_FADE_STATUS_UNCERTAIN を返します。

8.16.10.2 関数のステップ

- GLCD HAL ドライバの状態を返します
- レイヤーのフェード ステータスを返します

8.16.11 R_GLCD_VersionGet

```
ssp_err_t R_GLCD_VersionGet ( ssp_version_t * p_version )
```

8.16.11.1 詳細説明

表 710: 戻り値

名前	説明
p_version。	バージョン番号。

I：この関数は再入可能です。

8.16.12 API データ

8.16.12.1 glcd_clk_src_t

glcd_clk_src_t

詳細説明

クロック ソース選択

列挙値

名前	説明
GLCD_CLK_SRC_INTERNAL	内部。
GLCD_CLK_SRC_EXTERNAL	外部。

8.16.12.2 glcd_panel_clk_div_t

glcd_panel_clk_div_t

詳細説明

クロック周波数分周率

列挙値

名前	説明
GLCD_PANEL_CLK_DIVISOR_1	分周率は、1/1。
GLCD_PANEL_CLK_DIVISOR_2	分周率は、1/2。
GLCD_PANEL_CLK_DIVISOR_3	分周率は、1/3。
GLCD_PANEL_CLK_DIVISOR_4	分周率は、1/4。
GLCD_PANEL_CLK_DIVISOR_5	分周率は、1/5。
GLCD_PANEL_CLK_DIVISOR_6	分周率は、1/6。
GLCD_PANEL_CLK_DIVISOR_7	分周率は、1/7。
GLCD_PANEL_CLK_DIVISOR_8	分周率は、1/8。

名前	説明
GLCD_PANEL_CLK_DIVISOR_9	分周率は、1/9。
GLCD_PANEL_CLK_DIVISOR_12	分周率は、1/12。
GLCD_PANEL_CLK_DIVISOR_16	分周率は、1/16。
GLCD_PANEL_CLK_DIVISOR_24	分周率は、1/24。
GLCD_PANEL_CLK_DIVISOR_32	分周率は、1/32。

8.16.12.3 glcd_tcon_pin_t

glcd_tcon_pin_t

詳細説明

LCD TCON 出力ピン選択

列挙値

名前	説明
GLCD_TCON_PIN_NONE	出力なし。
GLCD_TCON_PIN_0	LCD_TCON0。
GLCD_TCON_PIN_1	LCD_TCON1。
GLCD_TCON_PIN_2	LCD_TCON2。
GLCD_TCON_PIN_3	LCD_TCON3。
GLCD_TCON_PIN_NUM	

8.16.12.4 glcd_bus_arbitration_t

glcd_bus_arbitration_t

詳細説明

バス アービトレーション設定

列挙値

名前	説明
GLCD_BUS_ARBITRATION_ROUNDROBIN	ラウンド ロビン。
GLCD_BUS_ARBITRATION_FIX_PRIORITY	固定。

8.16.12.5 glcd_correction_proc_order_t

glcd_correction_proc_order_t

詳細説明

訂正回路シーケンス制御

列挙値

名前	説明
GLCD_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA	明るさ -> コントラスト -> ガンマ修正。
GLCD_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST	ガンマ修正 -> 明るさ -> コントラスト。

8.16.12.6 glcd_tcon_signal_select_t

glcd_tcon_signal_select_t

詳細説明

LCD パネルを操作するためのタイミング信号

列挙値

名前	説明
GLCD_TCON_SIGNAL_SELECT_STVA_VS	STVA/VS。
GLCD_TCON_SIGNAL_SELECT_STVB_VE	STVB/VE。
GLCD_TCON_SIGNAL_SELECT_STHA_HS	STH/SP/HS。
GLCD_TCON_SIGNAL_SELECT_STHB_HE	STB/LP/HE。

名前	説明
GLCD_TCON_SIGNAL_SELECT_DE	DE。

8.16.12.7 glcd_clut_plane_t

glcd_clut_plane_t

詳細説明

シリアル RGB 出力のクロック位相調整

列挙値

名前	説明
GLCD_CLUT_PLANE_0	GLCD CLUT プレーン 0。
GLCD_CLUT_PLANE_1	GLCD CLUT プレーン 1。

8.16.12.8 glcd_dithering_mode_t

glcd_dithering_mode_t

詳細説明

ディザリング モード

列挙値

名前	説明
GLCD_DITHERING_MODE_TRUNCATE	ディザリングなし （切り捨て）
GLCD_DITHERING_MODE_ROUND_OFF	四捨五入のディザリング。
GLCD_DITHERING_MODE_2X2PATTERN	2x2 パターンのディザリング。
GLCD_DITHERING_MODE_SETTING_MAX	

8.16.12.9 glcd_dithering_pattern_t

glcd_dithering_pattern_t

詳細説明

ディザリング モード

列挙値

名前	説明
GLCD_DITHERING_PATTERN_00	2x2 パターン '00'
GLCD_DITHERING_PATTERN_01	2x2 パターン '01'
GLCD_DITHERING_PATTERN_10	2x2 パターン '10'
GLCD_DITHERING_PATTERN_11	2x2 パターン '11'

8.16.12.10 glcd_input_interface_format_t

glcd_input_interface_format_t

詳細説明

出力インタフェース フォーマット

列挙値

名前	説明
GLCD_INPUT_INTERFACE_FORMAT_RGB565	入力インタフェース フォーマットは、RGB565。
GLCD_INPUT_INTERFACE_FORMAT_RGB888	入力インタフェース フォーマットは、RGB888。
GLCD_INPUT_INTERFACE_FORMAT_ARGB1555	入力インタフェース フォーマットは、ARGB1555。
GLCD_INPUT_INTERFACE_FORMAT_ARGB4444	入力インタフェース フォーマットは、ARGB4444。
GLCD_INPUT_INTERFACE_FORMAT_ARGB8888	入力インタフェース フォーマットは、ARGB8888。
GLCD_INPUT_INTERFACE_FORMAT_CLUT8	入力インタフェース フォーマットは、CLUT8。
GLCD_INPUT_INTERFACE_FORMAT_CLUT4	入力インタフェース フォーマットは、CLUT4。
GLCD_INPUT_INTERFACE_FORMAT_CLUT1	入力インタフェース フォーマットは、CLUT1。

8.16.12.11 glcd_output_interface_format_t

glcd_output_interface_format_t

詳細説明

出力インタフェース フォーマット

列挙値

名前	説明
GLCD_OUTPUT_INTERFACE_FORMAT_RGB888	出力インタフェース フォーマットは、RGB888。
GLCD_OUTPUT_INTERFACE_FORMAT_RGB666	出力インタフェース フォーマットは、RGB666。
GLCD_OUTPUT_INTERFACE_FORMAT_RGB565	出力インタフェース フォーマットは、RGB565。
GLCD_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB	出力インタフェース フォーマットは、シリアル RGB。

8.16.12.12 glcd_dithering_output_format_t

glcd_dithering_output_format_t

詳細説明

ディザリング出力フォーマット

列挙値

名前	説明
GLCD_DITHERING_OUTPUT_FORMAT_RGB888	ディザリング出力フォーマットは、RGB888。
GLCD_DITHERING_OUTPUT_FORMAT_RGB666	ディザリング出力フォーマットは、RGB666。
GLCD_DITHERING_OUTPUT_FORMAT_RGB565	ディザリング出力フォーマットは、RGB565。

8.16.13 拡張

8.16.13.1 glcd_cfg_t

glcd_cfg_t

詳細説明

GLCD のハードウェア固有の設定

変数

- glcd_tcon_pin_t tcon_hsync

  GLCD TCON 出力ピン選択。
- glcd_tcon_pin_t tcon_vsync

  GLCD TCON 出力ピン選択。
- glcd_tcon_pin_t tcon_de

  GLCD TCON 出力ピン選択。
- glcd_correction_proc_order_t correction_proc_order

  補正制御ルート選択。
- glcd_clk_src_t clksrc

  クロック ソース選択。
- glcd_panel_clk_div_t clock_div_ratio

  ドット クロックのクロック分割比。
- glcd_dithering_mode_t dithering_mode

  ディザリング モード。
- glcd_dithering_pattern_t dithering_pattern_A

  ディザリング パターン A。
- glcd_dithering_pattern_t dithering_pattern_B

  ディザリング パターン B。
- glcd_dithering_pattern_t dithering_pattern_C

  ディザリング パターン C。
- glcd_dithering_pattern_t dithering_pattern_D

  ディザリング パターン D。

8.16.13.2 glcd_ctrl_t

glcd_ctrl_t

詳細説明

GLCD のハードウェア固有の制御ブロック

変数

- display_coordinate_t back_porch

  グラフィックス プレーンのゼロ座標 （バック ポーチ終点）

- uint16_t hsize

  1 ラインの水平ピクセル サイズ。

- uint16_t vsize

  1 フレームの垂直ピクセル サイズ。

- bsp_lock_t resource_lock

  リソース ロック。

- void * p_context

  display_ctrl_t 型のデータへのポインタ

8.16.14 モジュール

- GLCDC ビルドタイム構成

8.16.14.1 GLCDC ビルドタイム構成

定義

- #define GLCD_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックのコードを含めるかどうかを指定します この構成が 0 に設定された場合はパラメータチェックが除外されます。コードサイズの削減または性能の向上のために、パラメータチェックを無効にすることができます。BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.17 GPT

汎用 PWM タイマ （GPT）のドライバ。

8.17.1 概要

タイマインタフェース を拡張します。

このモジュールは、汎用 PWM タイマ （GPT）周辺機器 GPT32EH、GPT32E、GPT32 を使って タイマインタフェース を実装します。また、タイマ信号を GTIOC ピンに出力する出力比較拡張機能を提供します。

8.17.2 Functions

- R_GPT_TimerOpen
- R_GPT_Stop
- R_GPT_Start
- R_GPT_CounterGet
- R_GPT_Reset
- R_GPT_PeriodSet
- R_GPT_DutyCycleSet
- R_GPT_InfoGet
- R_GPT_Close
- R_GPT_VersionGet

8.17.3 定義

- #define GPT_CODE_VERSION_MAJOR

  初期値 :(1)
- #define GPT_CODE_VERSION_MINOR

  初期値 :(1)

8.17.4 R_GPT_TimerOpen

```
ssp_err_t R_GPT_TimerOpen ( timer_ctrl_t *const p_ctrl , timer_cfg_t const *const p_cfg )
```

8.17.4.1 概要説明

GPT を電源オンにして、ユーザーズマニュアルに説明されている必要な初期化を行います。open を実装します。

8.17.4.2 詳細説明

Open 関数は単一の GPT チャネルを構成し、チャネルを開始し、GPT API コントロールおよび Close 関数で使用するためにハンドルを提供します。Open 関数は単一の GPT チャネルを構成し、チャネルを開始し、GPT API コントロールおよび Close 関数で使用するためにハンドルを提供します。チャネルが開かれたら、関連する Close 関数を呼び出す前に同じチャネルに対して Open 関数を繰り返し呼び出さないようにする必要があります。

チャネルが開かれたら、関連する Close 関数を呼び出す前に同じチャネルに対して Open 関数を繰り返し呼び出さないようにする必要があります。GPT ハードウェアは、ワンショット機能をネイティブではサポートしません。

汎用タイマの GPT 実装は、timer_on_gpt_cfg_t 拡張パラメータを受け入れることができます。

表 711: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われ、タイマが開始しました。
p_ctrl の可能性があります。	次のいずれかのパラメータが不正確です。p_cfg が NULL か、または • p_ctrl が NULL か、または • p_cfg パラメータでリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できません。 • SSP_ERR_INVALID_ARGUMENT

表 711: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_POINTER	p_cfg->period: • p_cfg->period: 次の範囲内にする必要があります。 – 下限 : (1 / (PCLK 周波数) – 上限 : (0xFFFFFFFF * 1024 / (PCLK 周波数)) • p_cfg->p_callback は NULL ではありませんが、ISR が有効化されていません。コールバック関数を使用するには ISR を有効にする必要があります。bsp_irq_cfg.h でチャネルのオーバーフロー ISR を有効にします。 • p_cfg->mode は TIMER_MODE_ONE_SHOT ですが、ISR が有効化されていません。ワンショットモードを使用するには ISR を有効にする必要があります。bsp_irq_cfg.h でチャネルのオーバーフロー ISR を有効にします。
チャネルは現在動作中です。	指定したチャネルはすでに開かれています。構成は変更されていません。関連する Close 関数を呼び出すか、関連する Control コマンドを使用してチャネルを再構成します。

I : この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.17.4.3 関数のステップ

- 期間を計算して間隔変数を保存します
- ハードウェア レジスタを設定する前に GPT を電源オンにします。モード レジスタ、PCLK 除数レジスタ、およびカウンタ レジスタを設定する前に、カウンタが停止されていることを確認にします。
- デューティ サイクルを計算します

8.17.5 R_GPT_Stop

`ssp_err_t` R_GPT_Stop (`timer_ctrl_t` *const p_ctrl)

8.17.5.1 概要説明

タイマを停止します。stop を実装します。

8.17.5.2 詳細説明

表 712: 戻り値

名前	説明
SSP_SUCCESS	タイマが正常に停止されました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.5.3 関数のステップ

- タイマを停止します。

8.17.6 R_GPT_Start

`ssp_err_t` R_GPT_Start (`timer_ctrl_t` *const p_ctrl)

8.17.6.1 概要説明

タイマを開始します。start を実装します。

8.17.6.2 詳細説明

表 713: 戻り値

名前	説明
SSP_SUCCESS	タイマが正常に開始されました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

表 713: 戻り値 (続き)

名前	説明
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.6.3 関数のステップ

- タイマを開始します。

8.17.7 R_GPT_CounterGet

`ssp_err_t` R_GPT_CounterGet (`timer_ctrl_t` *const p_ctrl , `timer_size_t` *const p_value)

8.17.7.1 概要説明

提供された p_value ポインタでカウンタ値を設定します。counterGet を実装します。

8.17.7.2 詳細説明

表 714: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が読み取られ、p_value は有効です。
AMSSP_ERR_ASSERTION	p_ctrl または p_value パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.7.3 関数のステップ

- カウンタ値を読み取ります。

8.17.8 R_GPT_Reset

`ssp_err_t` R_GPT_Reset (`timer_ctrl_t` *const p_ctrl)

8.17.8.1 概要説明

カウンタ値を 0 にリセットします。reset を実装します。

8.17.8.2 詳細説明

表 715: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.8.3 関数のステップ

- カウンタ値を書き込みます

8.17.9 R_GPT_PeriodSet

`ssp_err_t` R_GPT_PeriodSet (`timer_ctrl_t` *const p_ctrl , `timer_size_t` const period , `timer_unit_t` const unit)

8.17.9.1 概要説明

指定された期間値をセットします。periodSet を実装します。

8.17.9.2 詳細説明

表 716: 戻り値

名前	説明
SSP_SUCCESS	周期値が正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

表 716: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_POINTER	次のいずれかが無効です。 • p_period->unit: timer_unit_t のいずれかのオプションにする必要があります • p_period->value: 次の範囲の期間になる必要があります。 – 下限 : (1 / (PCLK 周波数)) – 上限 : (0xFFFFFFFF * 1024 / (PCLK 周波数))
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.9.3 関数のステップ

- 期間が有効なことを確認してから、期間をセットします。

8.17.10 R_GPT_DutyCycleSet

```
ssp_err_t R_GPT_DutyCycleSet ( timer_ctrl_t *const p_ctrl , timer_size_t const duty_cycle ,
 timer_pwm_unit_t const unit , uint8_t const pin )
```

8.17.10.1 概要説明

指定された p_status ポインタ内のステータスをセットします。pwm_api_t::dutyCycleSet を実装します。

8.17.10.2 詳細説明

表 717: 戻り値

名前	説明
SSP_SUCCESS	カウンタ値が正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.10.3 関数のステップ

- レジスタで設定する前にデューティ サイクルを PCLK カウントに変換しました
- デューティ サイクルを設定します。

8.17.11 R_GPT_InfoGet

`ssp_err_t` R_GPT_InfoGet (`timer_ctrl_t` *const p_ctrl , `timer_info_t` *const p_info)

8.17.11.1 概要説明

タイマ情報を取得し、指定されたポインタ p_info に格納します。infoGet を実装します。

8.17.11.2 詳細説明

表 718: 戻り値

名前	説明
SSP_SUCCESS	期間、カウント方向、周波数、ステータスの値を呼び出し側の構造体に正常に書き込みました。
p_ctrl の可能性があります。	p_ctrl または p_info パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.11.3 関数のステップ

- 期間を取得して保存します
- クロック周波数を取得して保存します
- クロックのカウント方向を取得して保存します

8.17.12 R_GPT_Close

`ssp_err_t` R_GPT_Close (`timer_ctrl_t` *const p_ctrl)

8.17.12.1 概要説明

カウンタを停止して、割り込みを無効にし、出力ピンを無効にして、内部ドライバ データをクリアします。

8.17.12.2 詳細説明

表 719: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.17.12.3 関数のステップ

- クリーンアップ。割り込みを無効にし、カウンタを停止し、出力を無効にします。
- チャネルをロック解除します。
- 保存されている内部ドライバ データをクリアします。

8.17.13 R_GPT_VersionGet

```
ssp_err_t R_GPT_VersionGet ( ssp_version_t *const p_version )
```

8.17.13.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

8.17.13.2 詳細説明

表 720: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.17.14 API データ

8.17.14.1 gpt_pin_level_t

gpt_pin_level_t

詳細説明

GPT ピンのレベル

列挙値

名前	説明
GPT_PIN_LEVEL_LOW	ピンレベルが低くなっています。
GPT_PIN_LEVEL_HIGH	ピンレベルが高くなっています。
GPT_PIN_LEVEL_RETAINED	ピンレベルが保持されました。

8.17.14.2 gpt_trigger_t

gpt_trigger_t

詳細説明

ソースは、タイマの開始、タイマの停止、カウントアップ、またはカウントダウンに使用できます。

列挙値

名前	説明
GPT_TRIGGER_NONE	アクションが実行されませんでした。
GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_LOW	GTIOCB が低い間に GTIOCA 入力が上昇するとアクションが実行されます。
GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_HIGH	GTIOCB が高い間に GTIOCA 入力が上昇するとアクションが実行されます。
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_LOW	GTIOCB が低い間に GTIOCA 入力が低下するとアクションが実行されます。
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_HIGH	GTIOCB が高い間に GTIOCA 入力が低下するとアクションが実行されます。

名前	説明
GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_LOW	GTIOCA が低い間に GTIOCB 入力が上昇するとアクションが実行されます。
GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_HIGH	GTIOCA が高い間に GTIOCB 入力が上昇するとアクションが実行されます。
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_LOW	GTIOCA が低い間に GTIOCB 入力が低下するとアクションが実行されます。
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_HIGH	GTIOCA が高い間に GTIOCB 入力が低下するとアクションが実行されます。
GPT_TRIGGER_SOURCE_REGISTER_ENABLE	ソース選択レジスタで設定を有効にします。

8.17.14.3 gpt_output_t

gpt_output_t

詳細説明

比較一致またはサイクル終了で生じる事柄を選択する際に使用する出力レベルです。

列挙値

名前	説明
GPT_OUTPUT_RETAINED	出力が保持されました。
GPT_OUTPUT_LOW	出力が低くなっています。
GPT_OUTPUT_HIGH	出力が高くなっています。
GPT_OUTPUT_TOGGLED	出力が切り替わりました。

8.17.15 拡張

8.17.15.1 gpt_output_pin_t

gpt_output_pin_t

詳細説明

出力ピンの設定。

変数

- bool output_enabled

  出力を有効にするには true、無効にするには false に設定します。

- gpt_pin_level_t stop_level

  gpt_pin_level_t から停止レベルを選択します。

8.17.15.2 timer_on_gpt_cfg_t

timer_on_gpt_cfg_t

詳細説明

GPT 拡張は、GPT 向けに出力ピンを設定します。

変数

- gpt_output_pin_t gtioca

  GPT I/O ピン A の設定。

- gpt_output_pin_t gtiocb

  GPT I/O ピン B の設定。

8.17.16 モジュール

- ビルドタイム構成

8.17.16.1 ビルドタイム構成

定義

- #define GPT_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.18 GPT 入力キャプチャ

入力キャプチャ機能付き汎用 PWM タイマ （GPT）用ドライバ。

8.18.1 概要

入力キャプチャインタフェース を拡張します。

このモジュールは、汎用 PWM タイマ （GPT）周辺機器 GPT32EH、GPT32E、GPT32 に 入力キャプチャインタフェース を実装します。

8.18.2 Functions

- R_GPT_InputCaptureOpen
- R_GPT_InputCaptureClose
- R_GPT_InputCaptureVersionGet
- R_GPT_InputCaptureDisable
- R_GPT_InputCaptureEnable
- R_GPT_InputCaptureInfoGet
- R_GPT_InputCaptureLastCaptureGet

8.18.3 定義

- #define GPT_INPUT_CAPTURE_CODE_VERSION_MAJOR

  初期値 :(1)

  次が含まれます
- #define GPT_INPUT_CAPTURE_CODE_VERSION_MINOR

  初期値 :(1)
- #define GPT_INPUT_CAPTURE_MAX_COUNT

  初期値 :(0xFFFFFFFFUL)

  GPT カウンタの最大値。

8.18.4 R_GPT_InputCaptureOpen

```
ssp_err_t  R_GPT_InputCaptureOpen ( input_capture_ctrl_t   *const p_ctrl ,
 input_capture_cfg_t   const *const p_cfg )
```

8.18.4.1 概要説明

入力キャプチャのために GPT タイマを開きます。open を実装します。

8.18.4.2 詳細説明

Open 関数は入力キャプチャのために単一の GPT チャネルを構成し、他の入力キャプチャ API 関数で使用するためにハンドルを提供します。この関数は他のどの入力キャプチャ API 関数よりも前に、1 度呼び出す必要があります。チャネルが開かれたら、関連する Close 関数を呼び出す前に同じチャネルに対して Open 関数を繰り返し呼び出さないようにする必要があります。

表 721: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_cfg, p_ctrl, p_extend です。または、p_cfg パラメータでリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できない可能性があります。または p_cfg->mode が無効です。
SSP_ERR_INVALID_POINTER	次のいずれかが無効か範囲外です。 • p_extend->signal • p_extend->signal_filter • p_extend->clock_divider • p_extend->enable_level • p_extend->enable_filter • p_cfg->p_callback は NULL ではありませんが、ISR が有効化されていません。コールバック関数を使用するには ISR を有効にする必要があります。bsp_irq_cfg.h でチャネルのオーバーフロー ISR を有効にします。
チャネルは現在動作中です。	指定したチャネルはすでに開かれています。構成は変更されていません。関連する Close 関数を呼び出すか、関連する Control コマンドを使用してチャネルを再構成します。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.18.4.3 関数のステップ

- irq が使用可能であることを確認します
- まだ使用されていないチャネルを確認します。

8.18.5 R_GPT_InputCaptureClose

`ssp_err_t` R_GPT_InputCaptureClose (`input_capture_ctrl_t` *const p_ctrl)

8.18.5.1 概要説明

入力キャプチャのために GPT タイマ チャネルを閉じます。close を実装します。

8.18.5.2 詳細説明

タイマ設定をクリアし、割り込みを無効にし、間隔ドライバデータをクリアします。

表 722: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.18.5.3 関数のステップ

- クリーンアップ。割り込みを無効にし、測定を停止します。
- チャネルをロック解除します。
- 保存されている内部ドライバ データをクリアします。

8.18.6 R_GPT_InputCaptureVersionGet

`ssp_err_t` R_GPT_InputCaptureVersionGet (`ssp_version_t` *const p_version)

8.18.6.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。versionGet を実装します。

8.18.6.2 詳細説明

表 723: 戻り値

名前	説明
SSP_SUCCESS	成功。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.18.7 R_GPT_InputCaptureDisable

```
ssp_err_t  R_GPT_InputCaptureDisable ( input_capture_ctrl_t  const *const p_ctrl )
```

8.18.7.1 概要説明

NVIC で指定されたチャネルの GPT 入力キャプチャ RegA 割り込みを無効にします。disable を実装します。

8.18.7.2 詳細説明

表 724: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。
SSP_ERR_INVALID_POINTER	ISR が有効ではありません。bsp_irq_cfg.h でこの GPT チャネルのオーバーフロー ISR を有効にします。

8.18.7.3 関数のステップ

- 割り込みを無効にします

8.18.8 R_GPT_InputCaptureEnable

```
ssp_err_t  R_GPT_InputCaptureEnable ( input_capture_ctrl_t  const *const p_ctrl )
```

8.18.8.1 概要説明

NVIC で指定されたチャネルの GPT 入力キャプチャ RegA 割り込みを有効にします。enable を実装します。

8.18.8.2 詳細説明

表 725: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に有効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。
SSP_ERR_INVALID_POINTER	ISR が有効ではありません。bsp_irq_cfg.h でこの GPT チャネルのオーバーフロー ISR を有効にします。

8.18.8.3 関数のステップ

- 割り込みを有効にします

8.18.9 R_GPT_InputCaptureInfoGet

```
ssp_err_t  R_GPT_InputCaptureInfoGet ( input_capture_ctrl_t  const *const p_ctrl ,
 input_capture_info_t  *const p_info )
```

8.18.9.1 概要説明

提供された p_status ポインタにステータスを取得します。infoGet を実装します。

8.18.9.2 詳細説明

表 726: 戻り値

名前	説明
SSP_SUCCESS	成功。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.18.10 R_GPT_InputCaptureLastCaptureGet

```
ssp_err_t R_GPT_InputCaptureLastCaptureGet ( input_capture_ctrl_t const *const p_ctrl ,
 input_capture_capture_t *const p_capture )
```

8.18.10.1 概要説明

提供された p_counter ポインタでカウンタ値を設定します。lastCaptureGet を実装します。

8.18.10.2 詳細説明

表 727: 戻り値

名前	説明
SSP_SUCCESS	周期値が正常に書き込まれました。
AMSSP_ERR_ASSERTION	p_ctrl または p_value パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.18.10.3 関数のステップ

- キャプチャの値を設定します

8.18.11 API データ

8.18.11.1 gpt_input_capture_signal_t

gpt_input_capture_signal_t

詳細説明

入力キャプチャ信号選択

列挙値

名前	説明
GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCA	GTIOCxA ピン、x はチャネル番号です。
GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCB	GTIOCxB ピン、x はチャネル番号です。

8.18.11.2 gpt_input_capture_signal_filter_t

gpt_input_capture_signal_filter_t

詳細説明

入力キャプチャ信号ノイズフィルタ （デバウンス）設定。入力信号 GTIOCxA および GTIOCxB でのみ使用可能。ノイズフィルタは、PCLK をいずれかの値で割った間隔で外部信号をサンプリングします。3 つ連続したサンプルが同じレベル （高または低）の場合、そのレベルは信号の観測値として受け渡されます。ユーザーズマニュアルの「GPT」セクションにある「ノイズ フィルタ関数」を参照してください。

列挙値

名前	説明
GPT_INPUT_CAPTURE_SIGNAL_FILTER_NONE	なし - フィルタリングが行われません。
GPT_INPUT_CAPTURE_SIGNAL_FILTER_1	PCLK/1 - 速いサンプリング。
GPT_INPUT_CAPTURE_SIGNAL_FILTER_4	PCLK/4。
GPT_INPUT_CAPTURE_SIGNAL_FILTER_16	PCLK/16。
GPT_INPUT_CAPTURE_SIGNAL_FILTER_64	PCLK/64 - 遅いサンプリング。

8.18.11.3 gpt_input_capture_clock_divider_t

gpt_input_capture_clock_divider_t

詳細説明

入力キャプチャ PCLK 分周器。タイマカウンタのスケールに使用されます。

列挙値

名前	説明
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1	/ 1
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_4	/ 4
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_16	/ 16
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_64	/ 64
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_256	/ 256
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1024	/ 1024

8.18.12 拡張

8.18.12.1 gpt_input_capture_extend_t

gpt_input_capture_extend_t

概要説明

TU 入力キャプチャ用の拡張設定構造体。

詳細説明

詳細説明 p_extend

変数

- gpt_input_capture_signal_t signal

  gpt_input_capture_signal_t のいずれかです。

- gpt_input_capture_signal_filter_t signal_filter

  gpt_input_capture_signal_filter_t のいずれかです。

- gpt_input_capture_clock_divider_t clock_divider

  gpt_input_capture_clock_divider_t のいずれかです。

- input_capture_signal_level_t enable_level

  未使用の GTIOCx ピンは、キャプチャを有効化するためのイネーブル信号として用いることができます。入力キャプチャ信号ピンが GTIOCA の場合、イネーブル ピンは GTIOCB です。使用されている場合、有効レベルはここで設定されます。

- gpt_input_capture_signal_filter_t enable_filter

  gpt_input_capture_signal_filter_t のいずれかです。

8.18.13 モジュール

- ビルドタイム構成

8.18.13.1 ビルドタイム構成

定義

- #define GPT_INPUT_CAPTURE_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.19 ICU

割り込みコントローラ ユニット （ICU）の外部ピン割り込み機能用ドライバ。

8.19.1 概要

外部 IRQ インタフェース を拡張します。

このモジュールは、割り込みコントローラ ユニット （ICU）の外部入力ピンを使用して 外部 IRQ インタフェース を実装します。

8.19.2 Functions

- R_ICU_ExternalIrqOpen
- R_ICU_ExternalIrqEnable
- R_ICU_ExternalIrqDisable
- R_ICU_ExternalIrqTriggerSet
- R_ICU_ExternalIrqFilterEnable
- R_ICU_ExternalIrqFilterDisable
- R_ICU_ExternalIrqVersionGet
- R_ICU_ExternalIrqClose

8.19.3 定義

- #define ICU_CODE_VERSION_MAJOR

  初期値 :(1)
- #define ICU_CODE_VERSION_MINOR

  初期値 :(1)

8.19.4 R_ICU_ExternalIrqOpen

`ssp_err_t` R_ICU_ExternalIrqOpen (`external_irq_ctrl_t` *const p_ctrl , `external_irq_cfg_t` const *const p_cfg)

8.19.4.1 概要説明

ボタン インタフェースで使用する外部入力ピンを構成します。open を実装します。

8.19.4.2 詳細説明

Open 関数は、操作用に外部入力ピンを準備する役割を担っています。この関数は他のどの外部入力ピン API 関数よりも前に、1 度呼び出す必要があります。正常に完了すると、選択した外部入力ピンのステータスは「open」に設定されます。その後、次の関数を呼び出して、「close」を実行しない限り、同じ外部入力ピンを再度呼び出してはなりません。その後、R_ICU_ExternalIrqClose を呼び出して「close」を実行しない限り、同じ外部入力ピンでこの関数を再度呼び出してはなりません。

表 728: 戻り値

名前	説明
SSP_SUCCESS	コマンドが正常に完了しました。
p_ctrl の可能性があります。	次のいずれかが無効です。 • p_ctrl or p_cfg が NULL です • p_cfg でリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できません。
SSP_ERR_INVALID_POINTER	p_cfg->p_callback は NULL ではありませんが、ISR が有効化されていません。コールバック関数を使用するには ISR を有効にする必要があります。bsp_irq_cfg.h でチャネルのオーバーフロー ISR を有効にします。
チャネルは現在動作中です。	指定したチャネルはすでに開かれています。構成は変更されていません。関連する Close 関数を呼び出してチャネルを再構成します。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.19.4.3 関数のステップ

- まだ使用されていないチャネルを確認します。
- ISR で使用できるように制御ブロックを保存します
- IRQ 番号をルックアップします
- 割り込みを有効にします
- 制御ブロックを初期化します。

8.19.5 R_ICU_ExternalIrqEnable

`ssp_err_t` R_ICU_ExternalIrqEnable (`external_irq_ctrl_t` *const p_ctrl)

8.19.5.1 概要説明

NVIC で指定されたチャネルで外部割り込みを有効にします。enable を実装します。

8.19.5.2 詳細説明

表 729: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.19.6 R_ICU_ExternalIrqDisable

`ssp_err_t` R_ICU_ExternalIrqDisable (`external_irq_ctrl_t` *const p_ctrl)

8.19.6.1 概要説明

NVIC で指定されたチャネルで外部割り込みを無効にします。disable を実装します。

8.19.6.2 詳細説明

表 730: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.19.7 R_ICU_ExternalIrqTriggerSet

```
ssp_err_t  R_ICU_ExternalIrqTriggerSet ( external_irq_ctrl_t  *const p_ctrl ,
 external_irq_trigger_t  hw_trigger )
```

8.19.7.1 概要説明

指定されたトリガー値をセットします。triggerSet を実装します。

8.19.7.2 詳細説明

表 731: 戻り値

名前	説明
SSP_SUCCESS	周期値が正常に書き込まれました。
p_ctrl の可能性があります。	p_ctrl または p_period パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.19.8 R_ICU_ExternalIrqFilterEnable

```
ssp_err_t  R_ICU_ExternalIrqFilterEnable ( external_irq_ctrl_t  *const p_ctrl )
```

8.19.8.1 概要説明

指定されたチャネルで外部割り込みデジタル フィルタを有効にします。filterEnable を実装します。

8.19.8.2 詳細説明

表 732: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.19.9 R_ICU_ExternalIrqFilterDisable

ssp_err_t R_ICU_ExternalIrqFilterDisable (external_irq_ctrl_t *const p_ctrl)

8.19.9.1 概要説明

指定されたチャネルで外部割り込みデジタル フィルタを有効にします。filterDisable を実装します。

8.19.9.2 詳細説明

表 733: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.19.10 R_ICU_ExternalIrqVersionGet

ssp_err_t R_ICU_ExternalIrqVersionGet (ssp_version_t *const p_version)

8.19.10.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンをセットします。versionGet を実装します。

8.19.10.2 詳細説明

表 734: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.19.11 R_ICU_ExternalIrqClose

ssp_err_t R_ICU_ExternalIrqClose (external_irq_ctrl_t *const p_ctrl)

8.19.11.1 概要説明

外部割り込みを無効にします。close を実装します。

8.19.11.2 詳細説明

表 735: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.19.11.3 関数のステップ

- クリーンアップ。割り込みを無効にします
- 保存されている内部ドライバ データをクリアします。
- BSP ハードウェア ロックを解放します

8.19.12 モジュール

- ビルドタイム構成

8.19.12.1 ビルドタイム構成

定義

- #define ICU_BUTTON_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

8.20 IOPORT

I/O ポート用ドライバ。

IOPort HAL ドライバは、ビットおよびポートの両方のレベルでデバイスの I/O ポートにアクセスするための機能を提供します。ポートとピンの方向は変更できます。さらに、個々のピンの機能を変更するために数々の構成 API が提供されています。

8.20.1 Functions

- R_IOPORT_Init
- R_IOPORT_PinCfg
- R_IOPORT_PinRead
- R_IOPORT_PortRead
- R_IOPORT_PortWrite
- R_IOPORT_PinWrite
- R_IOPORT_PortDirectionSet
- R_IOPORT_PinDirectionSet
- R_IOPORT_PortEventInputRead
- R_IOPORT_PinEventInputRead
- R_IOPORT_PortEventOutputWrite
- R_IOPORT_PinEventOutputWrite
- R_IOPORT_VersionGet
- R_IOPORT_EthernetModeCfg

8.20.2 定義

- #define IOPORT_CODE_VERSION_MAJOR

  初期値 :(1)
- #define IOPORT_CODE_VERSION_MINOR

  初期値 :(1)

8.20.3 R_IOPORT_Init

```
ssp_err_t R_IOPORT_Init ( ioport_cfg_t const * p_cfg )
```

8.20.3.1 概要説明

ピン PFS レジスタに構成データを読み込んで、複数ピンの関数を構成します。init を実装します。

8.20.3.2 詳細説明

この関数は、値を指定して提供された PmnPFS レジスタのリストを初期化します。このデータは、ISDE ピン コンフィギュレータで生成するか、開発者が手動で生成できます。異なるピン構成を、ローパワー モードやテストなど、異なる状況のために読み込むことができます。*

表 736: 戻り値

名前	説明
SSP_SUCCESS	PFS レジスタに書き込まれたピン構成データ
p_ctrl の可能性があります。	Null ポインタ

8.20.3.3 関数のステップ

- g_ioport_version にアクセスできるのは ASSERT マクロのみなので、それにアクセスできないことを示す警告がコンパイラ ツールチェーンから発行されることがあります。以下のコードはこの警告を抑制するとともに、これらのデータ構造体が最適化によって除去されることを防ぎます。

8.20.4 R_IOPORT_PinCfg

```
ssp_err_t  R_IOPORT_PinCfg (  ioport_port_pin_t pin ,  uint32_t cfg )
```

8.20.4.1 概要説明

ピンの設定を構成します。pinCfg を実装します。

8.20.4.2 詳細説明

表 737: 戻り値

名前	説明
SSP_SUCCESS	ピンが構成されました。
SSP_ERR_INVALID_POINTER	無効なピン

I：この関数は異なるピンで再入可能です。この関数は、ピンの構成を新しい構成に変更します。たとえば、この関数では、ピンのドライブ強度を変更しながら、他のすべてのピン設定を無変更のままにすることはできません。これを行うには、必要な変更を加えた元の設定を、この関数で書き込む必要があります。

8.20.5 R_IOPORT_PinRead

`ssp_err_t` R_IOPORT_PinRead (ioport_port_pin_t pin , `ioport_level_t` * p_pin_value)

8.20.5.1 概要説明

ピンのレベルを読み取ります。pinRead を実装します。

8.20.5.2 詳細説明

表 738: 戻り値

名前	説明
SSP_SUCCESS	ピンが読み取られました。
SSP_ERR_INVALID_POINTER	無効な引数
p_ctrl の可能性があります。	Null ポインタ

I：この関数は異なるピンで再入可能です。

8.20.6 R_IOPORT_PortRead

`ssp_err_t` R_IOPORT_PortRead (ioport_port_t port , ioport_size_t * p_port_value)

8.20.6.1 概要説明

IO ポートの値を読み取ります。portRead を実装します。

8.20.6.2 詳細説明

指定したポートが読み取られ、すべてのピンのレベルが返されます。指定したポートが読み取られ、すべてのピンのレベルが返されます。値パラメータの各ビットは、ポートのビットに対応します。*

表 739: 戻り値

名前	説明
SSP_SUCCESS	ポートが読み取られました。
SSP_ERR_INVALID_POINTER	ポートが無効です。
p_ctrl の可能性があります。	Null ポインタ

I：この関数は異なるポートで再入可能です。

8.20.7 R_IOPORT_PortWrite

```
ssp_err_t R_IOPORT_PortWrite ( ioport_port_t port , ioport_size_t value , ioport_size_t mask )
```

8.20.7.1 概要説明

次を実装します :portWrite を実装します。

8.20.7.2 詳細説明

値パラメータの各ビットは、ポートのビットに対応します。入力値は、指定されたポートのために構成された ELC イベントが発生すると、そのポートに書き込まれます。値パラメータの各ビットは、ポートのビットに対応します。マスク パラメータの各ビットは、ポートのピンに対応します。

たとえば、値 = 0xFFFF、マスク = 0x0003 の場合、ビット 0 と 1 のみが更新されます。

表 740: 戻り値

名前	説明
SSP_SUCCESS	書き込み先のポート。
SSP_ERR_INVALID_POINTER	ポートおよび / またはマスクが無効です。

I：この関数は異なるポートで再入可能です。この関数は PCNTR3 レジスタを使用して、ポートで指定したピンのレベルを自動で変更します。

8.20.7.3 関数のステップ

- 高ビット
- 低ビット

8.20.8 R_IOPORT_PinWrite

`ssp_err_t` R_IOPORT_PinWrite (ioport_port_pin_t pin , `ioport_level_t` level)

8.20.8.1 概要説明

ピンの出力を高または低に設定します。pinWrite を実装します。

8.20.8.2 詳細説明

表 741: 戻り値

名前	説明
SSP_SUCCESS	書き込み先のピン。
SSP_ERR_INVALID_POINTER	ピンおよび / またはレベルが無効です。

I：この関数は異なるピンで再入可能です。この関数は PCNTR3 レジスタを使用して、ポートで指定したピンのレベルを自動で変更します。

8.20.9 R_IOPORT_PortDirectionSet

`ssp_err_t` R_IOPORT_PortDirectionSet (ioport_port_t port , ioport_size_t direction_values , ioport_size_t mask)

8.20.9.1 概要説明

ポートにある個々のピンの方向を設定します。portDirectionSet を実装します。

8.20.9.2 詳細説明

ポートにある複数のピンを一度に入力または出力に設定できます。マスク パラメータの各ビットは、ポートのピンに対応します。値パラメータの各ビットは、ポートのビットに対応します。ビットが 1 に設定されると、対応するピンは方向値で指定されたとおり、入力または出力に変更されます。マスク ビットが 0 に設定されると、ピンの方向は変更されません。

表 742: 戻り値

名前	説明
SSP_SUCCESS	ポートの方向が更新されました。
SSP_ERR_INVALID_POINTER	ポートおよび / またはマスクが無効です。

I：この関数は異なるポートで再入可能です。

8.20.9.3 関数のステップ

- 高ビット
- 低ビット
- ポート方向レジスタに書き込む新しい値

8.20.10 R_IOPORT_PinDirectionSet

`ssp_err_t` R_IOPORT_PinDirectionSet (ioport_port_pin_t pin , `ioport_direction_t` direction)

8.20.10.1 概要説明

ポートにある個々のピンの方向を設定します。pinDirectionSet を実装します。

8.20.10.2 詳細説明

表 743: 戻り値

名前	説明
SSP_SUCCESS	ピンの方向が更新されました。

表 743: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_POINTER	ピンおよび / または方向が無効です。

I：この関数は異なるピンで再入可能です。

8.20.11 R_IOPORT_PortEventInputRead

```
ssp_err_t R_IOPORT_PortEventInputRead ( ioport_port_t port , ioport_size_t * p_event_data )
```

8.20.11.1 概要説明

イベント入力データの値を読み取ります。portEventInputRead を実装します。

8.20.11.2 詳細説明

ポートのイベント入力データを読み取ります。指定したポートが読み取られ、すべてのピンのレベルが返されます。値パラメータの各ビットは、ポートのビットに対応します。

ポート イベント データは、ELC のトリガに応じてキャプチャされます。この関数はこのデータの読み取りを可能にします。イベント システムを使用すると、キャプチャされたデータを発生時に保存し、後で読み取ることができます。

表 744: 戻り値

名前	説明
SSP_SUCCESS	ポートが読み取られました。
SSP_ERR_INVALID_POINTER	ポートが無効です。
p_ctrl の可能性があります。	Null ポインタ

I：この関数は異なるポートで再入可能です。

8.20.12 R_IOPORT_PinEventInputRead

ssp_err_t R_IOPORT_PinEventInputRead (ioport_port_pin_t pin , ioport_level_t * p_pin_event)

8.20.12.1 概要説明

指定されたピンのイベント入力データの値を読み取ります。pinEventInputRead を実装します。

8.20.12.2 詳細説明

ピン イベント データは、ELC のトリガに応じてキャプチャされます。この関数はこのデータの読み取りを可能にします。イベント システムを使用すると、キャプチャされたデータを発生時に保存し、後で読み取ることができます。

表 745: 戻り値

名前	説明
SSP_SUCCESS	ピンが読み取られました。
SSP_ERR_INVALID_POINTER	ピンが無効です。
p_ctrl の可能性があります。	Null ポインタ

I：この関数は再入可能です。

8.20.13 R_IOPORT_PortEventOutputWrite

ssp_err_t R_IOPORT_PortEventOutputWrite (ioport_port_t port , ioport_size_t event_data , ioport_size_t mask_value)

8.20.13.1 概要説明

この関数は、ポートの設定およびリセット イベント出力を書き込みます。portEventOutputWrite を実装します。

8.20.13.2 詳細説明

イベント システムを使用すると、ポートで出力される前に、この関数がポートの状態を保存できるようになります。イベント システムを使用すると、ポートで出力される前に、この関数がポートの状態を保存できるようになります。

ポートへの出力は、ELC イベントが発生すると行われます。入力値は、指定されたポートのために構成された ELC イベントが発生すると、そのポートに書き込まれます。値パラメータの各ビットは、ポートのビットに対応します。マスク パラメータの各ビットは、ポートのピンに対応します。

表 746: 戻り値

名前	説明
SSP_SUCCESS	ポートイベントデータが書き込まれました。
SSP_ERR_INVALID_POINTER	ポートおよび / またはマスクが無効です。

I：この関数は異なるポートで再入可能です。

8.20.14 R_IOPORT_PinEventOutputWrite

```
ssp_err_t  R_IOPORT_PinEventOutputWrite (  ioport_port_pin_t pin ,  ioport_level_t   pin_value )
```

8.20.14.1 概要説明

この関数は、イベント出力データ値をピンに書き込みます。pinEventOutputWrite を実装します。

8.20.14.2 詳細説明

イベント システムを使用すると、ピンで出力される前に、この関数がピンの状態を保存できるようになります。イベント システムを使用すると、ピンで出力される前に、この関数がピンの状態を保存できるようになります。

表 747: 戻り値

名前	説明
SSP_SUCCESS	ピンイベントデータが書き込まれました。
SSP_ERR_INVALID_POINTER	ピンまたは値が無効です。

I：この関数は異なるポートで再入可能です。

8.20.15 R_IOPORT_VersionGet

ssp_err_t R_IOPORT_VersionGet (ssp_version_t * p_data)

8.20.15.1 概要説明

IOPort HAL ドライバのバージョンを返します。versionGet を実装します。

8.20.15.2 詳細説明

表 748: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報を読み取りました。

I：この関数は再入可能です。

8.20.16 R_IOPORT_EthernetModeCfg

ssp_err_t R_IOPORT_EthernetModeCfg (ioport_ethernet_channel_t channel ,
ioport_ethernet_mode_t mode)

8.20.16.1 概要説明

イーサネット チャネルの PHY モードを構成します。ioport_api_t::ethModeCfg を実装します。

8.20.16.2 詳細説明

表 749: 戻り値

名前	説明
SSP_SUCCESS	イーサネット PHY モードを設定しました。
SSP_ERR_INVALID_POINTER	チャネルまたはモードが無効です。
データ構造体が割り当てられませんでした。	イーサネット構成はこのデバイスではサポートされていません。

I：この関数は再入可能ではありません。

8.20.17 モジュール

- ビルドタイム構成

8.20.17.1 ビルドタイム構成

定義

- #define IOPORT_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.21 IWDT

独立ウォッチドッグ タイマ （IWDT）のドライバ。

8.21.1 概要

このモジュールは、独立ウォッチドッグ タイマ （IWDT）をサポートします。これは WDT インタフェースを実装します。WDT_API HAL レイヤードライバを拡張して、独立ウォッチドッグ タイマ （IWDT）周辺機器とのインタフェースを可能にします。

IWDT HAL API は、独立ウォッチドッグのリフレッシュ、タイマ値の読み取り、ステータス フラッグの読み取りとクリアの機能を提供します。NMI 出力モードで使用された場合、NMI ISR が呼び出すコールバックを登録できます。

8.21.2 Functions

- R_IWDT_Open
- R_IWDT_CfgGet
- R_IWDT_Refresh
- R_IWDT_StatusGet
- R_IWDT_StatusClear
- R_IWDT_CounterGet
- R_IWDT_TimeoutGet
- R_IWDT_VersionGet

8.21.3 定義

- #define IWDT_CODE_VERSION_MAJOR

  初期値 :(1)

- #define IWDT_CFG_PARAM_CHECKING_ENABLE

  初期値 :(#define BSP_CFG_PARAM_CHECKING_ENABLE)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.21.4 R_IWDT_Open

```
ssp_err_t  R_IWDT_Open ( wdt_ctrl_t  *const p_ctrl , wdt_cfg_t  const *const p_cfg )
```

8.21.4.1 概要説明

IWDT NMI コールバックを登録します。

8.21.4.2 詳細説明

表 750:　戻り値

名前	説明
SSP_SUCCESS	IWDT NMI コールバックが正常に構成されました。
p_ctrl の可能性があります。	Null ポインタ。
未サポートまたは不正確なモードです。	自動スタートモードで OFS0 レジスタが構成されていないときに IWDT を開こうとしました。
SSP_ERR_INVALID_ARGUMENT	IWDT モジュールはすでに呼び出されました。

I：この関数は再入可能ではありません。

8.21.4.3 関数のステップ

- g_iwdt_version にアクセスできるのは ASSERT マクロのみなので、それにアクセスできないことを示す警告がコンパイラ ツールチェーンから発行されることがあります。下のコードはこの警告を排除し、データ構造が最適化されてしまわないようにします。
- NMI 出力が使用されていない場合に、ツールチェーン警告を排除します。
- IWDT ハードウェア リソースをロックします
- WDT へのグローバル ポインタを NMI コールバックで使用できるように初期化します。
- NMI 出力モードを確認します
- NMI 出力モード
- IWDT アンダーフロー / リフレッシュ エラー割り込み （NMI を生成）を有効にします。

8.21.5 R_IWDT_CfgGet

`ssp_err_t` R_IWDT_CfgGet (`wdt_ctrl_t` *const p_ctrl , `wdt_cfg_t` *const p_cfg)

8.21.5.1 概要説明

IWDT の構成を読み取ります。cfgGet を実装します。

8.21.5.2 詳細説明

表 751: 戻り値

名前	説明
SSP_SUCCESS	IWDT の構成が正常に読み取られました。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	1 つまたは複数の構成オプションが無効です。

I：この関数は再入可能です。

8.21.5.3 関数のステップ

- OFS0 レジスタからタイムアウト値を取得します。

8.21.6 R_IWDT_Refresh

`ssp_err_t` R_IWDT_Refresh (`wdt_ctrl_t` *const p_ctrl)

8.21.6.1 概要説明

独立ウォッチドッグ タイマをリフレッシュします。refresh を実装します。

8.21.6.2 詳細説明

表 752: 戻り値

名前	説明
SSP_SUCCESS	IWDT が正常にリフレッシュされました。

I：この関数は再入可能です。この関数は、SSP_SUCCESS のみを返します。許可リフレッシュ期間の外側で行われたためにリフレッシュが失敗した場合、デバイスはリセットするか、NMI ISR をトリガーして実行します。

8.21.7 R_IWDT_StatusGet

```
ssp_err_t R_IWDT_StatusGet ( wdt_ctrl_t *const p_ctrl , wdt_status_t *const p_status )
```

8.21.7.1 概要説明

IWDT のステータス フラグを読み取ります。

8.21.7.2 詳細説明

ステータスとエラー状態の両方を示します。

表 753: 戻り値

名前	説明
SSP_SUCCESS	IWDT ステータスが正常に読み取られました。
p_ctrl の可能性があります。	NULL ポインタがパラメータ。

I：この関数は再入可能です。IWDT がアンダーフロー時またはリフレッシュエラー時にリセットを出力するように設定されている場合、リセット後にステータスとエラーフラグを読み取り、IWDT がリセットの原因かどうかを決定できます。NMI 出力モードでステータスとエラーフラグを読み取ると、IWDT によって NMI 割り込みが発生したかどうかが示されます。

8.21.8 R_IWDT_StatusClear

`ssp_err_t` R_IWDT_StatusClear (`wdt_ctrl_t` *const p_ctrl , `wdt_status_t` const status)

8.21.8.1 概要説明

IWDT のステータスおよびエラー フラグをクリアします。statusClear を実装します。

8.21.8.2 詳細説明

表 754: 戻り値

名前	説明
SSP_SUCCESS	IWDT フラグが正常にクリアされました。
p_ctrl の可能性があります。	NULL ポインタがパラメータ。

I：この関数は再入可能です。

8.21.8.3 関数のステップ

- フラグをクリアするためにゼロを書き込みます

8.21.9 R_IWDT_CounterGet

`ssp_err_t` R_IWDT_CounterGet (`wdt_ctrl_t` *const p_ctrl , uint32_t *const p_count)

8.21.9.1 概要説明

IWDT の現在のカウント値を読み取ります。counterGet を実装します。

8.21.9.2 詳細説明

表 755: 戻り値

名前	説明
SSP_SUCCESS	IWDT の現在のカウントが正常に読み取られました。
p_ctrl の可能性があります。	NULL ポインタがパラメータとして受け渡されました。

I：この関数は再入可能です。

8.21.10 R_IWDT_TimeoutGet

```
ssp_err_t R_IWDT_TimeoutGet ( wdt_ctrl_t *const p_ctrl , wdt_timeout_values_t
*const p_timeout )
```

8.21.10.1 概要説明

ウォッチドッグ タイマのタイムアウト情報を読み取ります。timeoutGet を実装します。

8.21.10.2 詳細説明

表 756: 戻り値

名前	説明
SSP_SUCCESS	WDT が正常にリフレッシュされました。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_ABORTED	このウォッチドッグのクロック分周器が無効です

I：この関数は再入可能です。この関数を呼び出す前に、R_WDT_Open を呼び出す必要があります。

8.21.11 R_IWDT_VersionGet

`ssp_err_t` R_IWDT_VersionGet (`ssp_version_t` *const p_data)

8.21.11.1 概要説明

IWDT HAL ドライバのバージョンを返します。versionGet を実装します。

8.21.11.2 詳細説明

表 757: 戻り値

名前	説明
SSP_SUCCESS	正常に呼び出しました。
p_ctrl の可能性があります。	NULL ポインタがパラメータとして受け渡されました。

I：この関数は再入可能です。

8.22 JPEG CODEC

JPEG コーデック用ドライバ。

8.22.1 Functions

- R_JPEG_Decode_Open
- R_JPEG_Decode_OutputBufferSet
- R_JPEG_Decode_LinesDecodedGet
- R_JPEG_Decode_InputBufferSet
- R_JPEG_Decode_ImageSubsampleSet
- R_JPEG_Decode_HorizontalStrideSet
- R_JPEG_Decode_Close
- R_JPEG_Decode_ImageSizeGet
- R_JPEG_Decode_StatusGet
- R_JPEG_Decode_PixelFormatGet
- R_JPEG_Decode_VersionGet

8.22.2 定義

- #define JPEG_DECODE_CODE_VERSION_MAJOR

  初期値 :(1)
- #define JPEG_DECODE_CODE_VERSION_MINOR

  初期値 :(1)
- #define JPEG_CODE_VERSION_MAJOR

  初期値 :(1)
- #define JPEG_CODE_VERSION_MINOR

  初期値 :(1)

8.22.3 R_JPEG_Decode_Open

`ssp_err_t` R_JPEG_Decode_Open (`jpeg_decode_ctrl_t` *const p_ctrl , `jpeg_decode_cfg_t` const *const p_cfg)

8.22.3.1 概要説明

JPEG コーデック モジュールを初期化します。この関数は、JPEG コーデックをデコード操作用に設定して、ユーザーが指定した設定パラメータに基づいてデータ形式とピクセル形式用のレジスタをセットアップします。画像サイズ読み取り操作とコールバック関数をサポートするために割り込みが有効にされます。

8.22.3.2 詳細説明

表 758: 戻り値

名前	説明
SSP_SUCCESS	JPEG コーデック モジュールが正しく設定され、入力データを取得する準備ができています。
チャネルは現在動作中です。	JPEG コーデックが使用中です。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_ARGUMENT	JPEG コーデック リソースがロックされています。

8.22.3.3 関数のステップ

- JPEG コーデックがまだ使用されていないことを確認します。
- 設定値を記録します。
- 水平ストライド値を初期化します。
- 出力バッファ サイズを初期化します。
- total_lines_decoded を初期化します
- JPEG モジュールに電源を供給します。
- バス リセットを実行します。
- 宛先バッファ アドレスをリセットします。
- ソース バッファ アドレスをリセットします。
- 水平ストライドをリセットします。
- JPEG モジュールをデコード操作用に設定します。
- デコードする画像の画像形式をセットします。
- 出力ピクセル形式が ARGB8888 の場合は、アルファ値も設定します。

- 出力ピクセル形式が ARGB8888 の場合は、アルファ値も設定します。
- デコードする画像のアルファ値をセットします。
- すべてのエラーに対する割り込みと画像サイズに対する割り込みが有効になります。
- ユーザーが入力したコールバック ルーチンを記録します。
- ctrl ステータスをセットします。
- ctrl ハンドルを記録します。ISR ルーチンは制御ブロックにアクセスする必要があります。
- すべてが完了しました。正常終了を返します。

8.22.4 R_JPEG_Decode_OutputBufferSet

```
ssp_err_t R_JPEG_Decode_OutputBufferSet ( jpeg_decode_ctrl_t * p_ctrl , void * p_output_buffer ,
uint32_t output_buffer_size )
```

8.22.4.1 概要説明

出力データを保存するために、出力バッファを JPEG コーデックに割り当てます。

8.22.4.2 詳細説明

I：デコードされるイメージ行数は、バッファのサイズと水平ストライド設定によって決まります。出力バッファサイズ、水平ストライド値、および入力ピクセル フォーマット （入力ピクセルフォーマットは、JPEG ヘッダーから JPEG デコーダが取得）が分かると、ドライバは出力バッファにデコードできる行数を自動で計算します。これらの行がデコードされた後、JPEG エンジンは一時停止してコールバック関数がトリガされるため、アプリケーションは JPEG モジュールが操作を再開できるように次のバッファを提供します。

JPEG デコーディング操作は、入力バッファと出力バッファの両方が設定され、出力バッファが少なくとも 8 行のデコードされたイメージ データを保有だきるだけの大きさの場合に、自動的に開始されます。

表 759: 戻り値

名前	説明
SSP_SUCCESS	出力バッファが JPEG コーデックデバイスに適切に割り当てられました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL か、output_buffer へのポインタが NULL か、output_buffer_size が 0 です。

表 759: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_ALIGNMENT	バッファ開始アドレスは 8 バイトに調整されていません。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.4.3 関数のステップ

- イメージ サイズの準備ができていない場合、ドライバは入力ピクセル フォーマットが分かりません。その情報がないと、ドライバはデコードするイメージの行数を計算できません。その場合、ドライバは出力バッファ サイズを記録します。すべての情報が整った後、ドライバはデコーディング処理の開始を試行します。
- デコーディング先アドレスを設定します。
- 出力バッファのサイズを記録します。
- デコーディング先アドレスを設定します。
- あるバッファ サイズで、イメージ サイズを取得したことが分かれば、デコードするイメージの行数を計算します。
- ドライバ ステータスが IMAGE_SIZE_READY で、他にフラグがない場合、ドライバが IMAGE_SIZE を受け取ったことを意味します。デコーディング処理はまだ開始していません。
- デフォルトで、出力カウントはオープンです。
- 入力バッファ、出力バッファ、および水平ストライドが設定されている場合、ドライバはデコードする行数を特定することができ、デコーディング動作を開始できます。
- 内部ステータス情報をセットします。
- コアを開始します。
- 現在のステータスが OUTPUT_PAUSE の場合、ドライバは動作を再開する必要があります。
- 内部ステータス情報をクリアします。
- 内部ステータス情報をセットします。
- jpeg コアを再開します。
- 出力バッファのサイズを記録します。

8.22.5 R_JPEG_Decode_LinesDecodedGet

```
ssp_err_t R_JPEG_Decode_LinesDecodedGet ( jpeg_decode_ctrl_t * p_ctrl , uint32_t * p_lines )
```

8.22.5.1 概要説明

出力バッファにデコードされた行数を返します。

8.22.5.2 詳細説明

I :JPEG が部分イメージをデコードした後に、出力バッファに書き込まれたイメージ行数を取得するには、この関数を使用します。アプリケーションは、水平ストライド設定と出力ピクセル フォーマットを組み合わせて、出力バッファから読み取るデータの量を計算できます。

表 760: 戻り値

名前	説明
SSP_SUCCESS	出力バッファが JPEG コーデックデバイスに適切に割り当てられました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL か、output_buffer へのポインタが NULL か、output_buffer_size が 0 です。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.6 R_JPEG_Decode_InputBufferSet

```
ssp_err_t  R_JPEG_Decode_InputBufferSet ( jpeg_decode_ctrl_t  *const p_ctrl ,  void
* p_data_buffer ,  uint32_t data_buffer_size )
```

8.22.6.1 概要説明

処理用の入力データ バッファを JPEG コーデックに割り当てます。

8.22.6.2 詳細説明

I : 一定量のデータを処理すると、JPEG ドライバは、フラグ JPEG_OPERATION_INPUT_PAUSE をセットしてコールバック関数をトリガーします。アプリケーションが次のデータ チャンクをドライバに提供するため、JPEG デコードを再開できます。

出力バッファはデコードする画像データの 1 行分以上を保存するのに十分な大きさです。

表 761: 戻り値

名前	説明
SSP_SUCCESS	入力データ バッファが正しく JPEG コーデック デバイスに割り当てられました。
p_ctrl の可能性があります。	制御ブロックへのポインタが NULL か、input_buffer へのポインタが NULL か、input_buffer_size が 0 です。
SSP_ERR_INVALID_ALIGNMENT	バッファ開始アドレスは 8 バイトに調整されていません。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.6.3 関数のステップ

- 入力バッファ アドレスを設定します。
- システムがアイドル状態の場合は、JPEG エンジンを開始します。これにより、システムは、画像情報 （画像サイズと入力ピクセル形式）を取得できます。この情報は、後でデコード プロセスを起動するときに必要です。
- バッファ サイズに基づいて、イン カウント モード設定を検出します。ドライバは入力データをチャンク単位で読み取ることができます。ただし、それぞれのチャンクのサイズは BUFFER_MAX_SIZE に制限されます。そのため、入力データ サイズが BUFFER_MAX_SIZE を上回っている場合は、ドライバが入力データ全体が渡されたと仮定して、それ以降の入力データを要求せずにデコードできます。それ以外の場合は、ドライバが入力ストリーム機能を有効にします。これは、全体の入力サイズが BUFFER_MAX_SIZE を下回っている場合でも機能します。
- 内部ステータスをセットします。
- 内部ステータス情報をクリアします。
- 内部ステータス情報をセットします。
- jpeg コアを再開します。

8.22.7 R_JPEG_Decode_ImageSubsampleSet

```
ssp_err_t  R_JPEG_Decode_ImageSubsampleSet ( jpeg_decode_ctrl_t  *const p_ctrl ,
 jpeg_decode_subsample_t   horizontal_subsample ,
 jpeg_decode_subsample_t   vertical_subsample )
```

8.22.7.1 概要説明

水平および垂直サブサンプルを構成します。

8.22.7.2 詳細説明

I：デコードされたイメージのスケーリングに使用します。

表 762: 戻り値

名前	説明
SSP_SUCCESS	水平ストライド値が適切に構成されました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.8 R_JPEG_Decode_HorizontalStrideSet

```
ssp_err_t  R_JPEG_Decode_HorizontalStrideSet ( jpeg_decode_ctrl_t  * p_ctrl ,
 uint32_t horizontal_stride )
```

8.22.8.1 概要説明

水平ストライドを設定します。

8.22.8.2 詳細説明

I：水平ストライドが画像幅と一致する必要があり、JPEG ドライバを開いたときに画像サイズが不明の場合に使用します。(open 呼び出しの前に画像サイズがわかっている場合は、jpef_cfg_t 構造体で水平ストライド値を渡します)。画像サイズが入手可能になったら、この関数を使用して水平ストライド値を更新します。ドライバが一度に 1 行ずつデコードする必要がある場合は、水平ストライドを 0 にセットできます。

表 763: 戻り値

名前	説明
SSP_SUCCESS	水平ストライド値が適切に構成されました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
SSP_ERR_INVALID_ALIGNMENT	水平ストライドが 8 バイトでアラインされていません。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.8.3 関数のステップ

- 水平ストライド値を制御ブロックに記録します。
- 水平ストライドをセットします。
- すべてのパラメータがセットされたら、コアによるデコードを再開します。
- 指定されたバッファ サイズに対して、デコードする行数を計算します。
- アウト カウント モードを設定します。

8.22.9 R_JPEG_Decode_Close

```
ssp_err_t  R_JPEG_Decode_Close ( jpeg_decode_ctrl_t  * p_ctrl )
```

8.22.9.1 概要説明

未処理の JPEG コーデック操作をキャンセルして、デバイスを閉じます。

8.22.9.2 詳細説明

表 764: 戻り値

名前	説明
SSP_SUCCESS	入力データ バッファが正しく JPEG コーデック デバイスに割り当てられました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.9.3 関数のステップ

- JPEG JINTE0 割り込みと JINTE1 割り込みをクリアします。
- NVIC で JEDI と JDTI を無効にします。
- JPEG コーデックの電源をオフにします。
- ドライバ内の jpeg ステータス フラグをリセットします。
- BSP レベルでモジュールをロック解除します。

8.22.10 R_JPEG_Decode_ImageSizeGet

ssp_err_t R_JPEG_Decode_ImageSizeGet (jpeg_decode_ctrl_t * p_ctrl , uint16_t * p_horizontal_size , uint16_t * p_vertical_size)

8.22.10.1 概要説明

この関数は、JPEG 画像をデコードする場合のみ機能します。この動作は、JPEG デコーディング動作中に有効です。

8.22.10.2 詳細説明

表 765: 戻り値

名前	説明
SSP_SUCCESS	イメージサイズが使用可能で、水平値および垂直値は p_horizontal_size および p_vertical_size がポイントするメモリーに保存されます。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
SSP_ERR_IMAGE_SIZE_UNKNOWN	イメージ サイズが不明です。さらに多くの入力データが必要な可能性があります。
未サポートまたは不正確なモードです。	JPEG Codec モジュールがデコードしていません。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.11 R_JPEG_Decode_StatusGet

ssp_err_t R_JPEG_Decode_StatusGet (jpeg_decode_ctrl_t * p_ctrl , jpeg_decode_status_t * p_status)

8.22.11.1 概要説明

JPEG コーデックのステータスを取得します。この関数はデバイスのポーリングにも使用できます。

8.22.11.2 詳細説明

表 766: 戻り値

名前	説明
SSP_SUCCESS	ステータス情報を正常に取得されました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.11.3 関数のステップ

- ハードウェアはエラーを報告しません。内部ステータス情報を返します。

8.22.12 R_JPEG_Decode_PixelFormatGet

```
ssp_err_t R_JPEG_Decode_PixelFormatGet ( jpeg_decode_ctrl_t * p_ctrl ,
 jpeg_decode_color_space_t * p_color_space )
```

8.22.12.1 概要説明

入力ピクセル形式を取得します。

8.22.12.2 詳細説明

表 767: 戻り値

名前	説明
SSP_SUCCESS	ステータス情報を正常に取得されました。
p_ctrl の可能性があります。	コントロールブロックへのポインタが NULL です。
タッチ パネルが設定されていません。	JPEG が開かれていません。

8.22.12.3 関数のステップ

- ハードウェアはエラーを報告しません。内部ステータス情報を返します。

8.22.13 R_JPEG_Decode_VersionGet

`ssp_err_t` R_JPEG_Decode_VersionGet (`ssp_version_t` * p_version)

8.22.13.1 概要説明

表示インタフェースと GLCD HAL コードのバージョンを取得します。

8.22.13.2 詳細説明

表 768: 戻り値

名前	説明
SSP_SUCCESS	バージョン番号

I：この関数は再入可能です。

8.22.14 モジュール

- ビルドタイム構成

8.22.14.1 ビルドタイム構成

定義

- #define JPEG_DECODE_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.23 キー割り込み

キー割り込み機能用ドライバ。

キー入力ドライバは、1 ～ 8 までのチャネルにまたはマトリックス形式で使用できます。このモジュールは、次のインタフェースを実装します : Key Matrix インタフェース。

8.23.1 Functions

- R_KINT_KEYMATRIX_Open
- R_KINT_KEYMATRIX_Close
- R_KINT_KEYMATRIX_Enable
- R_KINT_KEYMATRIX_Disable
- R_KINT_KEYMATRIX_TriggerSet
- R_KINT_VersionGet

8.23.2 定義

- #define KINT_CODE_VERSION_MAJOR

  初期値 :(1U)
- #define KINT_CODE_VERSION_MINOR

  初期値 :(0U)

8.23.3 R_KINT_KEYMATRIX_Open

`ssp_err_t` R_KINT_KEYMATRIX_Open (`keymatrix_ctrl_t` *const p_ctrl , `keymatrix_cfg_t` const *const p_cfg)

8.23.3.1 概要説明

KINT の電源をオンにして、ユーザーズマニュアルに記載されているように必要な初期化を処理します。open を実装します。

8.23.3.2 詳細説明

Open 関数は、すべてのキー入力 （KINT）チャネルを設定し、残りの KINT API 関数で使用するためのハンドルを提供します。この関数は、他の KINT API 関数を呼び出す前に 1 回以上呼び出す必要があります。ペリフェラルの初期化後は、Close 関数を呼び出す前に Open 関数を繰り返し呼び出さないようにする必要があります。

表 769: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	p_cfg パラメータと p_ctrl パラメータのどちらかまたはコールバックが NULL の可能性があります。
SSP_ERR_INVALID_POINTER	次のいずれかが無効になっている可能性があります。 • p_cfg->channel: 0 ～ KINT_MAX_NUM_CHANNELS の範囲内である必要があります • p_cfg->hw_trigger が有効な値ではありません。
SSP_ERR_INVALID_ARGUMENT	API がすでに開かれています。もう一度開く前に閉じる必要があります。

I：この関数は再入可能ではありません。

8.23.3.3 関数のステップ

- 渡された引数が null ポインタでないことを確認します。
- KINT ペリフェラルで割り込みを無効にします。
- KINT ペリフェラル内で保留中の割り込み要求をクリアします。
- ICU 内の割り込み要求をクリアします。
- トリガー エッジを設定します。トリガー エッジは必要に応じて TriggerSet 関数で変更できます。
- KEYMATRIX_TRIG_RISING 状態。
- キー割り込みフラグの用途を設定します。
- 割り込みが有効になっている場合は、それを選択されたチャネルに対してセットアップします。
- チャネルは後から IrqEnable 関数で変更できますが、コールバックとコンテキストを変更するには、API を閉じてから、新しいコールバックとコンテキストで開き直す必要があります。KINT ハードウェアは、すべてのチャネルに対して 1 つの割り込みしかサポートしません。
  I :KINT ハードウェアは、すべてのチャネルに対して 1 つの割り込みしかサポートしません。

KRM は 8 ビット レジスタのため、キャスト後に選択されたチャネルに対する割り込みを有効にします。

- 割り込みを有効にします。
- ユーザーに返す制御ブロックの値を保存します。
- KINT を初期化済みとしてマークします。

8.23.4 R_KINT_KEYMATRIX_Close

```
ssp_err_t  R_KINT_KEYMATRIX_Close ( keymatrix_ctrl_t  *const p_ctrl )
```

8.23.4.1 概要説明

KINT を無効にします。close を実装します。

8.23.4.2 詳細説明

関数 R_KINT_KEYMATRIX_Open の終了後に、Close 関数がペリフェラルと NVIC 内の割り込みを無効にして、保留になっている割り込み要求をすべてクリアします。API に対するハードウェア ロックも解除します。

表 770: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

8.23.4.3 関数のステップ

- ICU 内の割り込みを無効にします。
- KINT ペリフェラルで割り込みを無効にします。
- KINT ペリフェラル内で保留中の割り込み要求をクリアします。
- 割り込み要求ビットをクリアします。
- 保存されている内部ドライバ データをクリアします。
- KINT を未初期化としてマークします。

- ロックを解除します。

8.23.5 R_KINT_KEYMATRIX_Enable

`ssp_err_t` R_KINT_KEYMATRIX_Enable (`keymatrix_ctrl_t` *const p_ctrl)

8.23.5.1 概要説明

NVIC で指定されたチャネルの外部 irq を有効にします。enable を実装します。

8.23.5.2 詳細説明

この関数は、保留になっている要求を初めてクリアした後で、このペリフェラルに対する割り込みを割り込みレベルと NVIC 内の両方で有効にします。

表 771: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	ペリフェラルが開かれていません。

8.23.5.3 関数のステップ

- KINT ペリフェラル内で保留中の割り込み要求をクリアします。
- ICU 内の割り込み要求フラグをクリアします。
- KRM は 8 ビット レジスタのため、キャスト後に選択されたチャネルに対する割り込みを有効にします。
- 割り込みを有効にします。

8.23.6 R_KINT_KEYMATRIX_Disable

`ssp_err_t` R_KINT_KEYMATRIX_Disable (`keymatrix_ctrl_t` *const p_ctrl)

8.23.6.1 概要説明

NVIC で指定されたチャネルの外部 irq を無効にします。disable を実装します。

8.23.6.2 詳細説明

表 772: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	

表 773: 戻り値

名前	説明
SSP_SUCCESS	中断が正常に無効にされました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.23.6.3 関数のステップ

- KINT ペリフェラルで割り込みを無効にします。
- KINT ペリフェラル内で保留中の割り込み要求をクリアします。
- ICU 内の割り込みを無効にします。

8.23.7 R_KINT_KEYMATRIX_TriggerSet

```
ssp_err_t  R_KINT_KEYMATRIX_TriggerSet ( keymatrix_ctrl_t  *const p_ctrl ,
 keymatrix_trigger_t   hw_trigger )
```

8.23.7.1 概要説明

指定されたトリガー値をセットします。triggerSet を実装します。

8.23.7.2 詳細説明

表 774: 戻り値

名前	説明
SSP_SUCCESS	正常にトリガー値が書き込まれました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
SSP_ERR_INVALID_POINTER	トリガー入力が無効です。hw_trigger は button_trigger_t からのオプションのいずれかにする必要があります。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.23.7.3 関数のステップ

- トリガー エッジを設定します。
- KEYMATRIX_TRIG_RISING 状態。
- キー割り込みフラグの用途を設定します。

8.23.8 R_KINT_VersionGet

```
ssp_err_t  R_KINT_VersionGet ( ssp_version_t  *const p_version )
```

8.23.8.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンをセットします。

8.23.8.2 詳細説明

表 775: 戻り値

名前	説明
SSP_SUCCESS	正常な復帰。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.23.9 モジュール

- ビルドタイム構成

8.23.9.1 ビルドタイム構成

定義

- #define KINT_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.24 LPM

ローパワー モード用ドライバ。

この LPM （ローパワー モード）ドライバには、モジュールの停止、動作モードの選択、ローパワー モードの構成、ローパワー モードへの移行など、電力消費を制御する機能がいくつかあります。LPM ドライバは、LPM ハードウェア周辺機器を使用する MCU 動作モードと MCU ローパワー モードの設定をサポートしています。LPM ドライバは、動作モードの定電圧、低速、中速、高速、および副発振器モードをサポートしています。

また、LPM ドライバは、ディープ スタンバイ、スタンバイ、スリープ、およびスヌーズの低電力モードをサポートしています。LPM ドライバは、ディープ スタンバイ モード時、内部の電力供給制御と IO ポートのステータス再設定を通して電力消費の低減をサポートします。さらに、LPM ドライバは、MCU のその他のハードウェア周辺機器の無効化と有効化をサポートしています。

I：すべての MCU で、すべての動作モードが利用可能とは限りません。すべての MCU で、すべてのローパワー モードが利用可能とは限りません。

8.24.1 Functions

- R_LPM_Init
- R_LPM_MSTPCRSet
- R_LPM_MSTPCRGet
- R_LPM_ModuleStop
- R_LPM_ModuleStart
- R_LPM_OperatingPowerModeSet
- R_LPM_SnoozeEnable
- R_LPM_SnoozeDisable
- R_LPM_LowPowerConfigure
- R_LPM_WUPENSet
- R_LPM_WUPENGet
- R_LPM_DeepStandbyCancelRequestEnable
- R_LPM_DeepStandbyCancelRequestDisable
- R_LPM_LowPowerModeEnter

- R_LPM_VersionGet

8.24.2 定義

- #define INLINE_ATTRIBUTE

  初期値 :
- #define LPM_CODE_VERSION_MAJOR

  初期値 :(2)
- #define LPM_CODE_VERSION_MINOR

  初期値 :(1)

8.24.3 R_LPM_Init

```
ssp_err_t R_LPM_Init ( lpm_cfg_t const *const p_cfg )
```

8.24.3.1 概要説明

MCU 動作電力モードを初期化します。

8.24.3.2 詳細説明

設定構造体に渡された値に従って、MCU 動作電力モードを初期化します。

表 776: 戻り値

名前	説明
SSP_SUCCESS	設定が正常に終了しました。
p_ctrl の可能性があります。	p_cfg が NULL でした。

8.24.4 R_LPM_MSTPCRSet

```
ssp_err_t R_LPM_MSTPCRSet ( uint32_t mstpcra_value , uint32_t mstpcrb_value ,
 uint32_t mstpcrc_value , uint32_t mstpcrd_value )
```

8.24.4.1 概要説明

すべてのモジュール ストップ コントロール レジスタの値を設定します。

8.24.4.2 詳細説明

表 777: 戻り値

名前	説明
SSP_SUCCESS	値が正常に設定されました。

8.24.5 R_LPM_MSTPCRGet

ssp_err_t R_LPM_MSTPCRGet (uint32_t * mstpcra_value , uint32_t * mstpcrb_value , uint32_t * mstpcrc_value , uint32_t * mstpcrd_value)

8.24.5.1 概要説明

すべてのモジュール ストップ コントロール レジスタの値を取得します。

8.24.5.2 詳細説明

表 778: 戻り値

名前	説明
SSP_SUCCESS	値が正常に取得されました。
p_ctrl の可能性があります。	ポインタで渡された値が NULL でした。

8.24.6 R_LPM_ModuleStop

ssp_err_t R_LPM_ModuleStop (lpm_mstp_t module)

8.24.6.1 概要説明

モジュールを停止します。

8.24.6.2 詳細説明

モジュール停止レジスタ内の対応するビットをセットすることによって実行中のモジュールを停止します。

表 779: 戻り値

名前	説明
SSP_SUCCESS	設定が正常に終了しました。

8.24.7 R_LPM_ModuleStart

```
ssp_err_t R_LPM_ModuleStart ( lpm_mstp_t module )
```

8.24.7.1 概要説明

モジュールを実行します。

8.24.7.2 詳細説明

モジュール停止レジスタ内の対応するビットをクリアすることによって実行中のモジュールを開始します。

表 780: 戻り値

名前	説明
SSP_SUCCESS	設定が正常に終了しました。

8.24.8 R_LPM_OperatingPowerModeSet

```
ssp_err_t R_LPM_OperatingPowerModeSet ( lpm_operating_power_t power_mode ,
 lpm_subosc_t subosc )
```

8.24.8.1 概要説明

動作電力モードをセットします。

8.24.8.2 詳細説明

表 781: 戻り値

名前	説明
SSP_SUCCESS	動作電力モードが正常にセットされました。

8.24.9 R_LPM_SnoozeEnable

```
ssp_err_t R_LPM_SnoozeEnable ( lpm_snooze_rxd0_t rdx0_mode ,
 lpm_snooze_dtc_t dtc_mode , lpm_snooze_request_t requests , lpm_snooze_end_t triggers )
```

8.24.9.1 概要説明

スヌーズ モードを設定して有効にします。

8.24.9.2 詳細説明

この関数は、ソフトウェア スタンバイ モードに入る前に呼び出す必要があります。

表 782: 戻り値

名前	説明
SSP_SUCCESS	スヌーズ モードが正常に有効になりました。

8.24.10 R_LPM_SnoozeDisable

```
ssp_err_t R_LPM_SnoozeDisable ( void )
```

8.24.10.1 概要説明

スヌーズ モードを無効にします。

8.24.10.2 詳細説明

表 783: 戻り値

名前	説明
SSP_SUCCESS	スヌーズ モードが無効になりました

8.24.11 R_LPM_LowPowerConfigure

```
ssp_err_t R_LPM_LowPowerConfigure ( lpm_low_power_mode_t power_mode ,
 lpm_output_port_enable_t output_port_enable , lpm_power_supply_t power_supply ,
 lpm_io_port_t io_port_state )
```

8.24.11.1 概要説明

低電力モードを設定します。

8.24.11.2 詳細説明

注記 : この関数は低電力モードを開始するのではなく、モードのパラメータを設定するだけです。低電力モードを開始するには、WFI 命令を実行します。

表 784: 戻り値

名前	説明
SSP_SUCCESS	スリープ モードが正常に開始されました。

8.24.12 R_LPM_WUPENSet

```
ssp_err_t  R_LPM_WUPENSet (  uint32_t wupen_value )
```

8.24.12.1 概要説明

Wake Up 割り込み有効化レジスタ WUPEN の値を設定します。

8.24.12.2 詳細説明

表 785: 戻り値

名前	説明
SSP_SUCCESS	値が正常に設定されました。

8.24.13 R_LPM_WUPENGet

```
ssp_err_t  R_LPM_WUPENGet (  uint32_t * wupen_value )
```

8.24.13.1 概要説明

Wake Up 割り込み有効化レジスタ WUPEN の値を取得します。

8.24.13.2 詳細説明

表 786: 戻り値

名前	説明
SSP_SUCCESS	値が正常に取得されました。
p_ctrl の可能性があります。	ポインタで渡された値が NULL でした。

8.24.14 R_LPM_DeepStandbyCancelRequestEnable

```
ssp_err_t R_LPM_DeepStandbyCancelRequestEnable ( lpm_deep_standby_t pin_signal ,
 lpm_cancel_request_edge_t rising_falling )
```

8.24.14.1 概要説明

ディープ スタンバイのキャンセル要求を有効にします。

8.24.14.2 詳細説明

ディープ ソフトウェア スタンバイ モードをキャンセルするピンまたは信号を有効にします。

表 787: 戻り値

名前	説明
SSP_SUCCESS	ディープ ソフトウェア スタンバイ キャンセル要求が有効になりました。

8.24.15 R_LPM_DeepStandbyCancelRequestDisable

```
ssp_err_t R_LPM_DeepStandbyCancelRequestDisable ( lpm_deep_standby_t pin_signal )
```

8.24.15.1 概要説明

ディープ スタンバイのキャンセル要求を無効にします。

8.24.15.2 詳細説明

ディープ ソフトウェア スタンバイ モードをキャンセルするピンまたは信号を無効にします。

表 788: 戻り値

名前	説明
SSP_SUCCESS	ディープ ソフトウェア スタンバイ キャンセル要求が無効になりました。

8.24.16 R_LPM_LowPowerModeEnter

ssp_err_t R_LPM_LowPowerModeEnter (void)

8.24.16.1 概要説明

WFI マクロを使用して、ローパワー モード （スリープ / スタンバイ / ディープ スタンバイ）に移行します。

8.24.16.2 詳細説明

ローパワー モードが解除されると、関数が復帰します。

表 789: 戻り値

名前	説明
SSP_SUCCESS	成功しました。

8.24.17 R_LPM_VersionGet

ssp_err_t R_LPM_VersionGet (ssp_version_t *const p_version)

8.24.17.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

8.24.17.2 詳細説明

表 790: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。

表 790: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_PTR	p_version が NULL です。

8.24.18 モジュール

- ビルドタイム構成

8.24.18.1 ビルドタイム構成

<Enter concept body here>

8.25 LVD

低電圧検出用ドライバ。

LVD API の実装。詳細な説明は 低電圧検出ドライバインタフェース を参照してください。

8.25.1 Functions

- R_LVD_Open
- R_LVD_Close
- R_LVD_StatusGet
- R_LVD_StatusClear
- R_LVD_VersionGet

8.25.2 定義

- #define LVD_CODE_VERSION_MAJOR

  初期値 :(01U)
- #define LVD_CODE_VERSION_MINOR

  初期値 :(00U)

8.25.3 R_LVD_Open

```
ssp_err_t R_LVD_Open ( lvd_ctrl_t *const p_ctrl , lvd_cfg_t const *const p_cfg )
```

8.25.3.1 概要説明

渡された構成構造体に従って、低電圧検出ドライバを初期化します。構成構造体に従い、LVD 周辺機器を有効化します。

8.25.3.2 詳細説明

I：スタンバイ モードではデジタル フィルタは使用できません

表 791: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの監視制御構造体へのポインタ。
p_cfg	複数のビットを書き換えることもできます。	ドライバ インスタンスの構成構造体へのポインタ

表 792: 戻り値

名前	説明
SSP_SUCCESS	成功しました
p_ctrl の可能性があります。	p_ctrl が NULL、p_cfg が NULL、電圧監視の要求された構成が無効でした

8.25.4 R_LVD_Close

```
ssp_err_t  R_LVD_Close ( lvd_ctrl_t  *const p_ctrl )
```

8.25.4.1 概要説明

LVD 周辺機器を無効化します。ドライバ インスタンスを閉じます。

8.25.4.2 詳細説明

表 793: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの監視制御構造体へのポインタ。

表 794: 戻り値

名前	説明
SSP_SUCCESS	成功しました
p_ctrl の可能性があります。	p_ctrl が NULL でした、無効な監視番号

8.25.5 R_LVD_StatusGet

```
ssp_err_t R_LVD_StatusGet ( lvd_ctrl_t *const p_ctrl , lvd_status_t * p_lvd_status )
```

8.25.5.1 概要説明

監視の現在の状態を取得します （しきい値の超過が検出された、現在電圧が範囲内にある）

8.25.5.2 詳細説明

表 795: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの監視制御構造体へのポインタ。
p_lvd_status	入力 / 出力	lvd_status_t インスタンスへのポインタ

表 796: 戻り値

名前	説明
SSP_SUCCESS	成功しました
p_ctrl の可能性があります。	p_ctrl が NULL、p_lvd_status が NULL でした、無効な LVD 監視番号
タッチ パネルが設定されていません。	ドライバがオープンではありません、ステータスが有効になりません

8.25.6 R_LVD_StatusClear

ssp_err_t R_LVD_StatusClear (lvd_ctrl_t *const p_ctrl)

8.25.6.1 概要説明

監視のラッチ状態をクリアします。

8.25.6.2 詳細説明

表 797: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	ドライバ インスタンスの監視制御構造体へのポインタ。

表 798: 戻り値

名前	説明
SSP_SUCCESS	成功しました
p_ctrl の可能性があります。	p_ctrl が NULL でした、無効な LVD 監視番号
タッチ パネルが設定されていません。	ドライバがオープンではありません、ステータスが有効になりません

8.25.7 R_LVD_VersionGet

ssp_err_t R_LVD_VersionGet (ssp_version_t *const p_version)

8.25.7.1 概要説明

コンパイル時マクロに基づいて、LVD ドライバのバージョンを返します。

8.25.7.2 詳細説明

表 799: パラメータ

名前	方向	説明
p_version	入力 / 出力	バージョン構造体へのポインタ

表 800: 戻り値

名前	説明
SSP_SUCCESS	成功しました
p_ctrl の可能性があります。	p_version が NULL でした

8.25.8 API データ

8.25.8.1 lvd_sample_clock_t

lvd_sample_clock_t

詳細説明

サンプル クロック分周器、デジタル フィルタリングを無効化するには LVD_SAMPLE_CLOCK_DISABLED を使用します。

列挙値

名前	説明
LVD_SAMPLE_CLOCK_LOCO_DIV_2	デジタル フィルタ サンプル クロックは 2 分周の LOCO です。
LVD_SAMPLE_CLOCK_LOCO_DIV_4	デジタル フィルタ サンプル クロックは 4 分周の LOCO です。
LVD_SAMPLE_CLOCK_LOCO_DIV_8	デジタル フィルタ サンプル クロックは 8 分周の LOCO です。
LVD_SAMPLE_CLOCK_LOCO_DIV_16	デジタル フィルタ サンプル クロックは 16 分周の LOCO です。

名前	説明
LVD_SAMPLE_CLOCK_DISABLED	デジタル フィルタが無効化されています。

8.25.8.2 lvd_negation_delay_t

lvd_negation_delay_t

詳細説明

LVD 信号のネゲートは、リセットに続いて、または電圧が範囲内になった後で行われます

列挙値

名前	説明
LVD_NEGATION_DELAY_FROM_VOLTAGE	ネゲートは、VCC > Vdet1 が検出された後の安定化時間（tLVDn）に続いて行われます。ソフトウェア スタンバイまたはディープ ソフトウェア スタンバイへの移行が行われる場合、RN ビットの可能値は LVD_NEGATION_DELAY_FROM_VOLTAGE のみとなります
LVD_NEGATION_DELAY_FROM_RESET	ネゲートは、LVDn リセットのアサーション後の安定化時間 （tLVDn）に続いて行われます。ソフトウェア スタンバイまたはディープ ソフトウェア スタンバイへの移行が行われる場合、RN ビットの可能値は LVD_NEGATION_DELAY_FROM_VOLTAGE のみとなります

8.25.9 拡張

8.25.9.1 lvd_extend_t

lvd_extend_t

詳細説明

ハードウェア拡張構造体

変数

- lvd_negation_delay_t negation_delay

  LVD 信号のネゲートは、リセットに続いて、または電圧が範囲内になった後で行われます

- lvd_sample_clock_t sample_clock_divisor

  サンプル クロック分周器、デジタル フィルタリングを無効化するには LVD_SAMPLE_CLOCK_DISABLED を使用します

8.25.10 モジュール

- ビルドタイム構成

8.25.10.1 ビルドタイム構成

定義

- #define LVD_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.26 PDC

パラレルデータキャプチャユニット（PDC）のドライバ。

8.26.1 概要

PDC インタフェース を拡張します。PDC インタフェースは、カメラ モジュールからの画像キャプチャを許可します。

8.26.2 Functions

- R_PDC_Open
- R_PDC_Close
- R_PDC_CaptureStart
- R_PDC_StateGet
- R_PDC_VersionGet

8.26.3 定義

- #define PDC_CODE_VERSION_MAJOR

  初期値 :(01)
- #define PDC_CODE_VERSION_MINOR

  初期値 :(1)

8.26.4 R_PDC_Open

```
ssp_err_t R_PDC_Open ( pdc_ctrl_t *const p_ctrl , pdc_cfg_t const *const p_cfg )
```

8.26.4.1 概要説明

PDC の電源をオンにして、ユーザーズマニュアルに説明されている必要な初期化を行います。open を実装します。

8.26.4.2 詳細説明

open 関数は、PDC モジュールの初期構成を提供します。モジュールの電源をオンにし、PCLKO 出力および PIXCLK 入力を有効化します。それ以上の初期化を行うには、PIXCLK 入力を実行し、PDC を初期化の一部としてリセットする必要があります。このクロックはカメラ モジュールから入力され、リセットとそれ以上の初期化は captureStart で行われます。この関数は、その他の PDC API 関数の呼び出しを行う前に呼び

出す必要があります。PDC を開いた後は、Close 関数を呼び出す前に Open 関数を再度呼び出さないようにしてください。

表 801: 戻り値

名前	説明
SSP_SUCCESS	初期化が正常に行われました。
p_ctrl の可能性があります。	次のいずれかのパラメータが不正確です。p_cfg が NULL か、または • p_ctrl が NULL か、または • p_cfg パラメータでリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できません。 • p_cfg パラメータで指定したバッファが NULL であるか、または • p_cfg パラメータの転送インタフェースへのポインタが NULL です
SSP_ERR_INVALID_POINTER	次のいずれかの構成パラメータに誤りがあります。 p_cfg が NULL か、または • bytes_per_pixel がゼロ、または • x_resolution_pixels がゼロ、または • y_resolution_pixels がゼロ、または • x_resolution_pixels が x_capture_start_pixel + x_capture_pixels を下回る、または • y_resolution_pixels が y_capture_start_pixel + y_capture_pixels を下回る
SSP_ERR_ALREADY_OPEN	ドライバがすでに開いています。
SSP_ERR_INVALID_ARGUMENT	このモジュールに対する BSP ハードウェア ロックを保存できません。

I：この関数は再入可能ではありません。

8.26.4.3 関数のステップ

- ドライバがすでに開いていないか確認します。
- PDC ハードウェア リソースをロックします

8.26.5 R_PDC_Close

```
ssp_err_t  R_PDC_Close ( pdc_ctrl_t  *const p_ctrl )
```

8.26.5.1 概要説明

転送インタフェースを停止して閉じ、PDC を無効化し、PDC の電源をオフにし、内部ドライバ データをクリアして割り込みを無効化します。close を実装します。

8.26.5.2 詳細説明

表 802: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	次のいずれかのパラメータが不正確です。p_cfg が NULL か、または • p_cfg パラメータでリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できません。 • p_ctrl->p_transfer_pdc が NULL、または • a p_ctrl->p_transfer_pdc->api 関数呼び出しパラメータが無効です
タッチ パネルが設定されていません。	オープンが正常に呼び出されませんでした。

I：この API は PDC ドライバを閉じます。実行中のキャプチャは停止します。この関数は再入可能です。

8.26.5.3 関数のステップ

- すべての割り込みを無効化します。
- PDC ハードウェア リソースをロック解除します

8.26.6 R_PDC_CaptureStart

```
ssp_err_t  R_PDC_CaptureStart ( pdc_ctrl_t  *const p_ctrl ,  uint8_t *const p_buffer )
```

8.26.6.1 概要説明

キャプチャを開始します。割り込みを有効にします。captureStart を実装します。

8.26.6.2 詳細説明

PDC のデータを指定したバッファに転送するよう、転送インタフェースを設定します。過去に open API により設定したとおり、PDC 設定を構成します。PDC リセット動作に対し PIXCLK 入力はアクティブである必要があるため、これらの設定はここで構成されます。キャプチャが完了すると、open API 呼び出し中に登録されたコールバックが呼び出されます。

表 803: 戻り値

名前	説明
SSP_SUCCESS	キャプチャが正常に開始しました。
p_ctrl の可能性があります。	次のいずれかのパラメータが不正確です。p_cfg が NULL か、または • p_cfg パラメータでリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できません。 • p_ctrl->p_current_buffer が NULL です
タッチ パネルが設定されていません。	オープンが正常に呼び出されませんでした。
SSP_ERR_TIMEOUT	リセット動作がタイムアウトしました。

I：カメラ モジュールにより PIXCLK が生成される場合、カメラは open の呼び出し前、および captureStart の呼び出し後に構成する必要があります。この関数は再入可能ではありません。

ユーザーは、p_buffer が指すメモリが有効であり、かつ完全な画像の保存に十分なサイズを備えていることを、責任をもって確認する必要があります。必要な空き容量数 （バイト単位）は、以下のように計算できます。

サイズ （バイト）= 画像幅 （ピクセル）* 画像高さ （ライン）* ピクセルあたりのバイト数

8.26.6.3 関数のステップ

- 転送インタフェースを構成します
- 割り込みを有効化 : データ レディ割り込み、アンダーラン割り込み、オーバーラン割り込み、フレーム終了割り込み、垂直ライン設定エラー割り込み、水平バイト数設定設定エラー割り込みを受信します

8.26.7 R_PDC_StateGet

```
ssp_err_t R_PDC_StateGet ( pdc_ctrl_t *const p_ctrl , pdc_state_t * p_state )
```

8.26.7.1 概要説明

VSYNC および HSYNC ピンの状態を返します。stateGet を実装します。

8.26.7.2 詳細説明

表 804: 戻り値

名前	説明
SSP_SUCCESS	状態が正常に読み取られました。
p_ctrl の可能性があります。	次のいずれかのパラメータが不正確です。p_cfg が NULL か、または • p_cfg パラメータでリクエストされたチャネルは、r_bsp_cfg.h で選択されたデバイスでは使用できません。 • p_state が NULL です
タッチ パネルが設定されていません。	オープンが正常に呼び出されませんでした。

I：この関数は再入可能です。

8.26.8 R_PDC_VersionGet

```
ssp_err_t  R_PDC_VersionGet ( ssp_version_t  *const p_data )
```

8.26.8.1 概要説明

PDC HAL ドライバのバージョンを返します。versionGet を実装します。

8.26.8.2 詳細説明

表 805:　戻り値

名前	説明
SSP_SUCCESS	バージョン情報を正常に読み取りました。
p_ctrl の可能性があります。	Null ポインタがパラメータとして渡されました

I：この関数は再入可能です。

8.26.9 モジュール

- ビルドタイム構成

8.26.9.1 ビルドタイム構成

定義

- #define PDC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define  BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.27 QSPI

クワッド シリアルペリフェラルインタフェース （QSPI）用ドライバ。

これは、SPI 互換インタフェースを備えたシリアル ROM（シリアル フラッシュ メモリ、シリアル EEPROM、シリアル FeRAM などの不揮発性メモリ）に接続するためのメモリ コントローラである クワッド SPI モジュール （QSPI）用のドライバです。

8.27.1 概要

クワッド SPI フラッシュインタフェース を拡張します。

8.27.2 Functions

- R_QSPI_Open
- R_QSPI_Close
- R_QSPI_Read
- R_QSPI_PageProgram
- R_QSPI_SectorErase
- R_QSPI_StatusGet
- R_QSPI_BankSelect
- R_QSPI_VersionGet

8.27.3 変数

- g_qspi_on_qspi

8.27.4 定義

- #define INLINE_ATTRIBUTE

  初期値 :
- #define QSPI_CODE_VERSION_MAJOR

  初期値 :(1)
- #define QSPI_CODE_VERSION_MINOR

  初期値 :(1)

- #define QSPI_CFG_PARAM_CHECKING_ENABLE

  初期値 :(#define BSP_CFG_PARAM_CHECKING_ENABLE)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.27.5 R_QSPI_Open

```
ssp_err_t R_QSPI_Open ( qspi_ctrl_t * p_ctrl , qspi_cfg_t const *const p_cfg )
```

8.27.5.1 概要説明

QSPI ドライバ モジュールを開きます。

8.27.5.2 詳細説明

SPI プロトコル経由でフラッシュ メモリを読み書きするために直接通信モードで QSPI モジュール ドライバを開きます。

表 806: 戻り値

名前	説明
SSP_SUCCESS	構成が正常に行われました。
p_ctrl の可能性があります。	p_cfg が NULL でした。

8.27.6 R_QSPI_Close

```
ssp_err_t R_QSPI_Close ( qspi_ctrl_t * p_ctrl )
```

8.27.6.1 概要説明

QSPI ドライバ モジュールを閉じます。

8.27.6.2 詳細説明

QSPI モジュールを ROM アクセス モードに戻します。

表 807: 戻り値

名前	説明
SSP_SUCCESS	構成が正常に行われました。
p_ctrl の可能性があります。	p_cfg が NULL でした。

8.27.7 R_QSPI_Read

```
ssp_err_t R_QSPI_Read ( qspi_ctrl_t * p_ctrl , uint8_t * p_device_address , uint8_t
* p_memory_address , uint32_t byte_count )
```

8.27.7.1 概要説明

フラッシュからデータを読み取ります。

8.27.7.2 詳細説明

SPI フラッシュ デバイス上の特定のアドレスからデータのブロックを読み取ります。

表 808: 戻り値

名前	説明
SSP_SUCCESS	フラッシュが正常にプログラムされました。
未サポートまたは不正確なモードです。	QSPI コントローラは直接通信モードになっていません。

8.27.8 R_QSPI_PageProgram

```
ssp_err_t R_QSPI_PageProgram ( qspi_ctrl_t * p_ctrl , uint8_t * p_device_address , uint8_t
* p_memory_address , uint32_t byte_count )
```

8.27.8.1 概要説明

フラッシュ デバイスにページ単位でデータをプログラミングします。

8.27.8.2 詳細説明

表 809: 戻り値

名前	説明
SSP_SUCCESS	フラッシュが正常にプログラムされました。
未サポートまたは不正確なモードです。	QSPI コントローラは直接通信モードになっていません。

8.27.9 R_QSPI_SectorErase

ssp_err_t R_QSPI_SectorErase (qspi_ctrl_t * p_ctrl , uint8_t * p_device_address)

8.27.9.1 概要説明

フラッシュ上のセクターを消去します。

8.27.9.2 詳細説明

SPI フラッシュ デバイス上の 1 つのセクターを消去します。消去するセクター内のアドレスで渡されたすべてが受け入れられます。

表 810: 戻り値

名前	説明
SSP_SUCCESS	フラッシュのセクターを消去するコマンドが正常に実行されました。
未サポートまたは不正確なモードです。	QSPI コントローラは直接通信モードになっていません。

8.27.10 R_QSPI_StatusGet

ssp_err_t R_QSPI_StatusGet (qspi_ctrl_t * p_ctrl , bool * p_write_in_progress)

8.27.10.1 概要説明

フラッシュデバイスの書き込みまたは消去のステータスを取得します。

8.27.10.2 詳細説明

フラッシュの書き込みステータスを返します。これは、消去が完了したかどうかを判断するのに最適な方法です。

表 811: 戻り値

名前	説明
SSP_SUCCESS	書き込みステータスは正確です。
未サポートまたは不正確なモードです。	QSPI コントローラは直接通信モードになっていません。

8.27.11 R_QSPI_BankSelect

```
ssp_err_t  R_QSPI_BankSelect (  uint32_t bank )
```

8.27.11.1 概要説明

アクセスするバンクを選択します。

8.27.11.2 詳細説明

バンクは、フラッシュ メモリ空間への 64MB スライディング アクセス ウィンドウです。この関数は現在のバンクをセットします。

表 812: 戻り値

名前	説明
SSP_SUCCESS	バンクが正常に選択されました。

8.27.12 R_QSPI_VersionGet

```
ssp_err_t  R_QSPI_VersionGet ( ssp_version_t  *const p_version )
```

8.27.12.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

8.27.12.2 詳細説明

表 813: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
SSP_ERR_INVALID_PTR	p_version が NULL です。

8.27.13 g_qspi_on_qspi

qspi_api_t::g_qspi_on_qspi

8.27.13.1 次のように初期化されます

```
g_qspi_on_qspi=
{
    .open          = R_QSPI_Open,
    .close         = R_QSPI_Close,
    .read          = R_QSPI_Read,
    .pageProgram   = R_QSPI_PageProgram,
    .sectorErase   = R_QSPI_SectorErase,
    .statusGet     = R_QSPI_StatusGet,
    .bankSelect    = R_QSPI_BankSelect,
    .versionGet    =R_QSPI_VersionGet
}
```

8.28 IIC

I2C バス インタフェース （IIC）用ドライバ。

このモジュールは、Renesas Inter-Integrated Circuit （IIC）ペリフェラルをサポートします。次のインタフェースを実装します。

- I2C インタフェース r_i2c_api.h

8.28.1 Functions

- R_RIIC_MasterVersionGet
- R_RIIC_MasterOpen
- R_RIIC_MasterClose
- R_RIIC_MasterRead
- R_RIIC_MasterWrite
- R_RIIC_MasterReset

8.28.2 定義

- #define RIIC_MASTER_CODE_VERSION_MAJOR

  初期値 :(1)

- #define RIIC_MASTER_CODE_VERSION_MINOR

  初期値 :(1)

- #define RIIC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.28.3 R_RIIC_MasterVersionGet

`ssp_err_t` R_RIIC_MasterVersionGet (`ssp_version_t` *const p_version)

8.28.3.1 概要説明

バージョン情報を取得して、それを指定されたバージョン構造体に保存します。

8.28.3.2 詳細説明

表 814: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に取得されました。
Null ポインタが指定されました。	p_version が NULL です。

8.28.4 R_RIIC_MasterOpen

`ssp_err_t` R_RIIC_MasterOpen (`i2c_ctrl_t` *const p_ctrl , `i2c_cfg_t` const *const p_cfg)

8.28.4.1 概要説明

I2C デバイスを開きます。IIC ペリフェラルの電源をオンにして、ユーザーズマニュアルに記載された初期化を実行する場合があります。

8.28.4.2 詳細説明

以前よりも低いレートに設定されたデバイスが開かれた場合に、この関数がペリフェラルのクロックを再設定します。

表 815: 戻り値

名前	説明
SSP_SUCCESS	要求されたクロック レートが正しくセットされました。すでに開いているインスタンスの同じ設定を開きました。
チャネルは現在動作中です。	すでに開いているデバイス インスタンスを開こうとしました。
SSP_ERR_INVALID_RATE	要求されたレートを設定できません。

8.28.5 R_RIIC_MasterClose

`ssp_err_t` R_RIIC_MasterClose (`i2c_ctrl_t` *const p_ctrl)

8.28.5.1 概要説明

I2C デバイスを閉じます。IIC ペリフェラルの電源をオフにする場合があります。

8.28.5.2 詳細説明

この関数は、デバイスで進行中の I2C 転送を安全に終了します。転送が中止された場合は、中止イベントに伴うコールバックを介してユーザーに通知されます。コールバックがオプションのため、この関数は、この状況で特定のエラー コードも返します。

I：デバイスは複数のユーザーによって開かれる可能性があるため、転送が中止されるのは、デバイスのユーザーが 1 人しか存在しない場合だけです。

表 816: 戻り値

名前	説明
SSP_SUCCESS	デバイスが問題なく閉じられました。
SSP_ERR_ABORTED	転送が進行中にデバイスが閉じられました。

8.28.6 R_RIIC_MasterRead

`ssp_err_t` R_RIIC_MasterRead (`i2c_ctrl_t` *const p_ctrl , uint8_t *const p_dest , uint32_t const bytes , bool const restart)

8.28.6.1 概要説明

I2C デバイスからの読み取りを実行します。

8.28.6.2 詳細説明

この関数は、関連するチャネル上で I2C 転送が進行中の場合は失敗します。それ以外の場合は、I2C 読み取り操作が開始されます。ユーザーからコールバックが提供されない場合は、この関数がブロック化読み取りを実行します。それ以外の場合は、読み取り操作が非ブロック化になり、操作の完了時点でコールバック内の SF_EVENT_RX_COMPLETE を通して呼び出し元に通知されます。

表 817: 戻り値

名前	説明
SSP_SUCCESS	コールバックが提供されなかった場合は、プロセスが開始されています。
チャネルは現在動作中です。	別の転送が進行中でした。
SSP_ERR_ABORTED	転送が失敗しました。

8.28.7 R_RIIC_MasterWrite

ssp_err_t R_RIIC_MasterWrite (i2c_ctrl_t *const p_ctrl , uint8_t *const p_src , uint32_t const bytes , bool const restart)

8.28.7.1 概要説明

I2C デバイスへの書き込みを実行します。

8.28.7.2 詳細説明

この関数は、関連するチャネル上で I2C 転送が進行中の場合は失敗します。それ以外の場合は、I2C 書き込み操作が開始されます。ユーザーからコールバックが提供されない場合は、この関数がブロック化書き込みを実行します。それ以外の場合は、書き込み操作が非ブロック化になり、操作の完了時点でコールバック内の SF_EVENT_TX_COMPLETE を通して呼び出し元に通知されます。

表 818: 戻り値

名前	説明
SSP_SUCCESS	コールバックが提供されなかった場合は、プロセスが開始されています。
チャネルは現在動作中です。	別の転送が進行中でした。
SSP_ERR_ABORTED	転送が失敗しました。

8.28.8 R_RIIC_MasterReset

ssp_err_t R_RIIC_MasterReset (i2c_ctrl_t *const p_ctrl)

8.28.8.1 概要説明

進行中の転送を中止して、IIC ペリフェラルを強制的に準備完了状態にします。

8.28.8.2 詳細説明

この関数は、デバイスで進行中の I^2C 転送を安全に終了します。転送が中止された場合は、中止イベントに伴うコールバックを介してユーザーに通知されます。コールバックがオプションのため、この関数は、この状況で特定のエラー コードも返します。

表 819: 戻り値

名前	説明
SSP_SUCCESS	チャネルは問題なくリセットされました。
SSP_ERR_ABORTED	ハードウェアのリセット中に転送が中止されました。

8.29 SPI

シリアルペリフェラルインタフェース （SPI）のドライバ。

このモジュールは、SPI モジュール用の SPI シリアル通信をサポートします。SPI インタフェースは r_spi_api.h で定義されます

8.29.1 Functions

- R_RSPI_Open
- R_RSPI_Read
- R_RSPI_Write
- R_RSPI_WriteRead
- R_RSPI_Close
- R_RSPI_VersionGet

8.29.2 定義

- #define RSPI_API_VERSION_MAJOR

  初期値 :(1)
- #define RSPI_API_VERSION_MINOR

  初期値 :(0)
- #define RSPI_CODE_VERSION_MAJOR

  初期値 :(1)
- #define RSPI_CODE_VERSION_MINOR

  初期値 :(1)
- #define RSPI_NUM_CHANNELS

  初期値 :(02)

8.29.3 R_RSPI_Open

```
ssp_err_t R_RSPI_Open ( spi_ctrl_t * p_ctrl , spi_cfg_t const *const p_cfg )
```

8.29.3.1 概要説明

この関数は、SPI 通信モード用のチャネルを初期化します。

8.29.3.2 詳細説明

open を実装します この関数は、次のタスクを実行します。

パラメータ チェックを実行して、エラー状態を処理します。

SPI チャネルに電源を供給します。

割り込みを無効にします。

一部のデフォルト値とユーザー設定可能オプションを使用して関連するレジスタを初期化します。

他の API 関数で使用するチャネル制御を提供します。

必要に応じて、ユーザー設定可能ファイルを更新します。

表 820: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に初期化されました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
チャネルは現在動作中です。	チャネルは現在操作中です。先にチャネルを閉じてください。
Null ポインタが指定されました。	p_ctrl ポインタまたは p_ctrl ポインタが NULL です。
SSP_ERR_INVALID_POINTER	r_spi_cfg_t 構造体の要素に無効な値が含まれています。
SSP_ERR_INVALID_POINTER	r_spi_cfg_t 構造体の要素に無効な値が含まれています。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。

I：この関数は再入可能です。

8.29.4 R_RSPI_Read

```
ssp_err_t  R_RSPI_Read ( spi_ctrl_t  *const p_ctrl ,  void const * p_dest ,  uint32_t const length ,
 spi_bit_width_t  const bit_width )
```

8.29.4.1 概要説明

この関数は、SPI デバイスからデータを受信します。

8.29.4.2 詳細説明

read を実装します この関数は、次のタスクを実行します。

パラメータ チェックを実行して、エラー状態を処理します。

割り込みを無効にします。

SPI バスを無効にします。

ユーザー要求ごとのデータ ビット幅をセットアップします。

SPI バスを有効にします。

割り込みを有効にします。

送信バッファ エンプティ割り込みを介してダミー データを含むデータ伝送を開始します。

送信するデータをソース バッファから SPI データ レジスタにコピーします。

受信バッファ フル割り込みの発生からデータを受信し、そのデータを宛先のバッファにコピーします。

受信バッファ フル割り込み経由のデータ受信とダミー データの送信を終了します。

表 821: 戻り値

名前	説明
SSP_SUCCESS	読み取り操作が正常に終了しました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.29.5 R_RSPI_Write

```
ssp_err_t R_RSPI_Write ( spi_ctrl_t *const p_ctrl , void const * p_src , uint32_t const length ,
 spi_bit_width_t const bit_width )
```

8.29.5.1 概要説明

この関数は、SPI デバイスにデータを送信します。

8.29.5.2 詳細説明

write を実装します この関数は、次のタスクを実行します。

パラメータ チェックを実行して、エラー状態を処理します。

割り込みを無効にします。

SPI バスを無効にします。

ユーザー要求ごとのデータ ビット幅をセットアップします。

SPI バスを有効にします。

割り込みを有効にします。

送信バッファ エンプティ割り込みを介してダミー データを含むデータ伝送を開始します。

送信するデータをソース バッファから SPI データ レジスタにコピーします。

受信バッファ フル割り込みの発生からデータを受信して、受信したデータに対して何もしません。

受信バッファ フル割り込み経由のデータ送信を終了します。

表 822: 戻り値

名前	説明
SSP_SUCCESS	書き込み操作が正常に終了しました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。

表 822: 戻り値 (続き)

名前	説明
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.29.6 R_RSPI_WriteRead

ssp_err_t R_RSPI_WriteRead (spi_ctrl_t *const p_ctrl , void const * p_src , void const * p_dest , uint32_t const length , spi_bit_width_t const bit_width)

8.29.6.1 概要説明

この関数は、SPI デバイスからデータを受信しながら、SPI デバイスにデータを送信します （全二重)。

8.29.6.2 詳細説明

spi_api_t::writeread を実装します。この関数は、次のタスクを実行します。

パラメータ チェックを実行して、エラー状態を処理します。

割り込みを無効にします。

SPI バスを無効にします。

ユーザー要求ごとのデータ ビット幅をセットアップします。

SPI バスを有効にします。

割り込みを有効にします。

送信バッファ エンプティ割り込みを使用してデータ伝送を開始します。

送信するデータをソース バッファから SPI データ レジスタにコピーします。

受信バッファ フル割り込みの発生からデータを受信し、そのデータを宛先のバッファにコピーします。

受信バッファ フル割り込み経由のデータの送受信を終了します。

表 823: 戻り値

名前	説明
SSP_SUCCESS	書き込み操作が正常に終了しました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.29.7 R_RSPI_Close

```
ssp_err_t R_RSPI_Close ( spi_ctrl_t *const p_ctrl )
```

8.29.7.1 概要説明

この関数は、次のタスクによってチャネルの閉鎖を管理します。

8.29.7.2 詳細説明

close を実装します SPI バスを無効化することによって、SPI 操作を無効化します。

チャネルの電源をオフにします。

すべての関連する割り込みを無効にします。

チャネル ステータスを更新します。

表 824: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.29.8 R_RSPI_VersionGet

```
ssp_err_t R_RSPI_VersionGet ( ssp_version_t * p_version )
```

8.29.8.1 概要説明

この関数は、基礎となるドライバのバージョン情報を取得します。

8.29.8.2 詳細説明

spi_api_t::versionget を実装します

表 825: 戻り値

名前	説明
void	

I：この関数は再入可能です。

8.29.9 API データ

8.29.9.1 rspi_spcmd_bit_length_t

rspi_spcmd_bit_length_t

詳細説明

フレーム データ長

列挙値

名前	説明
RSPI_SPCMD_BIT_LENGTH_8	0100 ～ 0111 = 8 ビット データ長
RSPI_SPCMD_BIT_LENGTH_16	1111 = 16 ビット データ長
RSPI_SPCMD_BIT_LENGTH_32	0011 = 32 ビット データ長

8.29.9.2 rspi_operation_t

rspi_operation_t

詳細説明

SPCR （RSPI 制御レジスタ）SPMS （RSPI モード）選択

列挙値

名前	説明
RSPI_OPERATION_SPI	SPI 動作 （4 線方式）
RSPI_OPERATION_CLK_SYN	クロック同期動作 （3 線方式）

8.29.9.3 rspi_communication_t

rspi_communication_t

詳細説明

SPCR （RSPI 制御レジスタ）TXMD （通信動作モード）選択

列挙値

名前	説明
RSPI_COMMUNICATION_FULL_DUPLEX	全二重同期シリアル通信
RSPI_COMMUNICATION_TRANSMIT_ONLY	送信のみシリアル通信

8.29.9.4 rspi_sslp_t

rspi_sslp_t

詳細説明

SSLP （RSPI スレーブ選択極性レジスタ）選択の定義

列挙値

名前	説明
RSPI_SSLP_LOW	SSLP 信号極性アクティブ Low
RSPI_SSLP_HIGH	SSLP 信号極性アクティブ High

8.29.9.5 rspi_loopback1_t

rspi_loopback1_t

詳細説明

SPPCR （RSPI ピン制御レジスタ）Loopback1 選択

列挙値

名前	説明
RSPI_LOOPBACK1_NORMAL_DATA	Loopback1 は、ノーマル モード
RSPI_LOOPBACK1_INVERTED_DATA	Loopback1 は、反転データを使用

8.29.9.6 rspi_loopback2_t

rspi_loopback2_t

詳細説明

SPPCR （RSPI ピン制御レジスタ）Loopback2 選択

列挙値

名前	説明
RSPI_LOOPBACK2_NORMAL_DATA	Loopback2 は、ノーマル モード
RSPI_LOOPBACK2_NOT_INVERTED_DATA	Loopback2 は、非反転データを使用

8.29.9.7 rspi_mosi_idle_fixed_val_t

rspi_mosi_idle_fixed_val_t

詳細説明

SPPCR （RSPI ピン制御レジスタ）MOIFV 選択

列挙値

名前	説明
RSPI_MOSI_IDLE_FIXED_VAL_LOW	MOSI のアイドリングの間 MOSIn レベル Low
RSPI_MOSI_IDLE_FIXED_VAL_HIGH	MOSI のアイドリングの間 MOSIn レベル High

8.29.9.8 rspi_mosi_idle_val_fixing_t

rspi_mosi_idle_val_fixing_t

詳細説明

SPPCR （RSPI ピン制御レジスタ）MOIFE （MOSI アイドル値修正）選択

列挙値

名前	説明
RSPI_MOSI_IDLE_VAL_FIXING_ENABLE	MOSI 出力値 = 以前の転送で得た最終データ
RSPI_MOSI_IDLE_VAL_FIXING_DISABLE	MOSI 出力値 = MOIFV ビットに設定されている値

8.29.9.9 rspi_parity_state_t

rspi_parity_state_t

詳細説明

SPCR2 （RSPI 制御レジスタ 2）パリティ有効化選択

列挙値

名前	説明
RSPI_PARITY_STATE_DISABLE	パリティを無効化します
RSPI_PARITY_STATE_ENABLE	パリティを有効化します

8.29.9.10 rspi_parity_mode_t

rspi_parity_mode_t

詳細説明

SPCR2 （RSPI 制御レジスタ 2）パリティ選択

列挙値

名前	説明
RSPI_PARITY_MODE_ODD	偶数パリティを選択します
RSPI_PARITY_MODE_EVEN	奇数パリティを選択します

8.29.9.11 rspi_ssl_select_t

rspi_ssl_select_t

詳細説明

SPCMD （RSPI コマンド）レジスタ SSL 信号アサーション選択

列挙値

名前	説明
RSPI_SSL_SELECT_SSL0	SSL0 をスレーブとして選択します

名前	説明
RSPI_SSL_SELECT_SSL1	SSL1 をスレーブとして選択します
RSPI_SSL_SELECT_SSL2	SSL2 をスレーブとして選択します
RSPI_SSL_SELECT_SSL3	SSL3 をスレーブとして選択します

8.29.9.12 rspi_ssl_level_keep_t

rspi_ssl_level_keep_t

詳細説明

SPCMD （RSPI コマンド）レジスタ SSL 信号レベル維持選択

列挙値

名前	説明
RSPI_SSL_LEVEL_KEEP_NOT	転送完了時に、すべての SSL 信号をネゲートします
RSPI_SSL_LEVEL_KEEP	転送完了時に、SSL レベルを維持します

8.29.9.13 rspi_clock_delay_count_t

rspi_clock_delay_count_t

詳細説明

SPCKD （RSPI クロック遅延）レジスタ クロック遅延カウント選択

列挙値

名前	説明
RSPI_CLOCK_DELAY_COUNT_1	RSPCK クロック遅延 1 RSPCK に設定します
RSPI_CLOCK_DELAY_COUNT_2	RSPCK クロック遅延 2 RSPCK に設定します
RSPI_CLOCK_DELAY_COUNT_3	RSPCK クロック遅延 3 RSPCK に設定します
RSPI_CLOCK_DELAY_COUNT_4	RSPCK クロック遅延 4 RSPCK に設定します
RSPI_CLOCK_DELAY_COUNT_5	RSPCK クロック遅延 5 RSPCK に設定します

名前	説明
RSPI_CLOCK_DELAY_COUNT_6	RSPCK クロック遅延 6 RSPCK に設定します
RSPI_CLOCK_DELAY_COUNT_7	RSPCK クロック遅延 7 RSPCK に設定します
RSPI_CLOCK_DELAY_COUNT_8	RSPCK クロック遅延 8 RSPCK に設定します

8.29.9.14 rspi_clock_delay_state_t

rspi_clock_delay_state_t

詳細説明

SPCMD （RSPI コマンド）レジスタ RSPCK 遅延有効化 / 無効化選択 SCKDEN

列挙値

名前	説明
RSPI_CLOCK_DELAY_STATE_DISABLE	RSPCK 遅延 = 1 RSPCK
RSPI_CLOCK_DELAY_STATE_ENABLE	RSPCK 遅延 = SPCKD レジスタ設定

8.29.9.15 rspi_ssl_negation_delay_count_t

rspi_ssl_negation_delay_count_t

詳細説明

SSLND （RSPI スレーブ選択ネゲート遅延）レジスタ スレーブ選択ネゲート遅延カウント選択

列挙値

名前	説明
RSPI_SSL_NEGATION_DELAY_1	SSL ネゲート遅延を 1 RSPCK に設定します
RSPI_SSL_NEGATION_DELAY_2	SSL ネゲート遅延を 2 RSPCK に設定します
RSPI_SSL_NEGATION_DELAY_3	SSL ネゲート遅延を 3 RSPCK に設定します
RSPI_SSL_NEGATION_DELAY_4	SSL ネゲート遅延を 4 RSPCK に設定します
RSPI_SSL_NEGATION_DELAY_5	SSL ネゲート遅延を 5 RSPCK に設定します

名前	説明
RSPI_SSL_NEGATION_DELAY_6	SSL ネゲート遅延を 6 RSPCK に設定します
RSPI_SSL_NEGATION_DELAY_7	SSL ネゲート遅延を 7 RSPCK に設定します
RSPI_SSL_NEGATION_DELAY_8	SSL ネゲート遅延を 8 RSPCK に設定します

8.29.9.16 rspi_ssl_negation_delay_state_t

rspi_ssl_negation_delay_state_t

詳細説明

SPCMD （RSPI コマンド）レジスタ SSL ネゲート遅延選択 SLNDEN

列挙値

名前	説明
RSPI_SSL_NEGATION_DELAY_DISABLE	SSL ネゲート遅延 = 1 RSPCK
RSPI_SSL_NEGATION_DELAY_ENABLE	SSL ネゲート遅延 = SSLND レジスタ設定

8.29.9.17 rspi_next_access_delay_count_t

rspi_next_access_delay_count_t

詳細説明

SPND （RSPI 次アクセス遅延）レジスタ次アクセス遅延カウント選択

列挙値

名前	説明
RSPI_NEXT_ACCESS_DELAY_COUNT_1	次アクセス遅延を 1 RSPCK+2PCLK に設定します
RSPI_NEXT_ACCESS_DELAY_COUNT_2	次アクセス遅延を 2 RSPCK+2PCLK に設定します
RSPI_NEXT_ACCESS_DELAY_COUNT_3	次アクセス遅延を 3 RSPCK+2PCLK に設定します
RSPI_NEXT_ACCESS_DELAY_COUNT_4	次アクセス遅延を 4 RSPCK+2PCLK に設定します
RSPI_NEXT_ACCESS_DELAY_COUNT_5	次アクセス遅延を 5 RSPCK+2PCLK に設定します

名前	説明
RSPI_NEXT_ACCESS_DELAY_COUNT_6	次アクセス遅延を 6 RSPCK+2PCLK に設定します
RSPI_NEXT_ACCESS_DELAY_COUNT_7	次アクセス遅延を 7 RSPCK+2PCLK に設定します
RSPI_NEXT_ACCESS_DELAY_COUNT_8	次アクセス遅延を 8 RSPCK+2PCLK に設定します

8.29.9.18 rspi_next_access_delay_state_t

rspi_next_access_delay_state_t

詳細説明

SPCMD （RSPI コマンド）レジスタ次アクセス遅延選択 SPNDEN

列挙値

名前	説明
RSPI_NEXT_ACCESS_DELAY_STATE_DISABLE	次アクセス遅延 = 1 RSPCK + 2 PCLK
RSPI_NEXT_ACCESS_DELAY_STATE_ENABLE	次アクセス遅延 = SPND レジスタ設定

8.29.9.19 rspi_spcmd_br_div_t

rspi_spcmd_br_div_t

列挙値

名前	説明
RSPI_SPCMD_BR_DIV_1	
RSPI_SPCMD_BR_DIV_2	
RSPI_SPCMD_BR_DIV_4	
RSPI_SPCMD_BR_DIV_8	

8.29.9.20 rspi_spcmd_assert_ssl_t

rspi_spcmd_assert_ssl_t

詳細説明

転送動作でアサートされるスレーブ選択

列挙値

名前	説明
RSPI_SPCMD_ASSERT_SSL0	
RSPI_SPCMD_ASSERT_SSL1	
RSPI_SPCMD_ASSERT_SSL2	
RSPI_SPCMD_ASSERT_SSL3	

8.29.10 拡張

8.29.10.1 rspi_ssl_polarity_t

rspi_ssl_polarity_t

詳細説明

SSLP （RSPI スレーブ選択極性レジスタ）SSLnP 選択

変数

- rspi_sslp_t rspi_ssl2
- rspi_sslp_t rspi_ssl3
- rspi_sslp_t rspi_ssl0
- rspi_sslp_t rspi_ssl1

8.29.10.2 rspi_loopback_t

rspi_loopback_t

詳細説明

SPPCR （RSPI ピン制御レジスタ）ループバック選択

変数

- rspi_loopback1_t rspi_loopback1
- rspi_loopback2_t rspi_loopback2

8.29.10.3 rspi_mosi_idle_t

rspi_mosi_idle_t

詳細説明

SPPCR （RSPI ピン制御レジスタ）MOIFV （MOSI アイドル値）選択

変数

- rspi_mosi_idle_fixed_val_t rspi_mosi_idle_fixed_val
- rspi_mosi_idle_val_fixing_t rspi_mosi_idle_val_fixing

8.29.10.4 rspi_parity_t

rspi_parity_t

詳細説明

SPCR2 （RSPI 制御レジスタ 2）パリティ選択

変数

- rspi_parity_state_t rspi_parity
- rspi_parity_mode_t rspi_parity_mode

8.29.10.5 rspi_clock_delay_t

rspi_clock_delay_t

詳細説明

RSPI クロック遅延レジスタ （SPCKD）および SPCMD （RSPI コマンド）レジスタ クロック遅延状態（SCKDEN）を選択します

変数

- rspi_clock_delay_count_t rspi_clock_delay_count
- rspi_clock_delay_state_t rspi_clock_delay_state

8.29.10.6 rspi_ssl_negation_delay_t

rspi_ssl_negation_delay_t

詳細説明

SSL ネゲート遅延 （SSLND）および SPCMD レジスタ SSL ネゲート遅延状態 （SLNDEN）を選択します

変数

- rspi_ssl_negation_delay_count_t rspi_ssl_neg_delay_count
- rspi_ssl_negation_delay_state_t rspi_ssl_neg_delay_state

8.29.10.7 rspi_access_delay_t

rspi_access_delay_t

詳細説明

次アクセス遅延 （SPND）および SPCMD レジスタ次アクセス遅延状態 （SPNDEN）を選択します

変数

- rspi_next_access_delay_count_t rspi_next_access_delay_count
- rspi_next_access_delay_state_t rspi_next_access_delay_state

8.29.10.8 spi_on_rspi_cfg_t

spi_on_rspi_cfg_t

詳細説明

拡張 SPI インタフェース設定

変数

- rspi_operation_t rspi_clksyn
  動作モードを選択します （SPI またはクロック同期）
- rspi_communication_t rspi_comm
  通信方式を選択します （全二重または送信のみ）
- rspi_ssl_polarity_t ssl_polarity
  SSLn の信号極性を選択します
- rspi_loopback_t loopback
  loopback1 または loopback2 を選択します
- rspi_mosi_idle_t mosi_idle
  MOSI アイドル固定値および選択肢を選択します
- rspi_parity_t parity
  パリティ値を選択し、パリティ値の有効 / 無効を設定します

- rspi_ssl_select_t ssl_select

  使用するスレーブを選択します （0-SSL0、1-SSL1、2-SSL2、3-SSL3）
- rspi_ssl_level_keep_t ssl_level_keep

  転送完了後の SSL レベルを選択します （0- ネゲート、1- 維持）
- rspi_clock_delay_t clock_delay

  クロック遅延を 0 ～ 7 の範囲で選択します
- rspi_ssl_negation_delay_t ssl_neg_delay

  スレーブ起動ネゲート遅延を 0 ～ 7 の範囲で選択します
- rspi_access_delay_t access_delay

  次アクセス遅延を 0 ～ 7 の範囲で選択します

8.30 RTC

リアルタイム クロック （RTC）用ドライバ。

このモジュールは、リアルタイム クロック （RTC）をサポートします。次のインタフェースを実装します。

- RTC インタフェース

8.30.1 Functions

- R_RTC_Open
- R_RTC_Close
- R_RTC_CalendarTimeSet
- R_RTC_CalendarTimeGet
- R_RTC_CalendarAlarmSet
- R_RTC_CalendarAlarmGet
- R_RTC_CalendarCounterStart
- R_RTC_CalendarCounterStop
- R_RTC_IrqEnable
- R_RTC_IrqDisable
- R_RTC_PeriodicIrqRateSet
- R_RTC_InfoGet
- R_RTC_VersionGet

8.30.2 定義

- #define RTC_CODE_VERSION_MAJOR

  初期値 :(1)
- #define RTC_CODE_VERSION_MINOR

  初期値 :(1)

8.30.3 R_RTC_Open

```
ssp_err_t R_RTC_Open ( rtc_ctrl_t *const p_ctrl , rtc_cfg_t const *const p_cfg )
```

8.30.3.1 概要説明

RTC ドライバを開きます。

8.30.3.2 詳細説明

open を実装します。

RTC ドライバ モジュールを開いて設定します。設定には、クロック ソース、エラー修正、12/24 時間モード、および割り込みコールバック関数が含まれます。サブクロック オシレーターがクロック ソースの場合は、この関数で開始されます。

表 826: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、RTC が開始しました。
p_ctrl の可能性があります。	p_ctrl または p_cfg ポインタが無効です。

8.30.4 R_RTC_Close

```
ssp_err_t  R_RTC_Close ( rtc_ctrl_t  *const p_ctrl )
```

8.30.4.1 概要説明

RTC ドライバを閉じます。

8.30.4.2 詳細説明

次を実装します :close

表 827: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、RTC が開始しました。
p_ctrl の可能性があります。	p_cfg ポインタが無効です。

8.30.5 R_RTC_CalendarTimeSet

```
ssp_err_t  R_RTC_CalendarTimeSet ( rtc_ctrl_t  *const p_ctrl , rtc_time_t  * p_time ,
  bool clock_start )
```

8.30.5.1 概要説明

カレンダーの日付を設定します。

8.30.5.2 詳細説明

calendarTimeSet を実装します。

表 828: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、RTC が開始しました。
p_ctrl の可能性があります。	p_ctrl または p_time ポインタが無効です。

8.30.6 R_RTC_CalendarTimeGet

```
ssp_err_t R_RTC_CalendarTimeGet ( rtc_ctrl_t *const p_ctrl , rtc_time_t * p_time )
```

8.30.6.1 概要説明

カレンダーの時間を取得します。

8.30.6.2 詳細説明

次を実装します :calendarTimeGet

表 829: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、RTC が開始しました。
p_ctrl の可能性があります。	p_time ポインタが無効です。

8.30.7 R_RTC_CalendarAlarmSet

```
ssp_err_t R_RTC_CalendarAlarmSet ( rtc_ctrl_t *const p_ctrl , rtc_alarm_time_t * p_alarm ,
 bool interrupt_enable_flag )
```

8.30.7.1 概要説明

カレンダーのアラーム時間を設定します。

8.30.7.2 詳細説明

calendarAlarmSet を実装します。

I：カレンダー カウンタを実行しなければ、アラームをセットすることができません。

表 830: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、RTC が開始しました。
p_ctrl の可能性があります。	p_alarm ポインタが無効です。
未サポートまたは不正確なモードです。	カレンダー カウンタが実行していません。

8.30.8 R_RTC_CalendarAlarmGet

`ssp_err_t` R_RTC_CalendarAlarmGet (`rtc_ctrl_t` *const p_ctrl , `rtc_alarm_time_t` * p_alarm)

8.30.8.1 概要説明

カレンダーのアラーム時間を取得します。

8.30.8.2 詳細説明

次を実装します :calendarAlarmGet

表 831: 戻り値

名前	説明
SSP_SUCCESS	初期化が成功し、RTC が開始しました。
p_ctrl の可能性があります。	p_alarm ポインタが無効です。

8.30.9 R_RTC_CalendarCounterStart

ssp_err_t R_RTC_CalendarCounterStart (rtc_ctrl_t *const p_ctrl)

8.30.9.1 概要説明

カレンダー カウンタを開始します。

8.30.9.2 詳細説明

calendarCounterStart を実装します。

表 832: 戻り値

名前	説明
SSP_SUCCESS	カレンダー カウンタが開始しました。

8.30.10 R_RTC_CalendarCounterStop

ssp_err_t R_RTC_CalendarCounterStop (rtc_ctrl_t *const p_ctrl)

8.30.10.1 概要説明

カレンダー カウンタを停止します。

8.30.10.2 詳細説明

calendarCounterStop を実装します。

表 833: 戻り値

名前	説明
SSP_SUCCESS	カレンダー カウンタが停止しました。

8.30.11 R_RTC_IrqEnable

ssp_err_t R_RTC_IrqEnable (rtc_ctrl_t *const p_ctrl , rtc_event_t interrupt)

8.30.11.1 概要説明

アラーム 割り込みを有効にします。

8.30.11.2 詳細説明

rtc_api_t::interruptEnable を実装します。

表 834: 戻り値

名前	説明
SSP_SUCCESS	アラーム 割り込みが有効になりました。

8.30.12 R_RTC_IrqDisable

```
ssp_err_t  R_RTC_IrqDisable ( rtc_ctrl_t  *const p_ctrl , rtc_event_t  interrupt )
```

8.30.12.1 概要説明

アラーム 割り込みを無効にします。

8.30.12.2 詳細説明

rtc_api_t::interruptDisable を実装します。

表 835: 戻り値

名前	説明
SSP_SUCCESS	アラーム 割り込みが無効になりました。

8.30.13 R_RTC_PeriodicIrqRateSet

```
ssp_err_t  R_RTC_PeriodicIrqRateSet ( rtc_ctrl_t  *const p_ctrl ,
 rtc_periodic_irq_select_t  rate )
```

8.30.13.1 概要説明

周期割り込みレートをセットします。

8.30.13.2 詳細説明

rtc_api_t::periodicInterruptRateSet を実装します。

表 836: 戻り値

名前	説明
SSP_SUCCESS	周期割り込みレートが正常にセットされました。

8.30.14 R_RTC_InfoGet

`ssp_err_t` R_RTC_InfoGet (`rtc_ctrl_t` * p_ctrl , `rtc_info_t` * p_rtc_info)

8.30.14.1 概要説明

この関数は、ソース クロックなどのドライバに関する情報を返します。

8.30.14.2 詳細説明

次を実装します :infoGet

表 837: 戻り値

名前	説明
SSP_SUCCESS	成功。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。

8.30.15 R_RTC_VersionGet

`ssp_err_t` R_RTC_VersionGet (`ssp_version_t` * p_version)

8.30.15.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを取得します。

8.30.15.2 詳細説明

次を実装します :versionGet

表 838: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
SSP_ERR_INVALID_PTR	p_version パラメータは NULL です。

8.30.16 API データ

8.30.16.1 rtc_count_mode_t

rtc_count_mode_t

詳細説明

カウンティング モード

列挙値

名前	説明
RTC_CALENDAR_MODE	
RTC_BINARY_MODE	

8.30.17 モジュール

- ビルドタイム構成

8.30.17.1 ビルドタイム構成

定義

- #define RTC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータ チェック用のコードを含めるかどうかを指定します。
  BSP_CFG_PARAM_CHECKING_ENABLE にセットした場合はシステム デフォルト設定が使用されます。1 にセットした場合はパラメータ チェックが含まれます。0 はパラメータ チェックをコンパイルアウトします。

- #define RTC_CFG_CALCULATE_YDAY

  初期値 :(1)

8.31 SCI 共通

シリアル通信インタフェース （SCI）の共通機能のためのドライバ。

8.31.1 概要

このモジュールは、以下の SCI モードに共通です。

- UART - 調歩同期式モード。UART インタフェース は r_uart_api.h で定義されます。
- SSPI - 簡易 SPI モード。SPI インタフェース は r_spi_api.h で定義されます。
- SI2C - 簡易 I^2C モード。I2C インタフェース は r_i2c_api.h で定義されます

8.31.2 Functions

- sci_irq_status_clear

8.31.3 定義

- #define SCI_CH_MAX

  初期値 :(10)

  SCI 合計チャネル数 （S7G2）
- #define SCI_PHY_CH_MAX

  初期値 :(10)

  SCI 物理チャネルの最大数
- #define SCI_COMMON_ERROR_RETURN

  初期値 :`#define SSP_ERROR_RETURN`((a), (err), "sci_common", NULL)

8.31.4 sci_irq_status_clear

sci_irq_status_clear (uint32_t const channel)

8.31.4.1 詳細説明

すべての SCI チャネル用の TXI 割り込みルーチン

表 839: パラメータ

名前	方向	説明
none		

表 840: 戻り値

名前	説明
none	SCI チャネル用の RXI 割り込みルーチン。RXI ISR は受信割り込みサービス ルーチンです。この関数は TXI 割り込み発生時に実行されます。RXI 割り込みは、受信データが RDR に格納される場合 （FIFO 無効時）や、FRDRL レジスタに格納されるデータの数が FCRHL.RTRG に元々設定されていた値以上になる場合（FIFO 有効時）に、生成されます。

表 841: パラメータ

名前	方向	説明
none		

表 842: 戻り値

名前	説明
none	すべての SCI チャネル用の ERI 割り込みルーチン

表 843: パラメータ

名前	方向	説明
none		

表 844: 戻り値

名前	説明
none	SCI チャネル用の TEI 割り込みルーチン。

表 845: パラメータ

名前	方向	説明
none		

表 846: 戻り値

名前	説明
none	この関数は、特定のチャネルの送信 （TXI）、受信 （RXI）、エラー （ERI）IRQ ステータスをクリアします。

表 847: パラメータ

名前	方向	説明
channel	in	SCI モジュールのチャネル番号

表 848: 戻り値

名前	説明
void	

8.31.5 API データ

8.31.5.1 sci_mode_t

sci_mode_t

詳細説明

SCI 動作モード

列挙値

名前	説明
SCI_MODE_OFF	モード指定なし
SCI_MODE_ASYNC	UART モード
SCI_MODE_SYNC	クロック同期モード
SCI_MODE_SSPI	簡易 SPI モード
SCI_MODE_SIIC	簡易 I^2C モード
SCI_MODE_SMART	スマート カード インタフェース モード （このバージョンでは未サポート）
SCI_MODE_MAX	SCI モードの数

8.31.5.2 sci_clk_src_t

sci_clk_src_t

詳細説明

SCI クロック ソースの列挙

列挙値

名前	説明
SCI_CLK_SRC_INT	ボー生成に内部クロックを使用
SCI_CLK_SRC_EXT	ボー生成に外部クロックを使用
SCI_CLK_SRC_EXT8X	外部クロックの 8 倍のボーレートを使用
SCI_CLK_SRC_EXT16X	外部クロックの 16 倍のボーレートを使用

8.31.5.3 noise_cancel_lvl_t

noise_cancel_lvl_t

詳細説明

ノイズ フィルタ設定の定義

列挙値

名前	説明
NOISE_CANCEL_LVL1	ノイズ フィルタ レベル 1 （弱）
NOISE_CANCEL_LVL2	ノイズ フィルタ レベル 2
NOISE_CANCEL_LVL3	ノイズ フィルタ レベル 3
NOISE_CANCEL_LVL4	ノイズ フィルタ レベル 4 （強）

8.31.6 拡張

8.31.6.1 sci_ctrl_t

sci_ctrl_t

詳細説明

SCI チャネル制御ブロック < SCI チャネル制御 （ハンドル用）

変数

- sci_mode_t mode

  動作モード

- bool tx_busy

  送信ビジー フラグ

- void * p_context

  関数レベルのデバイス コンテキスト （例 :

- void(* p_extpin_ctrl)(uint32_t channel, uint32_t level)

  外部ピン制御

- bsp_lock_t resource_lock_tx

  送信用のリソース ロック

- bsp_lock_t resource_lock_rx

  受信用のリソース ロック

8.32 SCI 上の簡易 I²C

SCI 上の簡易 IIC 用のドライバ。

このモジュールは、I²C モードの SCI をサポートします。次のインタフェースを実装します。

- I2C インタフェース r_i2c_api.h

8.32.1 Functions

- R_SCI_SIIC_MasterVersionGet
- R_SCI_SIIC_MasterOpen
- R_SCI_SIIC_MasterClose
- R_SCI_SIIC_MasterRead
- R_SCI_SIIC_MasterWrite
- R_SCI_SIIC_MasterReset

8.32.2 定義

- #define SCI_SIIC_MASTER_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SCI_SIIC_MASTER_CODE_VERSION_MINOR

  初期値 :(1)

8.32.3 R_SCI_SIIC_MasterVersionGet

```
ssp_err_t R_SCI_SIIC_MasterVersionGet ( ssp_version_t *const p_version )
```

8.32.3.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

8.32.3.2 詳細説明

表 849: 戻り値

名前	説明
SSP_SUCCESS	バージョンが正常に取得されました。
Null ポインタが指定されました。	p_version が NULL です。

8.32.4 R_SCI_SIIC_MasterOpen

`ssp_err_t` R_SCI_SIIC_MasterOpen (`i2c_ctrl_t` *const p_ctrl , `i2c_cfg_t` const *const p_cfg)

8.32.4.1 概要説明

I²C デバイスを開きます。I²C ペリフェラルの電源をオンにして、ユーザーズマニュアルに記載された初期化を実行します。

8.32.4.2 詳細説明

以前よりも低いレートに設定されたデバイスが開かれた場合に、この関数がペリフェラルのクロックを再設定します。

表 850: 戻り値

名前	説明
SSP_SUCCESS	- 要求されたクロック レートが正しく設定されました。 • すでに開いているインスタンスの同じ設定を開きました。
SSP_ERR_APPROXIMATION	要求に近いクロック レートで実行する設定値が見つかりました。
SSP_ERR_CLAMPED	あらかじめ開かれているデバイスが低速であるため、クロック レートが制限されます。
SSP_ERR_ALREADY_OPEN	すでに開いているインスタンスと互換性のない設定を開こうとしました。
Null ポインタが指定されました。	p_ctrl or p_cfg が NULL です。
SSP_ERR_INVALID_CHANNEL	p_cfg で指定されたチャネルが不正です。

表 850: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_RATE	要求されたレートを設定できません。
データ構造体が割り当てられませんでした。	要求された機能は、まだサポートされていません。

8.32.5 R_SCI_SIIC_MasterClose

```
ssp_err_t  R_SCI_SIIC_MasterClose ( i2c_ctrl_t  *const p_ctrl )
```

8.32.5.1 概要説明

I²C デバイスを閉じます。I²C ペリフェラルの電源をオフにします。

8.32.5.2 詳細説明

この関数は、デバイスで進行中の I²C 転送を安全に終了します。転送が中止された場合は、中止イベントに伴うコールバックを介してユーザーに通知されます。コールバックがオプションのため、この関数は、この状況で特定のエラー コードも返します。

I : デバイスは複数のユーザーによって開かれる可能性があるため、転送が中止されるのは、デバイスのユーザーが 1 人しか存在しない場合だけです。

表 851: 戻り値

名前	説明
SSP_SUCCESS	デバイスが問題なく閉じられました。
Null ポインタが指定されました。	p_ctrl が NULL です。
SSP_ERR_ABORTED	転送の進行中にデバイスが閉じられました。
タッチ パネルが設定されていません。	デバイスが開かれていません。

8.32.6 R_SCI_SIIC_MasterRead

ssp_err_t R_SCI_SIIC_MasterRead (i2c_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const bytes , bool const restart)

8.32.6.1 概要説明

I2C デバイスからの読み取りを実行します。

8.32.6.2 詳細説明

この関数は、関連するチャネル上で I2C 転送が進行中の場合は失敗します。それ以外の場合は、I2C 読み取り操作が開始されます。ユーザーからコールバックが提供されない場合は、この関数がブロック化読み取りを実行します。それ以外の場合は、読み取り操作が非ブロック化になり、操作の完了時点でコールバック内の SF_EVENT_RX_COMPLETE を通して呼び出し元に通知されます。

表 852: 戻り値

名前	説明
SSP_SUCCESS	関数が問題なく実行されました。
Null ポインタが指定されました。	p_ctrl または p_dest が NULL です。
SSP_ERR_INVALID_POINTER	バイトが 0 です。
チャネルは現在動作中です。	別の転送が進行中でした。

8.32.7 R_SCI_SIIC_MasterWrite

ssp_err_t R_SCI_SIIC_MasterWrite (i2c_ctrl_t *const p_ctrl , uint8_t *const p_src , uint32_t const bytes , bool const restart)

8.32.7.1 概要説明

I2C デバイスへの書き込みを実行します。

8.32.7.2 詳細説明

この関数は、関連するチャネル上で I2C 転送が進行中の場合は失敗します。それ以外の場合は、I2C 書き込み操作が開始されます。ユーザーからコールバックが提供されない場合は、この関数がブロック化書き込みを実行します。それ以外の場合は、書き込み操作が非ブロック化になり、操作の完了時点でコールバック内の SF_EVENT_TX_COMPLETE を通して呼び出し元に通知されます。

表 853: 戻り値

名前	説明
SSP_SUCCESS	関数が問題なく実行されました。
Null ポインタが指定されました。	p_ctrl または p_src が NULL です。
SSP_ERR_INVALID_POINTER	バイトが 0 です。
チャネルは現在動作中です。	別の転送が進行中でした。

8.32.8 R_SCI_SIIC_MasterReset

```
ssp_err_t R_SCI_SIIC_MasterReset ( i2c_ctrl_t *const p_ctrl )
```

8.32.8.1 概要説明

進行中の転送を中止し、I²C ペリフェラルを強制的にレディ状態にします。

8.32.8.2 詳細説明

表 854: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	I²C 制御構造体へのポインタ。この関数は、デバイスで進行中の I²C 転送を安全に終了します。転送が中止された場合は、中止イベントに伴うコールバックを介してユーザーに通知されます。コールバックがオプションのため、この関数は、この状況で特定のエラー コードも返します。

表 855: 戻り値

名前	説明
SSP_SUCCESS	チャネルは問題なくリセットされました。

表 855: 戻り値 (続き)

名前	説明
Null ポインタが指定されました。	p_ctrl が NULL です。
SSP_ERR_ABORTED	ハードウェアのリセット中に転送が中止されました。
タッチ パネルが設定されていません。	デバイスが開かれていません。

8.32.9 モジュール

- ビルド時間設定

8.32.9.1 ビルド時間設定

定義

- #define SCI_SIIC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

- #define SCI_SIIC_CFG_PREREQUISITE_CHECKING_ENABLE

  初期値 :(BSP_CFG_PREREQUISITE_CHECKING_ENABLE)

  API 前提条件チェック用のコードを含めるかどうかを指定します。有効な設定値には BSP_CFG_PREREQUISITE_CHECKING_ENABLE が含まれます。bsp_cfg.h1 からのシステム デフォルト設定を使用します。前提条件 checking0 を含みます。前提条件チェックをコンパイルアウトします。

8.33 SCI 上の簡易 SPI

SCI 上の簡易 SPI 用のドライバ。

このモジュールは、マイクロコントローラの SCI ペリフェラルを使用するサンプル SPI シリアル通信をサポートします。このインタフェースは r_spi_api.h で定義されます。このモジュールは、SPI インタフェースを実装します。

8.33.1 Functions

- R_SCI_SSPI_Open
- R_SCI_SSPI_Read
- R_SCI_SSPI_Write
- R_SCI_SSPI_WriteRead
- R_SCI_SSPI_Close
- R_SCI_SSPI_VersionGet

8.33.2 変数

- g_spi_on_sci

8.33.3 定義

- #define SCI_SSPI_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SCI_SSPI_CODE_VERSION_MINOR

  初期値 :(1)

8.33.4 R_SCI_SSPI_Open

```
ssp_err_t R_SCI_SSPI_Open ( spi_ctrl_t * p_ctrl , spi_cfg_t const *const p_cfg )
```

8.33.4.1 概要説明

SPI 通信モードのチャネルを初期化します。open を実装します この関数は、パラメータ チェックを実行して、エラー状態を処理します。SPI チャネルに電源を供給します。割り込みを無効にします。デフォルト値とユーザー設定可能オプションを使用して関連するレジスタを初期化します。他の API 関数で使用するチャネル処理を提供します。必要に応じて、ユーザー設定可能ファイルを更新します。

8.33.4.2 詳細説明

表 856: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に初期化されました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
チャネルは現在動作中です。	先にチャネルを閉じてください。
Null ポインタが指定されました。	p_ctrl ポインタまたは p_ctrl ポインタが NULL です。
SSP_ERR_INVALID_POINTER	r_spi_cfg_t 構造体の要素に無効な値が含まれています。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。

I：この関数は再入可能です。

I :p_cfg 内のビットレート引数の範囲は、簡易 SPI の場合、PCLK=120 MHz で 2500 ～ 7.5m です。RSPI の場合は、BRDV が 0 に固定されて最大のビットレートが実現されます。範囲は、PCLK=120.0 MHz で 10.0 mbps ～ 30.0 mbps です

8.33.4.3 関数のステップ

- チャネルに電源を供給します
- 割り込みを無効にします

8.33.5 R_SCI_SSPI_Read

```
ssp_err_t  R_SCI_SSPI_Read ( spi_ctrl_t  *const p_ctrl ,  void const * p_dest ,  uint32_t const length ,
 spi_bit_width_t  const bit_width )
```

8.33.5.1 概要説明

SPI デバイスからデータを受信します。read を実装します。この関数は、次のタスクを実行します。

8.33.5.2 詳細説明

- パラメータ チェックを実行して、エラー状態を処理します。
- 割り込みを無効にします。
- ユーザー要求ごとのデータ ビット幅をセットアップします。
- トランスミッターを有効にします。
- レシーバーを有効にします。
- 割り込みを有効にします。
- 送信バッファ エンプティ割り込みを介してダミー データを含むデータ伝送を開始します。
- 送信するデータをソース バッファから SPI データ レジスタにコピーします。
- 受信バッファ フル割り込みの発生からデータを受信し、そのデータを宛先のバッファにコピーします。
- 受信バッファ フル割り込み経由のデータ受信とダミー データの送信を終了します。
- トランスミッターを無効にします。
- レシーバーを無効にします。
- 割り込みを無効にします。

表 857: 戻り値

名前	説明
SSP_SUCCESS	読み取り操作が正常に終了しました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.33.6 R_SCI_SSPI_Write

```
ssp_err_t  R_SCI_SSPI_Write ( spi_ctrl_t  *const p_ctrl ,  void const * p_src ,  uint32_t const length ,
 spi_bit_width_t  const bit_width )
```

8.33.6.1 概要説明

SPI デバイスにデータを送信します。write を実装します。

8.33.6.2 詳細説明

- この関数は、次のタスクを実行します。
- パラメータ チェックを実行して、エラー状態を処理します。
- 割り込みを無効にします。
- ユーザー要求ごとのデータ ビット幅をセットアップします。
- トランスミッターを有効にします。
- レシーバーを有効にします。
- 割り込みを有効にします。
- 送信バッファ エンプティ割り込みを介してデータを含むデータ伝送を開始します。
- 送信するデータをソース バッファから SPI データ レジスタにコピーします。
- 受信バッファ フル割り込みの発生からデータを受信して、受信したデータに対して何もしません。
- 受信バッファ フル割り込み経由のデータ送信を終了します。
- トランスミッターを無効にします。
- レシーバーを無効にします。
- 割り込みを無効にします。

表 858: 戻り値

名前	説明
SSP_SUCCESS	書き込み操作が正常に終了しました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
SSP_ERR_INVALID_ARGUMENT	2 ロックが取得できませんでした。チャネルはビジーです。

表 858: 戻り値 (続き)

名前	説明
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください

I：この関数は再入可能です。

8.33.7 R_SCI_SSPI_WriteRead

```
ssp_err_t R_SCI_SSPI_WriteRead ( spi_ctrl_t *const p_ctrl , void const * p_src , void const * p_dest , uint32_t const length , spi_bit_width_t const bit_width )
```

8.33.7.1 概要説明

SPI デバイスとの間で、データの送信と受信を同時に実行します （全二重通信）。writeRead を実装します。この関数は、次のタスクを実行します。

8.33.7.2 詳細説明

- パラメータ チェックを実行して、エラー状態を処理します。
- 割り込みを無効にします。ユーザー要求ごとのデータ ビット幅をセットアップします。トランスミッターを有効にします。レシーバーを有効にします。割り込みを有効にします。送信バッファ エンプティ割り込みを使用してデータ伝送を開始します。送信するデータをソース バッファから SPI データレジスタにコピーします。受信バッファ フル割り込みの発生からデータを受信し、そのデータを宛先のバッファにコピーします。受信バッファ フル割り込み経由のデータの送受信を終了します。トランスミッターを無効にします。レシーバーを無効にします。割り込みを無効にします。

表 859: 戻り値

名前	説明
SSP_SUCCESS	書き込み操作が正常に終了しました。
SSP_ERR_INVALID_POINTER	チャネル番号が無効です。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
SSP_ERR_INVALID_ARGUMENT	ロックが取得できませんでした。チャネルはビジーです。

表 859: 戻り値 (続き)

名前	説明
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.33.8 R_SCI_SSPI_Close

```
ssp_err_t R_SCI_SSPI_Close ( spi_ctrl_t *const p_ctrl )
```

8.33.8.1 概要説明

以下のタスクを実行することによって、チャネルのクローズを処理します。close を実装します チャネルの電源をオフにします。すべての関連する割り込みを無効にします。チャネル ステータスを更新します。

8.33.8.2 詳細説明

表 860: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました。
Null ポインタが指定されました。	必要なポインタ引数が NULL です。
ロックが取得できませんでした。	チャネルが開かれていません。先にチャネルを開いてください。

I：この関数は再入可能です。

8.33.8.3 関数のステップ

- < 割り込みを無効にします

- SCI デバイス コンテキストをクリアします
- < チャネルへの電源を遮断します

8.33.9 R_SCI_SSPI_VersionGet

`ssp_err_t` R_SCI_SSPI_VersionGet (`ssp_version_t` * p_version)

8.33.9.1 概要説明

基礎となるドライバのバージョン情報を取得します。versionGet を実装します。

8.33.9.2 詳細説明

表 861: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました。

I：この関数は再入可能です。

8.33.10 g_spi_on_sci

`spi_api_t`::g_spi_on_sci

8.33.10.1 詳細説明

このインスタンスのインタフェース API 構造体へのポインタの値を指定します。

8.33.11 モジュール

- ビルドタイム構成

8.33.11.1 ビルドタイム構成

定義

- #define SCI_SPI_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.34 SCI 上の UART

SCI 上の UART 用のドライバ。

8.34.1 概要

このモジュールは、SCI 上の UART をサポートします。これは UART インタフェースを実装し、SCI を全二重 UART 通信ポートとして駆動します。このモジュールは、すべての SCI チャネルと UART ポートとして駆動できます。

UART インタフェース を拡張します。

I：このモジュールは、16 ステージのハードウェア FIFO または DTC 転送実装のいずれかを使用して、複数バイトを書き込むことができます。

8.34.2 Functions

- R_SCI_UartOpen
- R_SCI_UartClose
- R_SCI_UartRead
- R_SCI_UartWrite
- R_SCI_UartBaudSet
- R_SCI_UartInfoGet
- R_SCI_UartVersionGet

8.34.3 定義

- #define SCI_UART_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SCI_UART_CODE_VERSION_MINOR

  初期値 :(1)

8.34.4 R_SCI_UartOpen

```
ssp_err_t R_SCI_UartOpen ( uart_ctrl_t *const p_ctrl , uart_cfg_t const *const p_cfg )
```

8.34.4.1 詳細説明

表 862: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に開かれました。
チャネルは現在動作中です。	チャネルはすでに使用中です。
p_ctrl の可能性があります。	UART 制御ブロックまたは設定構造体へのポインタが NULL です。
SSP_ERR_INVALID_ARGUMENT	チャネルがロックされています。
未サポートまたは不正確なモードです。	チャネルが非 UART モード用に使用されているか、設定されているモードが誤っています。
SSP_ERR_INVALID_POINTER	設定構造体に無効なパラメータ設定値が見つかりました。

I：この関数は再入可能です。

8.34.4.2 関数のステップ

- 指定された SCI チャネルをロックします
- チャネルに電源を供給します
- TX ピンのデフォルト値を 1 に設定します
- 割り込みを無効にします
- レシーバーを無効にします
- トランスミッターを無効にします
- FIFO 関連のレジスタを設定します
- RXD の立ち下がりエッジで受信を開始します
- RXD がローレベルになったときに受信を開始します

ノイズ キャンセルを有効にし、効果レベルを

最小限に固定します

- ノイズ キャンセルを無効にします
- UART 関連のレジスタを設定します
- デフォルト設定値を設定します
- チャネルへの電源を遮断します
- FIFO 関連のレジスタを設定します
- 受信データ レディが検出されたときに、RXI を選択します
- SCI HAL ドライバ内部の UART デバイス コンテキストを保存します
- 上位レイヤーからコールバック関数を登録します
- レシーバーを有効にします
- 受信割り込みを有効にします
- トランスミッターとその割り込みは、以下で有効になります : R_SCI_UartWrite

8.34.5 R_SCI_UartClose

`ssp_err_t` R_SCI_UartClose (`uart_ctrl_t` *const p_ctrl)

8.34.5.1 詳細説明

表 863: 戻り値

名前	説明
SSP_SUCCESS	チャネルが正常に閉じられました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。

I : この関数は再入可能です。

8.34.5.2 関数のステップ

- 転送インタフェースを閉じます。
- 制御ブロック パラメータをクリアします
- 関連する割り込みを無効にします

  レシーバーを無効にします

  トランスミッターを無効にします

  SCI チャネルへの電源を遮断します

  指定された SCI チャネルをロック解除します

  デバイス コンテキストをクリアします

8.34.6 R_SCI_UartRead

ssp_err_t R_SCI_UartRead (uart_ctrl_t *const p_ctrl , uint8_t const *const p_dest , uint32_t const bytes)

8.34.6.1 詳細説明

表 864: 戻り値

名前	説明
SSP_SUCCESS	データの読み取りが正常に終了しました。
SSP_ERR_INVALID_ARGUMENT	チャネルがロックされています。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
未サポートまたは不正確なモードです。	チャネルが非 UART モード用に使用されています。
SSP_ERR_INVALID_POINTER	宛先アドレスまたはデータ サイズが、データ長に対して無効です。

I：この関数は再入可能です。この API は、SCI_UART_CFG_RX_ENABLE が有効な場合のみ有効です。R_SCI_UartOpen 呼び出しで 9 ビット データ長が指定されている場合、宛先は 16 ビット境界にアラインする必要があります。

8.34.6.2 関数のステップ

- 何も行う必要はありません。
- データ バイト長を確認します
- SCI チャネルから読み取ります。
- 転送を行わない場合は、何も処理を行いません。バイトは、コールバックを介して届けられます。

8.34.7 R_SCI_UartWrite

`ssp_err_t` R_SCI_UartWrite (`uart_ctrl_t` *const p_ctrl , uint8_t const *const p_src , uint32_t const bytes)

8.34.7.1 詳細説明

表 865: 戻り値

名前	説明
SSP_SUCCESS	データの送信が正常に終了しました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
未サポートまたは不正確なモードです。	チャネルが非 UART モード用に使用されているか、ハンドルで設定されているモードが誤っています。
SSP_ERR_INVALID_POINTER	ソース アドレスまたはデータ サイズが、データ長に対して無効です。
SSP_ERR_INVALID_ARGUMENT	ハードウェアをロックできませんでした。

I：この関数は再入可能です。R_SCI_UartOpen 呼び出しで 9 ビット データ長が指定されている場合、ソースは 16 ビット境界にアラインする必要があります。

8.34.7.2 関数のステップ

- 送信ステータスを ON TRANSACTION に設定します

- 最後のバイトを除くすべてのバイトを転送する転送構成を行います。最後のバイトは、ISR で送信されて、送信終了 ISR を有効化します。これは、最後のバイトが DMAC または DTC で送信された場合には機能しません。
- ISR から送信する最後の文字を格納します。
- すべてのバイトは、ISR から送信されます。
- トランスミッターを有効にします

8.34.8 R_SCI_UartBaudSet

`ssp_err_t` R_SCI_UartBaudSet (`uart_ctrl_t` const * p_ctrl , uint32_t const baudrate)

8.34.8.1 詳細説明

表 866: 戻り値

名前	説明
SSP_SUCCESS	ボー レートが正常に変更されました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。
SSP_ERR_INVALID_POINTER	指定されたボー レート値が誤っています。
SSP_ERR_INVALID_ARGUMENT	ハードウェアをロックできませんでした。

I：この関数は再入可能です。この API ではクロック ソースを変更できません。

8.34.8.2 関数のステップ

- 割り込みを無効にします
- トランスミッターを無効にします。この API は送信を再開せず、終了します
- レシーバーを無効にします
- ボー レートに関連するレジスタを設定します
- レシーバーを有効にします
- 割り込みを有効にします

8.34.9 R_SCI_UartInfoGet

ssp_err_t R_SCI_UartInfoGet (uart_ctrl_t *const p_ctrl , uart_info_t *const p_info)

8.34.9.1 詳細説明

表 867: 戻り値

名前	説明
SSP_SUCCESS	ボー レートが正常に変更されました。
p_ctrl の可能性があります。	UART 制御ブロックへのポインタが NULL です。

I：この関数は再入可能です。

8.34.10 R_SCI_UartVersionGet

ssp_err_t R_SCI_UartVersionGet (ssp_version_t * p_version)

8.34.10.1 詳細説明

表 868: 戻り値

名前	説明
Version	番号

I：この関数は再入可能です。

8.34.11 拡張

8.34.11.1 uart_on_sci_cfg_t

uart_on_sci_cfg_t

詳細説明

SCI 上の UART デバイス設定

変数

- sci_clk_src_t clk_src

  SCI_CLK_SRC_INT/EXT8X/EXT16X を使用します

- bool baudclk_out

  ボー レート クロック出力を有効化します

- bool rx_edge_start

  立ち下がりエッジで受信を開始します

- bool noisecancel_en

  ノイズ キャンセルを有効化します

- void(* p_extpin_ctrl)(uint32_t channel, uint32_t level)

  RTS 信号として使用される外部 GPIO ピン制御に対するユーザー コールバック関数へのポインタ

8.34.12 モジュール

- ビルドタイム構成

8.34.12.1 ビルドタイム構成

定義

- #define SCI_UART_CFG_EXTERNAL_RTS_OPERATION

  初期値 :(1)

  GPIO ピンの 1 つを UART RTS ピンとして使用するかどうかを指定します。これは、GPIO のサポートを追加する UART の拡張機能です （1: GPIO を RTS ピンとして使用する、0: GPIO を RTS ピンとして使用しない）

- #define SCI_UART_CFG_RX_ENABLE

  初期値 :(1)

  受信をサポートするかどうかを指定します （1: 含める、0: 含めない）

- #define SCI_UART_CFG_TX_ENABLE

  初期値 :(1)

  送信をサポートするかどうかを指定します （1: 含める、0: 含めない）

- #define SCI_UART_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  API パラメータ チェック用のコードを含めるかどうかを指定します。
  BSP_CFG_PARAM_CHECKING_ENABLE に設定した場合は、システムのデフォルト設定が使用されます （1: パラメータ チェックを含める、0: パラメータ チェックをコンパイルアウトする）

8.35 SDMMC

SD/MMC ホスト インタフェース （SDHI）のドライバ。

eMMC を含む SD および MMC メモリ デバイスにアクセスするための SDMMC ドライバ。

8.35.1 Functions

- R_SDMMC_Open
- R_SDMMC_Close
- R_SDMMC_Read
- R_SDMMC_Write
- R_SDMMC_Control
- R_SDMMC_ReadIo
- R_SDMMC_WriteIo
- R_SDMMC_ReadIoExt
- R_SDMMC_WriteIoExt
- R_SDMMC_IoIntEnable
- R_SDMMC_VersionGet
- R_SDMMC_InfoGet
- R_SDMMC_Erase

8.35.2 定義

- #define SDMMC_CODE_VERSION_MAJOR

  初期値 :((uint8_t)1)
- #define SDMMC_CODE_VERSION_MINOR

  初期値 :((uint8_t)0)

8.35.3 R_SDMMC_Open

`ssp_err_t` R_SDMMC_Open (`sdmmc_ctrl_t` *const p_ctrl , `sdmmc_cfg_t` const *const p_cfg)

8.35.3.1 概要説明

読み書きおよび制御のためにデバイスを開きます。

8.35.3.2 詳細説明

Open 関数は、読み取り / 書き込みおよび制御のために SD または MMC デバイス ポートを開きます。この関数は、リセット以降に初めて呼び出されたときに SDMMC ドライバとハードウェアを初期化します。

表 869: 戻り値

名前	説明
SSP_SUCCESS	ポートが使用可能で、読み取り / 書き込み / 制御アクセス用に開いています。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	1 つまたは複数の構成オプションが無効です。
チャネルは現在動作中です。	指定したチャネルはすでに開かれています。構成は変更されていません。関連する Close 関数を呼び出すか、関連する Control コマンドを使用してチャネルを再構成します。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.35.3.3 関数のステップ

- < ハンドルを保存します。
- < 初期化済みでない場合、SD または eMMC を初期化します。
- < HAL ドライバ内部のデバイス コンテキストを保存します。
- < 上位レイヤーからコールバック関数を登録します。
- p_callback が選択されている状態で、割り込みを有効にし、p_callback を配列に格納して、ISR からアクセスできるようにします。

8.35.4 R_SDMMC_Close

```
ssp_err_t  R_SDMMC_Close ( sdmmc_ctrl_t  *const p_ctrl )
```

8.35.4.1 概要説明

開いているデバイス ポートを閉じます。

8.35.4.2 詳細説明

close 関数は、開いている SD/MMC デバイス ポートを閉じます。

表 870: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_ctrl パラメータは NULL です。
SSP_ERR_INVALID_POINTER	不正なチャネル番号。
タッチ パネルが設定されていません。	チャネルは開かれていません。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.35.5 R_SDMMC_Read

ssp_err_t R_SDMMC_Read (sdmmc_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const start_sector , uint32_t const sector_count)

8.35.5.1 概要説明

SD/MMC からデータを読み取ります。

8.35.5.2 詳細説明

Read 関数は、SD または MMC デバイス チャネルからデータを読み取ります。

表 871: 戻り値

名前	説明
SSP_SUCCESS	正常にデータが読み取られました。
p_ctrl の可能性があります。	NULL ポインタ。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.35.6 R_SDMMC_Write

ssp_err_t R_SDMMC_Write (sdmmc_ctrl_t *const p_ctrl , uint8_t const *const p_source , uint32_t const start_sector , uint32_t const sector_count)

8.35.6.1 概要説明

SDMMC チャネルにデータを書き込みます。

8.35.6.2 詳細説明

表 872: 戻り値

名前	説明
SSP_SUCCESS	カードの書き込みが正常に終了しました。
p_ctrl の可能性があります。	ハンドルまたはソース アドレスが NULL です。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。
SSP_ERR_WRITE_PROTECTED	SD または MMC カードが書き込み保護されています。

I：この関数は別のチャネルに対して再入可能です。

8.35.7 R_SDMMC_Control

ssp_err_t R_SDMMC_Control (sdmmc_ctrl_t *const p_ctrl , ssp_command_t const command , void * p_data)

8.35.7.1 概要説明

制御コマンドを送信して、SD/MMC チャネルのステータスを受信します。

8.35.7.2 詳細説明

Control 関数は、制御コマンドを送信して、SD/MMC チャネルのステータスを受信します。

表 873: 戻り値

名前	説明
SSP_SUCCESS	コマンドが正常に実行されました。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	コマンドが無効です。
SF_INFO_NOT_AVAILABLE	カードが取り外されているか、故障しているために情報が入手できない可能性があります。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.35.8 R_SDMMC_ReadIo

ssp_err_t R_SDMMC_ReadIo (sdmmc_ctrl_t *const p_ctrl , uint8_t *const p_data , uint32_t const function , uint32_t const address)

8.35.8.1 概要説明

SDIO からデータを読み取ります。

8.35.8.2 詳細説明

Read 関数は、SDIO チャネルからデータを読み取ります。

表 874: 戻り値

名前	説明
SSP_SUCCESS	正常にデータが読み取られました。
p_ctrl の可能性があります。	NULL ポインタ。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.35.9 R_SDMMC_WriteIo

ssp_err_t R_SDMMC_WriteIo (sdmmc_ctrl_t *const p_ctrl , uint8_t *const p_data , uint32_t const function , uint32_t const address , sdmmc_io_write_mode_t const read_after_write)

8.35.9.1 概要説明

SDIO にデータを書き込みます。

8.35.9.2 詳細説明

表 875: 戻り値

名前	説明
SSP_SUCCESS	カードの書き込みが正常に終了しました。
p_ctrl の可能性があります。	ハンドルまたはソース アドレスが NULL です。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはアドレス。

I：この関数は別のチャネルに対して再入可能です。

8.35.10 R_SDMMC_ReadIoExt

ssp_err_t R_SDMMC_ReadIoExt (sdmmc_ctrl_t *const p_ctrl , uint8_t *const p_dest , uint32_t const function , uint32_t const address , uint32_t *const count , sdmmc_io_transfer_mode_t transfer_mode , sdmmc_io_address_mode_t address_mode)

8.35.10.1 概要説明

SDIO からデータを読み取ります。

8.35.10.2 詳細説明

Read 関数は、SDIO チャネルからデータを読み取ります。

表 876: 戻り値

名前	説明
SSP_SUCCESS	正常にデータが読み取られました。
p_ctrl の可能性があります。	NULL ポインタ。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。

I：この関数は別のチャネルに対して再入可能です。同じチャネルに対しては再入可能ではありません。

8.35.11 R_SDMMC_WriteIoExt

ssp_err_t R_SDMMC_WriteIoExt (sdmmc_ctrl_t *const p_ctrl , uint8_t const *const p_source , uint32_t const function , uint32_t const address , uint32_t const count , sdmmc_io_transfer_mode_t transfer_mode , sdmmc_io_address_mode_t address_mode)

8.35.11.1 概要説明

SDIO にデータを書き込みます。

8.35.11.2 詳細説明

表 877: 戻り値

名前	説明
SSP_SUCCESS	カードの書き込みが正常に終了しました。
p_ctrl の可能性があります。	ハンドルまたはソース アドレスが NULL です。
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはアドレス。

I：この関数は別のチャネルに対して再入可能です。

8.35.12 R_SDMMC_IoIntEnable

`ssp_err_t` R_SDMMC_IoIntEnable (`sdmmc_ctrl_t` *const p_ctrl , bool enable)

8.35.12.1 概要説明

SDIO 割り込みを有効化します。

8.35.12.2 詳細説明

表 878: 戻り値

名前	説明
SSP_SUCCESS	Card enabled or disabled SDIO interrupts successfully.
SSP_ERR_NOT_ENABLED	操作が失敗しました。

I：この関数は別のチャネルに対して再入可能です。

8.35.13 R_SDMMC_VersionGet

ssp_err_t R_SDMMC_VersionGet (ssp_version_t *const p_version)

8.35.13.1 概要説明

SD/MMC ドライバのバージョンを取得します。

8.35.13.2 詳細説明

VersionGet 関数は、ファームウェアと API のバージョンを返します。

表 879: 戻り値

名前	説明
p_ctrl の可能性があります。	Null ポインタ。
SSP_SUCCESS	関数が正常に実行されました。

I：この関数は再入可能です。

8.35.14 R_SDMMC_InfoGet

ssp_err_t R_SDMMC_InfoGet (sdmmc_ctrl_t *const p_ctrl , sdmmc_info_t *const p_info)

8.35.14.1 概要説明

SD カードのカード固有のデータを取得します。

8.35.14.2 詳細説明

InfoGet 関数は、SD カードのカード固有のデータ （CSD）を返します。

I：この関数は再入可能です。

8.35.15 R_SDMMC_Erase

ssp_err_t R_SDMMC_Erase (sdmmc_ctrl_t *const p_ctrl , uint32_t const start_sector , uint32_t const sector_count)

8.35.15.1 概要説明

SDMMC カード内のセクターを消去します。

8.35.15.2 詳細説明

Erase 関数は SD カードを消去します。

表 880: 戻り値

名前	説明
SSP_ERR_INVALID_POINTER	不正なチャネル番号またはセクター アドレス。
SSP_SUCCESS	デバイス ステータスが入手可能です。

I：この関数は別のチャネルに対して再入可能です。

8.35.16 モジュール

- ビルドタイム構成

8.35.16.1 ビルドタイム構成

定義

- #define SDMMC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.36 SLCDC

セグメント LCD コントローラ （SLCDC）のドライバ。

8.36.1 概要

SLCDC インタフェース を拡張します。

8.36.2 Functions

- R_SLCDC_Open
- R_SLCDC_Write
- R_SLCDC_Modify
- R_SLCDC_Start
- R_SLCDC_Stop
- R_SLCDC_ContrastIncrease
- R_SLCDC_ContrastDecrease
- R_SLCDC_SetDisplayArea
- R_SLCDC_Close
- R_SLCDC_VersionGet

8.36.3 R_SLCDC_Open

```
ssp_err_t  R_SLCDC_Open ( slcdc_ctrl_t  *const p_ctrl , slcdc_cfg_t  const *const p_cfg )
```

8.36.3.1 詳細説明

表 881:　戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。

表 881: 戻り値 (続き)

名前	説明
SSP_ERR_INVALID_ARGUMENT	SLCDC リソースがロックされています。

8.36.3.2 関数のステップ

- SLCD リソースをロックします。

8.36.4 R_SLCDC_Write

```
ssp_err_t R_SLCDC_Write ( slcdc_ctrl_t *const p_ctrl , slcdc_size_t const start_segment ,
 slcdc_size_t const *const p_data , slcdc_size_t const segment_count )
```

8.36.4.1 詳細説明

表 882: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.5 R_SLCDC_Modify

```
ssp_err_t R_SLCDC_Modify ( slcdc_ctrl_t *const p_ctrl , slcdc_size_t const segment ,
 slcdc_size_t const data_mask , slcdc_size_t const data )
```

8.36.5.1 詳細説明

表 883: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。

表 883: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.6 R_SLCDC_Start

```
ssp_err_t  R_SLCDC_Start ( slcdc_ctrl_t  *const p_ctrl )
```

8.36.6.1 詳細説明

表 884: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.7 R_SLCDC_Stop

```
ssp_err_t  R_SLCDC_Stop ( slcdc_ctrl_t  *const p_ctrl )
```

8.36.7.1 詳細説明

表 885: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。

表 885: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.8 R_SLCDC_ContrastIncrease

`ssp_err_t` R_SLCDC_ContrastIncrease (`slcdc_ctrl_t` *const p_ctrl)

8.36.8.1 詳細説明

表 886: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.9 R_SLCDC_ContrastDecrease

`ssp_err_t` R_SLCDC_ContrastDecrease (`slcdc_ctrl_t` *const p_ctrl)

8.36.9.1 詳細説明

表 887: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。

表 887: 戻り値 (続き)

名前	説明
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.10 R_SLCDC_SetDisplayArea

ssp_err_t R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const p_ctrl , slcdc_display_area_t const display_area)

8.36.10.1 詳細説明

表 888: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
データ構造体が割り当てられませんでした。	サポートされていない操作です。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。
SSP_ERR_NOT_ENABLED	RTC が点滅操作に対して有効になっていません。

8.36.11 R_SLCDC_Close

ssp_err_t R_SLCDC_Close (slcdc_ctrl_t *const p_ctrl)

8.36.11.1 詳細説明

表 889: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に開かれました。
p_ctrl の可能性があります。	コントロールブロックまたは構成構造へのポインタが NULL です。
SSP_ERR_INVALID_POINTER	引数の無効なパラメータ。
タッチ パネルが設定されていません。	デバイスが開いていないか、初期化されていません。

8.36.12 R_SLCDC_VersionGet

`ssp_err_t` R_SLCDC_VersionGet (`ssp_version_t` * p_version)

8.36.12.1 詳細説明

表 890: 戻り値

名前	説明
p_version。	バージョン番号。

I：この関数は再入可能です。

8.36.13 モジュール

- ビルドタイム構成

8.36.13.1 ビルドタイム構成

定義

- #define SLCDC_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.37 SSI

シリアル サウンド インタフェース （SSI）のドライバ。

8.37.1 概要

I2S インタフェース を拡張します。

8.37.2 Functions

- R_SSI_Open
- R_SSI_Stop
- R_SSI_Close
- R_SSI_Write
- R_SSI_Read
- R_SSI_WriteRead
- R_SSI_Mute
- R_SSI_InfoGet
- R_SSI_VersionGet

8.37.3 定義

- #define SSI_CODE_VERSION_MAJOR

  初期値 :(1)
- #define SSI_CODE_VERSION_MINOR

  初期値 :(1)

8.37.4 R_SSI_Open

```
ssp_err_t R_SSI_Open ( i2s_ctrl_t *const p_ctrl , i2s_cfg_t const *const p_cfg )
```

8.37.4.1 概要説明

SSI を開きます。open を実装します。

8.37.4.2 詳細説明

この関数は、入力オーディオ クロック周波数と要求されたサンプリング周波数に基づき、クロックの除数を計算します。このクロック除数と、i2s_cfg_t で指定された構成を設定します。また、タイマと転送インタフェース （提供されている場合）を開きます。

表 891: 戻り値

名前	説明
SSP_SUCCESS	I2S 通信の準備が完了しています。
p_ctrl の可能性があります。	p_ctrl または p_cfg へのポインタが NULL でした。または、無効なチャネル数が指定されました。
チャネルは現在動作中です。	要求されたチャネルはすでに開かれています。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

8.37.4.3 関数のステップ

- タイマ インスタンスが提供されている場合、タイマ インスタンスを開きます。
- タイマ インスタンスが提供されており、WS 続行モードが有効化されている場合、タイマを開始します。
- 書き込み用に転送インスタンスが提供されている場合、転送インスタンスを開きます。
- 読み取り用に転送インスタンスが提供されている場合、転送インスタンスを開きます。

8.37.5 R_SSI_Stop

```
ssp_err_t  R_SSI_Stop ( i2s_ctrl_t  *const p_ctrl , i2s_dir_t  const dir )
```

8.37.5.1 概要説明

SSI を停止します。stop を実装します。

8.37.5.2 詳細説明

転送インタフェースが使用されている場合、この関数は、転送を無効化します。または、割り込み駆動モードが使用されている場合はデータ書き込みを停止するための停止信号を送信します。

表 892: 戻り値

名前	説明
SSP_SUCCESS	I2S 通信の停止要求が発行されました。
p_ctrl の可能性があります。	p_ctrl へのポインタが NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

8.37.5.3 関数のステップ

- 周辺機器がアイドル状態になった場合、停止します。
- 転送が使用されている場合、停止が要求された時点で転送を無効化します。
- 転送が使用されている場合、停止が要求された時点で転送を無効化します。

8.37.6 R_SSI_Close

```
ssp_err_t  R_SSI_Close ( i2s_ctrl_t  *const p_ctrl )
```

8.37.6.1 概要説明

SSI を閉じます。close を実装します。

8.37.6.2 詳細説明

この関数は、SSI の電源をオフにしてローレベル タイマを閉じ、ドライバを転送します （使用されている場合）。

表 893: 戻り値

名前	説明
SSP_SUCCESS	デバイスが正常に閉じられました。
p_ctrl の可能性があります。	p_ctrl へのポインタが NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

8.37.6.3 関数のステップ

- SSI 周辺機器へのクロックの供給を停止し、非アクティブ化します。
- タイマ インスタンスが提供されている場合、タイマ インスタンスを閉じます。
- 書き込み用に転送インスタンスが提供されている場合、転送インスタンスを閉じます。
- 読み取り用に転送インスタンスが提供されている場合、転送インスタンスを閉じます。
- HW ロックを解除します。

8.37.7 R_SSI_Write

`ssp_err_t` R_SSI_Write (`i2s_ctrl_t` *const p_ctrl , uint8_t const *const p_src , uint16_t const bytes)

8.37.7.1 概要説明

データ バッファを SSI に書き込みます。write を実装します。

8.37.7.2 詳細説明

この関数は、転送インタフェースが使用されている場合に転送をリセットします。または、FIFO に適したデータ長を書き込み、残りのライト バッファが ISR に書き込まれるよう制御ブロックに保存します。

表 894: 戻り値

名前	説明
SSP_SUCCESS	書き込みが正常に開始されました。
p_ctrl の可能性があります。	p_ctrl または p_src へのポインタが NULL でした。または、要求されたバイトが 0 でした。
チャネルは現在動作中です。	別の転送が進行中です。データは書き込まれませんでした。
タッチ パネルが設定されていません。	チャネルは開かれていません。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

8.37.7.3 関数のステップ

- 送信が有効化されていることを確認してください。
- 書き込み用に転送インスタンスが提供されている場合、転送をリセットします。
- それ以外の場合は、FIFO に直接書き込みを行います。

8.37.8 R_SSI_Read

`ssp_err_t` R_SSI_Read (`i2s_ctrl_t` *const p_ctrl , uint8_t *const p_dest , uint16_t const bytes)

8.37.8.1 概要説明

提供されたバッファにデータを読み出します。read を実装します。

8.37.8.2 詳細説明

この関数は、転送インタフェースが使用されている場合に転送をリセットします。または、FIFO 内のデータ長を読み取り、残りのリード バッファが ISR に書き込まれるよう制御ブロックに保存します。

表 895: 戻り値

名前	説明
SSP_SUCCESS	読み取りが正常に開始されました。
p_ctrl の可能性があります。	p_ctrl または p_dest へのポインタが NULL でした。または、要求されたバイトが 0 でした。
タッチ パネルが設定されていません。	チャネルは開かれていません。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

8.37.8.3 関数のステップ

- 受信が有効化されていることを確認してください。
- 読み取り用に転送インスタンスが提供されている場合、転送をリセットします。
- それ以外の場合は、FIFO から直接読み取りを行います。

8.37.9 R_SSI_WriteRead

ssp_err_t R_SSI_WriteRead (i2s_ctrl_t *const p_ctrl , uint8_t const *const p_src , uint8_t *const p_dest , uint16_t const bytes)

8.37.9.1 概要説明

ソース バッファから書き込みを行い、宛先バッファにデータを読み出します。writeRead を実装します。

8.37.9.2 詳細説明

この関数は、R_SSI_Write および R_SSI_Read を呼び出します。

表 896: 戻り値

名前	説明
SSP_SUCCESS	書き込みおよび読み取りが正常に開始されました。
p_ctrl の可能性があります。	p_ctrl または p_src、p_dest へのポインタが NULL でした。または、要求されたバイトが 0 でした。
タッチ パネルが設定されていません。	チャネルは開かれていません。

その他のリターン コードについては、一般的なエラーコードまたはローレベルのドライバを参照してください。

8.37.9.3 関数のステップ

- 書き込み、続いて読み取りを呼び出します。

8.37.10 R_SSI_Mute

ssp_err_t R_SSI_Mute (i2s_ctrl_t *const p_ctrl , i2s_mute_t const mute_enable)

8.37.10.1 概要説明

SSI をミュートにします。mute を実装します。

8.37.10.2 詳細説明

ミュートを有効化してもデータの書き込みは行われますが、送信回線はゼロを出力します。

表 897: 戻り値

名前	説明
SSP_SUCCESS	送信がミュートされています。
p_ctrl の可能性があります。	p_ctrl へのポインタが NULL でした。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.37.11 R_SSI_InfoGet

`ssp_err_t` R_SSI_InfoGet (`i2s_ctrl_t` *const p_ctrl , `i2s_info_t` *const p_info)

8.37.11.1 概要説明

I2S 情報を取得し、指定されたポインタ p_info に格納します。infoGet を実装します。

8.37.11.2 詳細説明

表 898: 戻り値

名前	説明
SSP_SUCCESS	情報が正常に格納されました。
p_ctrl の可能性があります。	p_ctrl または p_info パラメータは NULL です。
タッチ パネルが設定されていません。	チャネルは開かれていません。

8.37.12 R_SSI_VersionGet

`ssp_err_t` R_SSI_VersionGet (`ssp_version_t` *const p_version)

8.37.12.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

8.37.12.2 詳細説明

表 899: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.37.13 モジュール

- ビルドタイム構成

8.37.13.1 ビルドタイム構成

定義

- #define SSI_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`) TODO_SSI Add AMS_ prefix at driver layer

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.38 WDT

ウォッチドッグ タイマ （WDT）のドライバ。

8.38.1 概要

このモジュールは、ウォッチドッグ タイマ （WDT）をサポートします。これは WDT インタフェース を実装します。WDT HAL API では、WDT の動作の設定 （レジスタ スタート モードで使用される場合）、ウォッチドッグのリフレッシュ、タイマ値の読み取り、およびステータス フラグの読み取りとクリアを行うことができます。

8.38.2 Functions

- R_WDT_Open
- R_WDT_CfgGet
- R_WDT_TimeoutGet
- R_WDT_Refresh
- R_WDT_StatusGet
- R_WDT_StatusClear
- R_WDT_CounterGet
- R_WDT_VersionGet

8.38.3 定義

- #define WDT_CODE_VERSION_MAJOR

  初期値 :(1)
- #define WDT_CODE_VERSION_MINOR

  初期値 :(1)

8.38.4 R_WDT_Open

```
ssp_err_t R_WDT_Open ( wdt_ctrl_t *const p_ctrl , wdt_cfg_t const *const p_cfg )
```

8.38.4.1 概要説明

WDT をレジスタ スタート モードに設定します。auto-start_mode の場合は、NMI コールバックを登録できます。open を実装します。

8.38.4.2 詳細説明

WDT 設定レジスタは一度しか書き込むことができず、後続の呼び出しが無視されるため、この関数は一度しか呼び出せません。

表 900: 戻り値

名前	説明
SSP_SUCCESS	WDT が正常に設定されました。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_INVALID_POINTER	1 つまたは複数の構成オプションが無効です。
未サポートまたは不正確なモードです。	WDT をレジスタスタート モードで開こうとしましたが、OFS0 レジスタがオートスタート モードに設定されています。または WDT をオートスタート モードで開こうとしましたが、OSF0 レジスタがレジスタスタート モードに設定されています。

I：この関数は再入可能です。オートスタート モードでは、NMI 出力が選択されている場合に、NMI ISR のコールバックを登録する設定オプションのみが有効です。

8.38.4.3 関数のステップ

- g_wdt_version にアクセスできるのは ASSERT マクロのみなので、それにアクセスできないことを示す警告がコンパイラ ツールチェーンから発行されることがあります。以下のコードはこの警告を抑制するとともに、このデータ構造体が最適化によって除去されるのを防ぎます。
- NMI 出力が使用されていない場合に、ツールチェーン警告を排除します。
- 指定されたスタート モードが、OSF0 の設定と一致していることをチェックします。
- IWDT ハードウェア リソースをロックします
- WDT へのグローバル ポインタを NMI コールバックで使用できるように初期化します。
- 設定は、WDT がレジスタスタート モードで動作している場合のみ有効です。
- レジスタスタート モード。
- BSP NMI ISR を使用して、コールバックを登録します。
- WDT アンダーフロー / リフレッシュ エラー割り込み （NMI を生成）を有効にします。

- リフレッシュを実行して、タイマを開始します。

8.38.5 R_WDT_CfgGet

ssp_err_t R_WDT_CfgGet (wdt_ctrl_t *const p_ctrl , wdt_cfg_t *const p_cfg)

8.38.5.1 概要説明

レジスタスタート モードとオートスタート モードの両方で、WDT の設定を読み取ります。cfgGet を実装します。

8.38.5.2 詳細説明

表 901: 戻り値

名前	説明
SSP_SUCCESS	WDT が正常に設定されました。
p_ctrl の可能性があります。	Null ポインタ。

I：この関数は再入可能です。

8.38.5.3 関数のステップ

- レジスタスタート モード。
- WDTCR レジスタからタイムアウト値を取得します。

8.38.6 R_WDT_TimeoutGet

ssp_err_t R_WDT_TimeoutGet (wdt_ctrl_t *const p_ctrl , wdt_timeout_values_t *const p_timeout)

8.38.6.1 概要説明

ウォッチドッグ タイマのタイムアウト情報を読み取ります。timeoutGet を実装します。

8.38.6.2 詳細説明

表 902: 戻り値

名前	説明
SSP_SUCCESS	WDT が正常にリフレッシュされました。
p_ctrl の可能性があります。	Null ポインタ。
SSP_ERR_ABORTED	このウォッチドッグのクロック分周器が無効です

I：この関数は再入可能です。この関数を呼び出す前に、R_WDT_Open を呼び出す必要があります。

8.38.7 R_WDT_Refresh

```
ssp_err_t  R_WDT_Refresh ( wdt_ctrl_t  *const p_ctrl )
```

8.38.7.1 概要説明

ウォッチドッグ タイマをリフレッシュします。refresh を実装します。

8.38.7.2 詳細説明

この関数は、ウォッチドッグ カウンタのリフレッシュに加え、レジスタ スタート モードで使用してカウンタを開始することができます。

表 903: 戻り値

名前	説明
SSP_SUCCESS	WDT が正常にリフレッシュされました。

I：この関数は再入可能です。この関数は、SSP_SUCCESS のみを返します。許可リフレッシュ期間の外側で行われたためにリフレッシュが失敗した場合、デバイスはリセットするか、NMI ISR をトリガーして実行します。この関数を呼び出す前に、R_WDT_Open を呼び出す必要があります。

8.38.8 R_WDT_StatusGet

ssp_err_t R_WDT_StatusGet (wdt_ctrl_t *const p_ctrl , wdt_status_t *const p_status)

8.38.8.1 概要説明

WDT のステータス フラグを読み取ります。statusGet を実装します。

8.38.8.2 詳細説明

ステータスとエラー状態の両方を示します。

表 904: 戻り値

名前	説明
SSP_SUCCESS	WDT ステータスが正常に読み取られました。
p_ctrl の可能性があります。	NULL ポインタがパラメータ。

I：この関数は再入可能です。WDT がアンダーフロー時またはリフレッシュ エラー時にリセットを出力するように設定されている場合、ステータスやエラー フラグには、アンダーフローの発生やリフレッシュ エラーの存在が示されないため、それらを読み取っても意味がありません。ステータスやエラー フラグを読み取るのは、割り込み要求出力が有効な場合のみ役に立ちます。

8.38.9 R_WDT_StatusClear

ssp_err_t R_WDT_StatusClear (wdt_ctrl_t *const p_ctrl , wdt_status_t const status)

8.38.9.1 概要説明

WDT のステータスおよびエラー フラグをクリアします。statusClear を実装します。

8.38.9.2 詳細説明

表 905: 戻り値

名前	説明
SSP_SUCCESS	WDT フラグが正常にクリアされました。
p_ctrl の可能性があります。	NULL ポインタがパラメータ。

I：この関数は再入可能です。

8.38.9.3 関数のステップ

- ゼロを書き込んでフラグをクリアします。

8.38.10 R_WDT_CounterGet

`ssp_err_t` R_WDT_CounterGet (`wdt_ctrl_t` *const p_ctrl , uint32_t *const p_count)

8.38.10.1 概要説明

WDT の現在のカウント値を読み取ります。counterGet を実装します。

8.38.10.2 詳細説明

表 906: 戻り値

名前	説明
SSP_SUCCESS	WDT の現在のカウントが正常に読み取られました。
p_ctrl の可能性があります。	NULL ポインタがパラメータとして受け渡されました。

I：この関数は再入可能です。

8.38.11 R_WDT_VersionGet

`ssp_err_t` R_WDT_VersionGet (`ssp_version_t` *const p_data)

8.38.11.1 概要説明

WDT HAL ドライバのバージョンを返します。versionGet を実装します。

8.38.11.2 詳細説明

表 907: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報を正常に読み取りました。
p_ctrl の可能性があります。	Null ポインタがパラメータとして渡されました

I：この関数は再入可能です。

8.38.12 モジュール

- ビルドタイム構成

8.38.12.1 ビルドタイム構成

定義

- #define WDT_CFG_PARAM_CHECKING_ENABLE

  初期値 :(`#define BSP_CFG_PARAM_CHECKING_ENABLE`)

  API パラメータチェックにコードを含めるかどうかを指定します。有効な設定には次が含まれます :BSP_CFG_PARAM_CHECKING_ENABLE : bsp_cfg.h1 からのシステムデフォルト設定を使用します : パラメータのチェック 0 が含まれます : パラメータチェックをコンパイルアウトします

8.39 SCE モジュール

基本暗号関数。

SCE モジュールの暗号操作を初期化します。

SCE の暗号 API インタフェース

8.39.1 Functions

- R_SCE_Open
- R_SCE_VersionGet
- R_SCE_StatusGet
- R_SCE_Close

8.39.2 変数

- g_sce_crypto_api
- g_sce_ctrl_blk

8.39.3 R_SCE_Open

R_SCE_Open (`crypto_ctrl_t` *const p_ctrl , `crypto_cfg_t` const *const p_cfg)

8.39.4 R_SCE_VersionGet

R_SCE_VersionGet (`ssp_version_t` *const p_version)

8.39.4.1 概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

8.39.4.2 詳細説明

表 908: パラメータ

名前	方向	説明
p_version	out	SCE 実装のバージョン情報

表 909: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.39.5 R_SCE_StatusGet

R_SCE_StatusGet (uint32_t * p_status)

8.39.5.1 概要説明

この関数は、SCE が初期化されたかどうかを示します。

8.39.5.2 詳細説明

SCE モジュール初期化の SCE Get ステータス

表 910: パラメータ

名前	方向	説明
p_status	複数のビットを書き換えることもできます。	uint32_t へのポインタ。この記憶場所は SCE モジュール初期化で更新されます。

表 911: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

8.39.6 R_SCE_Close

R_SCE_Close (`crypto_ctrl_t` *const p_ctrl)

8.39.6.1 概要説明

SCE ドライバを閉じます。何もしません。常に正常終了を返します。

8.39.6.2 詳細説明

表 912: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SCE ブロックの制御構造体

表 913: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。

8.39.7 g_sce_crypto_api

crypto_api_t::g_sce_crypto_api

8.39.7.1 次のように初期化されます

```
g_sce_crypto_api=
{
   .open      = R_SCE_Open,
   .close     = R_SCE_Close,
   .statusGet  = R_SCE_StatusGet,
   .versionGet =R_SCE_VersionGet
}
```

8.39.8 g_sce_ctrl_blk

crypto_ctrl_t::g_sce_ctrl_blk

8.39.9 モジュール

- SCE AES
- SCE HRK AES

- SCE ARC4
- SCE DSA
- SCE_RSA
- SCE_HASH
- SCE_TDES
- SCE_TRNG

8.39.9.1 SCE AES

基本暗号関数。

AES 暗号化および復号化関数

暗号化および復号化関数の AES 256 ビット CBC モード実装

暗号化および復号化関数の AES 256 ビット CTR モード実装

暗号化および復号化関数の AES 256 ビット ECB モード実装

暗号化および復号化関数の AES 256 ビット XTS モード実装

Functions

- R_SCE_AES_Open
- R_SCE_AES_VersionGet
- R_SCE_AES_Close
- R_SCE_AES_128CbcEncrypt
- R_SCE_AES_128CbcDecrypt
- R_SCE_AES_128CtrEncrypt
- R_SCE_AES_128EcbEncrypt
- R_SCE_AES_128EcbDecrypt
- R_SCE_AES_128GcmOpen
- R_SCE_AES_128GcmAdditionalAuthenticationData
- R_SCE_AES_128GcmEncrypt
- R_SCE_AES_128GcmGetGcmTag
- R_SCE_AES_128GcmSetGcmTag
- R_SCE_AES_128GcmDecrypt
- R_SCE_AES_128GcmZeroPaddingEncrypt

参考資料

- R_SCE_AES_128GcmZeroPaddingDecrypt
- R_SCE_AES_128XtsEncrypt
- R_SCE_AES_128XtsDecrypt
- R_SCE_AES_192CbcEncrypt
- R_SCE_AES_192CbcDecrypt
- R_SCE_AES_192CtrEncrypt
- R_SCE_AES_192EcbEncrypt
- R_SCE_AES_192EcbDecrypt
- R_SCE_AES_192GcmOpen
- R_SCE_AES_192GcmAdditionalAuthenticationData
- R_SCE_AES_192GcmEncrypt
- R_SCE_AES_192GcmGetGcmTag
- R_SCE_AES_192GcmSetGcmTag
- R_SCE_AES_192GcmDecrypt
- R_SCE_AES_192GcmZeroPaddingEncrypt
- R_SCE_AES_192GcmZeroPaddingDecrypt
- R_SCE_AES_256CbcEncrypt
- R_SCE_AES_256CbcDecrypt
- R_SCE_AES_256CtrEncrypt
- R_SCE_AES_256EcbEncrypt
- R_SCE_AES_256EcbDecrypt
- R_SCE_AES_256GcmOpen
- R_SCE_AES_256GcmAdditionalAuthenticationData
- R_SCE_AES_256GcmEncrypt
- R_SCE_AES_256GcmGetGcmTag
- R_SCE_AES_256GcmSetGcmTag
- R_SCE_AES_256GcmDecrypt
- R_SCE_AES_256GcmZeroPaddingEncrypt
- R_SCE_AES_256GcmZeroPaddingDecrypt

- R_SCE_AES_256XtsEncrypt
- R_SCE_AES_256XtsDecrypt

変数

- g_aes128cbc_on_sce
- g_aes128ctr_on_sce
- g_aes128ecb_on_sce
- g_aes128gcm_on_sce
- g_aes128xts_on_sce
- g_aes192cbc_on_sce
- g_aes192ctr_on_sce
- g_aes192ecb_on_sce
- g_aes192gcm_on_sce
- g_aes256cbc_on_sce
- g_aes256ctr_on_sce
- g_aes256ecb_on_sce
- g_aes256gcm_on_sce
- g_aes256xts_on_sce

定義

- #define SCE_AES_CODE_VERSION_MAJOR

  初期値 :(01)
- #define SCE_AES_CODE_VERSION_MINOR

  初期値 :(00)
- #define SIZE_GCMTAG_BITS

  初期値 :(128)
- #define SIZE_GCMTAG_BYTES

  初期値 :((SIZE_GCMTAG_BITS) / 8)
- #define SIZE_GCMTAG_WORDS

  初期値 :((SIZE_GCMTAG_BITS) / 32)
- #define SIZE_AES_BLOCK_BITS

  初期値 :(128)

- #define SIZE_AES_BLOCK_BYTES
  初期値 :((SIZE_AES_BLOCK_BITS) / 8)
- #define SIZE_AES_BLOCK_WORDS
  初期値 :((SIZE_AES_BLOCK_BITS) / 32)
- #define SIZE_AES_128BIT_KEYLEN_BITS
  初期値 :(128)
- #define SIZE_AES_128BIT_KEYLEN_BYTES
  初期値 :((SIZE_AES_128BIT_KEYLEN_BITS) / 8)
- #define SIZE_AES_128BIT_KEYLEN_WORDS
  初期値 :((SIZE_AES_128BIT_KEYLEN_BITS) / 32)
- #define SIZE_AES_192BIT_KEYLEN_BITS
  初期値 :(192)
- #define SIZE_AES_192BIT_KEYLEN_BYTES
  初期値 :((SIZE_AES_192BIT_KEYLEN_BITS) / 8)
- #define SIZE_AES_192BIT_KEYLEN_WORDS
  初期値 :((SIZE_AES_192BIT_KEYLEN_BITS) / 32)
- #define SIZE_AES_256BIT_KEYLEN_BITS
  初期値 :(256)
- #define SIZE_AES_256BIT_KEYLEN_BYTES
  初期値 :((SIZE_AES_256BIT_KEYLEN_BITS) / 8)
- #define SIZE_AES_256BIT_KEYLEN_WORDS
  初期値 :((SIZE_AES_256BIT_KEYLEN_BITS) / 32)

g_aes128cbc_on_sce

aes_api_t::g_aes128cbc_on_sce

詳細説明

AES 128 ビット CBC モード実装

次のように初期化されます

```
g_aes128cbc_on_sce=
{
  .open      = R_SCE_AES_Open,
  .encrypt   = R_SCE_AES_128CbcEncrypt,
  .decrypt   = R_SCE_AES_128CbcDecrypt,
  .close     = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes128ctr_on_sce

aes_api_t::g_aes128ctr_on_sce

詳細説明

AES 128 ビット CTR モード実装

次のように初期化されます

```
g_aes128ctr_on_sce=
{
  .open      = R_SCE_AES_Open,
  .encrypt   = R_SCE_AES_128CtrEncrypt,
  .decrypt   = R_SCE_AES_128CtrEncrypt,
  .close     = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes128ecb_on_sce

aes_api_t::g_aes128ecb_on_sce

詳細説明

AES 128 ビット ECB モード実装

次のように初期化されます

```
g_aes128ecb_on_sce=
{
  .open      = R_SCE_AES_Open,
  .encrypt   = R_SCE_AES_128EcbEncrypt,
  .decrypt   = R_SCE_AES_128EcbDecrypt,
  .close     = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes128gcm_on_sce

aes_api_t::g_aes128gcm_on_sce

詳細説明

AES 128 ビット GCM モード実装

次のように初期化されます

```
g_aes128gcm_on_sce=
{
  .open            = R_SCE_AES_128GcmOpen,
  .encrypt         = R_SCE_AES_128GcmEncrypt,
  .decrypt         = R_SCE_AES_128GcmDecrypt,
  .getGcmTag       = R_SCE_AES_128GcmGetGcmTag,
  .setGcmTag       = R_SCE_AES_128GcmSetGcmTag,
  .close           = R_SCE_AES_Close,
  .versionGet      = R_SCE_AES_VersionGet,
  .zeroPaddingEncrypt = R_SCE_AES_128GcmZeroPaddingEncrypt,
  .zeroPaddingDecrypt =R_SCE_AES_128GcmZeroPaddingDecrypt
}
```

g_aes128xts_on_sce

aes_api_t::g_aes128xts_on_sce

詳細説明

AES 128 ビット CCM モード実装

次のように初期化されます

```
g_aes128xts_on_sce=
{
  .open     = R_SCE_AES_Open,
  .encrypt  = R_SCE_AES_128XtsEncrypt,
  .decrypt  = R_SCE_AES_128XtsDecrypt,
  .close    = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes192cbc_on_sce

aes_api_t::g_aes192cbc_on_sce

詳細説明

AES 192 ビット CBC モード実装

次のように初期化されます

```
g_aes192cbc_on_sce=
{
  .open     = R_SCE_AES_Open,
  .encrypt  = R_SCE_AES_192CbcEncrypt,
  .decrypt  = R_SCE_AES_192CbcDecrypt,
  .close    = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes192ctr_on_sce

aes_api_t::g_aes192ctr_on_sce

詳細説明

AES 192 ビット CTR モード実装

次のように初期化されます

```
g_aes192ctr_on_sce=
{
  .open      = R_SCE_AES_Open,
  .encrypt   = R_SCE_AES_192CtrEncrypt,
  .decrypt   = R_SCE_AES_192CtrEncrypt,
  .close     = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes192ecb_on_sce

aes_api_t::g_aes192ecb_on_sce

詳細説明

AES 192 ビット ECB モード実装

次のように初期化されます

```
g_aes192ecb_on_sce=
{
  .open      = R_SCE_AES_Open,
  .encrypt   = R_SCE_AES_192EcbEncrypt,
  .decrypt   = R_SCE_AES_192EcbDecrypt,
  .close     = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes192gcm_on_sce

aes_api_t::g_aes192gcm_on_sce

詳細説明

AES 192 ビット GCM モード実装

次のように初期化されます

```
g_aes192gcm_on_sce=
{
  .open            = R_SCE_AES_192GcmOpen,
  .encrypt         = R_SCE_AES_192GcmEncrypt,
  .decrypt         = R_SCE_AES_192GcmDecrypt,
  .getGcmTag       = R_SCE_AES_192GcmGetGcmTag,
  .setGcmTag       = R_SCE_AES_192GcmSetGcmTag,
  .close           = R_SCE_AES_Close,
  .versionGet      = R_SCE_AES_VersionGet,
  .zeroPaddingEncrypt = R_SCE_AES_192GcmZeroPaddingEncrypt,
  .zeroPaddingDecrypt =R_SCE_AES_192GcmZeroPaddingDecrypt
}
```

g_aes256cbc_on_sce

aes_api_t::g_aes256cbc_on_sce

詳細説明

AES 256 ビット CBC モード実装

次のように初期化されます

```
g_aes256cbc_on_sce=
{
  .open     = R_SCE_AES_Open,
  .encrypt  = R_SCE_AES_256CbcEncrypt,
  .decrypt  = R_SCE_AES_256CbcDecrypt,
  .close    = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes256ctr_on_sce

aes_api_t::g_aes256ctr_on_sce

詳細説明

AES 256 ビット CTR モード実装

次のように初期化されます

```
g_aes256ctr_on_sce=
{
  .open     = R_SCE_AES_Open,
  .encrypt  = R_SCE_AES_256CtrEncrypt,
  .decrypt  = R_SCE_AES_256CtrEncrypt,
  .close    = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

g_aes256ecb_on_sce

aes_api_t::g_aes256ecb_on_sce

詳細説明

AES 256 ビット ECB モード実装

次のように初期化されます

```
g_aes256ecb_on_sce=
{
   .open      = R_SCE_AES_Open,
   .encrypt    = R_SCE_AES_256EcbEncrypt,
   .decrypt    = R_SCE_AES_256EcbDecrypt,
   .close     = R_SCE_AES_Close,
   .versionGet =R_SCE_AES_VersionGet
}
```

g_aes256gcm_on_sce

aes_api_t::g_aes256gcm_on_sce

詳細説明

AES 256 ビット GCM モード実装

次のように初期化されます

```
g_aes256gcm_on_sce=
{
   .open            = R_SCE_AES_256GcmOpen,
   .encrypt          = R_SCE_AES_256GcmEncrypt,
   .decrypt          = R_SCE_AES_256GcmDecrypt,
   .getGcmTag        = R_SCE_AES_256GcmGetGcmTag,
   .setGcmTag        = R_SCE_AES_256GcmSetGcmTag,
   .close           = R_SCE_AES_Close,
   .versionGet       = R_SCE_AES_VersionGet,
   .zeroPaddingEncrypt = R_SCE_AES_256GcmZeroPaddingEncrypt,
   .zeroPaddingDecrypt =R_SCE_AES_256GcmZeroPaddingDecrypt
}
```

g_aes256xts_on_sce

aes_api_t::g_aes256xts_on_sce

詳細説明

AES 256 ビット XTS モード実装

次のように初期化されます

```
g_aes256xts_on_sce=
{
  .open      = R_SCE_AES_Open,
  .encrypt   = R_SCE_AES_256XtsEncrypt,
  .decrypt   = R_SCE_AES_256XtsDecrypt,
  .close     = R_SCE_AES_Close,
  .versionGet =R_SCE_AES_VersionGet
}
```

R_SCE_AES_Open

R_SCE_AES_Open (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

詳細説明

AES 初期化

表 914: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_AES_VersionGet

R_SCE_AES_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 915: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_AES_Close

R_SCE_AES_Close (aes_ctrl_t *const p_ctrl)

詳細説明

AES Close 関数

表 916: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_AES_128CbcEncrypt

R_SCE_AES_128CbcEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 128 ビット CBC モード実装

バッファ key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 917: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_128CbcDecrypt

R_SCE_AES_128CbcDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 918: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_128CtrEncrypt

R_SCE_AES_128CtrEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 128 ビット CTR モード実装

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 919: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_128EcbEncrypt

R_SCE_AES_128EcbEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 128 ビット ECB モード実装

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 920: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :ECB チェーン モードでは、p_iv バッファの内容が無視されます。

R_SCE_AES_128EcbDecrypt

R_SCE_AES_128EcbDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

復号化インタフェース API 用の AES 128 ビット ECB モード実装

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 921: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :ECB チェーン モードでは、p_iv バッファの内容が無視されます。

R_SCE_AES_128GcmOpen

R_SCE_AES_128GcmOpen (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

R_SCE_AES_128GcmAdditionalAuthenticationData

R_SCE_AES_128GcmAdditionalAuthenticationData (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source)

R_SCE_AES_128GcmEncrypt

R_SCE_AES_128GcmEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 922: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_128GcmGetGcmTag

R_SCE_AES_128GcmGetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_dest)

R_SCE_AES_128GcmSetGcmTag

R_SCE_AES_128GcmSetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_source)

R_SCE_AES_128GcmDecrypt

R_SCE_AES_128GcmDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 923: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

関数のステップ

- 認証タグを取得します。

R_SCE_AES_128GcmZeroPaddingEncrypt

R_SCE_AES_128GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_bytes バイトを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_bytes バイト以上のデータ用のスペースがあると見なされます。

表 924: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、(num_bytes/16+1)*16 ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

l : この関数はスレッド セーフではありません。

R_SCE_AES_128GcmZeroPaddingDecrypt

R_SCE_AES_128GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 925: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 925: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_128XtsEncrypt

R_SCE_AES_128XtsEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 128 ビット XTS モード実装。バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 926: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_128XtsDecrypt

R_SCE_AES_128XtsDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

復号化インタフェース API 用の AES 128 ビット XTS モード実装。バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 927: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_192CbcEncrypt

R_SCE_AES_192CbcEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 192 ビット CBC モード実装。バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 928: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 24 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_192CbcDecrypt

R_SCE_AES_192CbcDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

復号化インタフェース API 用の AES 192 ビット CBC モード実装。バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 929: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 24 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_192CtrEncrypt

R_SCE_AES_192CtrEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化および復号化インタフェース API 用の AES 192 ビット CTR モード実装。バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 930: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 24 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_192EcbEncrypt

R_SCE_AES_192EcbEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 192 ビット ECB モード実装

バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 931: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 24 バイトの AES キー データが保存され、

l :ECB チェーン モードでは、p_iv バッファの内容が無視されます。

R_SCE_AES_192EcbDecrypt

R_SCE_AES_192EcbDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

復号化インタフェース API 用の AES 192 ビット ECB モード実装。バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 932: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 24 バイトの AES キー データが保存され、

l :ECB チェーン モードでは、p_iv バッファの内容が無視されます。

R_SCE_AES_192GcmOpen

R_SCE_AES_192GcmOpen (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

R_SCE_AES_192GcmAdditionalAuthenticationData

R_SCE_AES_192GcmAdditionalAuthenticationData (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source)

R_SCE_AES_192GcmEncrypt

R_SCE_AES_192GcmEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 933: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_192GcmGetGcmTag

R_SCE_AES_192GcmGetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_dest)

R_SCE_AES_192GcmSetGcmTag

R_SCE_AES_192GcmSetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_source)

R_SCE_AES_192GcmDecrypt

R_SCE_AES_192GcmDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 934: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

関数のステップ

- 認証タグを取得します。

R_SCE_AES_192GcmZeroPaddingEncrypt

R_SCE_AES_192GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_bytes バイトを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_bytes バイト以上のデータ用のスペースがあると見なされます。

表 935: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_bytes ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります

l : この関数はスレッド セーフではありません。

R_SCE_AES_192GcmZeroPaddingDecrypt

R_SCE_AES_192GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 936: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_256CbcEncrypt

R_SCE_AES_256CbcEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 256 ビット CBC モード実装。バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 937: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_AES_256CbcDecrypt

R_SCE_AES_256CbcDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 938: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_AES_256CtrEncrypt

R_SCE_AES_256CtrEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化および復号化インタフェース API 用の AES 256 ビット CTR モード実装。バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 939: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_AES_256EcbEncrypt

R_SCE_AES_256EcbEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 256 ビット ECB モード実装。バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 940: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_AES_256EcbDecrypt

R_SCE_AES_256EcbDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 941: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_AES_256GcmOpen

R_SCE_AES_256GcmOpen (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

R_SCE_AES_256GcmAdditionalAuthenticationData

R_SCE_AES_256GcmAdditionalAuthenticationData (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source)

R_SCE_AES_256GcmEncrypt

R_SCE_AES_256GcmEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 256 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 942: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_256GcmGetGcmTag

R_SCE_AES_256GcmGetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_dest)

R_SCE_AES_256GcmSetGcmTag

R_SCE_AES_256GcmSetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_source)

R_SCE_AES_256GcmDecrypt

R_SCE_AES_256GcmDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 943: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

関数のステップ

- 認証タグを取得します。

R_SCE_AES_256GcmZeroPaddingEncrypt

R_SCE_AES_256GcmZeroPaddingEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_bytes バイトを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_bytes バイト以上のデータ用のスペースがあると見なされます。

表 944: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_bytes ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

I：この関数はスレッド セーフではありません。

R_SCE_AES_256GcmZeroPaddingDecrypt

R_SCE_AES_256GcmZeroPaddingDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 945: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

I :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

I :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_256XtsEncrypt

R_SCE_AES_256XtsEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

暗号化インタフェース API 用の AES 256 ビット XTS モード実装。バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 946: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_AES_256XtsDecrypt

R_SCE_AES_256XtsDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

復号化インタフェース API 用の AES 256 ビット XTS モード実装。バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 947: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

I :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

8.39.9.2 SCE HRK AES

基本暗号関数。

AES API の SCE/AES 実装。

SCE HRK 暗号化キー AES 暗号実装関数

SCE HRK 暗号化キー AES 暗号実装関数

Functions

- R_SCE_HRK_AES_Open
- R_SCE_HRK_AES_Close
- R_SCE_HRK_AES_Encrypt_Eimpl
- R_SCE_HRK_AES_Decrypt_Eimpl
- R_SCE_HRK_AES_VersionGet
- R_SCE_HRK_AES_128GcmOpen
- R_SCE_HRK_AES_128CreateEncryptedKey
- R_SCE_HRK_AES_128GcmAdditionalAuthenticationData
- R_SCE_HRK_AES_128GcmEncrypt
- R_SCE_HRK_AES_128GcmGetGcmTag
- R_SCE_HRK_AES_128GcmSetGcmTag
- R_SCE_HRK_AES_128GcmDecrypt
- R_SCE_HRK_AES_192GcmOpen
- R_SCE_HRK_AES_192CreateEncryptedKey
- R_SCE_HRK_AES_192GcmAdditionalAuthenticationData
- R_SCE_HRK_AES_192GcmEncrypt
- R_SCE_HRK_AES_192GcmGetGcmTag
- R_SCE_HRK_AES_192GcmSetGcmTag
- R_SCE_HRK_AES_192GcmDecrypt

- R_SCE_HRK_AES_256GcmOpen
- R_SCE_HRK_AES_256CreateEncryptedKey
- R_SCE_HRK_AES_256GcmAdditionalAuthenticationData
- R_SCE_HRK_AES_256GcmEncrypt
- R_SCE_HRK_AES_256GcmGetGcmTag
- R_SCE_HRK_AES_256GcmSetGcmTag
- R_SCE_HRK_AES_256GcmDecrypt

変数

- g_aes_eimpl_on_sceHrk
- g_aes128gcm_on_sceHrk
- g_aes192gcm_on_sceHrk
- g_aes256gcm_on_sceHrk

g_aes_eimpl_on_sceHrk

aes_api_t::g_aes_eimpl_on_sceHrk

次のように初期化されます

```
g_aes_eimpl_on_sceHrk=
{
   .open       = R_SCE_HRK_AES_Open,
   .encrypt    = R_SCE_HRK_AES_Encrypt_Eimpl,
   .decrypt    = R_SCE_HRK_AES_Decrypt_Eimpl,
   .close      = R_SCE_HRK_AES_Close,
   .versionGet =R_SCE_HRK_AES_VersionGet
}
```

g_aes128gcm_on_sceHrk

aes_api_t::g_aes128gcm_on_sceHrk

次のように初期化されます

```
g_aes128gcm_on_sceHrk=
{
  .open      = R_SCE_HRK_AES_128GcmOpen,
  .createKey  = R_SCE_HRK_AES_128CreateEncryptedKey,
  .encrypt    = R_SCE_HRK_AES_128GcmEncrypt,
  .decrypt    = R_SCE_HRK_AES_128GcmDecrypt,
  .getGcmTag  = R_SCE_HRK_AES_128GcmGetGcmTag,
  .setGcmTag  = R_SCE_HRK_AES_128GcmSetGcmTag,
  .close      = R_SCE_HRK_AES_Close,
  .versionGet =R_SCE_HRK_AES_VersionGet
}
```

g_aes192gcm_on_sceHrk

aes_api_t::g_aes192gcm_on_sceHrk

次のように初期化されます

```
g_aes192gcm_on_sceHrk=
{
  .open      = R_SCE_HRK_AES_192GcmOpen,
  .createKey  = R_SCE_HRK_AES_192CreateEncryptedKey,
  .encrypt    = R_SCE_HRK_AES_192GcmEncrypt,
  .decrypt    = R_SCE_HRK_AES_192GcmDecrypt,
  .getGcmTag  = R_SCE_HRK_AES_192GcmGetGcmTag,
  .setGcmTag  = R_SCE_HRK_AES_192GcmSetGcmTag,
  .close      = R_SCE_HRK_AES_Close,
  .versionGet =R_SCE_HRK_AES_VersionGet
}
```

g_aes256gcm_on_sceHrk

aes_api_t::g_aes256gcm_on_sceHrk

次のように初期化されます

```
g_aes256gcm_on_sceHrk=
{
  .open      = R_SCE_HRK_AES_256GcmOpen,
  .createKey  = R_SCE_HRK_AES_256CreateEncryptedKey,
  .encrypt    = R_SCE_HRK_AES_256GcmEncrypt,
  .decrypt    = R_SCE_HRK_AES_256GcmDecrypt,
  .getGcmTag  = R_SCE_HRK_AES_256GcmGetGcmTag,
  .setGcmTag  = R_SCE_HRK_AES_256GcmSetGcmTag,
  .close      = R_SCE_HRK_AES_Close,
  .versionGet =R_SCE_HRK_AES_VersionGet
}
```

R_SCE_HRK_AES_Open

R_SCE_HRK_AES_Open (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

詳細説明

AES 初期化

表 948: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_HRK_AES_Close

R_SCE_HRK_AES_Close (aes_ctrl_t *const p_ctrl)

詳細説明

AES 初期化

表 949: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_HRK_AES_Encrypt_Eimpl

R_SCE_HRK_AES_Encrypt_Eimpl (aes_ctrl_t *const p_ctrl , const uint32_t * key , uint32_t * iv , uint32_t imaxcnt , uint32_t * in , uint32_t * out)

R_SCE_HRK_AES_Decrypt_Eimpl

R_SCE_HRK_AES_Decrypt_Eimpl (aes_ctrl_t *const p_ctrl , const uint32_t * key , uint32_t * iv , uint32_t imaxcnt , uint32_t * in , uint32_t * out)

R_SCE_HRK_AES_VersionGet

R_SCE_HRK_AES_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 950:　戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_HRK_AES_128GcmOpen

R_SCE_HRK_AES_128GcmOpen (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

R_SCE_HRK_AES_128CreateEncryptedKey

R_SCE_HRK_AES_128CreateEncryptedKey (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_key)

R_SCE_HRK_AES_128GcmAdditionalAuthenticationData

R_SCE_HRK_AES_128GcmAdditionalAuthenticationData (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source)

R_SCE_HRK_AES_128GcmEncrypt

R_SCE_HRK_AES_128GcmEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 951:　戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_HRK_AES_128GcmGetGcmTag

R_SCE_HRK_AES_128GcmGetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_dest)

R_SCE_HRK_AES_128GcmSetGcmTag

R_SCE_HRK_AES_128GcmSetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_source)

R_SCE_HRK_AES_128GcmDecrypt

R_SCE_HRK_AES_128GcmDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 128 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 952: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

関数のステップ

- 認証タグを取得します。

R_SCE_HRK_AES_192GcmOpen

R_SCE_HRK_AES_192GcmOpen (`aes_ctrl_t` *const p_ctrl , `aes_cfg_t` const *const p_cfg)

R_SCE_HRK_AES_192CreateEncryptedKey

R_SCE_HRK_AES_192CreateEncryptedKey (`aes_ctrl_t` *const p_ctrl , uint32_t num_words , uint32_t * p_key)

R_SCE_HRK_AES_192GcmAdditionalAuthenticationData

R_SCE_HRK_AES_192GcmAdditionalAuthenticationData (`aes_ctrl_t` *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source)

R_SCE_HRK_AES_192GcmEncrypt

R_SCE_HRK_AES_192GcmEncrypt (`aes_ctrl_t` *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 192 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 953: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_HRK_AES_192GcmGetGcmTag

R_SCE_HRK_AES_192GcmGetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_dest)

R_SCE_HRK_AES_192GcmSetGcmTag

R_SCE_HRK_AES_192GcmSetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_source)

R_SCE_HRK_AES_192GcmDecrypt

R_SCE_HRK_AES_192GcmDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 192 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 954: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

関数のステップ

- 認証タグを取得します。

R_SCE_HRK_AES_256GcmOpen

R_SCE_HRK_AES_256GcmOpen (aes_ctrl_t *const p_ctrl , aes_cfg_t const *const p_cfg)

R_SCE_HRK_AES_256CreateEncryptedKey

R_SCE_HRK_AES_256CreateEncryptedKey (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_key)

R_SCE_HRK_AES_256GcmAdditionalAuthenticationData

R_SCE_HRK_AES_256GcmAdditionalAuthenticationData (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source)

R_SCE_HRK_AES_256GcmEncrypt

R_SCE_HRK_AES_256GcmEncrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 256 ビット AES p_key とバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 955: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

I :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

I :p_key バッファには 16 バイトの AES キー データが保存され、

l :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

R_SCE_HRK_AES_256GcmGetGcmTag

R_SCE_HRK_AES_256GcmGetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_dest)

R_SCE_HRK_AES_256GcmSetGcmTag

R_SCE_HRK_AES_256GcmSetGcmTag (aes_ctrl_t *const p_ctrl , uint32_t num_words , uint32_t * p_source)

R_SCE_HRK_AES_256GcmDecrypt

R_SCE_HRK_AES_256GcmDecrypt (aes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

詳細説明

バッファ p_key からの 256 ビット AES キーとバッファ p_iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 956: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには 16 バイトの AES キー データが保存され、

I :p_iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

関数のステップ

- 認証タグを取得します。

8.39.9.3 SCE ARC4

基本暗号関数。

ARC4 暗号化および復号化関数

Functions

- R_SCE_ARC4_Open
- R_SCE_ARC4_VersionGet
- R_SCE_ARC4_Close
- R_SCE_ARC4_2048Encrypt

定義

- #define SCE_ARC4_CODE_VERSION_MAJOR
  初期値 :(01)
- #define SCE_ARC4_CODE_VERSION_MINOR
  初期値 :(00)

R_SCE_ARC4_Open

R_SCE_ARC4_Open (cipher_ctrl_t *const p_ctrl , cipher_cfg_t const *const p_cfg)

詳細説明

ARC4 初期化

表 957: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました

表 957: 戻り値 (続き)

名前	説明
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE リソースがビジーです。
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE 内部 I/O が空ではありません。

R_SCE_ARC4_VersionGet

R_SCE_ARC4_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 958: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_ARC4_Close

R_SCE_ARC4_Close (cipher_ctrl_t *const p_ctrl)

詳細説明

ARC4 初期化

表 959: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE リソースがビジーです。
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE 内部 I/O が空ではありません。

R_SCE_ARC4_2048Encrypt

R_SCE_ARC4_2048Encrypt (cipher_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

ARC4 暗号化。

詳細説明

2048 ビット ARC4 キーを使用した暗号化入力データ

8.39.9.4 SCE DSA

基本暗号関数。

基本暗号関数 // (L=2048,N=256) DSA。

DSA 署名生成および検証関数

DSA 署名生成および検証関数 // (L=1024,N=160) DSA

DSA 署名生成および検証関数 // (L=2048,N=224) DSA

Functions

- R_SCE_DSA_Open
- R_SCE_DSA_VersionGet
- R_SCE_DSA_Close
- R_SCE_DSA_1024_160SignatureVerify
- R_SCE_DSA_1024_160SignatureGenerate
- R_SCE_DSA_1024_160HashSignatureVerify
- R_SCE_DSA_1024_160HashSignatureGenerate
- R_SCE_DSA_2048_224SignatureVerify
- R_SCE_DSA_2048_224SignatureGenerate
- R_SCE_DSA_2048_224HashSignatureVerify
- R_SCE_DSA_2048_224HashSignatureGenerate
- R_SCE_DSA_2048_256SignatureVerify
- R_SCE_DSA_2048_256SignatureGenerate
- R_SCE_DSA_2048_256HashSignatureVerify
- R_SCE_DSA_2048_256HashSignatureGenerate

変数

- g_dsa1024_160_on_sce
- g_dsa2048_224_on_sce
- g_dsa2048_256_on_sce

定義

- #define SCE_DSA_CODE_VERSION_MAJOR

  初期値 :(01)
- #define SCE_DSA_CODE_VERSION_MINOR

  初期値 :(00)
- #define DSA_PARAM_LENGTH_Q

  初期値 :(160)
- #define DSA_PARAM_LENGTH_P

  初期値 :(1024)
- #define DSA_PARAM_LENGTH_G

  初期値 :(DSA_PARAM_LENGTH_P)
- #define DSA_PRIVATE_KEYSIZE

  初期値 :((DSA_PARAM_LENGTH_Q) / 32)
- #define DSA_PUBLIC_KEYSIZE

  初期値 :((DSA_PARAM_LENGTH_P) / 32)
- #define DSA_DOMAIN_SIZE

  初期値 :(DSA_PRIVATE_KEYSIZE + 2 * (DSA_PUBLIC_KEYSIZE))
- #define DSA_SIGNATURE_SIZE

  初期値 :(2 * (DSA_PRIVATE_KEYSIZE))

dsa_domain_1024_160_t

dsa_domain_1024_160_t

詳細説明
変数

- uint32_t q[(160/32)]

  DSA (1024,160) ドメイン パラメータ Q。
- uint32_t p[(1024/32)]

  DSA (1024,160) ドメイン パラメータ P。
- uint32_t g[(1024/32)]

  DSA (1024,160) ドメイン パラメータ G。

dsa_signature_1024_160_t

dsa_signature_1024_160_t

詳細説明
変数

- uint32_t r[(160/32)]

  DSA (1024,160) 署名コンポーネント R。

- uint32_t s[(160/32)]

  DSA (1024,160) 署名コンポーネント S。

dsa_domain_2048_224_t

dsa_domain_2048_224_t

詳細説明
変数

- uint32_t q[(224/32)]

  DSA (2048,224) ドメイン パラメータ Q。

- uint32_t p[(2048/32)]

  DSA (2048,224) ドメイン パラメータ P。

- uint32_t g[(2048/32)]

  DSA (2048,224) ドメイン パラメータ G。

dsa_signature_2048_224_t

dsa_signature_2048_224_t

詳細説明
変数

- uint32_t r[(224/32)]

  DSA (2048,224) 署名コンポーネント R。

- uint32_t s[(224/32)]

  DSA (2048,224) 署名コンポーネント S。

dsa_domain_2048_256_t

dsa_domain_2048_256_t

詳細説明
変数

- uint32_t q[(256/32)]

  DSA (2048,256) ドメイン パラメータ Q。

- uint32_t p[(2048/32)]

  DSA (2048,256) ドメイン パラメータ P。

- uint32_t g[(2048/32)]

  DSA (2048,256) ドメイン パラメータ G。

dsa_signature_2048_256_t

dsa_signature_2048_256_t

詳細説明

変数

- uint32_t r[(256/32)]

  DSA (2048,256) 署名コンポーネント R。

- uint32_t s[(256/32)]

  DSA (2048,256) 署名コンポーネント S。

g_dsa1024_160_on_sce

dsa_api_t::g_dsa1024_160_on_sce

詳細説明

DSA API の SCE/DSA 実装。

次のように初期化されます

```
g_dsa1024_160_on_sce=
{
   .open      = R_SCE_DSA_Open,
   .sign      = R_SCE_DSA_1024_160SignatureGenerate,
   .verify    = R_SCE_DSA_1024_160SignatureVerify,
   .close     = R_SCE_DSA_Close,
   .versionGet = R_SCE_DSA_VersionGet,
   .hashSign   = R_SCE_DSA_1024_160HashSignatureGenerate,
   .hashVerify =R_SCE_DSA_1024_160HashSignatureVerify
}
```

g_dsa2048_224_on_sce

dsa_api_t::g_dsa2048_224_on_sce

詳細説明

DSA API の SCE/DSA 実装。

次のように初期化されます

```
g_dsa2048_224_on_sce=
{
  .open      = R_SCE_DSA_Open,
  .sign      = R_SCE_DSA_2048_224SignatureGenerate,
  .verify    = R_SCE_DSA_2048_224SignatureVerify,
  .close     = R_SCE_DSA_Close,
  .versionGet = R_SCE_DSA_VersionGet,
  .hashSign   = R_SCE_DSA_2048_224HashSignatureGenerate,
  .hashVerify = R_SCE_DSA_2048_224HashSignatureVerify,
}
```

g_dsa2048_256_on_sce

dsa_api_t::g_dsa2048_256_on_sce

詳細説明

DSA API の SCE/DSA 実装。

次のように初期化されます

```
g_dsa2048_256_on_sce=
{
  .open      = R_SCE_DSA_Open,
  .sign      = R_SCE_DSA_2048_256SignatureGenerate,
  .verify    = R_SCE_DSA_2048_256SignatureVerify,
  .close     = R_SCE_DSA_Close,
  .versionGet = R_SCE_DSA_VersionGet,
  .hashSign   = R_SCE_DSA_2048_256HashSignatureGenerate,
  .hashVerify =R_SCE_DSA_2048_256HashSignatureVerify
}
```

R_SCE_DSA_Open

R_SCE_DSA_Open (dsa_ctrl_t *const p_ctrl , dsa_cfg_t const *const p_cfg)

詳細説明

DSA 初期化

表 960: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_DSA_VersionGet

R_SCE_DSA_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 961: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_DSA_Close

R_SCE_DSA_Close (dsa_ctrl_t *const p_ctrl)

詳細説明

DSA ドライバを閉じます

表 962: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_DSA_1024_160SignatureVerify

R_SCE_DSA_1024_160SignatureVerify (const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

(1024 ビット ,160 ビット) DSA 公開キーを使用した署名検証。

詳細説明

バッファ p_key からの (L=1024,N=160) DSA 公開キーを使用して、長さ 2*imaxcnt ワードのバッファ p_signature からの DSA 署名データを検証します。バッファ p_paddedHash は、DSA 署名が生成されているはずのメッセージ バッファを示します。

表 963: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了、有効な署名
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	署名が正しくありません
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

I :p_key バッファには 32 ワードの DSA 公開キー データが保存されている必要があります

I :p_domain バッファには、(Q || P || G) というフォーマットで (20+128+128) バイトのデータが格納されている必要があります。Q は 5 ワード、P は 32 ワード、G は 32 ワードでなければなりません。

R_SCE_DSA_1024_160SignatureGenerate

R_SCE_DSA_1024_160SignatureGenerate (const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

(1024 ビット ,160 ビット) DSA 秘密キーを使用した署名生成。

詳細説明

バッファ p_key からの (L=1024,N=160) ビット DSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、2*imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 964: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

I :p_dest には、2*imaxcnt ワード以上のデータを保持するスペースが必要です。

I :p_key バッファには、有効な DSA 公開キー データが格納されている必要があります。p_domain には (Q || P || G) という順序で DSA ドメインが格納されている必要があります。Q は 5 ワード、P は 32 ワード、G は 32 ワードでなければなりません。

R_SCE_DSA_1024_160HashSignatureVerify

R_SCE_DSA_1024_160HashSignatureVerify (dsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

(1024 ビット ,160 ビット) DSA 公開キーを使用した署名検証。

詳細説明

バッファ p_key からの (L=1024,N=160) DSA 公開キーを使用して、長さ 2*imaxcnt ワードのバッファ p_signature からの DSA 署名データを検証します。バッファ p_paddedHash は、DSA 署名が生成されているはずのメッセージ バッファを示します。

表 965: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了、有効な署名

表 965: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	署名が正しくありません
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_key バッファには 32 ワードの DSA 公開キー データが保存されている必要があります

l :p_domain バッファには、(Q || P || G) というフォーマットで (20+128+128) バイトのデータが格納されている必要があります。Q は 5 ワード、P は 32 ワード、G は 32 ワードでなければなりません。

R_SCE_DSA_1024_160HashSignatureGenerate

R_SCE_DSA_1024_160HashSignatureGenerate (dsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

(1024 ビット ,160 ビット) DSA 秘密キーを使用した署名生成。

詳細説明

バッファ p_key からの (L=1024,N=160) ビット DSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、2*imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 966: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 966: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、2*imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、有効な DSA 公開キー データが格納されている必要があります。p_domain には (Q || P || G) という順序で DSA ドメインが格納されている必要があります。Q は 5 ワード、P は 32 ワード、G は 32 ワードでなければなりません。

R_SCE_DSA_2048_224SignatureVerify

R_SCE_DSA_2048_224SignatureVerify (const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

(2048 ビット ,224 ビット) DSA 公開キーを使用した署名検証。

詳細説明

バッファ p_key からの (L=2048,N=224) DSA 公開キーを使用して、長さ num_words のバッファ p_signature からの DSA 署名データを検証します。バッファ p_paddedHash は、DSA 署名が生成されているはずのメッセージ バッファを示します。

表 967: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Oepn() を呼び出すことによって初期化されている必要があります

l :p_key バッファには、128 バイトの DSA 公開キー データが格納されている必要があります。

l :p_domain バッファには、(Q || P || G) という形式で (28+256+256) バイトのデータが格納されている必要があります。

指定された DSA 公開キー p_key と、入力メッセージ ハッシュ p_paddedHash 用の p_domain からのドメイン パラメータを使用して、バッファ p_signature からの DSA 署名を検証します

R_SCE_DSA_2048_224SignatureGenerate

R_SCE_DSA_2048_224SignatureGenerate (const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

(2048 ビット ,224 ビット) DSA 秘密キーを使用した署名生成。

詳細説明

バッファ key からの (L=2048,N=224) ビット DSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 968: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、2*imaxcnt ワード以上のデータを保持するスペースが必要です。

l :key バッファには 7 ワードの DSA 秘密キー データが保存されている必要があります。

l :p_domain には有効な DSA ドメイン パラメータが含まれていなければなりません

R_SCE_DSA_2048_224HashSignatureVerify

R_SCE_DSA_2048_224HashSignatureVerify (dsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

(2048 ビット ,224 ビット) DSA 公開キーを使用した署名検証。

詳細説明

指定された DSA 公開キー p_key と、入力メッセージ ハッシュ p_paddedHash 用の p_domain からのドメイン パラメータを使用して、バッファ p_signature からの DSA 署名を検証します

R_SCE_DSA_2048_224HashSignatureGenerate

R_SCE_DSA_2048_224HashSignatureGenerate (dsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

(2048 ビット ,224 ビット) DSA 秘密キーを使用した署名生成。

詳細説明

バッファ key からの (L=2048,N=224) ビット DSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 969: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、2*imaxcnt ワード以上のデータを保持するスペースが必要です。

l :key バッファには 7 ワードの DSA 秘密キー データが保存されている必要があります。

l :p_domain には有効な DSA ドメイン パラメータが含まれていなければなりません

R_SCE_DSA_2048_256SignatureVerify

R_SCE_DSA_2048_256SignatureVerify (const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

(2048 ビット ,256 ビット) DSA 公開キーを使用した署名検証。

詳細説明

指定された DSA 公開キー p_key と、入力メッセージ ハッシュ p_paddedHash 用の p_domain からのドメイン パラメータを使用して、バッファ p_signature からの DSA 署名を検証します

バッファ p_key からの (L=2048,N=256) DSA 公開キーを使用して、長さ num_words のバッファ p_signature からの DSA 署名データを検証します。バッファ p_paddedHash は、DSA 署名が生成されているはずのメッセージ バッファを示します。

表 970: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_key バッファには 256 バイトの DSA 公開キー データが格納されている必要があります

l :p_domain バッファには、(Q || P || G) という形式で (32+256+256) バイトのデータが格納されている必要があります。

R_SCE_DSA_2048_256SignatureGenerate

R_SCE_DSA_2048_256SignatureGenerate (const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

(2048 ビット ,256 ビット) DSA 秘密キーを使用した署名生成。

詳細説明

ドメイン パラメータ p_domain に対して指定された DSA 秘密キー p_key を使用して、バッファ p_paddedHash に対する署名を生成します。結果はバッファ p_dest に書き込まれます

バッファ key からの (L=2048,N=256) ビット DSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 971: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

- :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

- :p_dest には、2*imaxcnt ワード以上のデータを保持するスペースが必要です。

- :key バッファには 8 ワードの DSA 秘密キー データが保存されている必要があります。

R_SCE_DSA_2048_256HashSignatureVerify

R_SCE_DSA_2048_256HashSignatureVerify (dsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

(2048 ビット ,256 ビット) DSA 公開キーを使用した署名検証。

詳細説明

指定された DSA 公開キー p_key と、入力メッセージ ハッシュ p_paddedHash 用の p_domain からのドメイン パラメータを使用して、バッファ p_signature からの DSA 署名を検証します

バッファ p_key からの (L=2048,N=256) DSA 公開キーを使用して、長さ num_words のバッファ p_signature からの DSA 署名データを検証します。バッファ p_paddedHash は、DSA 署名が生成されているはずのメッセージ バッファを示します。

表 972: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_key バッファには 256 バイトの DSA 公開キー データが格納されている必要があります

l :p_domain バッファには、(Q || P || G) という形式で (32+256+256) バイトのデータが格納されている必要があります。

R_SCE_DSA_2048_256HashSignatureGenerate

R_SCE_DSA_2048_256HashSignatureGenerate (dsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

(2048 ビット ,256 ビット) DSA 秘密キーを使用した署名生成。

詳細説明

ドメイン パラメータ p_domain に対して指定された DSA 秘密キー p_key を使用して、バッファ p_paddedHash に対する署名を生成します。結果はバッファ p_dest に書き込まれます

バッファ key からの (L=2048,N=256) ビット DSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 973: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

- :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

- :p_dest には、2*imaxcnt ワード以上のデータを保持するスペースが必要です。

- :key バッファには 8 ワードの DSA 秘密キー データが保存されている必要があります。

8.39.9.5 SCE_RSA

基本暗号関数。

暗号化、復号化、署名生成、および署名検証関数用の RSA 1024 ビット実装

暗号化、復号化、署名生成、および署名検証関数用の RSA 2048 ビット実装

Functions

- R_SCE_RSA_Open
- R_SCE_RSA_VersionGet
- R_SCE_RSA_Close
- R_SCE_RSA_1024PublicKeyEncrypt
- R_SCE_RSA_1024PublicKeyVerify
- R_SCE_RSA_1024PrivateKeyDecrypt
- R_SCE_RSA_1024PrivateKeySign
- R_SCE_RSA_1024PrivateCrtKeyDecrypt
- R_SCE_RSA_1024PrivateCrtKeySign
- R_SCE_RSA_2048PublicKeyEncrypt
- R_SCE_RSA_2048PublicKeyVerify
- R_SCE_RSA_2048PrivateKeyDecrypt
- R_SCE_RSA_2048PrivateKeySign
- R_SCE_RSA_2048PrivateCrtKeyDecrypt
- R_SCE_RSA_2048PrivateCrtKeySign

変数

- g_rsa1024_on_sce
- g_rsa2048_on_sce

定義

- #define SCE_RSA_CODE_VERSION_MAJOR

  初期値 :(01)

- #define SCE_RSA_CODE_VERSION_MINOR

  初期値 :(00)

- #define RSA_KEYLENGTH

  初期値 :(1024)

  RSA キーの長さ （ビット単位）

- #define UINT32_BITS

  初期値 :(32)

  uint32_t word size in bits

- #define RSA_MODULUS_SIZE

  初期値 :((#define RSA_KEYLENGTH)/(#define UINT32_BITS))

  ワード単位の rsa 係数データ サイズ

- #define RSA_PRIVATE_EXPONENT_SIZE

  初期値 :(#define RSA_MODULUS_SIZE)

  ワード単位の rsa 秘密指数データ サイズ

- #define RSA_PUBLIC_EXPONENT_SIZE

  初期値 :(1)

  ワード単位の rsa 公開指数データ サイズ

- #define RSA_PUBLIC_KEYSIZE

  初期値 :((#define RSA_PUBLIC_EXPONENT_SIZE)+(#define RSA_MODULUS_SIZE))

  ワード単位の rsa 公開キー サイズ

- #define RSA_PRIVATE_KEYSIZE

  初期値 :((#define RSA_PRIVATE_EXPONENT_SIZE)+(#define RSA_MODULUS_SIZE))

  ワード単位の rsa 秘密キー サイズ

- #define RSA_PRIVATE_CRT_KEYSIZE

  初期値 :((5*(#define RSA_MODULUS_SIZE))/2)

  ワード単位の rsa 秘密キー （CRT フォーマット）サイズ

g_rsa1024_on_sce

rsa_api_t::g_rsa1024_on_sce

詳細説明

RSA API の SCE/RSA 実装。

次のように初期化されます

```
g_rsa1024_on_sce=
{
  .open      = R_SCE_RSA_Open,
  .encrypt   = R_SCE_RSA_1024PublicKeyEncrypt,
  .decrypt   = R_SCE_RSA_1024PrivateKeyDecrypt,
  .decryptCrt = R_SCE_RSA_1024PrivateCrtKeyDecrypt,
  .sign      = R_SCE_RSA_1024PrivateKeySign,
  .signCrt   = R_SCE_RSA_1024PrivateCrtKeySign,
  .verify    = R_SCE_RSA_1024PublicKeyVerify,
  .close     = R_SCE_RSA_Close,
  .versionGet =R_SCE_RSA_VersionGet
}
```

g_rsa2048_on_sce

rsa_api_t::g_rsa2048_on_sce

次のように初期化されます

```
g_rsa2048_on_sce=
{
  .open      = R_SCE_RSA_Open,
  .encrypt   = R_SCE_RSA_2048PublicKeyEncrypt,
  .decrypt   = R_SCE_RSA_2048PrivateKeyDecrypt,
  .decryptCrt = R_SCE_RSA_2048PrivateCrtKeyDecrypt,
  .sign      = R_SCE_RSA_2048PrivateKeySign,
  .signCrt   = R_SCE_RSA_2048PrivateCrtKeySign,
  .verify    = R_SCE_RSA_2048PublicKeyVerify,
  .close     = R_SCE_RSA_Close,
  .versionGet =R_SCE_RSA_VersionGet
}
```

R_SCE_RSA_Open

R_SCE_RSA_Open (rsa_ctrl_t *const p_ctrl , rsa_cfg_t const *const p_cfg)

詳細説明

RSA 初期化

表 974: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_RSA_VersionGet

R_SCE_RSA_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 975: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_RSA_Close

R_SCE_RSA_Close (rsa_ctrl_t *const p_ctrl)

詳細説明

RSA ドライバを閉じます

表 976: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE リソースがビジーです。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE 内部 I/O が空ではありません。

R_SCE_RSA_1024PublicKeyEncrypt

R_SCE_RSA_1024PublicKeyEncrypt (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

1024 ビット RSA 公開キーを使用した暗号化。

詳細説明

バッファ p_key からの 1024 ビット RSA 公開キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 977: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、(public_exponent || public_modulus) というフォーマットで 132 バイトの RSA 公開キー データが格納されている必要があります。public_exponent は 1 ワードのデータで、public_modulus は 32 ワードのデータです

R_SCE_RSA_1024PublicKeyVerify

R_SCE_RSA_1024PublicKeyVerify (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t num_words , uint32_t * p_signature , uint32_t * p_padded_hash)

概要説明

1024 ビット RSA 公開キーを使用した署名検証。

詳細説明

1024 ビット RSA 公開キーを使用して、長さ num_words のバッファ p_signature からの RSA 署名データを検証します。バッファ p_padded_hash は、RSA 署名が生成されているはずのメッセージ バッファを示します。

表 978:　戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了、有効な署名
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	署名が正しくありません
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

I :p_key バッファには、(public_exponent || public_modulus) というフォーマットで 132 バイトの RSA 公開キー データが格納されている必要があります。public_exponent は 1 ワードで、public_modulus は 32 ワードです。

R_SCE_RSA_1024PrivateKeyDecrypt

R_SCE_RSA_1024PrivateKeyDecrypt (`rsa_ctrl_t` *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

1024 ビット RSA 秘密キー （標準フォーマット）を使用した復号化

詳細説明

バッファ p_key からの 1024 ビット RSA 秘密キーを使用して、バッファ p_source からの入力データの imaxcnt ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 979:　戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 979: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、32 ワードの RSA 秘密キー データと、それに続く 32 ワードの RSA キー係数データが保存されている必要があります

R_SCE_RSA_1024PrivateKeySign

R_SCE_RSA_1024PrivateKeySign (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

1024 ビット RSA 秘密キーを使用した署名生成。

詳細説明

バッファ p_key からの 1024 ビット RSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 980: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 980: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、32 ワードの RSA 秘密キー データと、それに続く 32 ワードの RSA キー係数データが保存されている必要があります

R_SCE_RSA_1024PrivateCrtKeyDecrypt

R_SCE_RSA_1024PrivateCrtKeyDecrypt (`rsa_ctrl_t` *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

1024 ビット RSA 秘密キー （CRT フォーマット）を使用した復号化

詳細説明

バッファ key からの 1024 ビット RSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 981: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 981: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

I :SCE モジュールが R_SCE_Open() により初期化されている必要があります

I :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

I :1024 ビット RSA キー CRT パラメータ (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p) を含む 80 ワード バッファへの p_key ポインタ

R_SCE_RSA_1024PrivateCrtKeySign

R_SCE_RSA_1024PrivateCrtKeySign (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

CRT フォーマットで表された 1024 ビットの RSA 秘密キーを使用した署名生成。

詳細説明

バッファ p_key からの 1024 ビット RSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 982: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 982: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :1024 ビット RSA キー CRT パラメータ (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p) を含む 80 ワード バッファへの p_key ポインタ

R_SCE_RSA_2048PublicKeyEncrypt

R_SCE_RSA_2048PublicKeyEncrypt (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

2048 ビット RSA 公開キーを使用した暗号化。

詳細説明

バッファ p_key からの 2048 ビット RSA 公開キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 983: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、(public_exponent || public_modulus) というフォーマットで 260 バイトの RSA 公開キー データが格納されている必要があります。public_exponent は 1 ワードのデータで、public_modulus は 64 ワードのデータです

R_SCE_RSA_2048PublicKeyVerify

R_SCE_RSA_2048PublicKeyVerify (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_signature , uint32_t * p_paddedHash)

概要説明

2048 ビット RSA 公開キーを使用した署名検証。

詳細説明

2048 ビット RSA 公開キーを使用して、長さ num_words のバッファ p_signature からの RSA 署名データを検証します。バッファ p_padded_hash は、RSA 署名が生成されているはずのメッセージ バッファを示します。

表 984: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了、有効な署名
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	署名が正しくありません
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

I :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

I :p_key バッファには、(public_exponent || public_modulus) というフォーマットで 260 バイトの RSA 公開キー データが格納されている必要があります。public_exponent は 1 ワードで、public_modulus は 64 ワードです。

R_SCE_RSA_2048PrivateKeyDecrypt

R_SCE_RSA_2048PrivateKeyDecrypt (`rsa_ctrl_t` *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

2048 ビット RSA 秘密キー （標準フォーマット）を使用した復号化

詳細説明

バッファ p_key からの 2048 ビット RSA 秘密キーを使用して、バッファ p_source からの入力データの imaxcnt ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 985: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 985: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、32 ワードの RSA 秘密キー データと、それに続く 64 ワードの RSA キー係数データが保存されている必要があります

R_SCE_RSA_2048PrivateKeySign

R_SCE_RSA_2048PrivateKeySign (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

2048 ビット RSA 秘密キーを使用した署名生成。

詳細説明

バッファ p_key からの 2048 ビット RSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 986: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 986: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、32 ワードの RSA 秘密キー データと、それに続く 64 ワードの RSA キー係数データが保存されている必要があります

R_SCE_RSA_2048PrivateCrtKeyDecrypt

R_SCE_RSA_2048PrivateCrtKeyDecrypt (rsa_ctrl_t *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

2048 ビット RSA 秘密キー （CRT フォーマット）を使用した復号化

詳細説明

バッファ p_key からの 2048 ビット RSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 987: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました

I :SCE モジュールが R_SCE_Open() により初期化されている必要があります

I :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

I :p_key バッファには、2048 ビット RSA キー CRT パラメータ (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p) を含む 160 ワード バッファへのポインタが格納されている必要があります

R_SCE_RSA_2048PrivateCrtKeySign

R_SCE_RSA_2048PrivateCrtKeySign (`rsa_ctrl_t` *const p_ctrl , const uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t * p_source , uint32_t * p_dest)

概要説明

CRT フォーマットで表された 2048 ビットの RSA 秘密キーを使用した署名生成。

詳細説明

バッファ p_key からの 2048 ビット RSA 秘密キーとバッファ p_domain からのドメイン パラメータを使用して、バッファ p_source からの入力データの imaxcnt ワードを署名します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、imaxcnt ワード以上のデータ用のスペースがあると見なされます。

表 988: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

l :SCE モジュールは関数 R_SCE_Open() を呼び出すことによって初期化されている必要があります

l :p_dest には、imaxcnt ワード以上のデータを保持するスペースが必要です。

l :p_key バッファには、2048 ビット RSA キー CRT パラメータ (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p) を含む 160 ワード バッファへのポインタが格納されている必要があります

8.39.9.6 SCE_HASH

基本暗号関数。

このドライバ固有の SCE_SHA1 定義 （多くがインタフェースに対する拡張)。SCE_SHA1 の例 : extern const r_sce_t g_<interface>_on_sce;

メッセージ ハッシュ / ダイジェスト関数

Functions

- R_SCE_SHA1_Open
- R_SCE_SHA1_VersionGet
- R_SCE_SHA1_UpdateHash
- R_SCE_SHA1_HashUpdate

- R_SCE_SHA1_Close
- R_SCE_SHA256_Open
- R_SCE_SHA256_VersionGet
- R_SCE_SHA256_UpdateHash
- R_SCE_SHA256_HashUpdate
- R_SCE_SHA256_Close

変数

- g_sha1_hash_on_sce
- g_sha256_hash_on_sce

定義

- #define SCE_SHA256_CODE_VERSION_MAJOR

  初期値 :(01)
- #define SCE_SHA256_CODE_VERSION_MINOR

  初期値 :(00)

g_sha1_hash_on_sce

hash_api_t::g_sha1_hash_on_sce

詳細説明

HASH API の SHA1 実装。

次のように初期化されます

```
g_sha1_hash_on_sce=
{
  .open      = R_SCE_SHA1_Open,
  .updateHash = R_SCE_SHA1_UpdateHash,
  .close     = R_SCE_SHA1_Close,
  .versionGet = R_SCE_SHA1_VersionGet,
  .hashUpdate =R_SCE_SHA1_HashUpdate
}
```

g_sha256_hash_on_sce

hash_api_t::g_sha256_hash_on_sce

詳細説明

HASH API の SHA256 実装。

次のように初期化されます

```
g_sha256_hash_on_sce=
{
  .open      = R_SCE_SHA256_Open,
  .updateHash = R_SCE_SHA256_UpdateHash,
  .close     = R_SCE_SHA256_Close,
  .versionGet = R_SCE_SHA256_VersionGet,
  .hashUpdate =R_SCE_SHA256_HashUpdate
}
```

R_SCE_SHA1_Open

R_SCE_SHA1_Open (hash_ctrl_t *const p_ctrl , hash_cfg_t const *const p_cfg)

詳細説明

SCE ブロックを使用した SCE SHA1 Open 関数

SHA1 HASH 初期化

表 989: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常に開かれました。

R_SCE_SHA1_VersionGet

R_SCE_SHA1_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

SCE SHA1 のバージョン取得

表 990: パラメータ

名前	方向	説明
p_version	複数のビットを書き換えることもできます。	バージョン情報が格納される ssp_version_t 構造体へのポインタ

表 991: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_SHA1_UpdateHash

R_SCE_SHA1_UpdateHash (const uint32_t * p_msg , uint32_t imaxcnt , uint32_t * p_digest)

詳細説明

長さが num_words ワードの指定された入力メッセージ バッファ p_msg の SHA1 メッセージ ダイジェストを計算します。メッセージ バッファの長さは 64 バイトの倍数にする必要があります。通常、メッセージ バッファの内容は「メッセージ || ストップ ビット || ゼロ パディング || メッセージ長」で指定されたようにパディングされた値です。

バッファ p_digest で指定されたように初期ハッシュ値が使用され、このバッファは計算された SHA1 メッセージ ダイジェスト値で更新されます。

表 992: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_SHA1_HashUpdate

R_SCE_SHA1_HashUpdate (hash_ctrl_t *const p_ctrl , const uint32_t * p_msg , uint32_t num_words , uint32_t * p_digest)

詳細説明

長さが num_words ワードの指定された入力メッセージ バッファ p_msg の SHA1 メッセージ ダイジェストを計算します。メッセージ バッファの長さは 64 バイトの倍数にする必要があります。通常、メッセージ バッファの内容は「メッセージ || ストップ ビット || ゼロ パディング || メッセージ長」で指定されたようにパディングされた値です。

バッファ p_digest で指定されたように初期ハッシュ値が使用され、このバッファは計算された SHA1 メッセージ ダイジェスト値で更新されます。

表 993: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_SHA1_Close

R_SCE_SHA1_Close (`hash_ctrl_t` *const p_ctrl)

詳細説明

HASH 初期化

表 994: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	SCE SHA1 Close 関数が正常に閉じられました。

表 995: パラメータ

名前	方向	説明
p_ctrl	複数のビットを書き換えることもできます。	SCE SHA1 の制御構造体

R_SCE_SHA256_Open

R_SCE_SHA256_Open (`hash_ctrl_t` *const p_ctrl , `hash_cfg_t` const *const p_cfg)

詳細説明

このドライバ固有の SCE_SHA256 定義 （多くがインタフェースに対する拡張)。SCE_SHA256 の例 :
extern const r_sce_t g_<interface>_on_sce;

SCE SHA256 HASH 初期化

表 996: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常に完了しました

R_SCE_SHA256_VersionGet

R_SCE_SHA256_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 997: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_SHA256_UpdateHash

R_SCE_SHA256_UpdateHash (const uint32_t * p_msg , uint32_t imaxcnt , uint32_t * p_digest)

概要説明

num_words ワードのバッファ p_source からの指定された入力メッセージを使用してハッシュ値を更新し、結果を p_digest に書き込みます

詳細説明

長さが num_words ワードの指定された入力メッセージ バッファ p_msg の SHA256 メッセージ ダイジェストを計算します。メッセージ バッファの長さは 64 バイトの倍数にする必要があります。通常、メッセージ バッファの内容は「メッセージ || ストップ ビット || ゼロ パディング || メッセージ長」で指定されたようにパディングされた値です。

バッファ p_digest で指定されたように初期ハッシュ値が使用され、このバッファは計算された SHA256 メッセージ ダイジェスト値で更新されます。

表 998: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了

表 998: 戻り値 (続き)

名前	説明
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_SHA256_HashUpdate

R_SCE_SHA256_HashUpdate (hash_ctrl_t *const p_ctrl , const uint32_t * p_msg , uint32_t imaxcnt , uint32_t * p_digest)

概要説明

num_words ワードのバッファ p_source からの指定された入力メッセージを使用してハッシュ値を更新し、結果を p_digest に書き込みます

詳細説明

長さが num_words ワードの指定された入力メッセージ バッファ p_msg の SHA256 メッセージ ダイジェストを計算します。メッセージ バッファの長さは 64 バイトの倍数にする必要があります。通常、メッセージ バッファの内容は「メッセージ || ストップ ビット || ゼロ パディング || メッセージ長」で指定されたようにパディングされた値です。

バッファ p_digest で指定されたように初期ハッシュ値が使用され、このバッファは計算された SHA256 メッセージ ダイジェスト値で更新されます。

表 999: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	正常終了
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	内部 I/O バッファが空ではありません。
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	リソース競合が発生しました。

R_SCE_SHA256_Close

R_SCE_SHA256_Close (hash_ctrl_t *const p_ctrl)

詳細説明

SCE SHA256 Close 関数

表 1000: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました

8.39.9.7 SCE_TDES

基本暗号関数。

Triple DES 暗号化および復号化関数

Functions

- R_SCE_TDES_Open
- R_SCE_TDES_VersionGet
- R_SCE_TDES_Close
- R_SCE_TDES_192CbcEncrypt
- R_SCE_TDES_192CbcDecrypt
- R_SCE_TDES_192CtrEncrypt
- R_SCE_TDES_192EcbEncrypt
- R_SCE_TDES_192EcbDecrypt

変数

- g_tdes192ecb_on_sce
- g_tdes192cbc_on_sce
- g_tdes192ctr_on_sce

定義

- #define SCE_TDES_CODE_VERSION_MAJOR

  初期値 :(01)
- #define SCE_TDES_CODE_VERSION_MINOR

  初期値 :(00)

g_tdes192ecb_on_sce

tdes_api_t::g_tdes192ecb_on_sce

詳細説明

TDES API の SCE/TDES 実装。

次のように初期化されます

```
g_tdes192ecb_on_sce=
{
   .open      = R_SCE_TDES_Open,
   .encrypt   = R_SCE_TDES_192EcbEncrypt,
   .decrypt   = R_SCE_TDES_192EcbDecrypt,
   .close     = R_SCE_TDES_Close,
   .versionGet =R_SCE_TDES_VersionGet
}
```

g_tdes192cbc_on_sce

tdes_api_t::g_tdes192cbc_on_sce

次のように初期化されます

```
g_tdes192cbc_on_sce=
{
   .open      = R_SCE_TDES_Open,
   .encrypt   = R_SCE_TDES_192CbcEncrypt,
   .decrypt   = R_SCE_TDES_192CbcDecrypt,
   .close     = R_SCE_TDES_Close,
   .versionGet =R_SCE_TDES_VersionGet
}
```

g_tdes192ctr_on_sce

tdes_api_t::g_tdes192ctr_on_sce

次のように初期化されます

```
g_tdes192ctr_on_sce=
{
   .open      = R_SCE_TDES_Open,
   .encrypt   = R_SCE_TDES_192CtrEncrypt,
   .decrypt   = R_SCE_TDES_192CtrEncrypt,
   .close     = R_SCE_TDES_Close,
   .versionGet =R_SCE_TDES_VersionGet
}
```

R_SCE_TDES_Open

R_SCE_TDES_Open (tdes_ctrl_t *const p_ctrl , tdes_cfg_t const *const p_cfg)

詳細説明

TDES 初期化

表 1001: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE リソースがビジーです。
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE 内部 I/O が空ではありません。

R_SCE_TDES_VersionGet

R_SCE_TDES_VersionGet (`ssp_version_t` *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 1002: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_TDES_Close

R_SCE_TDES_Close (`tdes_ctrl_t` *const p_ctrl)

詳細説明

Close TDES

表 1003: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE リソースがビジーです。
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE 内部 I/O が空ではありません。

R_SCE_TDES_192CbcEncrypt

R_SCE_TDES_192CbcEncrypt (tdes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

CBC モードを使用した Triple DES 暗号化。

詳細説明

バッファ key からの 192 ビット TDES キーとバッファ iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 1004: 戻り値

名前	説明
PASS	正常終了
FAIL	内部 I/O バッファが空ではありません。
RESOURCE_CONFLICT	リソース競合が発生しました。

l : 関数の R_SCE_SoftReset()、R_SCE_Initialization1()、および R_SCE_Initialization2() を呼び出すことによって、SCE モジュールが初期化され、RNG 品質チェックが実行されている必要があります。

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :key バッファには 16 バイトの TDES キー データが保存され、

l :iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

192 ビット TDES キーを使用して CBC モードで入力データを暗号化します

R_SCE_TDES_192CbcDecrypt

R_SCE_TDES_192CbcDecrypt (tdes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

CBC モードを使用した Triple DES 復号化。

詳細説明

バッファ key からの 192 ビット TDES キーとバッファ iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 1005: 戻り値

名前	説明
PASS	正常終了
FAIL	内部 I/O バッファが空ではありません。
RESOURCE_CONFLICT	リソース競合が発生しました。

l : 関数の R_SCE_SoftReset()、R_SCE_Initialization1()、および R_SCE_Initialization2() を呼び出すことによって、SCE モジュールが初期化され、RNG 品質チェックが実行されている必要があります。

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :key バッファには 16 バイトの TDES キー データが保存され、

l :iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

192 ビット TDES キーを使用して CBC モードで入力データを復号化します

R_SCE_TDES_192CtrEncrypt

R_SCE_TDES_192CtrEncrypt (tdes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

CTR モードを使用した Triple DES 暗号化。

詳細説明

バッファ key からの 192 ビット TDES キーとバッファ iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 1006: 戻り値

名前	説明
PASS	正常終了
FAIL	内部 I/O バッファが空ではありません。
RESOURCE_CONFLICT	リソース競合が発生しました。

l : 関数の R_SCE_SoftReset()、R_SCE_Initialization1()、および R_SCE_Initialization2() を呼び出すことによって、SCE モジュールが初期化され、RNG 品質チェックが実行されている必要があります。

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :key バッファには 16 バイトの TDES キー データが保存され、

l :iv バッファには 16 バイト以上のランダム データが保存されている必要があります。

192 ビット TDES キーを使用して CTR モードで入力データを暗号化します

R_SCE_TDES_192EcbEncrypt

R_SCE_TDES_192EcbEncrypt (tdes_ctrl_t *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

ECB モードを使用した Triple DES 暗号化。

詳細説明

バッファ key からの 192 ビット TDES キーとバッファ iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを暗号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 1007: 戻り値

名前	説明
PASS	正常終了
FAIL	内部 I/O バッファが空ではありません。
RESOURCE_CONFLICT	リソース競合が発生しました

l : 関数の R_SCE_SoftReset()、R_SCE_Initialization1()、および R_SCE_Initialization2() を呼び出すことによって、SCE モジュールが初期化され、RNG 品質チェックが実行されている必要があります。

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :key バッファには 16 バイトの TDES キー データが保存され、

l :ECB チェーニング モードでは、iv バッファの内容が無視されます。

192 ビット TDES キーを使用して ECB モードで入力データを暗号化します

R_SCE_TDES_192EcbDecrypt

R_SCE_TDES_192EcbDecrypt (`tdes_ctrl_t` *const p_ctrl , const uint32_t * p_key , uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t * p_dest)

概要説明

ECB モードを使用した Triple DES 復号化。

詳細説明

バッファ key からの 192 ビット TDES キーとバッファ iv からの初期化ベクターを使用して、バッファ p_source からの入力データの num_words ワードを復号化します。結果は p_dest からの出力バッファに書き込まれます。p_dest 配列は、num_words ワード以上のデータ用のスペースがあると見なされます。

表 1008: 戻り値

名前	説明
PASS	正常終了
FAIL	内部 I/O バッファが空ではありません。
RESOURCE_CONFLICT	リソース競合が発生しました。

l : 関数の R_SCE_SoftReset()、R_SCE_Initialization1()、および R_SCE_Initialization2() を呼び出すことによって、SCE モジュールが初期化され、RNG 品質チェックが実行されている必要があります。

l :p_dest には、num_words ワード以上のデータを保持するスペースが必要です。

l :key バッファには 16 バイトの TDES キー データが保存され、

l :ECB チェーニング モードでは、iv バッファの内容が無視されます。

192 ビット TDES キーを使用して ECB モードで入力データを復号化します

8.39.9.8 SCE_TRNG

基本暗号関数。

乱数生成関数

Functions

- R_SCE_TRNG_Open
- R_SCE_TRNG_VersionGet
- R_SCE_TRNG_Read
- R_SCE_TRNG_Close

変数

- g_trng_on_sce

定義

- #define SCE_TRNG_CODE_VERSION_MAJOR
  初期値 :(01)
- #define SCE_TRNG_CODE_VERSION_MINOR
  初期値 :(00)

g_trng_on_sce

trng_api_t::g_trng_on_sce

詳細説明

RNG API の SCE/TRNG 実装。

次のように初期化されます

```
g_trng_on_sce=
{
  .open      = R_SCE_TRNG_Open,
  .read      = R_SCE_TRNG_Read,
  .close     = R_SCE_TRNG_Close,
  .versionGet =R_SCE_TRNG_VersionGet
}
```

R_SCE_TRNG_Open

R_SCE_TRNG_Open (trng_ctrl_t *const p_ctrl , trng_cfg_t const *const p_cfg)

詳細説明

SCE_TRNG の例 : extern const r_sce_t g_<interface>_on_sce;

ハードウェア TRNG モジュール TRNG 初期化からランダム データを読み取るために TRNG ドライバを開きます。

表 1009: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE リソースがビジーです。
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE 内部 I/O が空ではありません。

R_SCE_TRNG_VersionGet

R_SCE_TRNG_VersionGet (ssp_version_t *const p_version)

概要説明

コンパイル時マクロに基づいて、ドライバのバージョンを設定します。

詳細説明

表 1010: 戻り値

名前	説明
SSP_SUCCESS	正常に閉じました。
p_ctrl の可能性があります。	p_version パラメータは NULL です。

R_SCE_TRNG_Read

R_SCE_TRNG_Read (trng_ctrl_t *const p_ctrl , uint32_t *const p_dest , uint32_t nwords)

詳細説明

num_words の乱数バイトを生成して、それらを p_dest バッファに保存します。SCE ハードウェア TRNG モジュールがランダム データの生成に使用されます。

表 1011: 戻り値

名前	説明
SF_CRYPTO_SUCCESS	乱数生成が成功しました
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE リソースがビジーです。
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE 内部 I/O が空ではありません。

R_SCE_TRNG_Close

R_SCE_TRNG_Close (trng_ctrl_t *const p_ctrl)

詳細説明

TRNG インタフェース ドライバを閉じます

参考資料

第 9 章 API リファレンス：ボード サポート パッケージ

9.1 Board Support Package

共通 BSP が含まれています。

BSP には、リセット以降の MCU からユーザー アプリケーション （メイン関数など）に到達できるようにする責任があります。ユーザー アプリケーションに到達する前に、BSP は、スタック、ヒープ、クロック、割り込み、および C ランタイム環境をセットアップします。BSP は、ユーザーが設計要件を満たすためにプロセスを変更できるように設定することができます。

9.2 サポートされているボード

このバージョンの BSP でサポートされているボード。

BSP 設定によって使用されるボードが決まります。

BSP は複数のボードのサポートを提供します。プロジェクトを使用してビルドされるボード BSP は、使用されている bsp_cfg.h ファイルに基づいて選択されます。各 ssp/src/bsp/board/"board"/ref フォルダで使用するボードの参考用 bsp_cfg.h ファイルが見つかります。

9.2.1 DK-S124 ボードの BSP

DK-S124 ボードの BSP。

DK-S124 は、LQFP64 パッケージに含まれる Renesas SynergyTM S124 マイクロコントローラ用の開発キットです。このボードは、アプリケーションの開発およびテスト用に、S124 のすべての周辺機器を対象としたアクセスしやすいインタフェースとコネクタ （USB デバイス、オーディオ出力 （DAC インタフェース）、温度およびライト センサー （ADC インタフェース）、Bluetooth ラジオ （SPI インタフェース）、SEGGER J-Link オンボード デバッグ、1 つの Pmod コネクタ、複数の DAC インジケーター）を提供します。

9.2.1.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。現時点で、次の設定値を構成するためのヘッダー ファイルが存在します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.1.2 ビルド時間設定 - 全般

このファイルには、全般 BSP 設定用のコンパイル時間設定オプションが保存されます。スタック サイズや ROM レジスタなどをこのファイルで設定します。

部品番号情報

MCU の製品部品番号を入力します。この情報は、パッケージ サイズやメモリ サイズなどの MCU に関する情報の取得に使用されます。この情報を解釈しやすくするために、複数のマクロを使って部品番号を定義します。

```
R7FS 1 2 4 7 7 3 A01 C FM
|  |||||||  || Macro Name                  Description
|  |||||||  ||__BSP_CFG_MCU_PART_PACKAGE     = Package type, number of pins
|  |||||||  |____not used                  = Quality ID
|  |||||||_________not used                   = Software ID
|  ||||||__________not used                   = Operating range
|  |||||_____________BSP_CFG_MCU_PART_MEMORY_SIZE  = ROM, RAM, and Data Flash Capacity
|  ||||______________BSP_CFG_MCU_PART_FEATURE_SET  = Superset, no encryption
|  |||__________________not used                  = Document index
|  ||____________________BSP_CFG_MCU_PART_CORE        = Core & frequency (CM0, 32MHz)
|  |_____________________BSP_CFG_MCU_PART_SERIES      = Performance category (High-performance, Low power)
|________________________________not used                  = Renesas Synergy MCU
```

- #define BSP_CFG_MCU_PART_PACKAGE

  初期値 :(0x1)

  パッケージ タイプ。次の値に基づいてマクロ定義をセットします。

```
Character(s) = Value for macro = Package Type/Number of Pins/Other Info
BJ          = 0x0           = BGA/121
FM          = 0x1           = LQFP/64
FP          = 0x2           = LQFP/100
FB          = 0x3           = LQFP/144
LJ          = 0x4           = LGA/100/7x7
LK          = 0x5           = LGA/145
NB          = 0x6           = WQFN/64
```

- #define BSP_CFG_MCU_PART_MEMORY_SIZE

  初期値 :(0x07)

  ROM、RAM、およびデータ フラッシュ容量。

```
Character(s) = Value for macro = ROM Size/Ram Size/Data Flash Size
6           = 0x06           = 64KB/16KB/4KB
7           = 0x07           = 128KB/16KB/4KB
```

- #define BSP_CFG_MCU_PART_CORE

  初期値 :(0x02)

  コアおよび周波数。

```
Character(s) = Value for macro = Description
2           = 0x02           = CM0+, 32MHz
```

- #define BSP_CFG_MCU_PART_SERIES

  初期値 :(0x1)

  シリーズ

```
Character(s) = Value for macro = Description
1          = 0x1          = Ultra-low Power (up to 32MHz)
```

スタックおよびヒープ サイズの設定

- #define BSP_CFG_STACK_MAIN_BYTES

  初期値 :(0x1000)

  バイト単位のメイン スタック サイズ。これは、リセット以降に使用される必須のスタックです。例外では常にメイン スタックが使用されます。

- #define BSP_CFG_STACK_PROCESS_BYTES

  初期値 :(0)

  バイト単位のプロセス スタック サイズ。プロセス スタックの使用は任意です。BSP はプロセス スタックを使用しません。ユーザーがアプリケーション内でプロセス スタックを使用する場合は、CONTROL レジスタの SPSEL ビットをセットする必要があります。

- #define BSP_CFG_HEAP_BYTES

  初期値 :(0x400)

  バイト単位のヒープ サイズ。

オプション設定メモリ （ROM レジスタ）の設定

特定のレジスタが ROM に保存され、リセット以降の MCU の設定に使用されます。

I：レジスタのデフォルト値を使用するには、マクロをすべて 0xFF のままにします。

I：すべてのオプションがデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_OFS0

  初期値 :(0xFFFFFFFF)

  WDT 設定と IWDT 設定を構成します。OFS0 - オプション関数選択レジスタ 0

- b31 予約済み （1 にセット）
- b30 WDTSTPCTL - WDT 停止制御 - （0= 集計を継続、1= スリープ モードに入ったら停止）
- b29 予約済み （1 にセット）
- b28 WDTRSTIRQS - WDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）

- b27:b26 WDTRPSS - WDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b25:b24 WDTRPES - WDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b23:b20 WDTCKS - WDT クロック周波数分割比 - （1=/4、4=/64、0xF=/128、6=/512、7=/2048、8=/8192）
- b19:b18 WDTTOPS - WDT タイムアウト期間選択 - （0=1024 サイクル、1=4096、2=8192、3=16384）
- b17 WDTSTRT - WDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b16:b15 予約済み （1 にセット）
- b14 IWDTSTPCTL - IWDT スリープ停止制御 - （0= 集計を継続、1= 一部の低電力モードで停止）
- b13 予約済み （1 にセット）
- b12 IWDTRSTIRQS - IWDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b11:b10 IWDTRPSS - IWDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b9:b8 IWDTRPES - IWDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b7:b4 IWDTCKS - IWDT クロック周波数分割比 - （0= なし、2=/16、3 = /32、4=/64、0xF=/128、5=/256）
- b3:b2 IWDTTOPS - IWDT タイムアウト期間選択 - （0=128 サイクル、1=512、2=1024、3=2048）
- b1 IWDTSTRT - IWDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b0 予約済み （1 にセット）

l :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

- #define BSP_CFG_ROM_REG_OFS1

  初期値 :(0xFFFFFFFF)

  電圧検出 0 回路と HOCO をリセット後に有効にするかどうかを設定します。OFS1 - オプション関数選択レジスタ 1

- b31:b15 予約済み （1 にセット）
- b14:b12 HOCOFRQ1 - HOCO 周波数設定 （bsp_clock_cfg.h 内の BSP_CFG_HOCO_FREQUENCY を使用してセット）

- b11:b9 予約済み （1 にセット）
- b8 HOCOEN - リセット後の HOCO 振動の有効化 / 無効化 （0= 有効、1= 無効）
- b7:b3 予約済み （1 にセット）
- b2 LVDAS - リセット後の電圧検出 0 回路の有効化 / 無効化の選択 （0= 有効、1= 無効）
- b1:b0 VDSEL - 電圧検出 0 レベル選択 （1=2.94V、2=2.87V、3=2.80V）

I :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

ID コード保護

必要な ID コードをセットします。この値をセットして MCU にプログラムしたら、その ID コードを覚えておく必要があります。これは、接続時にデバッガーから要求されるためです。ID コードは 16 バイト長です。以下のマクロは ID コードを 4 バイト セクションで定義します。

I :ID コードを使用しない（保護が無効になっている）場合は、デフォルト （すべて 0xFF）のままにします。

I a: 有効な ID コードの設定方法については、ユーザーズマニュアルを参照してください。

- #define BSP_CFG_ID_CODE_LONG_1

  初期値 :(0xFFFFFFFF)

  最下位 4 バイト セクション、アドレス 0x40120050。MSB から LSB へ : ID コード 16、ID コード 15、ID コード 14、ID コード 13。
- #define BSP_CFG_ID_CODE_LONG_2

  初期値 :(0xFFFFFFFF)

  第 2 ID コード セクション、アドレス 0x40120054。MSB から LSB へ : ID コード 12、ID コード 11、ID コード 10、ID コード 9。
- #define BSP_CFG_ID_CODE_LONG_3

  初期値 :(0xFFFFFFFF)

  第 3 ID コード セクション、アドレス 0x40120058。MSB から LSB へ : ID コード 8、ID コード 7、ID コード 6、ID コード 5。

- #define BSP_CFG_ID_CODE_LONG_4

  初期値 :(0xFFFFFFFF)

  第 4 ID コード セクション、アドレス 0x4012005C。MSB から LSB へ : ID コード 4、ID コード 3、ID コード 2、ID コード 1。

その他のハードウェア オプション

- #define BSP_CFG_MCU_VCC_MV

  初期値 :(3300)

  このマクロは、MCU に供給される電圧 （Vcc）を定義するために使用されます。また、このマクロはミリボルト単位で定義されます。実際に MCU 上の何かを変更するわけではありません。一部の AMS モジュールがこの情報を必要とするため、ここで定義します。

プロジェクト全体のソフトウェア オプション

- #define BSP_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  デフォルトで、AMS モジュールは入力パラメータが有効かどうかをチェックします。開発期間では有益ですが、実稼働コードでは無効にしたい場合があります。その目的は、実行時間とコード スペースを節約することです。このマクロは、パラメータ チェックを有効または無効にするためのグローバル設定です。各 AMS モジュールは、同じ目的で独自のローカル マクロも備えています。デフォルトで、ローカル マクロはここからグローバル値を取得しますが、オーバーライドすることができます。そのため、ローカル設定の方がこのグローバル設定より優先されます。パラメータ チェックの無効化は、入力が適切であることが分かっており、速度の向上またはコード スペースの削減が必要な場合にのみ使用してください。

- 0 = パラメータ チェックのグローバル設定が無効になっています。
- 1 = パラメータ チェックのグローバル設定が有効になっています （デフォルト）。
- #define BSP_CFG_RTOS

  初期値 :(0)

  使用する RTOS を指定します。

- 0 = RTOS なし
- 0 = RTOS なし
- 2 = その他

9.2.1.3 ビルド時間設定 - クロック

このファイルには、ビルド時間クロック設定オプションが保存されます。main() が実行される前に、BSP はこれらのマクロを使用して、ユーザーの MCU のクロックを設定します。

クロック設定オプション。入力クロック周波数が指定されてから、使用する乗算器を指定することによってシステム クロックがセットされます。乗算器の設定値はクロック レジスタのセットに使用されます。8MHz XTAL と 48MHz の CPU クロックを使用したボードの例を以下に示します。

```
BSP_CFG_XTAL_HZ = 12000000
BSP_CFG_PLL_DIV = 2 (/2)
BSP_CFG_PLL_MUL = 12 ((12MHz / 2) x 12 = 48MHz)
```

```
BSP_CFG_ICK_DIV =  1     : System Clock (ICLK)       =
                    (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_ICK_DIV)  = 48MHz
BSP_CFG_PCKA_DIV = 1     : Peripheral Clock A (PCLKA) =
                    (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKA_DIV) = 48MHz
BSP_CFG_PCKB_DIV = 2     : Peripheral Clock B (PCLKB) =
                    (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKB_DIV) = 24MHz
BSP_CFG_PCKD_DIV = 1     : Peripheral Clock D (PCLKD) =
                    (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKD_DIV) = 48MHz
```

I :USB クロックは除数を使用しません。

定義

- #define BSP_CFG_CLOCK_SOURCE

  初期値 :(CGC_CLOCK_HOCO)

  クロック ソース選択 （CKSEL）。選択されたクロックがシステム クロックとすべてのペリフェラル クロックに提供されるベース クロックになります。また、フラッシュ クロックと外部バス クロックにも使用されます。

  次のように、列挙のいずれかにマクロをセットします。

```
Clock                           -  Enumeration to use for macro
High Speed On-Chip Oscillator   (HOCO)     -  CGC_CLOCK_HOCO
Middle Speed On-Chip Oscillator (MOCO)     -  CGC_CLOCK_MOCO
Low Speed On-Chip Oscillator    (LOCO)     -  CGC_CLOCK_LOCO
Main Clock Oscillator                      -  CGC_CLOCK_MAIN_OSC
Sub-Clock Oscillator                       -  CGC_CLOCK_SUBCLOCK
```

- #define BSP_CFG_XTAL_HZ

  初期値 :(12000000)

  XTAL - Hz 単位の入力クロック周波数

- #define BSP_CFG_HOCO_FREQUENCY

  初期値 :(2)

  HOCO は複数の周波数で動作できます。下のマクロを使用していずれか 1 つを選択します。リセット以降に使用される周波数は OFS1.HOCOFRQ0 ビットによって異なります。

Available frequency settings:
0 = 24MHz
1 = 29.491MHz
2 = 32MHz
4 = 48MHz
5 = 64MHz

l :3 の値の設定は禁止されています。

- #define BSP_CFG_ICK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  PLL 出力周波数分割比選択 （PLIDIV）

  使用可能な除数 = - /1 (除算なし)、- /2、- /4

  l : マクロ定義を 'CGC_PLL_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKB_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック B 除算器 （PCKB）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  l : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKD_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  ペリフェラル モジュール クロック D 除算器 （PCKD）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  l : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

9.2.1.4 ビルド時間設定 - 割り込み

割り込みをトリガーするように設定可能な ELC イベントを以下に列挙します。ユーザーが特定のイベントで割り込みをトリガーしたい場合は、マクロの定義を BSP IRQ DISABLED から割り込みに使用する割り込み優先レベルに変更する必要があります。この MCU の有効な優先順位の範囲は 0 ～ 15 で、小さい数値ほど優先順位が高くなります （0 = 最高優先順位、15 = 最低優先順位）。BSP は、ベクター テーブルへの適切な関数の配置を処理して、割り込み優先レベルをセットし、リセットからユーザーのアプリケーションが呼び出されるまでの間に割り込みを生成するように ELC イベントを設定します。

定義

- #define BSP_IRQ_CFG_ICU_IRQ0

  初期値 :(4)

  PORT0 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ1

  初期値 :(1)

  PORT1 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ2

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT2 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ3

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT3 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ4

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT4 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ5

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT5 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ6

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT6 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ7

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT7 IRQ。
- #define BSP_IRQ_CFG_DTC_COMPLETE

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC COMPLETE。
- #define BSP_IRQ_CFG_DTC_END

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC DTC END。
- #define BSP_IRQ_CFG_ICU_SNOOZE_CANCEL

  初期値 :(#define BSP_IRQ_DISABLED)

  ICU CANCELING SNOOZE MODE。
- #define BSP_IRQ_CFG_FCU_FRDYI

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FRDYI。
- #define BSP_IRQ_CFG_LVD_LVD1

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD1 LVD1。
- #define BSP_IRQ_CFG_LVD_LVD2

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD2 LVD2。
- #define BSP_IRQ_CFG_VBATT_LVD

  初期値 :(#define BSP_IRQ_DISABLED)

  VBATT VBAT。
- #define BSP_IRQ_CFG_CGC_MOSC_STOP

  初期値 :(#define BSP_IRQ_DISABLED)

  MOSC OSC STOP。
- #define BSP_IRQ_CFG_LPM_SNOOZE_REQUEST

  初期値 :(#define BSP_IRQ_DISABLED)

  CPUSYS SNOOZE MODE ENTRY FLAG。

- #define BSP_IRQ_CFG_AGT0_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  AGT0 AGTI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_A
  初期値 :(#define BSP_IRQ_DISABLED)
  AGT0 AGTCMAI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_B
  初期値 :(#define BSP_IRQ_DISABLED)
  AGT0 AGTCMBI。
- #define BSP_IRQ_CFG_AGT1_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  AGT1 AGTI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_A
  初期値 :(#define BSP_IRQ_DISABLED)
  AGT1 AGTCMAI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_B
  初期値 :(#define BSP_IRQ_DISABLED)
  AGT1 AGTCMBI。
- #define BSP_IRQ_CFG_IWDT_UNDERFLOW
  初期値 :(#define BSP_IRQ_DISABLED)
  IWDT NMIUNDF N。
- #define BSP_IRQ_CFG_WDT_UNDERFLOW
  初期値 :(#define BSP_IRQ_DISABLED)
  CWDT NMIUNDF N。
- #define BSP_IRQ_CFG_RTC_ALARM
  初期値 :(#define BSP_IRQ_DISABLED)
  RTC ALM。
- #define BSP_IRQ_CFG_RTC_PERIOD
  初期値 :(#define BSP_IRQ_DISABLED)
  RTC PRD。

- #define BSP_IRQ_CFG_RTC_CARRY

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC CUP。
- #define BSP_IRQ_CFG_ADC0_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 ADI。
- #define BSP_IRQ_CFG_ADC0_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 GBADI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPAI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPBI。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MISMATCH。
- #define BSP_IRQ_CFG_COMP_LP_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C0IRQ。
- #define BSP_IRQ_CFG_COMP_LP_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C1IRQ.。
- #define BSP_IRQ_CFG_USBFS_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  USBFS USBI。

- #define BSP_IRQ_CFG_USBFS_RESUME

  初期値 :(#define BSP_IRQ_DISABLED)

  USBFS USBR。
- #define BSP_IRQ_CFG_IIC0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 RXI。
- #define BSP_IRQ_CFG_IIC0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 TXI。
- #define BSP_IRQ_CFG_IIC0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 TEI。
- #define BSP_IRQ_CFG_IIC0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 EEI。
- #define BSP_IRQ_CFG_IIC1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 RXI。
- #define BSP_IRQ_CFG_IIC1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TXI。
- #define BSP_IRQ_CFG_IIC1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TEI。
- #define BSP_IRQ_CFG_IIC1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 EEI。
- #define BSP_IRQ_CFG_CTSU_WRITE

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUWR。

- #define BSP_IRQ_CFG_CTSU_READ

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSURD。
- #define BSP_IRQ_CFG_CTSU_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUFN。
- #define BSP_IRQ_CFG_KEY_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  KEY INTKR。
- #define BSP_IRQ_CFG_DOC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DOC DOPCF。
- #define BSP_IRQ_CFG_CAC_FREQUENCY_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC FERRF。
- #define BSP_IRQ_CFG_CAC_MEASUREMENT_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC MENDF。
- #define BSP_IRQ_CFG_CAC_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC OVFF。
- #define BSP_IRQ_CFG_CAN0_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 ERS。
- #define BSP_IRQ_CFG_CAN0_FIFO_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXF。
- #define BSP_IRQ_CFG_CAN0_FIFO_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXF。

- #define BSP_IRQ_CFG_CAN0_MAILBOX_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXM。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXM。
- #define BSP_IRQ_CFG_IOPORT_EVENT_1

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP A。
- #define BSP_IRQ_CFG_IOPORT_EVENT_2

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP B。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_0

  初期値 :(#define BSP_IRQ_DISABLED)

  ELC0 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_1

  初期値 :(#define BSP_IRQ_DISABLED)

  ELC1 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_POEG0_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT0。
- #define BSP_IRQ_CFG_POEG1_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT1。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT B。

- #define BSP_IRQ_CFG_GPT0_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT0_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT0_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT0_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT1_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT1_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT1_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT1_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER UNDERFLOW。

- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT2_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT2_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT2_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT2_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT2_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT2_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT B。

- #define BSP_IRQ_CFG_GPT3_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT3_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT3_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT3_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT4_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT4_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT4_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT4_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER UNDERFLOW。

- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT5_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT5_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT5_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT5_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT6_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT6_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT D。

- #define BSP_IRQ_CFG_GPT6_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT6_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_OPS_UVW_EDGE

  初期値 :(#define BSP_IRQ_DISABLED)
- #define BSP_IRQ_CFG_SCI0_RXI

  初期値 :(2)

  SCI0 RXI。
- #define BSP_IRQ_CFG_SCI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TXI。
- #define BSP_IRQ_CFG_SCI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TEI。
- #define BSP_IRQ_CFG_SCI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 ERI。
- #define BSP_IRQ_CFG_SCI0_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 AM。
- #define BSP_IRQ_CFG_SCI0_RXI_OR_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI OR ERI。
- #define BSP_IRQ_CFG_SCI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 RXI。
- #define BSP_IRQ_CFG_SCI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TXI。

- #define BSP_IRQ_CFG_SCI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TEI。
- #define BSP_IRQ_CFG_SCI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 ERI。
- #define BSP_IRQ_CFG_SCI1_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 AM。
- #define BSP_IRQ_CFG_SCI9_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 RXI。
- #define BSP_IRQ_CFG_SCI9_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TXI。
- #define BSP_IRQ_CFG_SCI9_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TEI。
- #define BSP_IRQ_CFG_SCI9_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 ERI。
- #define BSP_IRQ_CFG_SCI9_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 AM。
- #define BSP_IRQ_CFG_SPI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPRI。
- #define BSP_IRQ_CFG_SPI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPTI。

- #define BSP_IRQ_CFG_SPI0_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPII。
- #define BSP_IRQ_CFG_SPI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPEI。
- #define BSP_IRQ_CFG_SPI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SP ELCTEND。
- #define BSP_IRQ_CFG_SPI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPRI。
- #define BSP_IRQ_CFG_SPI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPTI。
- #define BSP_IRQ_CFG_SPI1_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPII。
- #define BSP_IRQ_CFG_SPI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPEI。
- #define BSP_IRQ_CFG_SPI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SP ELCTEND。
- #define BSP_IRQ_CFG_AES_WRREQ

  初期値 :(#define BSP_IRQ_DISABLED)

  AES WRREQ。
- #define BSP_IRQ_CFG_AES_RDREQ

  初期値 :(#define BSP_IRQ_DISABLED)

  AES RDREQ。

- #define BSP_IRQ_CFG_TRNG_RDREQ

  初期値 :(#define BSP_IRQ_DISABLED)

  TRNG RDREQ。

9.2.1.5 ビルド時間設定 - ピン設定

このファイルには、ピン設定の配列が保存されます。起動時に main() が実行される前に、BSP はこの配列を反復処理し、配列内の設定に基づいて MCU のポート ピンを初期化します。

I：このファイルは ISDE Pin Configurator を使用して作成することをお勧めします。

9.2.2 カスタム ユーザー S124 ボードの BSP

カスタム ユーザー S124 ボードのテンプレート BSP。

9.2.2.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。現時点で、次の設定値を構成するためのヘッダー ファイルが存在します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.3 DK-S3A7 ボードの BSP

DK-S3A7 ボードの BSP。

DK-S3A7 は、LQFP144 パッケージに含まれる Renesas Synergy S3A7 マイクロコントローラ用の開発キットです。このキットには、メイン ボードとブレイクアウト ボードの 2 つのボードが付属しています。これらのボードはともに、アプリケーションの開発およびテストのために S3A7 のすべてのペリフェラルへのアクセスが容易なインタフェースとコネクタ （USB ホストとデバイス、SEGGER J-Link オンボード デバッグ、3 つの Pmod コネクタ、SD カード、および複数の LED インジケーター）を提供します。

9.2.3.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。現時点で、次の設定値を構成するためのヘッダー ファイルが存在します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.3.2 ビルド時間設定 - 全般

このファイルには、全般 BSP 設定用のコンパイル時間設定オプションが保存されます。スタック サイズや ROM レジスタなどをこのファイルで設定します。

部品番号情報

MCU の製品部品番号を入力します。この情報は、パッケージ サイズやメモリ サイズなどの MCU に関する情報の取得に使用されます。この情報を解釈しやすくするために、複数のマクロを使って部品番号を定義します。

```
R7FS 3 A 7 7 C 3 A01 C FB
|  |||||||  || Macro Name                  Description
|  |||||||  ||__BSP_CFG_MCU_PART_PACKAGE     = Package type, number of pins
|  |||||||  |____not used                   = Quality ID
|  |||||||_________not used                   = Software ID
|  ||||||__________not used                   = Operating range
|  |||||____________BSP_CFG_MCU_PART_MEMORY_SIZE  = ROM, RAM, and Data Flash Capacity
|  ||||_____________BSP_CFG_MCU_PART_FEATURE_SET  = Superset, no encryption
|  |||_________________not used                  = Document index
|  ||__________________BSP_CFG_MCU_PART_CORE        = Core & frequency (CM4, 48MHz)
|  |___________________BSP_CFG_MCU_PART_SERIES      = Performance category (High-efficiency -
                                        Low power)
|_____________________________not used                  = Renesas Synergy MCU
```

- #define BSP_CFG_MCU_PART_PACKAGE

  初期値 :(0x6)

  パッケージ タイプ。次の値に基づいてマクロ定義をセットします。

```
Character(s) = Value for macro = Package Type/Number of Pins/Other Info
BJ          = 0x0           = BGA/121
BG          = 0x1           = BGA/176
BD          = 0x2           = BGA/224
FL          = 0x3           = LQFP/48
FM          = 0x4           = LQFP/64
FP          = 0x5           = LQFP/100
FB          = 0x6           = LQFP/144
FC          = 0x7           = LQFP/176
LM          = 0x8           = LGA/36
LA          = 0x9           = LGA/100/5.5x5.5
LJ          = 0xA           = LGA/100/7x7
LK          = 0xB           = LGA/145
NF          = 0xC           = WQFN/40
NE          = 0xD           = WQFN/48
NB          = 0xE           = WQFN/64
```

- #define BSP_CFG_MCU_PART_MEMORY_SIZE

  初期値 :(0x0C)

  ROM、RAM、およびデータ フラッシュ容量。

```
Character(s) = Value for macro = ROM Size/Ram Size/Data Flash Size
6           = 0x06          = 64KB
7           = 0x07          = 128KB
8           = 0x08          = 256KB
A           = 0x0A          = 512KB
C           = 0x0C          = 1MB
E           = 0x0E          = 2MB
G           = 0x10          = 3MB
H           = 0x11          = 4MB/640KB/64KB
```

- #define BSP_CFG_MCU_PART_CORE

  初期値 :(0x0A)

  コアおよび周波数。

```
Character(s) = Value for macro = Description
2           = 0x02          = CM0+, 32MHz
A           = 0x0A          = CM4,  48MHz
D           = 0x0D          = CM4, 120MHz
G           = 0x10          = CM4, 240MHz
```

- #define BSP_CFG_MCU_PART_SERIES

  初期値 :(0x3)

  シリーズ

Character(s)	= Value for macro	= Description
1	= 0x1	= Ultra-low Power (up to 32MHz)
3	= 0x3	= High-efficiency (33-100MHz)
5	= 0x5	= High-integration (101MHz-200MHz)
7	= 0x7	= High-performance (201MHz-300MHz)

スタックおよびヒープ サイズの設定

- #define BSP_CFG_STACK_MAIN_BYTES

  初期値 :(0x1000)

  バイト単位のメイン スタック サイズ。これは、リセット以降に使用される必須のスタックです。例外では常にメイン スタックが使用されます。

- #define BSP_CFG_STACK_PROCESS_BYTES

  初期値 :(0)

  バイト単位のプロセス スタック サイズ。プロセス スタックの使用は任意です。BSP はプロセス スタックを使用しません。ユーザーがアプリケーション内でプロセス スタックを使用する場合は、CONTROL レジスタの SPSEL ビットをセットする必要があります。

- #define BSP_CFG_HEAP_BYTES

  初期値 :(0x400)

  バイト単位のヒープ サイズ。

オプション設定メモリ （ROM レジスタ）の設定

特定のレジスタが ROM に保存され、リセット以降の MCU の設定に使用されます。

I：レジスタのデフォルト値を使用するには、マクロをすべて 0xFF のままにします。

I：すべてのオプションがデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_OFS0

  初期値 :(0xFFFFFFFF)

  WDT 設定と IWDT 設定を構成します。OFS0 - オプション関数選択レジスタ 0

- b31 予約済み （1 にセット）
- b30 WDTSTPCTL - WDT 停止制御 - （0= 集計を継続、1= スリープ モードに入ったら停止）

- b29 予約済み （1 にセット）
- b28 WDTRSTIRQS - WDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b27:b26 WDTRPSS - WDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b25:b24 WDTRPES - WDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b23:b20 WDTCKS - WDT クロック周波数分割比 - （1=/4、4=/64、0xF=/128、6=/512、7=/2048、8=/8192）
- b19:b18 WDTTOPS - WDT タイムアウト期間選択 - （0=1024 サイクル、1=4096、2=8192、3=16384）
- b17 WDTSTRT - WDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b16:b15 予約済み （1 にセット）
- b14 IWDTSTPCTL - IWDT Sleep Stop Control - （0=counting continues, 1=stop w/selected low power modes）
- b13 予約済み （1 にセット）
- b12 IWDTRSTIRQS - IWDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b11:b10 IWDTRPSS - IWDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b9:b8 IWDTRPES - IWDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b7:b4 IWDTCKS - IWDT クロック周波数分割比 - （0= なし、2=/16、3 = /32、4=/64、0xF=/128、5=/256）
- b3:b2 IWDTTOPS - IWDT タイムアウト期間選択 - （0=128 サイクル、1=512、2=1024、3=2048）
- b1 IWDTSTRT - IWDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b0 予約済み （1 にセット）

l :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

- #define BSP_CFG_ROM_REG_OFS1

  初期値 :(0xFFFFFFFF)

  電圧検出 0 回路と HOCO をリセット後に有効にするかどうかを設定します。OFS1 - オプション関数選択レジスタ 1

- b31:b15 予約済み （1 にセット）
- b14:b12 HOCOFRQ1 - HOCO 周波数設定 （bsp_clock_cfg.h 内の BSP_CFG_HOCO_FREQUENCY を使用してセット）
- b11:b9 予約済み （1 にセット）
- b8 HOCOEN - リセット後の HOCO 振動の有効化 / 無効化 （0= 有効、1= 無効）
- b7:b3 予約済み （1 にセット）
- b2 LVDAS - リセット後の電圧検出 0 回路の有効化 / 無効化の選択 （0= 有効、1= 無効）
- b1:b0 VDSEL - 電圧検出 0 レベル選択 （1=2.94V、2=2.87V、3=2.80V）

  I :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

セキュリティ MPU オプション

以下のセキュリティ MPU 領域のそれぞれに 3 つずつのマクロがあります。

- オプションを有効 / 無効にします。1 = 無効、0 = 有効を使用します。
- 開始アドレス - この MPU 領域の開始アドレス。
- 終了アドレス - この MPU 領域の終了アドレス。

I : 領域によって、セット可能なアドレスに対する制限が異なります。

I : すべての領域がデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_MPU_PC0_ENABLE

  初期値 :(1)

  PC 領域 0 を有効または無効にします。
- 0 = 有効
- 1 = 無効

- #define BSP_CFG_ROM_REG_MPU_PC0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_PC1_ENABLE

  初期値 :(1)

  PC 領域 1 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_PC1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION0_ENABLE

  初期値 :(1)

  メモリ領域 0 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0x00FFFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0x00FFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION1_ENABLE

  初期値 :(1)

  メモリ領域 1 を有効または無効にします。
- 0 = 有効

- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION2_ENABLE

  初期値 :(1)

  メモリ領域 2 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION2_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION2_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION3_ENABLE

  初期値 :(1)

  メモリ領域 3 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION3_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION3_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF

ID コード保護

必要な ID コードをセットします。この値をセットして MCU にプログラムしたら、その ID コードを覚えておく必要があります。これは、接続時にデバッガーから要求されるためです。ID コードは 16 バイト長です。以下のマクロは ID コードを 4 バイト セクションで定義します。

I :ID コードを使用しない（保護が無効になっている）場合は、デフォルト （すべて 0xFF）のままにします。

I a: 有効な ID コードの設定方法については、ユーザーズマニュアルを参照してください。

- #define BSP_CFG_ID_CODE_LONG_1

  初期値 :(0xFFFFFFFF)

  最下位 4 バイト セクション、アドレス 0x40120050。MSB から LSB へ : ID コード 16、ID コード 15、ID コード 14、ID コード 13。

- #define BSP_CFG_ID_CODE_LONG_2

  初期値 :(0xFFFFFFFF)

  第 2 ID コード セクション、アドレス 0x40120054。MSB から LSB へ : ID コード 12、ID コード 11、ID コード 10、ID コード 9。

- #define BSP_CFG_ID_CODE_LONG_3

  初期値 :(0xFFFFFFFF)

  第 3 ID コード セクション、アドレス 0x40120058。MSB から LSB へ : ID コード 8、ID コード 7、ID コード 6、ID コード 5。

- #define BSP_CFG_ID_CODE_LONG_4

  初期値 :(0xFFFFFFFF)

  第 4 ID コード セクション、アドレス 0x4012005C。MSB から LSB へ : ID コード 4、ID コード 3、ID コード 2、ID コード 1。

その他のハードウェア オプション

- #define BSP_CFG_MCU_VCC_MV

  初期値 :(3300)

  このマクロは、MCU に供給される電圧 （Vcc）を定義するために使用されます。また、このマクロはミリボルト単位で定義されます。実際に MCU 上の何かを変更するわけではありません。選択したモジュールがこの情報を必要とするため、ここで定義します。

プロジェクト全体のソフトウェア オプション

- #define BSP_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  デフォルトで、SSP モジュールは入力パラメータが有効かどうかをチェックします。これは開発において役立ちますが、状況に応じて実稼働コードに対し無効化することもできます。その目的は、実行時間とコード スペースを節約することです。このマクロは、パラメータ チェックを有効または無効にするためのグローバル設定です。各モジュールは、同じ目的の独自のローカル マクロも備えています。デフォルトで、ローカル マクロはここからグローバル値を取得しますが、オーバーライドすることができます。そのため、ローカル設定の方がこのグローバル設定より優先されます。パラメータ チェックの無効化は、入力が適切であることが分かっており、速度の向上またはコード スペースの削減が必要な場合にのみ使用してください。

- 0 = パラメータ チェックのグローバル設定が無効になっています。
- 1 = パラメータ チェックのグローバル設定が有効になっています （デフォルト）。
- #define BSP_CFG_RTOS

  初期値 :(1)

  使用する RTOS を指定します。

- 0 = RTOS なし
- 0 = RTOS なし
- 2 = その他

9.2.3.3 ビルド時間設定 - クロック

このファイルには、ビルド時間クロック設定オプションが保存されます。main() が実行される前に、BSP はこれらのマクロを使用して、ユーザーの MCU のクロックを設定します。

クロック設定オプション。入力クロック周波数が指定されてから、使用する乗算器を指定することによってシステム クロックがセットされます。乗算器の設定値はクロック レジスタのセットに使用されます。8MHz XTAL と 48MHz の CPU クロックを使用したボードの例を以下に示します。

```
BSP_CFG_XTAL_HZ = 12000000
BSP_CFG_PLL_DIV = 2 (/2)
BSP_CFG_PLL_MUL = 12 ((12MHz / 2) x 12 = 48MHz)
```

```
BSP_CFG_ICK_DIV =  1     : System Clock (ICLK)       =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_ICK_DIV)  = 48MHz
BSP_CFG_PCKA_DIV = 1     : Peripheral Clock A (PCLKA) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKA_DIV) = 48MHz
BSP_CFG_PCKB_DIV = 2     : Peripheral Clock B (PCLKB) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKB_DIV) = 24MHz
BSP_CFG_PCKC_DIV = 1     : Peripheral Clock C (PCLKC) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKC_DIV) = 48MHz
BSP_CFG_PCKD_DIV = 1     : Peripheral Clock D (PCLKD) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKD_DIV) = 48MHz
BSP_CFG_FCK_DIV =  2     : Flash IF Clock (FCLK)     =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_FCK_DIV)  = 24MHz
BSP_CFG_BCK_DIV =  2     : External Bus Clock (BCK)   =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_BCK_DIV)  = 24MHz
```

I :USB クロックは除数を使用しません。

定義

- #define BSP_CFG_CLOCK_SOURCE

  初期値 :(CGC_CLOCK_PLL)

  クロック ソース選択 （CKSEL）。選択されたクロックがシステム クロックとすべてのペリフェラル クロックに提供されるベース クロックになります。また、フラッシュ クロックと外部バス クロックにも使用されます。

  次のように、列挙のいずれかにマクロをセットします。

```
Clock                           -  Enumeration to use for macro
High Speed On-Chip Oscillator   (HOCO)     -  CGC_CLOCK_HOCO
Middle Speed On-Chip Oscillator (MOCO)     -  CGC_CLOCK_MOCO
Low Speed On-Chip Oscillator    (LOCO)     -  CGC_CLOCK_LOCO
Main Clock Oscillator                      -  CGC_CLOCK_MAIN_OSC
Sub-Clock Oscillator                       -  CGC_CLOCK_SUBCLOCK
PLL Circuit                                -  CGC_CLOCK_PLL
```

- #define BSP_CFG_XTAL_HZ

  初期値 :(12000000)

  XTAL - Hz 単位の入力クロック周波数

- #define BSP_CFG_HOCO_FREQUENCY

  初期値 :(0)

  HOCO は複数の周波数で動作できます。下のマクロを使用していずれか 1 つを選択します。リセット以降に使用される周波数は OFS1.HOCOFRQ0 ビットによって異なります。

```
Available frequency settings:
0 = 24MHz
1 = 29.491MHz
2 = 32MHz
4 = 48MHz
5 = 64MHz
```

  l :3 の値の設定は禁止されています。

- #define BSP_CFG_PLL_DIV

  初期値 :(CGC_PLL_DIV_2)

  PLL 出力周波数分割比選択 (PLIDIV)

  使用可能な除数 = - /1 (除算なし)、- /2、- /4

  l : マクロ定義を 'CGC_PLL_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PLL_MUL

  初期値 :(8)

  PLL 周波数増倍係数選択 (PLLMUL)

  使用可能な乗算器 = x8 ～ x31 の 1 ずつのインクリメント (10、11、12、13 ... 30、31 など)

- #define BSP_CFG_ICK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  システム クロック除算器 (ICK)

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  l : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKA_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  ペリフェラル モジュール クロック A 除算器 （PCKA）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKB_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック B 除算器 （PCKB）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKC_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  ペリフェラル モジュール クロック C 除算器 （PCKC）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKD_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  ペリフェラル モジュール クロック D 除算器 （PCKD）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  外部バス クロック除算器 （BCLK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I：マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_FCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  フラッシュ IF クロック除算器 （FCK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I：マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

9.2.3.4 ビルド時間設定 - 割り込み

割り込みをトリガーするように設定可能な ELC イベントを以下に列挙します。ユーザーが特定のイベントで割り込みをトリガーしたい場合は、マクロの定義を BSP IRQ DISABLED から割り込みに使用する割り込み優先レベルに変更する必要があります。この MCU の有効な優先順位の範囲は 0 ～ 15 で、小さい数値ほど優先順位が高くなります （0 = 最高優先順位、15 = 最低優先順位）。BSP は、ベクター テーブルへの適切な関数の配置を処理して、割り込み優先レベルをセットし、リセットからユーザーのアプリケーションが呼び出されるまでの間に割り込みを生成するように ELC イベントを設定します。

定義

- #define BSP_IRQ_CFG_ICU_IRQ0

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT0 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ1

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT1 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ2

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT2 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ3

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT3 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ4

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT4 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ5

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT5 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ6

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT6 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ7

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT7 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ8

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT8 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ9

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT9 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ10

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT10 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ11

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT11 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ12

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT12 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ13

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT13 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ14

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT14 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ15

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT15 IRQ。
- #define BSP_IRQ_CFG_DMAC0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC0 DMAC。
- #define BSP_IRQ_CFG_DMAC1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC1 DMAC。
- #define BSP_IRQ_CFG_DMAC2_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC2 DMAC。
- #define BSP_IRQ_CFG_DMAC3_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC3 DMAC。
- #define BSP_IRQ_CFG_DTC_COMPLETE

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC COMPLETE。
- #define BSP_IRQ_CFG_DTC_END

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC DTC END。
- #define BSP_IRQ_CFG_ICU_SNOOZE_CANCEL

  初期値 :(#define BSP_IRQ_DISABLED)

  ICU CANCELING SNOOZE MODE。

- #define BSP_IRQ_CFG_FCU_FRDYI

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FRDYI。
- #define BSP_IRQ_CFG_LVD_LVD1

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD1 LVD1。
- #define BSP_IRQ_CFG_LVD_LVD2

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD2 LVD2。
- #define BSP_IRQ_CFG_VBATT_LVD

  初期値 :(#define BSP_IRQ_DISABLED)

  VBATT VBAT。
- #define BSP_IRQ_CFG_CGC_MOSC_STOP

  初期値 :(#define BSP_IRQ_DISABLED)

  MOSC OSC STOP。
- #define BSP_IRQ_CFG_LPM_SNOOZE_REQUEST

  初期値 :(#define BSP_IRQ_DISABLED)

  CPUSYS SNOOZE MODE ENTRY FLAG。
- #define BSP_IRQ_CFG_AGT0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMAI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMBI。
- #define BSP_IRQ_CFG_AGT1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTI。

- #define BSP_IRQ_CFG_AGT1_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMAI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMBI。
- #define BSP_IRQ_CFG_IWDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  IWDT NMIUNDF N。
- #define BSP_IRQ_CFG_WDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CWDT NMIUNDF N。
- #define BSP_IRQ_CFG_RTC_ALARM

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC ALM。
- #define BSP_IRQ_CFG_RTC_PERIOD

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC PRD。
- #define BSP_IRQ_CFG_RTC_CARRY

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC CUP。
- #define BSP_IRQ_CFG_ADC0_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 ADI。
- #define BSP_IRQ_CFG_ADC0_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 GBADI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPAI。

- #define BSP_IRQ_CFG_ADC0_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPBI。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MISMATCH。
- #define BSP_IRQ_CFG_COMP_HS_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP OC0 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD1 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_LP_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C0IRQ。
- #define BSP_IRQ_CFG_COMP_LP_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C1IRQ.。
- #define BSP_IRQ_CFG_USBFS_FIFO_0

  初期値 :(#define BSP_IRQ_DISABLED)

  USBFS D0FIFO。
- #define BSP_IRQ_CFG_USBFS_FIFO_1

  初期値 :(#define BSP_IRQ_DISABLED)

  USBFS D1FIFO.。
- #define BSP_IRQ_CFG_USBFS_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  USBFS USBI。

- #define BSP_IRQ_CFG_USBFS_RESUME

  初期値 :(#define BSP_IRQ_DISABLED)

  USBFS USBR。
- #define BSP_IRQ_CFG_IIC0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 RXI。
- #define BSP_IRQ_CFG_IIC0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 TXI。
- #define BSP_IRQ_CFG_IIC0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 TEI。
- #define BSP_IRQ_CFG_IIC0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 EEI。
- #define BSP_IRQ_CFG_IIC0_WUI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 WUI。
- #define BSP_IRQ_CFG_IIC1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 RXI。
- #define BSP_IRQ_CFG_IIC1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TXI。
- #define BSP_IRQ_CFG_IIC1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TEI。
- #define BSP_IRQ_CFG_IIC1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 EEI。

- #define BSP_IRQ_CFG_IIC2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 RXI。
- #define BSP_IRQ_CFG_IIC2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 TXI。
- #define BSP_IRQ_CFG_IIC2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 TEI。
- #define BSP_IRQ_CFG_IIC2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 EEI。
- #define BSP_IRQ_CFG_SSI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSITXI。
- #define BSP_IRQ_CFG_SSI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIRXI。
- #define BSP_IRQ_CFG_SSI0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIF。
- #define BSP_IRQ_CFG_SSI1_TXI_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIRT。
- #define BSP_IRQ_CFG_SSI1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIF。
- #define BSP_IRQ_CFG_CTSU_WRITE

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUWR。

- #define BSP_IRQ_CFG_CTSU_READ

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSURD。
- #define BSP_IRQ_CFG_CTSU_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUFN。
- #define BSP_IRQ_CFG_KEY_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  KEY INTKR。
- #define BSP_IRQ_CFG_DOC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DOC DOPCF。
- #define BSP_IRQ_CFG_CAC_FREQUENCY_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC FERRF。
- #define BSP_IRQ_CFG_CAC_MEASUREMENT_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC MENDF。
- #define BSP_IRQ_CFG_CAC_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC OVFF。
- #define BSP_IRQ_CFG_CAN0_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 ERS。
- #define BSP_IRQ_CFG_CAN0_FIFO_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXF。
- #define BSP_IRQ_CFG_CAN0_FIFO_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXF。

- #define BSP_IRQ_CFG_CAN0_MAILBOX_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXM。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXM。
- #define BSP_IRQ_CFG_IOPORT_EVENT_1

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP A。
- #define BSP_IRQ_CFG_IOPORT_EVENT_2

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP B。
- #define BSP_IRQ_CFG_IOPORT_EVENT_3

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP C。
- #define BSP_IRQ_CFG_IOPORT_EVENT_4

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP D。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_0

  初期値 :(#define BSP_IRQ_DISABLED)

  ELC0 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_1

  初期値 :(#define BSP_IRQ_DISABLED)

  ELC1 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_POEG0_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT0。
- #define BSP_IRQ_CFG_POEG1_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT1。

- #define BSP_IRQ_CFG_POEG2_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT2。
- #define BSP_IRQ_CFG_POEG3_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT3。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT0_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT0_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT0_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT0_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT0_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT0_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER UNDERFLOW。

- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT1_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT1_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT1_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT1_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT1_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT1_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT B。

- #define BSP_IRQ_CFG_GPT2_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT2_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT2_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT2_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT2_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT2_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT3_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT3_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT D。

- #define BSP_IRQ_CFG_GPT3_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT3_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT3_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT3_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT4_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT4_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT4_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT4_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT F。

- #define BSP_IRQ_CFG_GPT4_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT4_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT5_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT5_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT5_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT5_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT5_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT5_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER UNDERFLOW。

- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT6_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT6_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT6_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT6_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT6_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT6_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT B。

- #define BSP_IRQ_CFG_GPT7_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT7_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT7_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT7_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT7_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT7_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT8_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT8_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT D。

- #define BSP_IRQ_CFG_GPT8_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT8_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT8_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT8_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT9_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT9_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT9_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT9_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT F。

- #define BSP_IRQ_CFG_GPT9_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT9_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_OPS_UVW_EDGE

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT UVW EDGE。
- #define BSP_IRQ_CFG_SCI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI。
- #define BSP_IRQ_CFG_SCI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TXI。
- #define BSP_IRQ_CFG_SCI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TEI。
- #define BSP_IRQ_CFG_SCI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 ERI。
- #define BSP_IRQ_CFG_SCI0_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 AM。
- #define BSP_IRQ_CFG_SCI0_RXI_OR_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI OR ERI。
- #define BSP_IRQ_CFG_SCI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 RXI。

- #define BSP_IRQ_CFG_SCI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TXI。
- #define BSP_IRQ_CFG_SCI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TEI。
- #define BSP_IRQ_CFG_SCI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 ERI。
- #define BSP_IRQ_CFG_SCI1_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 AM。
- #define BSP_IRQ_CFG_SCI2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 RXI。
- #define BSP_IRQ_CFG_SCI2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TXI。
- #define BSP_IRQ_CFG_SCI2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TEI。
- #define BSP_IRQ_CFG_SCI2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 ERI。
- #define BSP_IRQ_CFG_SCI2_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 AM。
- #define BSP_IRQ_CFG_SCI3_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 RXI。

- #define BSP_IRQ_CFG_SCI3_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TXI。
- #define BSP_IRQ_CFG_SCI3_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TEI。
- #define BSP_IRQ_CFG_SCI3_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 ERI。
- #define BSP_IRQ_CFG_SCI3_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 AM。
- #define BSP_IRQ_CFG_SCI4_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 RXI。
- #define BSP_IRQ_CFG_SCI4_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TXI。
- #define BSP_IRQ_CFG_SCI4_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TEI。
- #define BSP_IRQ_CFG_SCI4_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 ERI。
- #define BSP_IRQ_CFG_SCI4_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 AM。
- #define BSP_IRQ_CFG_SCI9_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 RXI。

- #define BSP_IRQ_CFG_SCI9_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  SCI9 TXI。
- #define BSP_IRQ_CFG_SCI9_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  SCI9 TEI。
- #define BSP_IRQ_CFG_SCI9_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  SCI9 ERI。
- #define BSP_IRQ_CFG_SCI9_AM
  初期値 :(#define BSP_IRQ_DISABLED)
  SCI9 AM。
- #define BSP_IRQ_CFG_SPI0_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPRI。
- #define BSP_IRQ_CFG_SPI0_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPTI。
- #define BSP_IRQ_CFG_SPI0_IDLE
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPII。
- #define BSP_IRQ_CFG_SPI0_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPEI。
- #define BSP_IRQ_CFG_SPI0_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SP ELCTEND。
- #define BSP_IRQ_CFG_SPI1_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI1 SPRI。

- #define BSP_IRQ_CFG_SPI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPTI。
- #define BSP_IRQ_CFG_SPI1_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPII。
- #define BSP_IRQ_CFG_SPI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPEI。
- #define BSP_IRQ_CFG_SPI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SP ELCTEND。
- #define BSP_IRQ_CFG_QSPI_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  QSPI INTR。
- #define BSP_IRQ_CFG_SDHIMMC0_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC0_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC0_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 CARD。
- #define BSP_IRQ_CFG_SDHIMMC0_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ODMSDBREQ。
- #define BSP_IRQ_CFG_SDHIMMC1_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ACCS。

- #define BSP_IRQ_CFG_SDHIMMC1_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC1_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 CARD。
- #define BSP_IRQ_CFG_SDHIMMC1_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ODMSDBREQ。
- #define BSP_IRQ_CFG_DIVIDER_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  EXT DIVIDER INTMD。
- #define BSP_IRQ_CFG_SCE_PROC_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP PROC BUSY N。
- #define BSP_IRQ_CFG_SCE_ROMOK

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP ROMOK N。
- #define BSP_IRQ_CFG_SCE_LONG_PLG

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP LONG PLG N。
- #define BSP_IRQ_CFG_SCE_TEST_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP TEST BUSY N。
- #define BSP_IRQ_CFG_SCE_WRRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 0 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 1 N。

- #define BSP_IRQ_CFG_SCE_WRRDY_4

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 4 N。
- #define BSP_IRQ_CFG_SCE_RDRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 0 N。
- #define BSP_IRQ_CFG_SCE_RDRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 1 N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_WRRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE WRRDY N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_RDRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE RDRDY N。
- #define BSP_IRQ_CFG_GLCDC_LINE_DETECT

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 0。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_1

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 1。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_2

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 2。
- #define BSP_IRQ_CFG_DRW_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  TWOD ENGINE IRQ。
- #define BSP_IRQ_CFG_JPEG_JEDI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JEDI。

- #define BSP_IRQ_CFG_JPEG_JDTI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JDTI。

9.2.3.5 ビルド時間設定 - ピン設定

このファイルには、ピン設定の配列が保存されます。起動時に main() が実行される前に、BSP はこの配列を反復処理し、配列内の設定に基づいて MCU のポート ピンを初期化します。

I：このファイルは ISDE Pin Configurator を使用して作成することをお勧めします。

9.2.4 カスタム ユーザー S3A7 ボードの BSP

カスタム ユーザー S3A7 ボードのテンプレート BSP。

9.2.4.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。現時点で、次の設定値を構成するためのヘッダー ファイルが存在します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.5 DK-S7G2 ボードの BSP

DK-S7G2 ボードの BSP。

DK-S7G2 は、BGA224 パッケージに含まれる Renesas SynergyTM S7G2 マイクロコントローラ用の開発キットです。このキットには、メイン ボードとブレイクアウト ボードの 2 つのボードが付属しています。これらのボードはともに、アプリケーションの開発およびテストのために S7G2 のペリフェラルへのアクセスが容易なインタフェースとコネクタ （TFTLCD グラフィクス、イーサネット、USB ホストとデバイス、SEGGER J-Link オンボード デバッグ、4 つの Pmod コネクタ、SD カード、および複数の LED インジケーター）を提供します。

9.2.5.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。現時点で、次の設定値を構成するためのヘッダー ファイルが存在します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.5.2 ビルド時間設定 - 全般

このファイルには、全般 BSP 設定用のコンパイル時間設定オプションが保存されます。スタック サイズや ROM レジスタなどをこのファイルで設定します。

部品番号情報

MCU の製品部品番号を入力します。この情報は、パッケージ サイズやメモリ サイズなどの MCU に関する情報の取得に使用されます。この情報を解釈しやすくするために、複数のマクロを使って部品番号を定義します。

```
R7FS 7 G 2 7 H 2 A01 C BD
|  ||||||| || Macro Name                  Description
|  ||||||| ||__BSP_CFG_MCU_PART_PACKAGE    = Package type, number of pins
|  ||||||| |____not used                   = Quality ID
|  |||||||_________not used                  = Software ID
|  |||||___________not used                  = Operating range
|  ||||_____________BSP_CFG_MCU_PART_MEMORY_SIZE  = ROM, RAM, and Data Flash Capacity
|  |||_______________BSP_CFG_MCU_PART_FEATURE_SET  = Superset, no encryption
|  ||_________________not used                  = Document index
|  |___________________BSP_CFG_MCU_PART_CORE       = Core & frequency (CM4, 240MHz)
|  |_____________________BSP_CFG_MCU_PART_SERIES     = Performance category (High-performance)
|_____________________________not used                  = Renesas Synergy MCU
```

- #define BSP_CFG_MCU_PART_PACKAGE

  初期値 :(0x2)

  パッケージ タイプ。次の値に基づいてマクロ定義をセットします。

Character(s)	Value for macro	Package Type/Number of Pins/Other Info
BJ	0x0	BGA/121
BG	0x1	BGA/176
BD	0x2	BGA/224
FL	0x3	LQFP/48
FM	0x4	LQFP/64
FP	0x5	LQFP/100
FB	0x6	LQFP/144
FC	0x7	LQFP/176
LM	0x8	LGA/36
LA	0x9	LGA/100/5.5x5.5
LJ	0xA	LGA/100/7x7
LK	0xB	LGA/145
NF	0xC	WQFN/40
NE	0xD	WQFN/48
NB	0xE	WQFN/64

- #define BSP_CFG_MCU_PART_MEMORY_SIZE

  初期値 :(0x11)

  ROM、RAM、およびデータ フラッシュ容量。

Character(s)	Value for macro	ROM Size/Ram Size/Data Flash Size
6	0x06	64KB
7	0x07	128KB
8	0x08	256KB
A	0x0A	512KB
C	0x0C	1MB
E	0x0E	2MB
G	0x10	3MB
H	0x11	4MB/640KB/64KB

- #define BSP_CFG_MCU_PART_CORE

  初期値 :(0x10)

  コアおよび周波数。

Character(s)	Value for macro	Description
2	0x02	CM0+, 32MHz
A	0x0A	CM4, 48MHz
D	0x0D	CM4, 120MHz
G	0x10	CM4, 240MHz

- #define BSP_CFG_MCU_PART_SERIES

  初期値 :(0x7)

  シリーズ

Character(s)	= Value for macro	= Description
1	= 0x1	= Ultra-low Power (up to 32MHz)
3	= 0x3	= High-efficiency (33-100MHz)
5	= 0x5	= High-integration (101MHz-200MHz)
7	= 0x7	= High-performance (201MHz-300MHz)

スタックおよびヒープ サイズの設定

- #define BSP_CFG_STACK_MAIN_BYTES

  初期値 :(0x1000)

  バイト単位のメイン スタック サイズ。これは、リセット以降に使用される必須のスタックです。例外では常にメイン スタックが使用されます。

- #define BSP_CFG_STACK_PROCESS_BYTES

  初期値 :(0)

  バイト単位のプロセス スタック サイズ。プロセス スタックの使用は任意です。BSP はプロセス スタックを使用しません。ユーザーがアプリケーション内でプロセス スタックを使用する場合は、CONTROL レジスタの SPSEL ビットをセットする必要があります。

- #define BSP_CFG_HEAP_BYTES

  初期値 :(0x400)

  バイト単位のヒープ サイズ。

オプション設定メモリ （ROM レジスタ）の設定

特定のレジスタが ROM に保存され、リセット以降の MCU の設定に使用されます。

I：レジスタのデフォルト値を使用するには、マクロをすべて 0xFF のままにします。

I：すべてのオプションがデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_OFS0

  初期値 :(0xFFFFFFFF)

  WDT 設定と IWDT 設定を構成します。OFS0 - オプション関数選択レジスタ 0

- b31 予約済み （1 にセット）
- b30 WDTSTPCTL - WDT 停止制御 - （0= 集計を継続、1= スリープ モードに入ったら停止）
- b29 予約済み （1 にセット）
- b28 WDTRSTIRQS - WDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）

- b27:b26 WDTRPSS - WDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b25:b24 WDTRPES - WDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b23:b20 WDTCKS - WDT クロック周波数分割比 - （1=/4、4=/64、0xF=/128、6=/512、7=/2048、8=/8192）
- b19:b18 WDTTOPS - WDT タイムアウト期間選択 - （0=1024 サイクル、1=4096、2=8192、3=16384）
- b17 WDTSTRT - WDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b16:b15 予約済み （1 にセット）
- b14 IWDTSTPCTL - IWDT Sleep Stop Control - （0=counting continues, 1=stop w/selected low power modes）
- b13 予約済み （1 にセット）
- b12 IWDTRSTIRQS - IWDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b11:b10 IWDTRPSS - IWDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b9:b8 IWDTRPES - IWDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b7:b4 IWDTCKS - IWDT クロック周波数分割比 - （0= なし、2=/16、3 = /32、4=/64、0xF=/128、5=/256）
- b3:b2 IWDTTOPS - IWDT タイムアウト期間選択 - （0=128 サイクル、1=512、2=1024、3=2048）
- b1 IWDTSTRT - IWDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b0 予約済み （1 にセット）

l :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

- #define BSP_CFG_ROM_REG_OFS1

  初期値 :(0xFFFFFFFF)

  電圧検出 0 回路と HOCO をリセット後に有効にするかどうかを設定します。OFS1 - オプション関数選択レジスタ 1

- b31:b9 予約済み （1 にセット）
- b10:b9 HOCOFRQ0 - HOCO 周波数設定 （bsp_clock_cfg.h 内の BSP_CFG_HOCO_FREQUENCY を使用してセット）
- b8 HOCOEN - リセット後の HOCO 振動の有効化 / 無効化 （0= 有効、1= 無効）

- b7:b3 予約済み （1 にセット）
- b2 LVDAS - リセット後の電圧検出 0 回路の有効化 / 無効化の選択 （0= 有効、1= 無効）
- b1:b0 VDSEL - 電圧検出 0 レベル選択 （1=2.94V、2=2.87V、3=2.80V）

  I :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

セキュリティ MPU オプション

以下のセキュリティ MPU 領域のそれぞれに 3 つずつのマクロがあります。

- オプションを有効 / 無効にします。1 = 無効、0 = 有効を使用します。
- 開始アドレス - この MPU 領域の開始アドレス。
- 終了アドレス - この MPU 領域の終了アドレス。

I : 領域によって、セット可能なアドレスに対する制限が異なります。

I : すべての領域がデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_MPU_PC0_ENABLE

  初期値 :(1)

  PC 領域 0 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_PC0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF

- #define BSP_CFG_ROM_REG_MPU_PC1_ENABLE

  初期値 :(1)

  PC 領域 1 を有効または無効にします。

- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_PC1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC

- #define BSP_CFG_ROM_REG_MPU_PC1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF

- #define BSP_CFG_ROM_REG_MPU_REGION0_ENABLE

  初期値 :(1)

  メモリ領域 0 を有効または無効にします。

- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0x00FFFFFC

- #define BSP_CFG_ROM_REG_MPU_REGION0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0x00FFFFFF

- #define BSP_CFG_ROM_REG_MPU_REGION1_ENABLE

  初期値 :(1)

  メモリ領域 1 を有効または無効にします。

- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC

- #define BSP_CFG_ROM_REG_MPU_REGION1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION2_ENABLE

  初期値 :(1)

  メモリ領域 2 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION2_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION2_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION3_ENABLE

  初期値 :(1)

  メモリ領域 3 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION3_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION3_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF

ID コード保護

必要な ID コードをセットします。この値をセットして MCU にプログラムしたら、その ID コードを覚えておく必要があります。これは、接続時にデバッガーから要求されるためです。ID コードは 16 バイト長です。以下のマクロは ID コードを 4 バイト セクションで定義します。

I :ID コードを使用しない（保護が無効になっている）場合は、デフォルト（すべて 0xFF）のままにします。

I a: 有効な ID コードの設定方法については、ユーザーズマニュアルを参照してください。

- #define BSP_CFG_ID_CODE_LONG_1

  初期値 :(0xFFFFFFFF)

  最下位 4 バイト セクション、アドレス 0x40120050。MSB から LSB へ : ID コード 16、ID コード 15、ID コード 14、ID コード 13。

- #define BSP_CFG_ID_CODE_LONG_2

  初期値 :(0xFFFFFFFF)

  第 2 ID コード セクション、アドレス 0x40120054。MSB から LSB へ : ID コード 12、ID コード 11、ID コード 10、ID コード 9。

- #define BSP_CFG_ID_CODE_LONG_3

  初期値 :(0xFFFFFFFF)

  第 3 ID コード セクション、アドレス 0x40120058。MSB から LSB へ : ID コード 8、ID コード 7、ID コード 6、ID コード 5。

- #define BSP_CFG_ID_CODE_LONG_4

  初期値 :(0xFFFFFFFF)

  第 4 ID コード セクション、アドレス 0x4012005C。MSB から LSB へ : ID コード 4、ID コード 3、ID コード 2、ID コード 1。

その他のハードウェア オプション

- #define BSP_CFG_MCU_VCC_MV

  初期値 :(3300)

  このマクロは、MCU に供給される電圧 （Vcc）を定義するために使用されます。また、このマクロはミリボルト単位で定義されます。実際に MCU 上の何かを変更するわけではありません。選択したモジュールがこの情報を必要とするため、ここで定義します。

プロジェクト全体のソフトウェア オプション

- #define BSP_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  デフォルトで、SSP モジュールは入力パラメータが有効かどうかをチェックします。これは開発において役立ちますが、状況に応じて実稼働コードに対し無効化することもできます。その目的は、実行時間とコード スペースを節約することです。このマクロは、パラメータ チェックを有効または無効にするためのグローバル設定です。各モジュールは、同じ目的の独自のローカル マクロも備えています。デフォルトで、ローカル マクロはここからグローバル値を取得しますが、オーバーライドすることができます。そのため、ローカル設定の方がこのグローバル設定より優先されます。パラメータ

チェックの無効化は、入力が適切であることが分かっており、速度の向上またはコード スペースの削減が必要な場合にのみ使用してください。

- 0 = パラメータ チェックのグローバル設定が無効になっています。
- 1 = パラメータ チェックのグローバル設定が有効になっています （デフォルト）。
- #define BSP_CFG_RTOS

  初期値 :(1)

  使用する RTOS を指定します。
- 0 = RTOS なし
- 0 = RTOS なし
- 2 = その他
- #define BSP_CFG_ASSERT

  初期値 :(0)

  SSP_ASSERT が失敗した場合の処理を指定します。
- 0 = SSP_ERR_ASSERTION を返します。
- 1 = ssp_error_log を呼び出してから、SSP_ERR_ASSERTION を返します。ssp_error_log は脆弱な関数であり、ssp/src/bsp/mcu/all/bsp_common.h 内のプロトタイプに基づいてユーザー コード内でオーバーライドする必要があることに注意してください。
- 2 = 標準のアサート ライブラリを使用して実行を停止します。
- #define BSP_CFG_ERROR_LOG

  初期値 :(0)

  SSP 関数からエラー コードが返された場合の処理を指定します。
- 0 = エラー コードを返します。
- 1 = ssp_error_log を呼び出してから、エラー コードを返します。ssp_error_log は脆弱な関数であり、ssp/src/bsp/mcu/all/bsp_common.h 内のプロトタイプに基づいてユーザー コード内でオーバーライドする必要があることに注意してください。

9.2.5.3 ビルド時間設定 - クロック

このファイルには、ビルド時間クロック設定オプションが保存されます。main() が実行される前に、BSP はこれらのマクロを使用して、ユーザーの MCU のクロックを設定します。

クロック設定オプション。入力クロック周波数が指定されてから、使用する乗算器を指定することによってシステム クロックがセットされます。乗算器の設定値はクロック レジスタのセットに使用されます。12MHz XTAL と 240MHz の CPU クロックを使用したボードの例を以下に示します。

```
BSP_CFG_XTAL_HZ = 12000000
BSP_CFG_PLL_DIV = 1  (no division)
BSP_CFG_PLL_MUL = 20 (12MHz x 20 = 240MHz)
```

```
BSP_CFG_ICK_DIV =  1     : System Clock (ICLK)       =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_ICK_DIV)  = 240MHz
BSP_CFG_PCKA_DIV = 2     : Peripheral Clock A (PCLKA) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKA_DIV) = 120MHz
BSP_CFG_PCKB_DIV = 4     : Peripheral Clock B (PCLKB) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                    BSP_CFG_PCKB_DIV) = 60MHz
BSP_CFG_PCKC_DIV = 4     : Peripheral Clock C (PCLKC) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKC_DIV) = 60MHz
BSP_CFG_PCKD_DIV = 2     : Peripheral Clock D (PCLKD) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKD_DIV) = 120MHz
BSP_CFG_FCK_DIV =  4     : Flash IF Clock (FCLK)     =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_FCK_DIV)  = 60MHz
BSP_CFG_BCK_DIV =  2     : External Bus Clock (BCK)   =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_BCK_DIV)  = 120MHz
BSP_CFG_UCK_DIV =  5     : USB Clock (UCLK)          =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_UCK_DIV)  = 48MHz
```

このファイルには、ビルド時間クロック設定オプションが保存されます。main() が実行される前に、BSP はこれらのマクロを使用して、ユーザーの MCU のクロックを設定します。

定義

- #define BSP_CFG_CLOCK_SOURCE

  初期値 :(CGC_CLOCK_PLL)

  クロック ソース選択 （CKSEL）。選択されたクロックがシステム クロックとすべてのペリフェラル クロックに提供されるベース クロックになります。また、フラッシュ クロックと外部バス クロックにも使用されます。

  次のように、列挙のいずれかにマクロをセットします。

```
Clock                           -  Enumeration to use for macro
High Speed On-Chip Oscillator   (HOCO)     -  CGC_CLOCK_HOCO
Middle Speed On-Chip Oscillator (MOCO)     -  CGC_CLOCK_MOCO
Low Speed On-Chip Oscillator    (LOCO)     -  CGC_CLOCK_LOCO
Main Clock Oscillator                      -  CGC_CLOCK_MAIN_OSC
Sub-Clock Oscillator                       -  CGC_CLOCK_SUBCLOCK
PLL Circuit                                -  CGC_CLOCK_PLL
```

- #define BSP_CFG_XTAL_HZ

  初期値 :(24000000)

  XTAL - Hz 単位の入力クロック周波数

- #define BSP_CFG_HOCO_FREQUENCY

  初期値 :(0)

  HOCO は複数の周波数で動作できます。下のマクロを使用していずれか 1 つを選択します。リセット以降に使用される周波数は OFS1.HOCOFRQ0 ビットによって異なります。

```
Available frequency settings:
0 = 16MHz
1 = 18MHz
2 = 20MHz
```

- #define BSP_CFG_PLL_SOURCE

  初期値 :(CGC_CLOCK_MAIN_OSC)

  PLL クロック ソース （PLLSRCEL）。PLL 回路に入力するクロック ソースを選択します。

```
Available clock sources:    -  Enumeration to use for macro
0 = Main clock (default)    -  CGC_CLOCK_MAIN_OSC
1 = HOCO                    -  CGC_CLOCK_HOCO
```

- #define BSP_CFG_PLL_DIV

  初期値 :(CGC_PLL_DIV_2)

  PLL 入力周波数分割比選択 （PLIDIV）

  使用可能な除数 = - /1 (除算なし)、- /2、- /3

  l : マクロ定義を 'CGC_PLL_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PLL_MUL

  初期値 :(20.0)

  PLL 周波数増倍係数選択 （PLLMUL）

  使用可能な乗算器 = x10.0 ～ x30.0 の 0.5 ずつのインクリメント （10.0、10.5、11.0、11.5 ... 29.0、29.5、30.0 など）

- #define BSP_CFG_ICK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  システム クロック除算器 （ICK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKA_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック A 除算器 （PCKA）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKB_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  ペリフェラル モジュール クロック B 除算器 （PCKB）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKC_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  ペリフェラル モジュール クロック C 除算器 （PCKC）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKD_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック D 除算器 （PCKD）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  外部バス クロック除算器 （BCLK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_FCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  フラッシュ IF クロック除算器 （FCK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_UCK_DIV

  初期値 :(CGC_USB_CLOCK_DIV_5)

  USB クロック除算器選択。

  使用可能な除数 = /3、/4、/5

  I : マクロ定義を 'CGC_USB_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCLK_OUTPUT

  初期値 :(1)

  BCLK 出力ピンの設定

  使用可能なオプション :

- 0 = 出力なし
- 1 = BCK 周波数
- 2 = BCK/2 周波数

  I : このマクロは、外部バスが有効になっている場合にのみ効果があります。

- #define BSP_CFG_SDCLK_OUTPUT

  初期値 :(1)

  SDCLK 出力ピンの設定

  使用可能なオプション :

- 0 = 出力なし
- 1 = BCK 周波数

  I : このマクロは、外部バスが有効になっている場合にのみ効果があります。

9.2.5.4 ビルド時間設定 - 割り込み

割り込みをトリガーするように設定可能な ELC イベントを以下に列挙します。ユーザーが特定のイベントで割り込みをトリガーしたい場合は、マクロの定義を BSP IRQ DISABLED から割り込みに使用する割り込み優先レベルに変更する必要があります。この MCU の有効な優先順位の範囲は 0 ～ 15 で、小さい数値ほど優先順位が高くなります （0 = 最高優先順位、15 = 最低優先順位）。BSP は、ベクター テーブルへの適切な関数の配置を処理して、割り込み優先レベルをセットし、リセットからユーザーのアプリケーションが呼び出されるまでの間に割り込みを生成するように ELC イベントを設定します。

定義

- #define BSP_IRQ_CFG_ICU_IRQ0

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT0 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ1

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT1 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ2

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT2 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ3

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT3 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ4

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT4 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ5

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT5 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ6

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT6 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ7

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT7 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ8

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT8 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ9

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT9 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ10
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT10 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ11
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT11 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ12
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT12 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ13
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT13 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ14
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT14 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ15
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT15 IRQ。
- #define BSP_IRQ_CFG_DMAC0_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC0 DMAC。
- #define BSP_IRQ_CFG_DMAC1_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC1 DMAC。
- #define BSP_IRQ_CFG_DMAC2_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC2 DMAC。
- #define BSP_IRQ_CFG_DMAC3_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC3 DMAC。

- #define BSP_IRQ_CFG_DMAC4_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC4 DMAC。
- #define BSP_IRQ_CFG_DMAC5_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC5 DMAC。
- #define BSP_IRQ_CFG_DMAC6_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC6 DMAC。
- #define BSP_IRQ_CFG_DMAC7_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC7 DMAC。
- #define BSP_IRQ_CFG_DTC_COMPLETE

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC COMPLETE。
- #define BSP_IRQ_CFG_DTC_END

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC DTC END。
- #define BSP_IRQ_CFG_ICU_SNOOZE_CANCEL

  初期値 :(#define BSP_IRQ_DISABLED)

  ICU CANCELING SNOOZE MODE。
- #define BSP_IRQ_CFG_FCU_FIFERR

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FIFERR。
- #define BSP_IRQ_CFG_FCU_FRDYI

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FRDYI。
- #define BSP_IRQ_CFG_LVD_LVD1

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD1 LVD1。

- #define BSP_IRQ_CFG_LVD_LVD2

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD2 LVD2。
- #define BSP_IRQ_CFG_CGC_MOSC_STOP

  初期値 :(#define BSP_IRQ_DISABLED)

  MOSC OSC STOP。
- #define BSP_IRQ_CFG_LPM_SNOOZE_REQUEST

  初期値 :(#define BSP_IRQ_DISABLED)

  CPUSYS SNOOZE MODE ENTRY FLAG。
- #define BSP_IRQ_CFG_AGT0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMAI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMBI。
- #define BSP_IRQ_CFG_AGT1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMAI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMBI。
- #define BSP_IRQ_CFG_IWDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  IWDT NMIUNDF N。

- #define BSP_IRQ_CFG_WDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CWDT NMIUNDF N。
- #define BSP_IRQ_CFG_RTC_ALARM

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC ALM。
- #define BSP_IRQ_CFG_RTC_PERIOD

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC PRD。
- #define BSP_IRQ_CFG_RTC_CARRY

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC CUP。
- #define BSP_IRQ_CFG_ADC0_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 ADI。
- #define BSP_IRQ_CFG_ADC0_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 GBADI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPAI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPBI。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MISMATCH。

- #define BSP_IRQ_CFG_ADC1_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 ADI。
- #define BSP_IRQ_CFG_ADC1_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 GBADI。
- #define BSP_IRQ_CFG_ADC1_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 CMPAI。
- #define BSP_IRQ_CFG_ADC1_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 CMPBI。
- #define BSP_IRQ_CFG_ADC1_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC1_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 COMPARE MISMATCH。
- #define BSP_IRQ_CFG_COMP_HS_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP OC0 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD1 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_2

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD2 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_3

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD3 COMP IRQ。

- #define BSP_IRQ_CFG_COMP_HS_4
  初期値 :(#define BSP_IRQ_DISABLED)
  COMP RD4 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_5
  初期値 :(#define BSP_IRQ_DISABLED)
  COMP RD5 COMP IRQ。
- #define BSP_IRQ_CFG_USBFS_FIFO_0
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS D0FIFO。
- #define BSP_IRQ_CFG_USBFS_FIFO_1
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS D1FIFO.。
- #define BSP_IRQ_CFG_USBFS_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS USBI。
- #define BSP_IRQ_CFG_USBFS_RESUME
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS USBR。
- #define BSP_IRQ_CFG_IIC0_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 RXI。
- #define BSP_IRQ_CFG_IIC0_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 TXI。
- #define BSP_IRQ_CFG_IIC0_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 TEI。
- #define BSP_IRQ_CFG_IIC0_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 EEI。

- #define BSP_IRQ_CFG_IIC0_WUI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 WUI。
- #define BSP_IRQ_CFG_IIC1_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC1 RXI。
- #define BSP_IRQ_CFG_IIC1_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC1 TXI。
- #define BSP_IRQ_CFG_IIC1_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC1 TEI。
- #define BSP_IRQ_CFG_IIC1_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC1 EEI。
- #define BSP_IRQ_CFG_IIC2_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC2 RXI。
- #define BSP_IRQ_CFG_IIC2_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC2 TXI。
- #define BSP_IRQ_CFG_IIC2_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC2 TEI。
- #define BSP_IRQ_CFG_IIC2_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC2 EEI。
- #define BSP_IRQ_CFG_SSI0_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  SSI0 SSITXI。

- #define BSP_IRQ_CFG_SSI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIRXI。
- #define BSP_IRQ_CFG_SSI0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIF。
- #define BSP_IRQ_CFG_SSI1_TXI_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIRT。
- #define BSP_IRQ_CFG_SSI1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIF。
- #define BSP_IRQ_CFG_SRC_INPUT_FIFO_EMPTY

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC IDEI。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_FULL

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC ODFI。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC OVF。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC UDF。
- #define BSP_IRQ_CFG_SRC_CONVERSION_END

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC CEF。
- #define BSP_IRQ_CFG_PDC_RECEIVE_DATA_READY

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCDFI。

- #define BSP_IRQ_CFG_PDC_FRAME_END

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCFEI。
- #define BSP_IRQ_CFG_PDC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCERI。
- #define BSP_IRQ_CFG_CTSU_WRITE

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUWR。
- #define BSP_IRQ_CFG_CTSU_READ

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSURD。
- #define BSP_IRQ_CFG_CTSU_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUFN。
- #define BSP_IRQ_CFG_KEY_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  KEY INTKR。
- #define BSP_IRQ_CFG_DOC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DOC DOPCF。
- #define BSP_IRQ_CFG_CAC_FREQUENCY_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC FERRF。
- #define BSP_IRQ_CFG_CAC_MEASUREMENT_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC MENDF。
- #define BSP_IRQ_CFG_CAC_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC OVFF。

- #define BSP_IRQ_CFG_CAN0_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 ERS。
- #define BSP_IRQ_CFG_CAN0_FIFO_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXF。
- #define BSP_IRQ_CFG_CAN0_FIFO_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXF。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXM。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXM。
- #define BSP_IRQ_CFG_CAN1_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 ERS。
- #define BSP_IRQ_CFG_CAN1_FIFO_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 RXF。
- #define BSP_IRQ_CFG_CAN1_FIFO_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 TXF。
- #define BSP_IRQ_CFG_CAN1_MAILBOX_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 RXM。
- #define BSP_IRQ_CFG_CAN1_MAILBOX_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 TXM。

- #define BSP_IRQ_CFG_IOPORT_EVENT_1
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP A。
- #define BSP_IRQ_CFG_IOPORT_EVENT_2
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP B。
- #define BSP_IRQ_CFG_IOPORT_EVENT_3
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP C。
- #define BSP_IRQ_CFG_IOPORT_EVENT_4
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP D。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_0
  初期値 :(#define BSP_IRQ_DISABLED)
  ELC0 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_1
  初期値 :(#define BSP_IRQ_DISABLED)
  ELC1 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_POEG0_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT0。
- #define BSP_IRQ_CFG_POEG1_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT1。
- #define BSP_IRQ_CFG_POEG2_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT2。
- #define BSP_IRQ_CFG_POEG3_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT3。

- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT0_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT0_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT0_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT0_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT0_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT0_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT0_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 AD TRIG A。
- #define BSP_IRQ_CFG_GPT0_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 AD TRIG B。

- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT1_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT1_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT1_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT1_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT1_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT1_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT1_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 AD TRIG A。
- #define BSP_IRQ_CFG_GPT1_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 AD TRIG B。

- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT2_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT2_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT2_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT2_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT2_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT2_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT2_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 AD TRIG A。
- #define BSP_IRQ_CFG_GPT2_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 AD TRIG B。

- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT3_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT3_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT3_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT3_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT3_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT3_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT3_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 AD TRIG A。
- #define BSP_IRQ_CFG_GPT3_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 AD TRIG B。

- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT4_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT4_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT4_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT4_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT4_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT4_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT4_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 AD TRIG A。
- #define BSP_IRQ_CFG_GPT4_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 AD TRIG B。

- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT5_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT5_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT5_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT5_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT5_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT5_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT5_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 AD TRIG A。
- #define BSP_IRQ_CFG_GPT5_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 AD TRIG B。

- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT6_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT6_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT6_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT6_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT6_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT6_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT6_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 AD TRIG A。
- #define BSP_IRQ_CFG_GPT6_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 AD TRIG B。

- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT7_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT7_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT7_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT7_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT7_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT7_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT7_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 AD TRIG A。
- #define BSP_IRQ_CFG_GPT7_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 AD TRIG B。

- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT8_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT8_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT8_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT8_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT8_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT8_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT8_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 AD TRIG A。
- #define BSP_IRQ_CFG_GPT8_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 AD TRIG B。

- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT9_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT9_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT9_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT9_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT9_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT9_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT9_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 AD TRIG A。
- #define BSP_IRQ_CFG_GPT9_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 AD TRIG B。

- #define BSP_IRQ_CFG_GPT10_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT10_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT10_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT10_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT10_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT10_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT10_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT10_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT10_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 AD TRIG A。
- #define BSP_IRQ_CFG_GPT10_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 AD TRIG B。

- #define BSP_IRQ_CFG_GPT11_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT11_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT11_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT11_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT11_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT11_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT11_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT11_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT11_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 AD TRIG A。
- #define BSP_IRQ_CFG_GPT11_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 AD TRIG B。

- #define BSP_IRQ_CFG_GPT12_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT12_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT12_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT12_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT12_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT12_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT12_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT12_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT12_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 AD TRIG A。
- #define BSP_IRQ_CFG_GPT12_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 AD TRIG B。

- #define BSP_IRQ_CFG_GPT13_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT13_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT13_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT13_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT13_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT13_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT13_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT13_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT13_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 AD TRIG A。
- #define BSP_IRQ_CFG_GPT13_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 AD TRIG B。

- #define BSP_IRQ_CFG_OPS_UVW_EDGE

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT UVW EDGE。
- #define BSP_IRQ_CFG_EPTPC_IPLS

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER IPLS。
- #define BSP_IRQ_CFG_EPTPC_MINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER MINT。
- #define BSP_IRQ_CFG_EPTPC_PINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER PINT。
- #define BSP_IRQ_CFG_EDMAC0_EINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER EINT0。
- #define BSP_IRQ_CFG_EDMAC1_EINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER EINT1。
- #define BSP_IRQ_CFG_EPTPC_TIMER0_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER0 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER1_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER1 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER2_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER2 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER3_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER3 RISE。

- #define BSP_IRQ_CFG_EPTPC_TIMER4_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER4 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER5_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER5 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER0_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER0 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER1_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER1 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER2_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER2 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER3_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER3 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER4_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER4 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER5_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER5 FALL。
- #define BSP_IRQ_CFG_USBHS_FIFO_0

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS D0FIFO。
- #define BSP_IRQ_CFG_USBHS_FIFO_1

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS D1FIFO.。

- #define BSP_IRQ_CFG_USBHS_USB_INT_RESUME

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS USBIR。
- #define BSP_IRQ_CFG_SCI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI。
- #define BSP_IRQ_CFG_SCI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TXI。
- #define BSP_IRQ_CFG_SCI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TEI。
- #define BSP_IRQ_CFG_SCI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 ERI。
- #define BSP_IRQ_CFG_SCI0_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 AM。
- #define BSP_IRQ_CFG_SCI0_RXI_OR_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI OR ERI。
- #define BSP_IRQ_CFG_SCI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 RXI。
- #define BSP_IRQ_CFG_SCI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TXI。
- #define BSP_IRQ_CFG_SCI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TEI。

- #define BSP_IRQ_CFG_SCI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 ERI。
- #define BSP_IRQ_CFG_SCI1_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 AM。
- #define BSP_IRQ_CFG_SCI2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 RXI。
- #define BSP_IRQ_CFG_SCI2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TXI。
- #define BSP_IRQ_CFG_SCI2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TEI。
- #define BSP_IRQ_CFG_SCI2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 ERI。
- #define BSP_IRQ_CFG_SCI2_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 AM。
- #define BSP_IRQ_CFG_SCI3_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 RXI。
- #define BSP_IRQ_CFG_SCI3_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TXI。
- #define BSP_IRQ_CFG_SCI3_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TEI。

- #define BSP_IRQ_CFG_SCI3_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 ERI。
- #define BSP_IRQ_CFG_SCI3_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 AM。
- #define BSP_IRQ_CFG_SCI4_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 RXI。
- #define BSP_IRQ_CFG_SCI4_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TXI。
- #define BSP_IRQ_CFG_SCI4_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TEI。
- #define BSP_IRQ_CFG_SCI4_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 ERI。
- #define BSP_IRQ_CFG_SCI4_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 AM。
- #define BSP_IRQ_CFG_SCI5_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 RXI。
- #define BSP_IRQ_CFG_SCI5_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 TXI。
- #define BSP_IRQ_CFG_SCI5_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 TEI.

- #define BSP_IRQ_CFG_SCI5_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 ERI。
- #define BSP_IRQ_CFG_SCI5_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 AM。
- #define BSP_IRQ_CFG_SCI6_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 RXI。
- #define BSP_IRQ_CFG_SCI6_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 TXI。
- #define BSP_IRQ_CFG_SCI6_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 TEI.
- #define BSP_IRQ_CFG_SCI6_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 ERI。
- #define BSP_IRQ_CFG_SCI6_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 AM。
- #define BSP_IRQ_CFG_SCI7_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 RXI。
- #define BSP_IRQ_CFG_SCI7_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 TXI。
- #define BSP_IRQ_CFG_SCI7_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 TEI.

- #define BSP_IRQ_CFG_SCI7_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 ERI。
- #define BSP_IRQ_CFG_SCI7_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 AM。
- #define BSP_IRQ_CFG_SCI8_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 RXI。
- #define BSP_IRQ_CFG_SCI8_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 TXI。
- #define BSP_IRQ_CFG_SCI8_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 TEI.
- #define BSP_IRQ_CFG_SCI8_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 ERI。
- #define BSP_IRQ_CFG_SCI8_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 AM。
- #define BSP_IRQ_CFG_SCI9_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 RXI。
- #define BSP_IRQ_CFG_SCI9_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TXI。
- #define BSP_IRQ_CFG_SCI9_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TEI。

- #define BSP_IRQ_CFG_SCI9_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 ERI。
- #define BSP_IRQ_CFG_SCI9_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 AM。
- #define BSP_IRQ_CFG_SPI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPRI。
- #define BSP_IRQ_CFG_SPI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPTI。
- #define BSP_IRQ_CFG_SPI0_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPII。
- #define BSP_IRQ_CFG_SPI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPEI。
- #define BSP_IRQ_CFG_SPI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SP ELCTEND。
- #define BSP_IRQ_CFG_SPI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPRI。
- #define BSP_IRQ_CFG_SPI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPTI。
- #define BSP_IRQ_CFG_SPI1_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPII。

- #define BSP_IRQ_CFG_SPI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPEI。
- #define BSP_IRQ_CFG_SPI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SP ELCTEND。
- #define BSP_IRQ_CFG_QSPI_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  QSPI INTR。
- #define BSP_IRQ_CFG_SDHIMMC0_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC0_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC0_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 CARD。
- #define BSP_IRQ_CFG_SDHIMMC0_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ODMSDBREQ。
- #define BSP_IRQ_CFG_SDHIMMC1_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC1_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC1_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 CARD。

- #define BSP_IRQ_CFG_SDHIMMC1_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ODMSDBREQ。
- #define BSP_IRQ_CFG_DIVIDER_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  EXT DIVIDER INTMD。
- #define BSP_IRQ_CFG_SCE_PROC_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP PROC BUSY N。
- #define BSP_IRQ_CFG_SCE_ROMOK

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP ROMOK N。
- #define BSP_IRQ_CFG_SCE_LONG_PLG

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP LONG PLG N。
- #define BSP_IRQ_CFG_SCE_TEST_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP TEST BUSY N。
- #define BSP_IRQ_CFG_SCE_WRRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 0 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 1 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_4

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 4 N。
- #define BSP_IRQ_CFG_SCE_RDRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 0 N。

- #define BSP_IRQ_CFG_SCE_RDRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 1 N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_WRRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE WRRDY N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_RDRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE RDRDY N。
- #define BSP_IRQ_CFG_GLCDC_LINE_DETECT

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 0。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_1

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 1。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_2

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 2。
- #define BSP_IRQ_CFG_DRW_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  TWOD ENGINE IRQ。
- #define BSP_IRQ_CFG_JPEG_JEDI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JEDI。
- #define BSP_IRQ_CFG_JPEG_JDTI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JDTI。
- #define BSP_IRQ_CFG_VBATT_LVD

  初期値 :(#define BSP_IRQ_DISABLED)

  VBATT VBAT。

- #define BSP_IRQ_CFG_COMP_LP_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C0IRQ。
- #define BSP_IRQ_CFG_COMP_LP_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C1IRQ.。

9.2.5.5 ビルド時間設定 - ピン設定

このファイルには、ピン設定の配列が保存されます。起動時に main() が実行される前に、BSP はこの配列を反復処理し、配列内の設定に基づいて MCU のポート ピンを初期化します。

I：このファイルは ISDE Pin Configurator を使用して作成することをお勧めします。

9.2.6 PE-HMI1 ボードの BSP

PE-HMI1 ボードの BSP。

PE-HMI1 は、ヒューマン マシン インタフェース（HMI）アプリケーションで Renesas SynergyTM S7G2 マイクロコントローラを評価するための開発ボードです。このボードは、サーモスタット、セキュリティ パネル、医療モニタリング装置などの HMI アプリケーションの迅速な開発を可能にする静電容量式タッチ 7 インチ WVGA TFTLCD グラフィックス ディスプレイと複数の有線および無線インタフェースを特徴とします。TFTLCD ディスプレイと通信機能が付属した PE-HMI1 キットは、実際の HMI 最終製品の機能と外観に厳密に適合します。

9.2.6.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。ヘッダー ファイルで次の設定を構成します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.6.2 ビルド時間設定 - 全般

このファイルには、全般 BSP 設定用のコンパイル時間設定オプションが保存されます。スタック サイズや ROM レジスタなどをこのファイルで設定します。

部品番号情報

MCU の製品部品番号を入力します。この情報は、パッケージ サイズやメモリ サイズなどの MCU に関する情報の取得に使用されます。この情報を解釈しやすくするために、複数のマクロを使って部品番号を定義します。

```
R7FS 7 G 2 7 H 2 A01 C BD
|  |||||||  ||  Macro Name                  Description
|  |||||||  ||__BSP_CFG_MCU_PART_PACKAGE     = Package type, number of pins
|  |||||||  |____not used                    = Quality ID
|  |||||||_________not used                   = Software ID
|  ||||||____________not used                  = Operating range
|  |||||_______________BSP_CFG_MCU_PART_MEMORY_SIZE  = ROM, RAM, and Data Flash Capacity
|  ||||________________BSP_CFG_MCU_PART_FEATURE_SET  = Superset, no encryption
|  |||___________________not used                  = Document index
|  ||______________________BSP_CFG_MCU_PART_CORE        = Core & frequency (CM4, 240MHz)
|  |_________________________BSP_CFG_MCU_PART_SERIES      = Performance category (High-performance)
|____________________________not used                  = Renesas Synergy MCU
```

- #define BSP_CFG_MCU_PART_PACKAGE

  初期値 :(0x2)

  パッケージ タイプ。次の値に基づいてマクロ定義をセットします。

```
Character(s) = Value for macro = Package Type/Number of Pins/Other Info
BJ          = 0x0           = BGA/121
BG          = 0x1           = BGA/176
BD          = 0x2           = BGA/224
FL          = 0x3           = LQFP/48
FM          = 0x4           = LQFP/64
FP          = 0x5           = LQFP/100
FB          = 0x6           = LQFP/144
FC          = 0x7           = LQFP/176
LM          = 0x8           = LGA/36
LA          = 0x9           = LGA/100/5.5x5.5
LJ          = 0xA           = LGA/100/7x7
LK          = 0xB           = LGA/145
NF          = 0xC           = WQFN/40
NE          = 0xD           = WQFN/48
NB          = 0xE           = WQFN/64
```

- #define BSP_CFG_MCU_PART_MEMORY_SIZE

  初期値 :(0x11)

  ROM、RAM、およびデータ フラッシュ容量。

```
Character(s) = Value for macro = ROM Size/Ram Size/Data Flash Size
6          = 0x06          = 64KB
7          = 0x07          = 128KB
8          = 0x08          = 256KB
A          = 0x0A          = 512KB
C          = 0x0C          = 1MB
E          = 0x0E          = 2MB
G          = 0x10          = 3MB
H          = 0x11          = 4MB/640KB/64KB
```

- #define BSP_CFG_MCU_PART_CORE

  初期値 :(0x10)

  コアおよび周波数。

```
Character(s) = Value for macro = Description
2          = 0x02          = CM0+, 32MHz
A          = 0x0A          = CM4,  48MHz
D          = 0x0D          = CM4, 120MHz
G          = 0x10          = CM4, 240MHz
```

- #define BSP_CFG_MCU_PART_SERIES

  初期値 :(0x7)

  シリーズ

```
Character(s) = Value for macro = Description
1          = 0x1           = Ultra-low Power (up to 32MHz)
3          = 0x3           = High-efficiency (33-100MHz)
5          = 0x5           = High-integration (101MHz-200MHz)
7          = 0x7           = High-performance (201MHz-300MHz)
```

スタックおよびヒープ サイズの設定

- #define BSP_CFG_STACK_MAIN_BYTES

  初期値 :(0x1000)

  バイト単位のメイン スタック サイズ。これは、リセット以降に使用される必須のスタックです。例外では常にメイン スタックが使用されます。

- #define BSP_CFG_STACK_PROCESS_BYTES

  初期値 :(0)

  バイト単位のプロセス スタック サイズ。プロセス スタックの使用は任意です。BSP はプロセス スタックを使用しません。ユーザーがアプリケーション内でプロセス スタックを使用する場合は、CONTROL レジスタの SPSEL ビットをセットする必要があります。

- #define BSP_CFG_HEAP_BYTES

  初期値 :(0x400)

  バイト単位のヒープ サイズ。

オプション設定メモリ （ROM レジスタ）の設定

特定のレジスタが ROM に保存され、リセット以降の MCU の設定に使用されます。

I：レジスタのデフォルト値を使用するには、マクロをすべて 0xFF のままにします。

I：すべてのオプションがデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_OFS0

  初期値 :(0xFFFFFFFF)

  WDT 設定と IWDT 設定を構成します。OFS0 - オプション関数選択レジスタ 0

- b31 予約済み （1 にセット）
- b30 WDTSTPCTL - WDT 停止制御 - （0= 集計を継続、1= スリープ モードに入ったら停止）
- b29 予約済み （1 にセット）
- b28 WDTRSTIRQS - WDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b27:b26 WDTRPSS - WDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b25:b24 WDTRPES - WDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b23:b20 WDTCKS - WDT クロック周波数分割比 - （1=/4、4=/64、0xF=/128、6=/512、7=/2048、8=/8192）
- b19:b18 WDTTOPS - WDT タイムアウト期間選択 - （0=1024 サイクル、1=4096、2=8192、3=16384）

- b17 WDTSTRT - WDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b16:b15 予約済み （1 にセット）
- b14 IWDTSTPCTL - IWDT Sleep Stop Control - （0=counting continues, 1=stop w/selected low power modes)
- b13 予約済み （1 にセット）
- b12 IWDTRSTIRQS - IWDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b11:b10 IWDTRPSS - IWDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b9:b8 IWDTRPES - IWDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b7:b4 IWDTCKS - IWDT クロック周波数分割比 - （0= なし、2=/16、3 = /32、4=/64、0xF=/128、5=/256）
- b3:b2 IWDTTOPS - IWDT タイムアウト期間選択 - （0=128 サイクル、1=512、2=1024、3=2048）
- b1 IWDTSTRT - IWDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b0 予約済み （1 にセット）

l :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

- #define BSP_CFG_ROM_REG_OFS1

  初期値 :(0xFFFFFFFF)

  電圧検出 0 回路と HOCO をリセット後に有効にするかどうかを設定します。OFS1 - オプション関数選択レジスタ 1

- b31:b9 予約済み （1 にセット）
- b10:b9 HOCOFRQ0 - HOCO 周波数設定 （bsp_clock_cfg.h 内の BSP_CFG_HOCO_FREQUENCY を使用してセット）
- b8 HOCOEN - リセット後の HOCO 振動の有効化 / 無効化 （0= 有効、1= 無効）
- b7:b3 予約済み （1 にセット）
- b2 LVDAS - リセット後の電圧検出 0 回路の有効化 / 無効化の選択 （0= 有効、1= 無効）
- b1:b0 VDSEL - 電圧検出 0 レベル選択 （1=2.94V、2=2.87V、3=2.80V）

l :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

セキュリティ MPU オプション

以下のセキュリティ MPU 領域のそれぞれに 3 つずつのマクロがあります。

- オプションを有効 / 無効にします。1 = 無効、0 = 有効を使用します。
- 開始アドレス - この MPU 領域の開始アドレス。
- 終了アドレス - この MPU 領域の終了アドレス。

I : 領域によって、セット可能なアドレスに対する制限が異なります。

I : すべての領域がデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_MPU_PC0_ENABLE
  初期値 :(1)
  PC 領域 0 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_PC0_START
  初期値 :(0xFFFFFFFF)
  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC0_END
  初期値 :(0xFFFFFFFF)
  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_PC1_ENABLE
  初期値 :(1)
  PC 領域 1 を有効または無効にします。
- 0 = 有効
- 1 = 無効

- #define BSP_CFG_ROM_REG_MPU_PC1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION0_ENABLE

  初期値 :(1)

  メモリ領域 0 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0x00FFFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0x00FFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION1_ENABLE

  初期値 :(1)

  メモリ領域 1 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION2_ENABLE

  初期値 :(1)

  メモリ領域 2 を有効または無効にします。
- 0 = 有効

- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION2_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION2_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION3_ENABLE

  初期値 :(1)

  メモリ領域 3 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION3_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION3_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF

ID コード保護

必要な ID コードをセットします。この値をセットして MCU にプログラムしたら、その ID コードを覚えておく必要があります。これは、接続時にデバッガーから要求されるためです。ID コードは 16 バイト長です。以下のマクロは ID コードを 4 バイト セクションで定義します。

I :ID コードを使用しない（保護が無効になっている）場合は、デフォルト （すべて 0xFF）のままにします。

I a: 有効な ID コードの設定方法については、ユーザーズマニュアルを参照してください。

- #define BSP_CFG_ID_CODE_LONG_1

  初期値 :(0xFFFFFFFF)

  最下位 4 バイト セクション、アドレス 0x40120050。MSB から LSB へ : ID コード 16、ID コード 15、ID コード 14、ID コード 13。

- #define BSP_CFG_ID_CODE_LONG_2

  初期値 :(0xFFFFFFFF)

  第 2 ID コード セクション、アドレス 0x40120054。MSB から LSB へ : ID コード 12、ID コード 11、ID コード 10、ID コード 9。

- #define BSP_CFG_ID_CODE_LONG_3

  初期値 :(0xFFFFFFFF)

  第 3 ID コード セクション、アドレス 0x40120058。MSB から LSB へ : ID コード 8、ID コード 7、ID コード 6、ID コード 5。

- #define BSP_CFG_ID_CODE_LONG_4

  初期値 :(0xFFFFFFFF)

  第 4 ID コード セクション、アドレス 0x4012005C。MSB から LSB へ : ID コード 4、ID コード 3、ID コード 2、ID コード 1。

その他のハードウェア オプション

- #define BSP_CFG_MCU_VCC_MV

  初期値 :(3300)

  このマクロは、MCU に供給される電圧 （Vcc）を定義するために使用されます。また、このマクロはミリボルト単位で定義されます。実際に MCU 上の何かを変更するわけではありません。選択したモジュールがこの情報を必要とするため、ここで定義します。

プロジェクト全体のソフトウェア オプション

- #define BSP_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  デフォルトで、SSP モジュールは入力パラメータが有効かどうかをチェックします。これは開発において役立ちますが、状況に応じて実稼働コードに対し無効化することもできます。その目的は、実行時間とコード スペースを節約することです。このマクロは、パラメータ チェックを有効または無効にするためのグローバル設定です。各モジュールは、同じ目的の独自のローカル マクロも備えています。デフォルトで、ローカル マクロはここからグローバル値を取得しますが、オーバーライドすることができます。そのため、ローカル設定の方がこのグローバル設定より優先されます。パラメータ チェックの無効化は、入力が適切であることが分かっており、速度の向上またはコード スペースの削減が必要な場合にのみ使用してください。

- 0 = パラメータ チェックのグローバル設定が無効になっています。
- 1 = パラメータ チェックのグローバル設定が有効になっています （デフォルト）。

- #define BSP_CFG_RTOS

  初期値 :(1)

  使用する RTOS を指定します。

- 0 = RTOS なし
- 0 = RTOS なし
- 2 = その他
- #define BSP_CFG_ASSERT

  初期値 :(0)

  SSP_ASSERT が失敗した場合の処理を指定します。

- 0 = SSP_ERR_ASSERTION を返します。
- 1 = ssp_error_log を呼び出してから、SSP_ERR_ASSERTION を返します。ssp_error_log は脆弱な関数であり、ssp/src/bsp/mcu/all/bsp_common.h 内のプロトタイプに基づいてユーザー コード内でオーバーライドする必要があることに注意してください。
- 2 = 標準のアサート ライブラリを使用して実行を停止します。
- #define BSP_CFG_ERROR_LOG

  初期値 :(0)

  SSP 関数からエラー コードが返された場合の処理を指定します。

- 0 = エラー コードを返します。
- 1 = ssp_error_log を呼び出してから、エラー コードを返します。ssp_error_log は脆弱な関数であり、ssp/src/bsp/mcu/all/bsp_common.h 内のプロトタイプに基づいてユーザー コード内でオーバーライドする必要があることに注意してください。

9.2.6.3 ビルド時間設定 - クロック

このファイルには、ビルド時間クロック設定オプションが保存されます。main() が実行される前に、BSP はこれらのマクロを使用して、ユーザーの MCU のクロックを設定します。

クロック設定オプション。入力クロック周波数が指定されてから、使用する乗算器を指定することによってシステム クロックがセットされます。乗算器の設定値はクロック レジスタのセットに使用されます。12MHz XTAL と 240MHz の CPU クロックを使用したボードの例を以下に示します。

```
BSP_CFG_XTAL_HZ = 12000000
BSP_CFG_PLL_DIV = 1  (no division)
BSP_CFG_PLL_MUL = 20 (12MHz x 20 = 240MHz)
```

```
BSP_CFG_ICK_DIV =  1     : System Clock (ICLK)       =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_ICK_DIV)  = 240MHz
BSP_CFG_PCKA_DIV = 2     : Peripheral Clock A (PCLKA) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKA_DIV) = 120MHz
BSP_CFG_PCKB_DIV = 4     : Peripheral Clock B (PCLKB) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKB_DIV) = 60MHz
BSP_CFG_PCKC_DIV = 4     : Peripheral Clock C (PCLKC) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKC_DIV) = 60MHz
BSP_CFG_PCKD_DIV = 2     : Peripheral Clock D (PCLKD) =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_PCKD_DIV) = 120MHz
BSP_CFG_FCK_DIV =  4     : Flash IF Clock (FCLK)     =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_FCK_DIV)  = 60MHz
BSP_CFG_BCK_DIV =  2     : External Bus Clock (BCK)   =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_BCK_DIV)  = 120MHz
BSP_CFG_UCK_DIV =  5     : USB Clock (UCLK)          =
                   (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) /
                   BSP_CFG_UCK_DIV)  = 48MHz
```

定義

- #define BSP_CFG_CLOCK_SOURCE

  初期値 :(CGC_CLOCK_PLL)

  クロック ソース選択 （CKSEL）。選択されたクロックがシステム クロックとすべてのペリフェラル クロックに提供されるベース クロックになります。また、フラッシュ クロックと外部バス クロックにも使用されます。

  次のように、列挙のいずれかにマクロをセットします。

```
Clock                              -  Enumeration to use for macro
High Speed On-Chip Oscillator  (HOCO)     -  CGC_CLOCK_HOCO
Middle Speed On-Chip Oscillator (MOCO)    -  CGC_CLOCK_MOCO
Low Speed On-Chip Oscillator   (LOCO)     -  CGC_CLOCK_LOCO
Main Clock Oscillator                -  CGC_CLOCK_MAIN_OSC
Sub-Clock Oscillator                 -  CGC_CLOCK_SUBCLOCK
PLL Circuit                        -  CGC_CLOCK_PLL
```

- #define BSP_CFG_XTAL_HZ

  初期値 :(24000000)

  XTAL - Hz 単位の入力クロック周波数

- #define BSP_CFG_HOCO_FREQUENCY

  初期値 :(0)

  HOCO は複数の周波数で動作できます。下のマクロを使用していずれか 1 つを選択します。リセット以降に使用される周波数は OFS1.HOCOFRQ0 ビットによって異なります。

```
Available frequency settings:
0 = 16MHz
1 = 18MHz
2 = 20MHz
```

- #define BSP_CFG_PLL_SOURCE

  初期値 :(CGC_CLOCK_MAIN_OSC)

  PLL クロック ソース （PLLSRCEL）。PLL 回路に入力するクロック ソースを選択します。

```
Available clock sources:    -  Enumeration to use for macro
0 = Main clock (default)    -  CGC_CLOCK_MAIN_OSC
1 = HOCO                    -  CGC_CLOCK_HOCO
```

- #define BSP_CFG_PLL_DIV

  初期値 :(CGC_PLL_DIV_2)

  PLL 入力周波数分割比選択 （PLIDIV）

  使用可能な除数 = - /1 (除算なし)、- /2、- /3

  I ：マクロ定義を 'CGC_PLL_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PLL_MUL

  初期値 :(20.0)

  PLL 周波数増倍係数選択 （PLLMUL）

  使用可能な乗算器 = x10.0 ～ x30.0 の 0.5 ずつのインクリメント （10.0、10.5、11.0、11.5 ... 29.0、29.5、30.0 など）

- #define BSP_CFG_ICK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  システム クロック除算器 （ICK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I ：マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKA_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック A 除算器 （PCKA）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKB_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  ペリフェラル モジュール クロック B 除算器 （PCKB）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKC_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  ペリフェラル モジュール クロック C 除算器 （PCKC）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKD_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック D 除算器 （PCKD）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  外部バス クロック除算器 （BCLK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_FCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  フラッシュ IF クロック除算器 （FCK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_UCK_DIV

  初期値 :(CGC_USB_CLOCK_DIV_5)

  USB クロック除算器選択。

  使用可能な除数 = /3、/4、/5

  I : マクロ定義を 'CGC_USB_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCLK_OUTPUT

  初期値 :(1)

  BCLK 出力ピンの設定

  使用可能なオプション :

- 0 = 出力なし
- 1 = BCK 周波数
- 2 = BCK/2 周波数

  I : このマクロは、外部バスが有効になっている場合にのみ効果があります。

- #define BSP_CFG_SDCLK_OUTPUT

  初期値 :(1)

  SDCLK 出力ピンの設定

  使用可能なオプション :

- 0 = 出力なし
- 1 = BCK 周波数

  I : このマクロは、外部バスが有効になっている場合にのみ効果があります。

9.2.6.4 ビルド時間設定 - 割り込み

割り込みをトリガーするように設定可能な ICU ELC モジュールに対するイベント入力を以下に列挙します。特定のイベントで割り込みをトリガーできるようにするには、マクロの定義を BSP IRQ DISABLED から割り込みに使用する割り込み優先レベルに変更します。この MCU の有効な優先順位の範囲は 0 ～ 15 で、小さい数値ほど優先順位が高くなります （0 = 最高優先順位、15 = 最低優先順位）。BSP は、ベクター テーブルへの適切な関数の配置を処理して、割り込み優先レベルをセットし、リセットからユーザーのアプリケーションが呼び出されるまでの間に割り込みを生成するように ELC イベントを設定します。

定義

- #define BSP_IRQ_CFG_ICU_IRQ0

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT0 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ1

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT1 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ2

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT2 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ3

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT3 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ4

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT4 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ5

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT5 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ6

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT6 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ7

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT7 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ8

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT8 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ9

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT9 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ10
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT10 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ11
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT11 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ12
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT12 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ13
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT13 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ14
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT14 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ15
  初期値 :(#define BSP_IRQ_DISABLED)
  PORT15 IRQ。
- #define BSP_IRQ_CFG_DMAC0_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC0 DMAC。
- #define BSP_IRQ_CFG_DMAC1_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC1 DMAC。
- #define BSP_IRQ_CFG_DMAC2_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC2 DMAC。
- #define BSP_IRQ_CFG_DMAC3_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  DMAC3 DMAC。

- #define BSP_IRQ_CFG_DMAC4_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC4 DMAC。
- #define BSP_IRQ_CFG_DMAC5_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC5 DMAC。
- #define BSP_IRQ_CFG_DMAC6_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC6 DMAC。
- #define BSP_IRQ_CFG_DMAC7_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC7 DMAC。
- #define BSP_IRQ_CFG_DTC_COMPLETE

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC COMPLETE。
- #define BSP_IRQ_CFG_DTC_END

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC DTC END。
- #define BSP_IRQ_CFG_ICU_SNOOZE_CANCEL

  初期値 :(#define BSP_IRQ_DISABLED)

  ICU CANCELING SNOOZE MODE。
- #define BSP_IRQ_CFG_FCU_FIFERR

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FIFERR。
- #define BSP_IRQ_CFG_FCU_FRDYI

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FRDYI。
- #define BSP_IRQ_CFG_LVD_LVD1

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD1 LVD1。

- #define BSP_IRQ_CFG_LVD_LVD2

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD2 LVD2。
- #define BSP_IRQ_CFG_CGC_MOSC_STOP

  初期値 :(#define BSP_IRQ_DISABLED)

  MOSC OSC STOP。
- #define BSP_IRQ_CFG_LPM_SNOOZE_REQUEST

  初期値 :(#define BSP_IRQ_DISABLED)

  CPUSYS SNOOZE MODE ENTRY FLAG。
- #define BSP_IRQ_CFG_AGT0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMAI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMBI。
- #define BSP_IRQ_CFG_AGT1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMAI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMBI。
- #define BSP_IRQ_CFG_IWDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  IWDT NMIUNDF N。

- #define BSP_IRQ_CFG_WDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CWDT NMIUNDF N。
- #define BSP_IRQ_CFG_RTC_ALARM

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC ALM。
- #define BSP_IRQ_CFG_RTC_PERIOD

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC PRD。
- #define BSP_IRQ_CFG_RTC_CARRY

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC CUP。
- #define BSP_IRQ_CFG_ADC0_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 ADI。
- #define BSP_IRQ_CFG_ADC0_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 GBADI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPAI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPBI。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MISMATCH。

- #define BSP_IRQ_CFG_ADC1_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 ADI。
- #define BSP_IRQ_CFG_ADC1_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 GBADI。
- #define BSP_IRQ_CFG_ADC1_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 CMPAI。
- #define BSP_IRQ_CFG_ADC1_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 CMPBI。
- #define BSP_IRQ_CFG_ADC1_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC1_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 COMPARE MISMATCH。
- #define BSP_IRQ_CFG_COMP_HS_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP OC0 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD1 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_2

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD2 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_3

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD3 COMP IRQ。

- #define BSP_IRQ_CFG_COMP_HS_4
  初期値 :(#define BSP_IRQ_DISABLED)
  COMP RD4 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_5
  初期値 :(#define BSP_IRQ_DISABLED)
  COMP RD5 COMP IRQ。
- #define BSP_IRQ_CFG_USBFS_FIFO_0
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS D0FIFO。
- #define BSP_IRQ_CFG_USBFS_FIFO_1
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS D1FIFO.。
- #define BSP_IRQ_CFG_USBFS_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS USBI。
- #define BSP_IRQ_CFG_USBFS_RESUME
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS USBR。
- #define BSP_IRQ_CFG_IIC0_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 RXI。
- #define BSP_IRQ_CFG_IIC0_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 TXI。
- #define BSP_IRQ_CFG_IIC0_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 TEI。
- #define BSP_IRQ_CFG_IIC0_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 EEI。

- #define BSP_IRQ_CFG_IIC0_WUI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC0 WUI。
- #define BSP_IRQ_CFG_IIC1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 RXI。
- #define BSP_IRQ_CFG_IIC1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TXI。
- #define BSP_IRQ_CFG_IIC1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TEI。
- #define BSP_IRQ_CFG_IIC1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 EEI。
- #define BSP_IRQ_CFG_IIC2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 RXI。
- #define BSP_IRQ_CFG_IIC2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 TXI。
- #define BSP_IRQ_CFG_IIC2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 TEI。
- #define BSP_IRQ_CFG_IIC2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 EEI。
- #define BSP_IRQ_CFG_SSI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSITXI。

- #define BSP_IRQ_CFG_SSI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIRXI。
- #define BSP_IRQ_CFG_SSI0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIF。
- #define BSP_IRQ_CFG_SSI1_TXI_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIRT。
- #define BSP_IRQ_CFG_SSI1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIF。
- #define BSP_IRQ_CFG_SRC_INPUT_FIFO_EMPTY

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC IDEI。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_FULL

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC ODFI。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC OVF。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC UDF。
- #define BSP_IRQ_CFG_SRC_CONVERSION_END

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC CEF。
- #define BSP_IRQ_CFG_PDC_RECEIVE_DATA_READY

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCDFI。

- #define BSP_IRQ_CFG_PDC_FRAME_END

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCFEI。
- #define BSP_IRQ_CFG_PDC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCERI。
- #define BSP_IRQ_CFG_CTSU_WRITE

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUWR。
- #define BSP_IRQ_CFG_CTSU_READ

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSURD。
- #define BSP_IRQ_CFG_CTSU_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUFN。
- #define BSP_IRQ_CFG_KEY_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  KEY INTKR。
- #define BSP_IRQ_CFG_DOC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DOC DOPCF。
- #define BSP_IRQ_CFG_CAC_FREQUENCY_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC FERRF。
- #define BSP_IRQ_CFG_CAC_MEASUREMENT_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC MENDF。
- #define BSP_IRQ_CFG_CAC_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC OVFF。

- #define BSP_IRQ_CFG_CAN0_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 ERS。
- #define BSP_IRQ_CFG_CAN0_FIFO_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXF。
- #define BSP_IRQ_CFG_CAN0_FIFO_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXF。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 RXM。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 TXM。
- #define BSP_IRQ_CFG_CAN1_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 ERS。
- #define BSP_IRQ_CFG_CAN1_FIFO_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 RXF。
- #define BSP_IRQ_CFG_CAN1_FIFO_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 TXF。
- #define BSP_IRQ_CFG_CAN1_MAILBOX_RX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 RXM。
- #define BSP_IRQ_CFG_CAN1_MAILBOX_TX

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN21 TXM。

- #define BSP_IRQ_CFG_IOPORT_EVENT_1
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP A。
- #define BSP_IRQ_CFG_IOPORT_EVENT_2
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP B。
- #define BSP_IRQ_CFG_IOPORT_EVENT_3
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP C。
- #define BSP_IRQ_CFG_IOPORT_EVENT_4
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP D。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_0
  初期値 :(#define BSP_IRQ_DISABLED)
  ELC0 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_1
  初期値 :(#define BSP_IRQ_DISABLED)
  ELC1 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_POEG0_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT0。
- #define BSP_IRQ_CFG_POEG1_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT1。
- #define BSP_IRQ_CFG_POEG2_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT2。
- #define BSP_IRQ_CFG_POEG3_EVENT
  初期値 :(#define BSP_IRQ_DISABLED)
  POEG GROUP EVENT3。

- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT0_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT0_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT0_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT0_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT0_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT0_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT0_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 AD TRIG A。
- #define BSP_IRQ_CFG_GPT0_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 AD TRIG B。

- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT1_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT1_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT1_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT1_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT1_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT1_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT1_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 AD TRIG A。
- #define BSP_IRQ_CFG_GPT1_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 AD TRIG B。

- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT2_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT2_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT2_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT2_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT2_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT2_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT2_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 AD TRIG A。
- #define BSP_IRQ_CFG_GPT2_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 AD TRIG B。

- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT3_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT3_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT3_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT3_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT3_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT3_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT3_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 AD TRIG A。
- #define BSP_IRQ_CFG_GPT3_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 AD TRIG B。

- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT4_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT4_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT4_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT4_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT4_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT4_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT4_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 AD TRIG A。
- #define BSP_IRQ_CFG_GPT4_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 AD TRIG B。

- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT5_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT5_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT5_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT5_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT5_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT5_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT5_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 AD TRIG A。
- #define BSP_IRQ_CFG_GPT5_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 AD TRIG B。

- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT6_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT6_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT6_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT6_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT6_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT6_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT6_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 AD TRIG A。
- #define BSP_IRQ_CFG_GPT6_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 AD TRIG B。

- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT7_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT7_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT7_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT7_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT7_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT7_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT7_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 AD TRIG A。
- #define BSP_IRQ_CFG_GPT7_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 AD TRIG B。

- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT8_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT8_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT8_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT8_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT8_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT8_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT8_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 AD TRIG A。
- #define BSP_IRQ_CFG_GPT8_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 AD TRIG B。

- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT9_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT9_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT9_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT9_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT9_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT9_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT9_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 AD TRIG A。
- #define BSP_IRQ_CFG_GPT9_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 AD TRIG B。

- #define BSP_IRQ_CFG_GPT10_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT10_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT10_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT10_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT10_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT10_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT10_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT10_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT10_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 AD TRIG A。
- #define BSP_IRQ_CFG_GPT10_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 AD TRIG B。

- #define BSP_IRQ_CFG_GPT11_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT11_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT11_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT11_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT11_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT11_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT11_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT11_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT11_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 AD TRIG A。
- #define BSP_IRQ_CFG_GPT11_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 AD TRIG B。

- #define BSP_IRQ_CFG_GPT12_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT12_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT12_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT12_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT12_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT12_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT12_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT12_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT12_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 AD TRIG A。
- #define BSP_IRQ_CFG_GPT12_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 AD TRIG B。

- #define BSP_IRQ_CFG_GPT13_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 CAPTURE COMPARE INT A。
- #define BSP_IRQ_CFG_GPT13_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT13_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT13_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT13_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT13_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT13_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT13_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT13_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 AD TRIG A。
- #define BSP_IRQ_CFG_GPT13_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 AD TRIG B。

- #define BSP_IRQ_CFG_OPS_UVW_EDGE

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT UVW EDGE。
- #define BSP_IRQ_CFG_EPTPC_IPLS

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER IPLS。
- #define BSP_IRQ_CFG_EPTPC_MINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER MINT。
- #define BSP_IRQ_CFG_EPTPC_PINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER PINT。
- #define BSP_IRQ_CFG_EDMAC0_EINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER EINT0。
- #define BSP_IRQ_CFG_EDMAC1_EINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER EINT1。
- #define BSP_IRQ_CFG_EPTPC_TIMER0_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER0 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER1_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER1 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER2_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER2 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER3_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER3 RISE。

- #define BSP_IRQ_CFG_EPTPC_TIMER4_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER4 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER5_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER5 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER0_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER0 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER1_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER1 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER2_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER2 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER3_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER3 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER4_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER4 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER5_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER5 FALL。
- #define BSP_IRQ_CFG_USBHS_FIFO_0

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS D0FIFO。
- #define BSP_IRQ_CFG_USBHS_FIFO_1

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS D1FIFO.。

- #define BSP_IRQ_CFG_USBHS_USB_INT_RESUME

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS USBIR。
- #define BSP_IRQ_CFG_SCI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI。
- #define BSP_IRQ_CFG_SCI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TXI。
- #define BSP_IRQ_CFG_SCI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TEI。
- #define BSP_IRQ_CFG_SCI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 ERI。
- #define BSP_IRQ_CFG_SCI0_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 AM。
- #define BSP_IRQ_CFG_SCI0_RXI_OR_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI OR ERI。
- #define BSP_IRQ_CFG_SCI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 RXI。
- #define BSP_IRQ_CFG_SCI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TXI。
- #define BSP_IRQ_CFG_SCI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TEI。

- #define BSP_IRQ_CFG_SCI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 ERI。
- #define BSP_IRQ_CFG_SCI1_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 AM。
- #define BSP_IRQ_CFG_SCI2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 RXI。
- #define BSP_IRQ_CFG_SCI2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TXI。
- #define BSP_IRQ_CFG_SCI2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TEI。
- #define BSP_IRQ_CFG_SCI2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 ERI。
- #define BSP_IRQ_CFG_SCI2_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 AM。
- #define BSP_IRQ_CFG_SCI3_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 RXI。
- #define BSP_IRQ_CFG_SCI3_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TXI。
- #define BSP_IRQ_CFG_SCI3_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TEI。

- #define BSP_IRQ_CFG_SCI3_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 ERI。
- #define BSP_IRQ_CFG_SCI3_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 AM。
- #define BSP_IRQ_CFG_SCI4_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 RXI。
- #define BSP_IRQ_CFG_SCI4_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TXI。
- #define BSP_IRQ_CFG_SCI4_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TEI。
- #define BSP_IRQ_CFG_SCI4_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 ERI。
- #define BSP_IRQ_CFG_SCI4_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 AM。
- #define BSP_IRQ_CFG_SCI5_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 RXI。
- #define BSP_IRQ_CFG_SCI5_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 TXI。
- #define BSP_IRQ_CFG_SCI5_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 TEI.

- #define BSP_IRQ_CFG_SCI5_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 ERI。
- #define BSP_IRQ_CFG_SCI5_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 AM。
- #define BSP_IRQ_CFG_SCI6_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 RXI。
- #define BSP_IRQ_CFG_SCI6_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 TXI。
- #define BSP_IRQ_CFG_SCI6_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 TEI.
- #define BSP_IRQ_CFG_SCI6_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 ERI。
- #define BSP_IRQ_CFG_SCI6_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 AM。
- #define BSP_IRQ_CFG_SCI7_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 RXI。
- #define BSP_IRQ_CFG_SCI7_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 TXI。
- #define BSP_IRQ_CFG_SCI7_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 TEI.

- #define BSP_IRQ_CFG_SCI7_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 ERI。
- #define BSP_IRQ_CFG_SCI7_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 AM。
- #define BSP_IRQ_CFG_SCI8_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 RXI。
- #define BSP_IRQ_CFG_SCI8_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 TXI。
- #define BSP_IRQ_CFG_SCI8_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 TEI.
- #define BSP_IRQ_CFG_SCI8_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 ERI。
- #define BSP_IRQ_CFG_SCI8_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 AM。
- #define BSP_IRQ_CFG_SCI9_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 RXI。
- #define BSP_IRQ_CFG_SCI9_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TXI。
- #define BSP_IRQ_CFG_SCI9_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TEI。

- #define BSP_IRQ_CFG_SCI9_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  SCI9 ERI。
- #define BSP_IRQ_CFG_SCI9_AM
  初期値 :(#define BSP_IRQ_DISABLED)
  SCI9 AM。
- #define BSP_IRQ_CFG_SPI0_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPRI。
- #define BSP_IRQ_CFG_SPI0_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPTI。
- #define BSP_IRQ_CFG_SPI0_IDLE
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPII。
- #define BSP_IRQ_CFG_SPI0_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SPEI。
- #define BSP_IRQ_CFG_SPI0_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI0 SP ELCTEND。
- #define BSP_IRQ_CFG_SPI1_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI1 SPRI。
- #define BSP_IRQ_CFG_SPI1_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI1 SPTI。
- #define BSP_IRQ_CFG_SPI1_IDLE
  初期値 :(#define BSP_IRQ_DISABLED)
  RSPI1 SPII。

- #define BSP_IRQ_CFG_SPI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPEI。
- #define BSP_IRQ_CFG_SPI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SP ELCTEND。
- #define BSP_IRQ_CFG_QSPI_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  QSPI INTR。
- #define BSP_IRQ_CFG_SDHIMMC0_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC0_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC0_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 CARD。
- #define BSP_IRQ_CFG_SDHIMMC0_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ODMSDBREQ。
- #define BSP_IRQ_CFG_SDHIMMC1_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC1_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC1_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 CARD。

- #define BSP_IRQ_CFG_SDHIMMC1_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ODMSDBREQ。
- #define BSP_IRQ_CFG_DIVIDER_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  EXT DIVIDER INTMD。
- #define BSP_IRQ_CFG_SCE_PROC_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP PROC BUSY N。
- #define BSP_IRQ_CFG_SCE_ROMOK

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP ROMOK N。
- #define BSP_IRQ_CFG_SCE_LONG_PLG

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP LONG PLG N。
- #define BSP_IRQ_CFG_SCE_TEST_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP TEST BUSY N。
- #define BSP_IRQ_CFG_SCE_WRRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 0 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 1 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_4

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 4 N。
- #define BSP_IRQ_CFG_SCE_RDRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 0 N。

- #define BSP_IRQ_CFG_SCE_RDRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 1 N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_WRRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE WRRDY N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_RDRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE RDRDY N。
- #define BSP_IRQ_CFG_GLCDC_LINE_DETECT

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 0。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_1

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 1。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_2

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 2。
- #define BSP_IRQ_CFG_DRW_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  TWOD ENGINE IRQ。
- #define BSP_IRQ_CFG_JPEG_JEDI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JEDI。
- #define BSP_IRQ_CFG_JPEG_JDTI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JDTI。

9.2.6.5 ビルド時間設定 - ピン設定

このファイルには、ピン設定の配列が保存されます。起動時に main() が実行される前に、BSP はこの配列を反復処理し、配列内の設定に基づいて MCU のポート ピンを初期化します。

I：このファイルは ISDE Pin Configurator を使用して作成することをお勧めします。

9.2.7 SK-S7G2 ボードの BSP

SK-S7G2 ボードの BSP。

SK-S7G2 は、LQFP176 パッケージに含まれる S7G2 用のスターター キットです。

9.2.7.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。現時点で、次の設定値を構成するためのヘッダー ファイルが存在します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.2.7.2 ビルド時間設定 - 全般

このファイルには、全般 BSP 設定用のコンパイル時間設定オプションが保存されます。スタック サイズや ROM レジスタなどをこのファイルで設定します。

部品番号情報

MCU の製品部品番号を入力します。この情報は、パッケージ サイズやメモリ サイズなどの MCU に関する情報の取得に使用されます。この情報を解釈しやすくするために、複数のマクロを使って部品番号を定義します。

```
R7FS 7 G 2 7 H 3 A01 C FC
|  |||||||  || Macro Name                  Description
|  |||||||  ||__BSP_CFG_MCU_PART_PACKAGE    = Package type, number of pins
|  |||||||  |____not used                   = Quality ID
|  |||||||_________not used                  = Software ID
|  ||||||__________not used                  = Operating range
|  |||||_____________BSP_CFG_MCU_PART_MEMORY_SIZE  = ROM, RAM, and Data Flash Capacity
|  ||||_______________BSP_CFG_MCU_PART_FEATURE_SET  = Superset, no encryption
|  |||__________________not used                  = Document index
|  ||____________________BSP_CFG_MCU_PART_CORE       = Core & frequency (CM4, 240MHz)
|  |_____________________BSP_CFG_MCU_PART_SERIES     = Performance category (High-performance)
|________________________not used                  = Renesas Synergy MCU
```

- #define BSP_CFG_MCU_PART_PACKAGE

  初期値 :(0x7)

  パッケージ タイプ。次の値に基づいてマクロ定義をセットします。

```
Character(s) = Value for macro = Package Type/Number of Pins/Other Info
BJ      = 0x0       = BGA/121
BG      = 0x1       = BGA/176
BD      = 0x2       = BGA/224
FL      = 0x3       = LQFP/48
FM      = 0x4       = LQFP/64
FP      = 0x5       = LQFP/100
FB      = 0x6       = LQFP/144
FC      = 0x7       = LQFP/176
LM      = 0x8       = LGA/36
LA      = 0x9       = LGA/100/5.5x5.5
LJ      = 0xA       = LGA/100/7x7
LK      = 0xB       = LGA/145
NF      = 0xC       = WQFN/40
NE      = 0xD       = WQFN/48
NB      = 0xE       = WQFN/64
```

- #define BSP_CFG_MCU_PART_MEMORY_SIZE

  初期値 :(0x11)

  ROM、RAM、およびデータ フラッシュ容量。

```
Character(s) = Value for macro = ROM Size/Ram Size/Data Flash Size
6       = 0x06      = 64KB
7       = 0x07      = 128KB
8       = 0x08      = 256KB
A       = 0x0A      = 512KB
C       = 0x0C      = 1MB
E       = 0x0E      = 2MB
G       = 0x10      = 3MB
H       = 0x11      = 4MB/640KB/64KB
```

- #define BSP_CFG_MCU_PART_CORE

  初期値 :(0x10)

  コアおよび周波数。

```
Character(s) = Value for macro = Description
2       = 0x02      = CM0+, 32MHz
A       = 0x0A      = CM4,  48MHz
D       = 0x0D      = CM4, 120MHz
G       = 0x10      = CM4, 240MHz
```

- #define BSP_CFG_MCU_PART_SERIES

  初期値 :(0x7)

  シリーズ

```
Character(s) = Value for macro = Description
1       = 0x1       = Ultra-low Power (up to 32MHz)
3       = 0x3       = High-efficiency (33-100MHz)
5       = 0x5       = High-integration (101MHz-200MHz)
7       = 0x7       = High-performance (201MHz-300MHz)
```

スタックおよびヒープ サイズの設定

- #define BSP_CFG_STACK_MAIN_BYTES

  初期値 :(0x1000)

  バイト単位のメイン スタック サイズ。これは、リセット以降に使用される必須のスタックです。例外では常にメイン スタックが使用されます。

- #define BSP_CFG_STACK_PROCESS_BYTES

  初期値 :(0)

  バイト単位のプロセス スタック サイズ。プロセス スタックの使用は任意です。BSP はプロセス スタックを使用しません。ユーザーがアプリケーション内でプロセス スタックを使用する場合は、CONTROL レジスタの SPSEL ビットをセットする必要があります。

- #define BSP_CFG_HEAP_BYTES

  初期値 :(0x400)

  バイト単位のヒープ サイズ。

オプション設定メモリ （ROM レジスタ）の設定

特定のレジスタが ROM に保存され、リセット以降の MCU の設定に使用されます。

I : レジスタのデフォルト値を使用するには、マクロをすべて 0xFF のままにします。

I : すべてのオプションがデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_OFS0

  初期値 :(0xFFFFFFFF)

  WDT 設定と IWDT 設定を構成します。OFS0 - オプション関数選択レジスタ 0

- b31 予約済み （1 にセット）
- b30 WDTSTPCTL - WDT 停止制御 - （0= 集計を継続、1= スリープ モードに入ったら停止）
- b29 予約済み （1 にセット）
- b28 WDTRSTIRQS - WDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b27:b26 WDTRPSS - WDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b25:b24 WDTRPES - WDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b23:b20 WDTCKS - WDT クロック周波数分割比 - （1=/4、4=/64、0xF=/128、6=/512、7=/2048、8=/8192）
- b19:b18 WDTTOPS - WDT タイムアウト期間選択 - （0=1024 サイクル、1=4096、2=8192、3=16384）
- b17 WDTSTRT - WDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b16:b15 予約済み （1 にセット）
- b14 IWDTSTPCTL - IWDT Sleep Stop Control - （0=counting continues, 1=stop w/selected low power modes）
- b13 予約済み （1 にセット）
- b12 IWDTRSTIRQS - IWDT リセット割り込み要求 - アンダーフロー時の処理 （0= 割り込みの受け入れ、1=MCU のリセット）
- b11:b10 IWDTRPSS - IWDT ウィンドウ開始位置選択 - （0=25%、1=50%、2=75%、3=100% （使用禁止））
- b9:b8 IWDTRPES - IWDT ウィンドウ終了位置選択 - （0=75%、1=50%、2=25%、3=0% （使用禁止））
- b7:b4 IWDTCKS - IWDT クロック周波数分割比 - （0= なし、2=/16、3 = /32、4=/64、0xF=/128、5=/256）
- b3:b2 IWDTTOPS - IWDT タイムアウト期間選択 - （0=128 サイクル、1=512、2=1024、3=2048）
- b1 IWDTSTRT - IWDT 開始モード選択 - （0= リセット後自動起動、1= リセット後停止）
- b0 予約済み （1 にセット）

l :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

- #define BSP_CFG_ROM_REG_OFS1

  初期値 :(0xFFFFFFFF)

  電圧検出 0 回路と HOCO をリセット後に有効にするかどうかを設定します。OFS1 - オプション関数選択レジスタ 1

- b31:b9 予約済み （1 にセット）
- b10:b9 HOCOFRQ0 - HOCO 周波数設定 （bsp_clock_cfg.h 内の BSP_CFG_HOCO_FREQUENCY を使用してセット）
- b8 HOCOEN - リセット後の HOCO 振動の有効化 / 無効化 （0= 有効、1= 無効）
- b7:b3 予約済み （1 にセット）
- b2 LVDAS - リセット後の電圧検出 0 回路の有効化 / 無効化の選択 （0= 有効、1= 無効）
- b1:b0 VDSEL - 電圧検出 0 レベル選択 （1=2.94V、2=2.87V、3=2.80V）

  I :0xFFFFFFFF の値はデフォルトで、すべての機能が無効になります。

セキュリティ MPU オプション

以下のセキュリティ MPU 領域のそれぞれに 3 つずつのマクロがあります。

- オプションを有効 / 無効にします。1 = 無効、0 = 有効を使用します。
- 開始アドレス - この MPU 領域の開始アドレス。
- 終了アドレス - この MPU 領域の終了アドレス。

I : 領域によって、セット可能なアドレスに対する制限が異なります。

I : すべての領域がデフォルトで無効になっています。

- #define BSP_CFG_ROM_REG_MPU_PC0_ENABLE

  初期値 :(1)

  PC 領域 0 を有効または無効にします。

- 0 = 有効

- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_PC0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_PC1_ENABLE

  初期値 :(1)

  PC 領域 1 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_PC1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_PC1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION0_ENABLE

  初期値 :(1)

  メモリ領域 0 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION0_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0x00FFFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION0_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0x00FFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION1_ENABLE

  初期値 :(1)

  メモリ領域 1 を有効または無効にします。

- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION1_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000000 ～ 0xFFFFFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION1_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x00000003 ～ 0xFFFFFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION2_ENABLE

  初期値 :(1)

  メモリ領域 2 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION2_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION2_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF
- #define BSP_CFG_ROM_REG_MPU_REGION3_ENABLE

  初期値 :(1)

  メモリ領域 3 を有効または無効にします。
- 0 = 有効
- 1 = 無効
- #define BSP_CFG_ROM_REG_MPU_REGION3_START

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000000 ～ 0x407FFFFC
- #define BSP_CFG_ROM_REG_MPU_REGION3_END

  初期値 :(0xFFFFFFFF)

  有効範囲 : 0x40000003 ～ 0x407FFFFF

ID コード保護

必要な ID コードをセットします。この値をセットして MCU にプログラムしたら、その ID コードを覚えておく必要があります。これは、接続時にデバッガーから要求されるためです。ID コードは 16 バイト長です。以下のマクロは ID コードを 4 バイト セクションで定義します。

I :ID コードを使用しない（保護が無効になっている）場合は、デフォルト （すべて 0xFF）のままにします。

I a: 有効な ID コードの設定方法については、ユーザーズマニュアルを参照してください。

- #define BSP_CFG_ID_CODE_LONG_1

  初期値 :(0xFFFFFFFF)

  最下位 4 バイト セクション、アドレス 0x40120050。MSB から LSB へ : ID コード 16、ID コード 15、ID コード 14、ID コード 13。

- #define BSP_CFG_ID_CODE_LONG_2

  初期値 :(0xFFFFFFFF)

  第 2 ID コード セクション、アドレス 0x40120054。MSB から LSB へ : ID コード 12、ID コード 11、ID コード 10、ID コード 9。

- #define BSP_CFG_ID_CODE_LONG_3

  初期値 :(0xFFFFFFFF)

  第 3 ID コード セクション、アドレス 0x40120058。MSB から LSB へ : ID コード 8、ID コード 7、ID コード 6、ID コード 5。

- #define BSP_CFG_ID_CODE_LONG_4

  初期値 :(0xFFFFFFFF)

  第 4 ID コード セクション、アドレス 0x4012005C。MSB から LSB へ : ID コード 4、ID コード 3、ID コード 2、ID コード 1。

その他のハードウェア オプション

- #define BSP_CFG_MCU_VCC_MV

  初期値 :(3300)

  このマクロは、MCU に供給される電圧 （Vcc）を定義するために使用されます。また、このマクロはミリボルト単位で定義されます。実際に MCU 上の何かを変更するわけではありません。選択したモジュールがこの情報を必要とするため、ここで定義します。

プロジェクト全体のソフトウェア オプション

- #define BSP_CFG_PARAM_CHECKING_ENABLE

  初期値 :(1)

  デフォルトで、SSP モジュールは入力パラメータが有効かどうかをチェックします。これは開発において役立ちますが、状況に応じて実稼働コードに対し無効化することもできます。その目的は、実行時間とコード スペースを節約することです。このマクロは、パラメータ チェックを有効または無効にするためのグローバル設定です。各モジュールは、同じ目的の独自のローカル マクロも備えています。デフォルトで、ローカル マクロはここからグローバル値を取得しますが、オーバーライドすることができます。そのため、ローカル設定の方がこのグローバル設定より優先されます。パラメータ チェックの無効化は、入力が適切であることが分かっており、速度の向上またはコード スペースの削減が必要な場合にのみ使用してください。

- 0 = パラメータ チェックのグローバル設定が無効になっています。
- 1 = パラメータ チェックのグローバル設定が有効になっています （デフォルト）。
- #define BSP_CFG_RTOS

  初期値 :(1)

  使用する RTOS を指定します。

- 0 = RTOS なし
- 0 = RTOS なし
- 2 = その他
- #define BSP_CFG_ASSERT

  初期値 :(0)

  SSP_ASSERT が失敗した場合の処理を指定します。

- 0 = SSP_ERR_ASSERTION を返します。
- 1 = ssp_error_log を呼び出してから、SSP_ERR_ASSERTION を返します。ssp_error_log は脆弱な関数であり、ssp/src/bsp/mcu/all/bsp_common.h 内のプロトタイプに基づいてユーザー コード内でオーバーライドする必要があることに注意してください。
- 2 = 標準のアサート ライブラリを使用して実行を停止します。
- #define BSP_CFG_ERROR_LOG

  初期値 :(0)

  SSP 関数からエラー コードが返された場合の処理を指定します。

- 0 = エラー コードを返します。
- 1 = ssp_error_log を呼び出してから、エラー コードを返します。ssp_error_log は脆弱な関数であり、ssp/src/bsp/mcu/all/bsp_common.h 内のプロトタイプに基づいてユーザー コード内でオーバーライドする必要があることに注意してください。

9.2.7.3 ビルド時間設定 - クロック

このファイルには、ビルド時間クロック設定オプションが保存されます。main() が実行される前に、BSP はこれらのマクロを使用して、ユーザーの MCU のクロックを設定します。

定義

- #define BSP_CFG_CLOCK_SOURCE

  初期値 :(CGC_CLOCK_PLL)

  クロック ソース選択 （CKSEL）。選択されたクロックがシステム クロックとすべてのペリフェラル クロックに提供されるベース クロックになります。また、フラッシュ クロックと外部バス クロックにも使用されます。

  次のように、列挙のいずれかにマクロをセットします。

```
Clock                           -  Enumeration to use for macro
High Speed On-Chip Oscillator  (HOCO)     -  CGC_CLOCK_HOCO
Middle Speed On-Chip Oscillator (MOCO)     -  CGC_CLOCK_MOCO
Low Speed On-Chip Oscillator   (LOCO)     -  CGC_CLOCK_LOCO
Main Clock Oscillator                -  CGC_CLOCK_MAIN_OSC
Sub-Clock Oscillator                 -  CGC_CLOCK_SUBCLOCK
PLL Circuit                      -  CGC_CLOCK_PLL
```

- #define BSP_CFG_XTAL_HZ

  初期値 :(24000000)

  XTAL - Hz 単位の入力クロック周波数

- #define BSP_CFG_HOCO_FREQUENCY

  初期値 :(0)

  HOCO は複数の周波数で動作できます。下のマクロを使用していずれか 1 つを選択します。リセット以降に使用される周波数は OFS1.HOCOFRQ0 ビットによって異なります。

```
Available frequency settings:
0 = 16MHz
1 = 18MHz
2 = 20MHz
```

- #define BSP_CFG_PLL_SOURCE

  初期値 :(CGC_CLOCK_MAIN_OSC)

  PLL クロック ソース （PLLSRCEL）。PLL 回路に入力するクロック ソースを選択します。

```
Available clock sources:    -  Enumeration to use for macro
0 = Main clock (default)    -  CGC_CLOCK_MAIN_OSC
1 = HOCO                -  CGC_CLOCK_HOCO
```

- #define BSP_CFG_PLL_DIV

  初期値 :(CGC_PLL_DIV_2)

  PLL 入力周波数分割比選択（PLIDIV）

  使用可能な除数 = - /1 (除算なし)、- /2、- /3

  l : マクロ定義を 'CGC_PLL_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PLL_MUL

  初期値 :(20.0)

  PLL 周波数増倍係数選択 （PLLMUL）

  使用可能な乗算器 = x10.0 ～ x30.0 の 0.5 ずつのインクリメント （10.0、10.5、11.0、11.5 ... 29.0、29.5、30.0 など）

- #define BSP_CFG_ICK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_1)

  システム クロック除算器 （ICK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  l : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKA_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック A 除算器 （PCKA）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  l : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKB_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  ペリフェラル モジュール クロック B 除算器 （PCKB）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKC_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  ペリフェラル モジュール クロック C 除算器 （PCKC）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_PCKD_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  ペリフェラル モジュール クロック D 除算器 （PCKD）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_2)

  外部バス クロック除算器 （BCLK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_FCK_DIV

  初期値 :(CGC_SYS_CLOCK_DIV_4)

  フラッシュ IF クロック除算器 （FCK）

  使用可能な除数 = /1 (除算なし)、/2、/4、/8、/16、/32、/64

  I : マクロ定義を 'CGC_SYS_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_UCK_DIV

  初期値 :(CGC_USB_CLOCK_DIV_5)

  USB クロック除算器選択。

  使用可能な除数 = /3、/4、/5

  I : マクロ定義を 'CGC_USB_CLOCK_DIV_' + 除算器選択にセットします。

- #define BSP_CFG_BCLK_OUTPUT

  初期値 :(1)

  BCLK 出力ピンの設定

  使用可能なオプション :

- 0 = 出力なし
- 1 = BCK 周波数
- 2 = BCK/2 周波数

  I : このマクロは、外部バスが有効になっている場合にのみ効果があります。

- #define BSP_CFG_SDCLK_OUTPUT

  初期値 :(1)

  SDCLK 出力ピンの設定

  使用可能なオプション :

- 0 = 出力なし
- 1 = BCK 周波数

  I : このマクロは、外部バスが有効になっている場合にのみ効果があります。

9.2.7.4 ビルド時間設定 - 割り込み

割り込みをトリガーするように設定可能な ELC イベントを以下に列挙します。ユーザーが特定のイベントで割り込みをトリガーしたい場合は、マクロの定義を BSP IRQ DISABLED から割り込みに使用する割り込み優先レベルに変更する必要があります。この MCU の有効な優先順位の範囲は 0 ～ 15 で、小さい数値ほど優先順位が高くなります （0 = 最高優先順位、15 = 最低優先順位）。BSP は、ベクター テーブルへの適切な関数の配置を処理して、割り込み優先レベルをセットし、リセットからユーザーのアプリケーションが呼び出されるまでの間に割り込みを生成するように ELC イベントを設定します。

定義

- #define BSP_IRQ_CFG_ICU_IRQ0

  初期値 :(`#define BSP_IRQ_DISABLED`)

  PORT0 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ1

  初期値 :(`#define BSP_IRQ_DISABLED`)

  PORT1 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ2

  初期値 :(`#define BSP_IRQ_DISABLED`)

  PORT2 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ3

  初期値 :(`#define BSP_IRQ_DISABLED`)

  PORT3 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ4

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT4 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ5

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT5 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ6

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT6 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ7

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT7 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ8

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT8 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ9

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT9 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ10

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT10 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ11

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT11 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ12

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT12 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ13

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT13 IRQ。

- #define BSP_IRQ_CFG_ICU_IRQ14

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT14 IRQ。
- #define BSP_IRQ_CFG_ICU_IRQ15

  初期値 :(#define BSP_IRQ_DISABLED)

  PORT15 IRQ。
- #define BSP_IRQ_CFG_DMAC0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC0 DMAC。
- #define BSP_IRQ_CFG_DMAC1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC1 DMAC。
- #define BSP_IRQ_CFG_DMAC2_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC2 DMAC。
- #define BSP_IRQ_CFG_DMAC3_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC3 DMAC。
- #define BSP_IRQ_CFG_DMAC4_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC4 DMAC。
- #define BSP_IRQ_CFG_DMAC5_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC5 DMAC。
- #define BSP_IRQ_CFG_DMAC6_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC6 DMAC。
- #define BSP_IRQ_CFG_DMAC7_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DMAC7 DMAC。

- #define BSP_IRQ_CFG_DTC_COMPLETE

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC COMPLETE。
- #define BSP_IRQ_CFG_DTC_END

  初期値 :(#define BSP_IRQ_DISABLED)

  DTC DTC END。
- #define BSP_IRQ_CFG_ICU_SNOOZE_CANCEL

  初期値 :(#define BSP_IRQ_DISABLED)

  ICU CANCELING SNOOZE MODE。
- #define BSP_IRQ_CFG_FCU_FIFERR

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FIFERR。
- #define BSP_IRQ_CFG_FCU_FRDYI

  初期値 :(#define BSP_IRQ_DISABLED)

  FCU FRDYI。
- #define BSP_IRQ_CFG_LVD_LVD1

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD1 LVD1。
- #define BSP_IRQ_CFG_LVD_LVD2

  初期値 :(#define BSP_IRQ_DISABLED)

  LVD2 LVD2。
- #define BSP_IRQ_CFG_VBATT_LVD

  初期値 :(#define BSP_IRQ_DISABLED)

  VBATT VBAT。
- #define BSP_IRQ_CFG_CGC_MOSC_STOP

  初期値 :(#define BSP_IRQ_DISABLED)

  MOSC OSC STOP。
- #define BSP_IRQ_CFG_LPM_SNOOZE_REQUEST

  初期値 :(#define BSP_IRQ_DISABLED)

  CPUSYS SNOOZE MODE ENTRY FLAG。

- #define BSP_IRQ_CFG_AGT0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMAI。
- #define BSP_IRQ_CFG_AGT0_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT0 AGTCMBI。
- #define BSP_IRQ_CFG_AGT1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMAI。
- #define BSP_IRQ_CFG_AGT1_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  AGT1 AGTCMBI。
- #define BSP_IRQ_CFG_IWDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  IWDT NMIUNDF N。
- #define BSP_IRQ_CFG_WDT_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CWDT NMIUNDF N。
- #define BSP_IRQ_CFG_RTC_ALARM

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC ALM。
- #define BSP_IRQ_CFG_RTC_PERIOD

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC PRD。

- #define BSP_IRQ_CFG_RTC_CARRY

  初期値 :(#define BSP_IRQ_DISABLED)

  RTC CUP。
- #define BSP_IRQ_CFG_ADC0_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 ADI。
- #define BSP_IRQ_CFG_ADC0_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 GBADI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPAI。
- #define BSP_IRQ_CFG_ADC0_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 CMPBI。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC0_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD0 COMPARE MISMATCH。
- #define BSP_IRQ_CFG_ADC1_SCAN_END

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 ADI。
- #define BSP_IRQ_CFG_ADC1_SCAN_END_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 GBADI。
- #define BSP_IRQ_CFG_ADC1_WINDOW_A

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 CMPAI。

- #define BSP_IRQ_CFG_ADC1_WINDOW_B

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 CMPBI。
- #define BSP_IRQ_CFG_ADC1_COMPARE_MATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 COMPARE MATCH。
- #define BSP_IRQ_CFG_ADC1_COMPARE_MISMATCH

  初期値 :(#define BSP_IRQ_DISABLED)

  S12AD1 COMPARE MISMATCH。
- #define BSP_IRQ_CFG_COMP_HS_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP OC0 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_1

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD1 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_2

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD2 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_3

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD3 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_4

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD4 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_HS_5

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP RD5 COMP IRQ。
- #define BSP_IRQ_CFG_COMP_LP_0

  初期値 :(#define BSP_IRQ_DISABLED)

  COMP LP COMP C0IRQ。

- #define BSP_IRQ_CFG_COMP_LP_1
  初期値 :(#define BSP_IRQ_DISABLED)
  COMP LP COMP C1IRQ.。
- #define BSP_IRQ_CFG_USBFS_FIFO_0
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS D0FIFO。
- #define BSP_IRQ_CFG_USBFS_FIFO_1
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS D1FIFO.。
- #define BSP_IRQ_CFG_USBFS_INT
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS USBI。
- #define BSP_IRQ_CFG_USBFS_RESUME
  初期値 :(#define BSP_IRQ_DISABLED)
  USBFS USBR。
- #define BSP_IRQ_CFG_IIC0_RXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 RXI。
- #define BSP_IRQ_CFG_IIC0_TXI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 TXI。
- #define BSP_IRQ_CFG_IIC0_TEI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 TEI。
- #define BSP_IRQ_CFG_IIC0_ERI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 EEI。
- #define BSP_IRQ_CFG_IIC0_WUI
  初期値 :(#define BSP_IRQ_DISABLED)
  RIIC0 WUI。

- #define BSP_IRQ_CFG_IIC1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 RXI。
- #define BSP_IRQ_CFG_IIC1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TXI。
- #define BSP_IRQ_CFG_IIC1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 TEI。
- #define BSP_IRQ_CFG_IIC1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC1 EEI。
- #define BSP_IRQ_CFG_IIC2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 RXI。
- #define BSP_IRQ_CFG_IIC2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 TXI。
- #define BSP_IRQ_CFG_IIC2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 TEI。
- #define BSP_IRQ_CFG_IIC2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RIIC2 EEI。
- #define BSP_IRQ_CFG_SSI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSITXI。
- #define BSP_IRQ_CFG_SSI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIRXI。

- #define BSP_IRQ_CFG_SSI0_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI0 SSIF。
- #define BSP_IRQ_CFG_SSI1_TXI_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIRT。
- #define BSP_IRQ_CFG_SSI1_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  SSI1 SSIF。
- #define BSP_IRQ_CFG_SRC_INPUT_FIFO_EMPTY

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC IDEI。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_FULL

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC ODFI。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC OVF。
- #define BSP_IRQ_CFG_SRC_OUTPUT_FIFO_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC UDF。
- #define BSP_IRQ_CFG_SRC_CONVERSION_END

  初期値 :(#define BSP_IRQ_DISABLED)

  SRC CEF。
- #define BSP_IRQ_CFG_PDC_RECEIVE_DATA_READY

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCDFI。
- #define BSP_IRQ_CFG_PDC_FRAME_END

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCFEI。

- #define BSP_IRQ_CFG_PDC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  PDC PCERI。
- #define BSP_IRQ_CFG_CTSU_WRITE

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUWR。
- #define BSP_IRQ_CFG_CTSU_READ

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSURD。
- #define BSP_IRQ_CFG_CTSU_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CTSU CTSUFN。
- #define BSP_IRQ_CFG_KEY_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  KEY INTKR。
- #define BSP_IRQ_CFG_DOC_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  DOC DOPCF。
- #define BSP_IRQ_CFG_CAC_FREQUENCY_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC FERRF。
- #define BSP_IRQ_CFG_CAC_MEASUREMENT_END

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC MENDF。
- #define BSP_IRQ_CFG_CAC_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  CAC OVFF。
- #define BSP_IRQ_CFG_CAN0_ERROR

  初期値 :(#define BSP_IRQ_DISABLED)

  RCAN20 ERS。

- #define BSP_IRQ_CFG_CAN0_FIFO_RX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN20 RXF。
- #define BSP_IRQ_CFG_CAN0_FIFO_TX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN20 TXF。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_RX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN20 RXM。
- #define BSP_IRQ_CFG_CAN0_MAILBOX_TX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN20 TXM。
- #define BSP_IRQ_CFG_CAN1_ERROR
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN21 ERS。
- #define BSP_IRQ_CFG_CAN1_FIFO_RX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN21 RXF。
- #define BSP_IRQ_CFG_CAN1_FIFO_TX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN21 TXF。
- #define BSP_IRQ_CFG_CAN1_MAILBOX_RX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN21 RXM。
- #define BSP_IRQ_CFG_CAN1_MAILBOX_TX
  初期値 :(#define BSP_IRQ_DISABLED)
  RCAN21 TXM。
- #define BSP_IRQ_CFG_IOPORT_EVENT_1
  初期値 :(#define BSP_IRQ_DISABLED)
  GPIO PORT GROUP A。

- #define BSP_IRQ_CFG_IOPORT_EVENT_2

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP B。
- #define BSP_IRQ_CFG_IOPORT_EVENT_3

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP C。
- #define BSP_IRQ_CFG_IOPORT_EVENT_4

  初期値 :(#define BSP_IRQ_DISABLED)

  GPIO PORT GROUP D。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_0

  初期値 :(#define BSP_IRQ_DISABLED)

  ELC0 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_ELC_SOFTWARE_EVENT_1

  初期値 :(#define BSP_IRQ_DISABLED)

  ELC1 SOFTWARE EVENT。
- #define BSP_IRQ_CFG_POEG0_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT0。
- #define BSP_IRQ_CFG_POEG1_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT1。
- #define BSP_IRQ_CFG_POEG2_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT2。
- #define BSP_IRQ_CFG_POEG3_EVENT

  初期値 :(#define BSP_IRQ_DISABLED)

  POEG GROUP EVENT3。
- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT0_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT0_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT0_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT0_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT0_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT0_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT0_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT0_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 AD TRIG A。
- #define BSP_IRQ_CFG_GPT0_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT0 AD TRIG B。
- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT1_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT1_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT1_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT1_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT1_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT1_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT1_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT1_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 AD TRIG A。
- #define BSP_IRQ_CFG_GPT1_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT1 AD TRIG B。
- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT2_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT2_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT2_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT2_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT2_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT2_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT2_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT2_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 AD TRIG A。
- #define BSP_IRQ_CFG_GPT2_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT2 AD TRIG B。
- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT3_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT3_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT3_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT3_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT3_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT3_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT3_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT3_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 AD TRIG A。
- #define BSP_IRQ_CFG_GPT3_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT3 AD TRIG B。
- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT4_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT4_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT4_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT4_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT4_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT4_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT4_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT4_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 AD TRIG A。
- #define BSP_IRQ_CFG_GPT4_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT4 AD TRIG B。
- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT5_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT5_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT5_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT5_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT5_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT5_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT5_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT5_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 AD TRIG A。
- #define BSP_IRQ_CFG_GPT5_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT5 AD TRIG B。
- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT6_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT6_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT6_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT6_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT6_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT6_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT6_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT6_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 AD TRIG A。
- #define BSP_IRQ_CFG_GPT6_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT6 AD TRIG B。
- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT7_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT7_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT7_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT7_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT7_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT7_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT7_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT7_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 AD TRIG A。
- #define BSP_IRQ_CFG_GPT7_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT7 AD TRIG B。
- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT8_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT8_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT8_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT8_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT8_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT8_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT8_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT8_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 AD TRIG A。
- #define BSP_IRQ_CFG_GPT8_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT8 AD TRIG B。
- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT9_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT9_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT9_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT9_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT9_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT9_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT9_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT9_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 AD TRIG A。
- #define BSP_IRQ_CFG_GPT9_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT9 AD TRIG B。
- #define BSP_IRQ_CFG_GPT10_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT10_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT10_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT10_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT10_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT10_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT10_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT10_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT10_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 AD TRIG A。
- #define BSP_IRQ_CFG_GPT10_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT10 AD TRIG B。
- #define BSP_IRQ_CFG_GPT11_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT11_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT11_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT11_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT11_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT11_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT11_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT11_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT11_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 AD TRIG A。
- #define BSP_IRQ_CFG_GPT11_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT11 AD TRIG B。
- #define BSP_IRQ_CFG_GPT12_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT12_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT12_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT12_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT12_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT12_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT12_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT12_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT12_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 AD TRIG A。
- #define BSP_IRQ_CFG_GPT12_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT12 AD TRIG B。
- #define BSP_IRQ_CFG_GPT13_CAPTURE_COMPARE_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 CAPTURE COMPARE INT A。

- #define BSP_IRQ_CFG_GPT13_CAPTURE_COMPARE_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 CAPTURE COMPARE INT B。
- #define BSP_IRQ_CFG_GPT13_COMPARE_C

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT C。
- #define BSP_IRQ_CFG_GPT13_COMPARE_D

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT D。
- #define BSP_IRQ_CFG_GPT13_COMPARE_E

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT E。
- #define BSP_IRQ_CFG_GPT13_COMPARE_F

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COMPARE INT F。
- #define BSP_IRQ_CFG_GPT13_COUNTER_OVERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COUNTER OVERFLOW。
- #define BSP_IRQ_CFG_GPT13_COUNTER_UNDERFLOW

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 COUNTER UNDERFLOW。
- #define BSP_IRQ_CFG_GPT13_AD_TRIG_A

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 AD TRIG A。
- #define BSP_IRQ_CFG_GPT13_AD_TRIG_B

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT13 AD TRIG B。
- #define BSP_IRQ_CFG_OPS_UVW_EDGE

  初期値 :(#define BSP_IRQ_DISABLED)

  GPT UVW EDGE。

- #define BSP_IRQ_CFG_EPTPC_IPLS

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER IPLS。
- #define BSP_IRQ_CFG_EPTPC_MINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER MINT。
- #define BSP_IRQ_CFG_EPTPC_PINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER PINT。
- #define BSP_IRQ_CFG_EDMAC0_EINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER EINT0。
- #define BSP_IRQ_CFG_EDMAC1_EINT

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER EINT1。
- #define BSP_IRQ_CFG_EPTPC_TIMER0_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER0 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER1_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER1 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER2_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER2 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER3_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER3 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER4_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER4 RISE。

- #define BSP_IRQ_CFG_EPTPC_TIMER5_RISE

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER5 RISE。
- #define BSP_IRQ_CFG_EPTPC_TIMER0_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER0 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER1_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER1 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER2_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER2 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER3_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER3 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER4_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER4 FALL。
- #define BSP_IRQ_CFG_EPTPC_TIMER5_FALL

  初期値 :(#define BSP_IRQ_DISABLED)

  ETHER ETHER5 FALL。
- #define BSP_IRQ_CFG_USBHS_FIFO_0

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS D0FIFO。
- #define BSP_IRQ_CFG_USBHS_FIFO_1

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS D1FIFO.。
- #define BSP_IRQ_CFG_USBHS_USB_INT_RESUME

  初期値 :(#define BSP_IRQ_DISABLED)

  USBHS USBIR。

- #define BSP_IRQ_CFG_SCI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI。
- #define BSP_IRQ_CFG_SCI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TXI。
- #define BSP_IRQ_CFG_SCI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 TEI。
- #define BSP_IRQ_CFG_SCI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 ERI。
- #define BSP_IRQ_CFG_SCI0_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 AM。
- #define BSP_IRQ_CFG_SCI0_RXI_OR_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI0 RXI OR ERI。
- #define BSP_IRQ_CFG_SCI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 RXI。
- #define BSP_IRQ_CFG_SCI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TXI。
- #define BSP_IRQ_CFG_SCI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 TEI。
- #define BSP_IRQ_CFG_SCI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 ERI。

- #define BSP_IRQ_CFG_SCI1_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI1 AM。
- #define BSP_IRQ_CFG_SCI2_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 RXI。
- #define BSP_IRQ_CFG_SCI2_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TXI。
- #define BSP_IRQ_CFG_SCI2_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 TEI。
- #define BSP_IRQ_CFG_SCI2_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 ERI。
- #define BSP_IRQ_CFG_SCI2_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI2 AM。
- #define BSP_IRQ_CFG_SCI3_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 RXI。
- #define BSP_IRQ_CFG_SCI3_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TXI。
- #define BSP_IRQ_CFG_SCI3_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 TEI。
- #define BSP_IRQ_CFG_SCI3_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 ERI。

- #define BSP_IRQ_CFG_SCI3_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI3 AM。
- #define BSP_IRQ_CFG_SCI4_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 RXI。
- #define BSP_IRQ_CFG_SCI4_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TXI。
- #define BSP_IRQ_CFG_SCI4_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 TEI。
- #define BSP_IRQ_CFG_SCI4_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 ERI。
- #define BSP_IRQ_CFG_SCI4_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI4 AM。
- #define BSP_IRQ_CFG_SCI5_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 RXI。
- #define BSP_IRQ_CFG_SCI5_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 TXI。
- #define BSP_IRQ_CFG_SCI5_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 TEI.
- #define BSP_IRQ_CFG_SCI5_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 ERI。

- #define BSP_IRQ_CFG_SCI5_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI5 AM。
- #define BSP_IRQ_CFG_SCI6_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 RXI。
- #define BSP_IRQ_CFG_SCI6_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 TXI。
- #define BSP_IRQ_CFG_SCI6_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 TEI.
- #define BSP_IRQ_CFG_SCI6_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 ERI。
- #define BSP_IRQ_CFG_SCI6_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI6 AM。
- #define BSP_IRQ_CFG_SCI7_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 RXI。
- #define BSP_IRQ_CFG_SCI7_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 TXI。
- #define BSP_IRQ_CFG_SCI7_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 TEI.
- #define BSP_IRQ_CFG_SCI7_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 ERI。

- #define BSP_IRQ_CFG_SCI7_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI7 AM。
- #define BSP_IRQ_CFG_SCI8_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 RXI。
- #define BSP_IRQ_CFG_SCI8_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 TXI。
- #define BSP_IRQ_CFG_SCI8_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 TEI.
- #define BSP_IRQ_CFG_SCI8_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 ERI。
- #define BSP_IRQ_CFG_SCI8_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI8 AM。
- #define BSP_IRQ_CFG_SCI9_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 RXI。
- #define BSP_IRQ_CFG_SCI9_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TXI。
- #define BSP_IRQ_CFG_SCI9_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 TEI。
- #define BSP_IRQ_CFG_SCI9_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 ERI。

- #define BSP_IRQ_CFG_SCI9_AM

  初期値 :(#define BSP_IRQ_DISABLED)

  SCI9 AM。
- #define BSP_IRQ_CFG_SPI0_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPRI。
- #define BSP_IRQ_CFG_SPI0_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPTI。
- #define BSP_IRQ_CFG_SPI0_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPII。
- #define BSP_IRQ_CFG_SPI0_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SPEI。
- #define BSP_IRQ_CFG_SPI0_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI0 SP ELCTEND。
- #define BSP_IRQ_CFG_SPI1_RXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPRI。
- #define BSP_IRQ_CFG_SPI1_TXI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPTI。
- #define BSP_IRQ_CFG_SPI1_IDLE

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPII。
- #define BSP_IRQ_CFG_SPI1_ERI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SPEI。

- #define BSP_IRQ_CFG_SPI1_TEI

  初期値 :(#define BSP_IRQ_DISABLED)

  RSPI1 SP ELCTEND。
- #define BSP_IRQ_CFG_QSPI_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  QSPI INTR。
- #define BSP_IRQ_CFG_SDHIMMC0_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC0_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC0_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 CARD。
- #define BSP_IRQ_CFG_SDHIMMC0_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC0 ODMSDBREQ。
- #define BSP_IRQ_CFG_SDHIMMC1_ACCS

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ACCS。
- #define BSP_IRQ_CFG_SDHIMMC1_SDIO

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 SDIO。
- #define BSP_IRQ_CFG_SDHIMMC1_CARD

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 CARD。
- #define BSP_IRQ_CFG_SDHIMMC1_DMA_REQ

  初期値 :(#define BSP_IRQ_DISABLED)

  SDHI MMC1 ODMSDBREQ。

- #define BSP_IRQ_CFG_DIVIDER_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  EXT DIVIDER INTMD。
- #define BSP_IRQ_CFG_SCE_PROC_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP PROC BUSY N。
- #define BSP_IRQ_CFG_SCE_ROMOK

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP ROMOK N。
- #define BSP_IRQ_CFG_SCE_LONG_PLG

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP LONG PLG N。
- #define BSP_IRQ_CFG_SCE_TEST_BUSY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP TEST BUSY N。
- #define BSP_IRQ_CFG_SCE_WRRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 0 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 1 N。
- #define BSP_IRQ_CFG_SCE_WRRDY_4

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP WRRDY 4 N。
- #define BSP_IRQ_CFG_SCE_RDRDY_0

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 0 N。
- #define BSP_IRQ_CFG_SCE_RDRDY_1

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP RDRDY 1 N。

- #define BSP_IRQ_CFG_SCE_INTEGRATE_WRRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE WRRDY N。
- #define BSP_IRQ_CFG_SCE_INTEGRATE_RDRDY

  初期値 :(#define BSP_IRQ_DISABLED)

  TSIP INTEGRATE RDRDY N。
- #define BSP_IRQ_CFG_GLCDC_LINE_DETECT

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 0。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_1

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 1。
- #define BSP_IRQ_CFG_GLCDC_UNDERFLOW_2

  初期値 :(#define BSP_IRQ_DISABLED)

  LCDC LCDC LEVEL 2。
- #define BSP_IRQ_CFG_DRW_INT

  初期値 :(#define BSP_IRQ_DISABLED)

  TWOD ENGINE IRQ。
- #define BSP_IRQ_CFG_JPEG_JEDI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JEDI。
- #define BSP_IRQ_CFG_JPEG_JDTI

  初期値 :(#define BSP_IRQ_DISABLED)

  JPEG JDTI。

9.2.7.5 ビルド時間設定 - ピン設定

このファイルには、ピン設定の配列が保存されます。起動時に main() が実行される前に、BSP はこの配列を反復処理し、配列内の設定に基づいて MCU のポート ピンを初期化します。

I：このファイルは ISDE Pin Configurator を使用して作成することをお勧めします。

9.2.8 カスタム ユーザー S7G2 ボードの BSP

カスタム ユーザー S7G2 ボードのテンプレート BSP。

9.2.8.1 ビルドタイム構成

BSP には、ビルド時間設定オプションを含む複数のヘッダー ファイルが付属しています。ヘッダー ファイルで次の設定を構成します。

- 全般 BSP オプション
- [Clocks]
- このモジュールが動作するためには、割り込みが有効になっている
- ピン設定

9.3 サポートされている MCU

このバージョンの BSP でサポートされている MCU。

BSP には MCU とボードに固有のコードが含まれています。MCU に固有のコードはその MCU を使用するボード間で共有することができます。

9.3.1 Functions

- R_SSP_VersionGet

9.3.2 R_SSP_VersionGet

`ssp_err_t` R_SSP_VersionGet (`ssp_pack_version_t` *const p_version)

9.3.2.1 概要説明

コンパイル時マクロに基づいて SSP バージョンをセットします。

9.3.2.2 詳細説明

表 1012: パラメータ

名前	方向	説明
p_version	out	バージョン情報を返す先のメモリ アドレス。

表 1013: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報が保存されました。

9.3.3 S124

S124 に共通するコード。

S124 に共通する関数を実装します。

9.3.3.1 定義

- #define BSP_PACKAGE_BGA

  初期値 :
- #define BSP_PACKAGE_PINS

  初期値 :(121)
- #define BSP_LOCO_HZ

  初期値 :(32768)

  低速オンチップ オシレーター。
- #define BSP_MOCO_HZ

  初期値 :(8000000)

  中速オンチップ オシレーター。
- #define BSP_SUB_CLOCK_HZ

  初期値 :(32768)

  サブ クロック オシレーター。
- #define BSP_HOCO_HZ

  初期値 :(24000000)

  高速オンチップ オシレーター。
- #define BSP_VECTOR_TABLE_ENTRIES

  初期値 :(48)

9.3.3.2 elc_peripheral_t

イベント信号にリンクされる可能性のある周辺機能

9.3.3.3 elc_event_t

他の周辺機器または CPU1 にリンクされるイベント信号のソース。
I：このリストは、デバイスに応じて変更される可能性があります。これは、S124 に対応したリストです。

9.3.3.4 クロック初期化

このファイル内の関数は、bsp_clock_cfg.h 内のマクロに基づいてシステム クロックを設定します。

9.3.3.5 グループ化割り込みのサポート

グループ化割り込みのサポート。グループ化割り込みは、複数の割り込みイベントが同じイベント ベクターをトリガーしたときに発生します。この共通ベクターがトリガーされた場合は、アクティベーション ソースを発見する必要があります。このファイル内の関数を使用すれば、ユーザーは、割り込みグループ内のシングル割り込みソース用のコールバック関数を登録することができます。

グループ化割り込みのサポート。グループ化割り込みは、複数の割り込みイベントが同じイベント ベクターをトリガーしたときに発生します。この共通ベクターがトリガーされた場合は、アクティベーション ソースを発見する必要があります。このファイル内の関数を使用すれば、ユーザーは、割り込みグループ内のシングル割り込みソース用のコールバック関数を登録することができます。

Functions

- R_BSP_GroupIrqWrite

R_BSP_GroupIrqWrite

```
ssp_err_t  R_BSP_GroupIrqWrite (  bsp_grp_irq_t irq ,  void(*)(bsp_grp_irq_t irq) p_callback )
```

概要説明

サポートされている割り込みのコールバック関数を登録します。コールバック引数に NULL が渡された場合は、過去に登録されたコールバックが登録解除されます。

詳細説明

表 1014: パラメータ

名前	方向	説明
irq	複数のビットを書き換えることもできます。	コールバックを登録するための割り込み。
p_callback	複数のビットを書き換えることもできます。	割り込みの発生時に呼び出す関数へのポインタ。

表 1015: 戻り値

名前	説明
SSP_SUCCESS	コールバックが登録されました。

関数のステップ

- 有効なアドレスをチェックします。
- コールバックが NULL でした。コールバックを登録解除します。
- コールバックを登録します。

9.3.3.6 グループ化割り込みのサポート

このモジュールは、NVIC 割り込みをトリガーできるように特定の ELC イベントを設定します。割り込みとして使用されるイベントは bsp_irq_cfg.h 内の設定によって異なります。

Functions

- R_BSP_IrqStatusClear

R_BSP_IrqStatusClear

R_BSP_IrqStatusClear (IRQn_Type irq)

概要説明

特定の割り込みの割り込みステータス フラグ （IR）をクリアします。: 特定の割り込みの割り込みステータス フラグ （IR）をクリアします。割り込みがトリガーされると、IR ビットがセットされます。

詳細説明

表 1016: パラメータ

名前	方向	説明
irq	複数のビットを書き換えることもできます。	IR ビットをクリアするための割り込み。

I :IRQn_Type 値が <0 になるシステム例外では、これが機能しません。

9.3.3.7 アトミック ロック

このモジュールは、アトミック ロック メカニズムを実装します。

Functions

- R_BSP_SoftwareLock
- R_BSP_SoftwareUnlock
- R_BSP_HardwareLock
- R_BSP_HardwareUnlock
- bsp_init_hardware_locks
- R_BSP_SoftwareLockInit

データ構造体

- bsp_lock_t

R_BSP_SoftwareLock

ssp_err_t R_BSP_SoftwareLock (bsp_lock_t * p_lock)

概要説明

送信されたロックの取得を試みます。

詳細説明

表 1017: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	取得するロックを含む構造体へのポインタ。

表 1018: 戻り値

名前	説明
SSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

R_BSP_SoftwareUnlock

R_BSP_SoftwareUnlock (bsp_lock_t * p_lock)

概要説明

ホールドオン ロックを解除します。

詳細説明

表 1019: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	ロック解除するロックを含む構造体へのポインタ。

R_BSP_HardwareLock

ssp_err_t R_BSP_HardwareLock (bsp_hw_lock_t hw_resource)

概要説明

ハードウェア リソース ロックの予約を試みます。

詳細説明

表 1020: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロックするハードウェア リソースに対するロック配列内のインデックス。

表 1021: 戻り値

名前	説明
BSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

表 1022: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロックするハードウェア リソースに対するロック配列内のインデックス。

表 1023: 戻り値

名前	説明
SSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

R_BSP_HardwareUnlock

R_BSP_HardwareUnlock (bsp_hw_lock_t hw_resource)

概要説明

ホールドオン ロックを解除します。

詳細説明

表 1024: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロック解除するハードウェア リソースに対するロック配列内のインデックス。

bsp_init_hardware_locks

bsp_init_hardware_locks (void)

概要説明

すべてのハードウェア ロックを BSP_LOCK_UNLOCKED に初期化します。

R_BSP_SoftwareLockInit

R_BSP_SoftwareLockInit (bsp_lock_t * p_lock)

概要説明

ロック値をロック済みに初期化します。

詳細説明

表 1025: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	初期化するロックを含む構造体へのポインタ。

bsp_lock_t

bsp_lock_t

詳細説明

ロック構造体。ソフトウェア ロッキング関数に渡されます (R_BSP_SoftwareLock や R_BSP_SoftwareUnLock など)。

変数

- uint8_t lock

  サイズは 8 ビットである必要があるので、enum の代わりに uint8_t が使用されます。

9.3.3.8 レジスタ保護

一部のレジスタは書き込み保護されています。このモジュールは、これらのレジスタの保護を設定するための API を提供します。適切な動作を保証するために参照カウンタが使用されます。

Functions

- R_BSP_RegisterProtectEnable
- R_BSP_RegisterProtectDisable

R_BSP_RegisterProtectEnable

R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect)

概要説明

レジスタ保護を有効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ （PRCR）と MPC の書き込み保護レジスタ （PWPR）を使用することで可能になります。

詳細説明

表 1026: パラメータ

名前	方向	説明
regs_to_protect	複数のビットを書き換えることもできます。	書き込み保護が有効になっているレジスタ。

関数のステップ

- 保護カウンタをデクリメントします。
- PRCR レジスタを使用して保護を有効にします。
- PRCR レジスタに書き込む場合は、上位 8 ビットを正しいキーにする必要があります。下位ビットを 0 にセットして書き込みを無効にします。
- b15:b8 PRKEY - 0xA5 を上位バイトに書き込んで、下位バイトへの書き込みを有効にします。
  b7:b4 予約済み （0 にセット）
  b3 PRC3 - LVD に関連したレジスタ （LVD1CR1、LVD1SR、LVD2CR1、LVD2SR、LVCMPCR、LVDLVLR、LVD1CR0、LVD2CR0）への書き込みを有効にします。
  b2 予約済み （0 にセット）
  b1 PRC1 - ローパワー モードに関連したレジスタ （SBYCR、SNZCR、SNZEDCR、SNZREQCR、FLSTOP、OPCCR、SOPCCR、SYOCDCR）への書き込みを有効にします。
  b0 PRC0 - クロック発生回路に関連したレジスタ （SCKDIVCR、SCKSCR、MOSCCR、MOCOCR、CKOCR、OSTDCR、OSTDSR、MOCOUTCR、HOCOUTCR、MOSCWTCR、MOMCR、SOSCCR、SOMCR、LOCOCR、LOCOUTCR、HOCOWTCR）への書き込みを有効にします。

R_BSP_RegisterProtectDisable

R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect)

概要説明

レジスタ保護を無効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ （PRCR）と MPC の書き込み保護レジスタ （PWPR）を使用して無効にします。

詳細説明

表 1027: パラメータ

名前	方向	説明
regs_to_unprotect	複数のビットを書き換えることもできます。	書き込み保護が無効になっているレジスタ。

関数のステップ

- PRCR レジスタに書き込む場合は、上位 8 ビットを正しいキーにする必要があります。下位ビットを 1 にセットして書き込みを有効にします。
- b15:b8 PRKEY - 0xA5 を上位バイトに書き込んで、下位バイトへの書き込みを有効にします。
  b7:b4 予約済み （0 にセット）
  b3 PRC3 - LVD に関連したレジスタ （LVCMPCR、LVDLVLR、LVD1CR0、LVD1CR1、LVD1SR、LVD2CR0、LVD2CR1、LVD2SR）への書き込みを有効にします。
  b2 予約済み （0 にセット）
  b1 PRC1 - 動作モード、低電力消費、クロック発生回路、およびソフトウェア リセットに関連したレジスタ（SYSCR1、SBYCR、MSTPCRA、MSTPCRB、MSTPCRC、OPCCR、RSTCKCR、SOPCCR、MOFCR、MOSCWTCR、SWRR）への書き込みを有効にします。
  b0 PRC0 - クロック発生回路に関連したレジスタ（SCKCR、SCKCR3、PLLCR、PLLCR2、MOSCCR、SOSCCR、LOCOCR、ILOCOCR、HOCOCR、OSTDCR、OSTDSR、CKOCR）への書き込みを有効にします。
- 保護カウンタをインクリメントします。

9.3.3.9 ROM レジスタ

ROM （OFS など）内に存在し、コンパイル時にセットする必要のある MCU レジスタを定義します。レジスタはすべて bsp_cfg.h を使用してセットできます。

9.3.4 S3A7

S3A7 に共通するコード。

S3A7 に共通する関数を実装します。

9.3.4.1 定義

- #define BSP_PACKAGE_BGA

  初期値 :
- #define BSP_PACKAGE_PINS

  初期値 :(121)
- #define BSP_LOCO_HZ

  初期値 :(32768)

  低速オンチップ オシレーター。
- #define BSP_MOCO_HZ

  初期値 :(8000000)

  中速オンチップ オシレーター。
- #define BSP_SUB_CLOCK_HZ

  初期値 :(32768)

  サブ クロック オシレーター。
- #define BSP_HOCO_HZ

  初期値 :(24000000)

  高速オンチップ オシレーター。
- #define BSP_VECTOR_TABLE_ENTRIES

  初期値 :(80)
- #define BSP_MCU_VBATT_SUPPORT

  初期値 :(1)

9.3.4.2 elc_peripheral_t

イベント信号にリンクされる可能性のある周辺機能

9.3.4.3 elc_event_t

他の周辺機器または CPU1 にリンクされるイベント信号のソース。
I：このリストは、デバイスに応じて変更される可能性があります。これは S3A7 に対応したリストです。

9.3.4.4 クロック初期化

このファイル内の関数は、bsp_clock_cfg.h 内のマクロに基づいてシステム クロックを設定します。

9.3.4.5 グループ化割り込みのサポート

グループ化割り込みのサポート。グループ化割り込みは、複数の割り込みイベントが同じイベント ベクターをトリガーしたときに発生します。この共通ベクターがトリガーされた場合は、アクティベーション ソースを発見する必要があります。このファイル内の関数を使用すれば、ユーザーは、割り込みグループ内のシングル割り込みソース用のコールバック関数を登録することができます。

グループ化割り込みのサポート。グループ化割り込みは、複数の割り込みイベントが同じイベント ベクターをトリガーしたときに発生します。この共通ベクターがトリガーされた場合は、アクティベーション ソースを発見する必要があります。このファイル内の関数を使用すれば、ユーザーは、割り込みグループ内のシングル割り込みソース用のコールバック関数を登録することができます。

Functions

- R_BSP_GroupIrqWrite

R_BSP_GroupIrqWrite

```
ssp_err_t  R_BSP_GroupIrqWrite (  bsp_grp_irq_t irq ,  void(*)(bsp_grp_irq_t irq) p_callback )
```

概要説明

サポートされている割り込みのコールバック関数を登録します。コールバック引数に NULL が渡された場合は、過去に登録されたコールバックが登録解除されます。

詳細説明

表 1028: パラメータ

名前	方向	説明
irq	複数のビットを書き換えることもできます。	コールバックを登録するための割り込み。
p_callback	複数のビットを書き換えることもできます。	割り込みの発生時に呼び出す関数へのポインタ。

表 1029: 戻り値

名前	説明
SSP_SUCCESS	コールバックが登録されました。

関数のステップ

- 有効なアドレスをチェックします。
- コールバックが NULL でした。コールバックを登録解除します。
- コールバックを登録します。

9.3.4.6 グループ化割り込みのサポート

このモジュールは、NVIC 割り込みをトリガーできるように特定の ELC イベントを設定します。割り込みとして使用されるイベントは bsp_irq_cfg.h 内の設定によって異なります。

Functions

- R_BSP_IrqStatusClear

R_BSP_IrqStatusClear

R_BSP_IrqStatusClear (IRQn_Type irq)

概要説明

特定の割り込みの割り込みステータス フラグ （IR）をクリアします。: 特定の割り込みの割り込みステータス フラグ （IR）をクリアします。割り込みがトリガーされると、IR ビットがセットされます。

詳細説明

表 1030: パラメータ

名前	方向	説明
irq	複数のビットを書き換えることもできます。	IR ビットをクリアするための割り込み。

I :IRQn_Type 値が <0 になるシステム例外では、これが機能しません。

9.3.4.7 アトミック ロック

このモジュールは、アトミック ロック メカニズムを実装します。

Functions

- R_BSP_SoftwareLock
- R_BSP_SoftwareUnlock

- R_BSP_HardwareLock
- R_BSP_HardwareUnlock
- bsp_init_hardware_locks
- R_BSP_SoftwareLockInit

データ構造体

- bsp_lock_t

R_BSP_SoftwareLock

`ssp_err_t` R_BSP_SoftwareLock (`bsp_lock_t` * p_lock)

概要説明

送信されたロックの取得を試みます。

詳細説明

表 1031: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	取得するロックを含む構造体へのポインタ。

表 1032: 戻り値

名前	説明
SSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

R_BSP_SoftwareUnlock

R_BSP_SoftwareUnlock (`bsp_lock_t` * p_lock)

概要説明

ホールドオン ロックを解除します。

詳細説明

表 1033: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	ロック解除するロックを含む構造体へのポインタ。

R_BSP_HardwareLock

`ssp_err_t` R_BSP_HardwareLock (bsp_hw_lock_t hw_resource)

概要説明

ハードウェア リソース ロックの予約を試みます。

詳細説明

表 1034: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロックするハードウェア リソースに対するロック配列内のインデックス。

表 1035: 戻り値

名前	説明
BSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

表 1036: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロックするハードウェア リソースに対するロック配列内のインデックス。

表 1037: 戻り値

名前	説明
SSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

R_BSP_HardwareUnlock

R_BSP_HardwareUnlock (bsp_hw_lock_t hw_resource)

概要説明

ホールドオン ロックを解除します。

詳細説明

表 1038: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロック解除するハードウェア リソースに対するロック配列内のインデックス。

bsp_init_hardware_locks

bsp_init_hardware_locks (void)

概要説明

すべてのハードウェア ロックを BSP_LOCK_UNLOCKED に初期化します。

R_BSP_SoftwareLockInit

R_BSP_SoftwareLockInit (bsp_lock_t * p_lock)

概要説明

ロック値をロック済みに初期化します。

詳細説明

表 1039: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	初期化するロックを含む構造体へのポインタ。

bsp_lock_t

bsp_lock_t

詳細説明

ロック構造体。ソフトウェア ロッキング関数に渡されます （R_BSP_SoftwareLock や R_BSP_SoftwareUnLock など）。

変数

- uint8_t lock

  サイズは 8 ビットである必要があるので、enum の代わりに uint8_t が使用されます。

9.3.4.8 レジスタ保護

重要なレジスタは書き込み保護されています。このモジュールは、これらのレジスタの保護を設定するための API を提供します。適切な動作を保証するために参照カウンタが使用されます。

Functions

- R_BSP_RegisterProtectEnable
- R_BSP_RegisterProtectDisable

R_BSP_RegisterProtectEnable

R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect)

概要説明

レジスタ保護を有効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ （PRCR）と MPC の書き込み保護レジスタ （PWPR）を使用することで可能になります。

詳細説明

表 1040: パラメータ

名前	方向	説明
regs_to_protect	複数のビットを書き換えることもできます。	書き込み保護が有効になっているレジスタ。

関数のステップ

- S3A7 用 PRCR レジスタを使用して保護を有効にします。
- PRCR レジスタに書き込む場合は、上位 8 ビットを正しいキーにする必要があります。下位ビットを 0 にセットして書き込みを無効にします。S3A7 用 PRCR レジスタの説明です :
- b15:b8 PRKEY - 0xA5 を上位バイトに書き込んで、下位バイトへの書き込みを有効にします。
  b7:b4 予約済み （0 にセット）

b3 PRC3 - LVD に関連したレジスタ
LVD1CR1, LVD1SR, LVD2CR1, LVD2SR, LVCMPCR, LVDLVLR, LVD1CR0, LVD2CR0.
b2 予約済み （0 にセット）
b1 PRC1 - ローパワー モードに関連したレジスタ
SBYCR, SNZCR, SNZEDCR, SNZREQCR, FLSTOP, PSMCR, OPCCR, SOPCCR, SYOCDCR.
バックアップ機能に関連したレジスタへの書き込みを有効にします :
VBTCR1, VBTCR2, VBTSR, VBTCMPCR, VBTLVDICR, VBTWCTLR, VBTWCH0OTSR, VBTWCH1OTSR, VBTWCH2OTSR, VBTICTLR, VBTOCTLR, VBTWTER, VBTWEGR, VBTWFR, VBTBKRn （n = 0 to 511）
b0 PRC0 - クロック発生回路に関連したレジスタ
SCKDIVCR, SCKSCR, PLLCR, PLLCCR2, BCKCR, MEMWAIT, MOSCCR, HOCOCR, MOCOCR, CKOCR, TRCKCR, OSTDCR, OSTDSR, SLCDSCKCR, EBCKOCR, MOCOUTCR, HOCOUTCR, MOSCWTCR, MOMCR, SOSCCR, SOMCR, LOCOCR, LOCOUTCR, HOCOWTCR.

R_BSP_RegisterProtectDisable

R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect)

概要説明

レジスタ保護を無効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ （PRCR）と MPC の書き込み保護レジスタ （PWPR）を使用して無効にします。

詳細説明

表 1041: パラメータ

名前	方向	説明
regs_to_unprotect	複数のビットを書き換えることもできます。	書き込み保護が無効になっているレジスタ。

関数のステップ

- S3A7 の PRCR レジスタを使用して保護を無効にします。
- PRCR レジスタに書き込む場合は、上位 8 ビットを正しいキーにする必要があります。下位ビットを 0 にセットして書き込みを無効にします。S3A7 用 PRCR レジスタの説明です :
- b15:b8 PRKEY - 0xA5 を上位バイトに書き込んで、下位バイトへの書き込みを有効にします。
  b7:b4 予約済み （0 にセット）
  b3 PRC3 - LVD に関連したレジスタ
  LVD1CR1, LVD1SR, LVD2CR1, LVD2SR, LVCMPCR, LVDLVLR, LVD1CR0, LVD2CR0.
  b2 予約済み （0 にセット）
  b1 PRC1 - ローパワー モードに関連したレジスタ
  SBYCR, SNZCR, SNZEDCR, SNZREQCR, FLSTOP, PSMCR, OPCCR, SOPCCR, SYOCDCR.
  バックアップ機能に関連したレジスタへの書き込みを有効にします :
  VBTCR1, VBTCR2, VBTSR, VBTCMPCR, VBTLVDICR, VBTWCTLR, VBTWCH0OTSR, VBTWCH1OTSR,

VBTWCH2OTSR, VBTICTLR, VBTOCTLR, VBTWTER, VBTWEGR, VBTWFR, VBTBKRn （n = 0 to 511）
b0 PRC0 - クロック発生回路に関連したレジスタ
SCKDIVCR, SCKSCR, PLLCR, PLLCCR2, BCKCR, MEMWAIT, MOSCCR, HOCOCR, MOCOCR, CKOCR, TRCKCR, OSTDCR, OSTDSR, SLCDSCKCR, EBCKOCR, MOCOUTCR, HOCOUTCR, MOSCWTCR, MOMCR, SOSCCR, SOMCR, LOCOCR, LOCOUTCR, HOCOWTCR.

9.3.4.9 ROM レジスタ

ROM （OFS など）内に存在し、コンパイル時にセットする必要のある MCU レジスタを定義します。レジスタはすべて bsp_cfg.h を使用してセットできます。

9.3.5 S7G2

このモジュールは、これらのレジスタの保護を設定するための API を提供します。

S7G2 に共通する関数を実装します。

9.3.5.1 elc_peripheral_t

イベント信号にリンクされる可能性のある周辺機能

9.3.5.2 elc_event_t

他の周辺機器または CPU1 にリンクされるイベント信号のソース。
I：このリストは、デバイスに応じて変更される可能性があります。これは、S7G2 に対応したリストです。

9.3.5.3 クロック初期化

このファイル内の関数は、bsp_clock_cfg.h 内のマクロに基づいてシステム クロックを設定します。

9.3.5.4 グループ化割り込みのサポート

グループ化割り込みのサポート。グループ化割り込みは、複数の割り込みイベントが同じイベント ベクターをトリガーしたときに発生します。この共通ベクターがトリガーされた場合は、アクティベーション ソースを発見する必要があります。このファイル内の関数を使用すれば、ユーザーは、割り込みグループ内のシングル割り込みソース用のコールバック関数を登録することができます。

グループ化割り込みのサポート。グループ化割り込みは、複数の割り込みイベントが同じイベント ベクターをトリガーしたときに発生します。この共通ベクターがトリガーされた場合は、アクティベーション

ソースを発見する必要があります。このファイル内の関数を使用すれば、ユーザーは、割り込みグループ内のシングル割り込みソース用のコールバック関数を登録することができます。

Functions

- R_BSP_GroupIrqWrite

R_BSP_GroupIrqWrite

```
ssp_err_t  R_BSP_GroupIrqWrite (  bsp_grp_irq_t irq ,  void(*)(bsp_grp_irq_t irq) p_callback )
```

概要説明

サポートされている割り込みのコールバック関数を登録します。コールバック引数に NULL が渡された場合は、過去に登録されたコールバックが登録解除されます。

詳細説明

表 1042: パラメータ

名前	方向	説明
irq	複数のビットを書き換えることもできます。	コールバックを登録するための割り込み。
p_callback	複数のビットを書き換えることもできます。	割り込みの発生時に呼び出す関数へのポインタ。

表 1043: 戻り値

名前	説明
SSP_SUCCESS	コールバックが登録されました。

関数のステップ

- 有効なアドレスをチェックします。
- コールバックが NULL でした。コールバックを登録解除します。
- コールバックを登録します。

9.3.5.5 グループ化割り込みのサポート

このモジュールは、NVIC 割り込みをトリガーできるように特定の ELC イベントを設定します。割り込みとして使用されるイベントは bsp_irq_cfg.h 内の設定によって異なります。

Functions

- R_BSP_IrqStatusClear

R_BSP_IrqStatusClear

R_BSP_IrqStatusClear (IRQn_Type irq)

概要説明

特定の割り込みの割り込みステータス フラグ （IR）をクリアします。: 特定の割り込みの割り込みステータス フラグ （IR）をクリアします。割り込みがトリガーされると、IR ビットがセットされます。

詳細説明

表 1044: パラメータ

名前	方向	説明
irq	複数のビットを書き換えることもできます。	IR ビットをクリアするための割り込み。

I :IRQn_Type 値が <0 になるシステム例外では、これが機能しません。

9.3.5.6 アトミック ロック

このモジュールは、アトミック ロック メカニズムを実装します。

Functions

- R_BSP_SoftwareLock
- R_BSP_SoftwareUnlock
- R_BSP_HardwareLock
- R_BSP_HardwareUnlock
- bsp_init_hardware_locks
- R_BSP_SoftwareLockInit

データ構造体

- bsp_lock_t

R_BSP_SoftwareLock

```
ssp_err_t R_BSP_SoftwareLock ( bsp_lock_t * p_lock )
```

概要説明

送信されたロックの取得を試みます。

詳細説明

表 1045: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	取得するロックを含む構造体へのポインタ。

表 1046: 戻り値

名前	説明
SSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

R_BSP_SoftwareUnlock

R_BSP_SoftwareUnlock (bsp_lock_t * p_lock)

概要説明

ホールドオン ロックを解除します。

詳細説明

表 1047: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	ロック解除するロックを含む構造体へのポインタ。

R_BSP_HardwareLock

ssp_err_t R_BSP_HardwareLock (bsp_hw_lock_t hw_resource)

概要説明

ハードウェア リソース ロックの予約を試みます。

詳細説明

表 1048: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロックするハードウェア リソースに対するロック配列内のインデックス。

表 1049: 戻り値

名前	説明
BSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

表 1050: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロックするハードウェア リソースに対するロック配列内のインデックス。

表 1051: 戻り値

名前	説明
SSP_SUCCESS	ロックが取得されました。
チャネルは現在動作中です。	ロックが取得されませんでした。

R_BSP_HardwareUnlock

R_BSP_HardwareUnlock (bsp_hw_lock_t hw_resource)

概要説明

ホールドオン ロックを解除します。

詳細説明

表 1052: パラメータ

名前	方向	説明
hw_resource	複数のビットを書き換えることもできます。	ロック解除するハードウェア リソースに対するロック配列内のインデックス。

bsp_init_hardware_locks

bsp_init_hardware_locks (void)

概要説明

すべてのハードウェア ロックを BSP_LOCK_UNLOCKED に初期化します。

R_BSP_SoftwareLockInit

R_BSP_SoftwareLockInit (bsp_lock_t * p_lock)

概要説明

ロック値をロック済みに初期化します。

詳細説明

表 1053: パラメータ

名前	方向	説明
p_lock	複数のビットを書き換えることもできます。	初期化するロックを含む構造体へのポインタ。

bsp_lock_t

bsp_lock_t

詳細説明

ロック構造体。ソフトウェア ロッキング関数に渡されます （R_BSP_SoftwareLock や R_BSP_SoftwareUnLock など）。

変数

- uint8_t lock

  サイズは 8 ビットである必要があるので、enum の代わりに uint8_t が使用されます。

9.3.5.7 レジスタ保護

重要なレジスタは書き込み保護されています。このモジュールは、これらのレジスタの保護を設定するための API を提供します。適切な動作を保証するために参照カウンタが使用されます。

Functions

- R_BSP_RegisterProtectEnable
- R_BSP_RegisterProtectDisable

R_BSP_RegisterProtectEnable

R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect)

概要説明

レジスタ保護を有効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ （PRCR）と MPC の書き込み保護レジスタ （PWPR）を使用することで可能になります。

詳細説明

表 1054: パラメータ

名前	方向	説明
regs_to_protect	複数のビットを書き換えることもできます。	書き込み保護が有効になっているレジスタ。

関数のステップ

- S7G2 用 PRCR レジスタを使用して保護を有効にします。
- PRCR レジスタに書き込む場合は、上位 8 ビットを正しいキーにする必要があります。下位ビットを 0 にセットして書き込みを無効にします。S7G2 用 PRCR レジスタの説明です :
- b15:b8 PRKEY - 0xA5 を上位バイトに書き込んで、下位バイトへの書き込みを有効にします。
  b7:b4 予約済み （0 にセット）
  b3 PRC3 - LVD に関連したレジスタ
  LVD1CR1, LVD1SR, LVD2CR1, LVD2SR, LVCMPCR, LVDLVLR, LVD1CR0, LVD2CR0.
  b2 予約済み （0 にセット）
  b1 PRC1 - ローパワー モードに関連したレジスタ
  SBYCR, SNZCR, SNZEDCR, SNZREQCR, OPCCR, SOPCCR, DPSBYCR, DPSIER0-3, DPSIFR0-3, DPSIEGR0-3, SYOCDCR, STCONR.
  バックアップ機能に関連したレジスタへの書き込みを有効にします :
  VBTBKRn （n = 0 to 511）
  b0 PRC0 - クロック発生回路に関連したレジスタ
  SCKDIVCR, SCKDIVCR2, SCKSCR, PLLCCR, PLLCR, BCKCR, MOSCCR, HOCOCR, MOCOCR, CKOCR, TRCKCR, OSTDCR, OSTDSR, EBCKOCR, SDCKOCR, MOCOUTCR, HOCOUTCR, MOSCWTCR, MOMCR, SOSCCR, SOMCR, LOCOCR, LOCOUTCR, HOCOWTCR.

R_BSP_RegisterProtectDisable

R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect)

概要説明

レジスタ保護を無効にします。保護されたレジスタに書き込むことはできません。レジスタ保護は、保護レジスタ （PRCR）と MPC の書き込み保護レジスタ （PWPR）を使用して無効にします。

詳細説明

表 1055: パラメータ

名前	方向	説明
regs_to_unprotect	複数のビットを書き換えることもできます。	書き込み保護が無効になっているレジスタ。

関数のステップ

- S7G2 用 PRCR レジスタを使用して保護を無効にします。
- PRCR レジスタに書き込む場合は、上位 8 ビットを正しいキーにする必要があります。下位ビットを 0 にセットして書き込みを無効にします : S7G2 用 PRCR レジスタの説明です :
- b15:b8 PRKEY - 0xA5 を上位バイトに書き込んで、下位バイトへの書き込みを有効にします。
  b7:b4 予約済み （0 にセット）
  b3 PRC3 - LVD に関連したレジスタ
  LVD1CR1, LVD1SR, LVD2CR1, LVD2SR, LVCMPCR, LVDLVLR, LVD1CR0, LVD2CR0.
  b2 予約済み （0 にセット）
  b1 PRC1 - ローパワー モードに関連したレジスタ
  SBYCR, SNZCR, SNZEDCR, SNZREQCR, OPCCR, SOPCCR, DPSBYCR, DPSIER0-3, DPSIFR0-3, DPSIEGR0-3, SYOCDCR, STCONR.
  バックアップ機能に関連したレジスタへの書き込みを有効にします :
  VBTBKRn （n = 0 to 511）
  b0 PRC0 - クロック発生回路に関連したレジスタ
  SCKDIVCR, SCKDIVCR2, SCKSCR, PLLCCR, PLLCR, BCKCR, MOSCCR, HOCOCR, MOCOCR, CKOCR, TRCKCR, OSTDCR, OSTDSR, EBCKOCR, SDCKOCR, MOCOUTCR, HOCOUTCR, MOSCWTCR, MOMCR, SOSCCR, SOMCR, LOCOCR, LOCOUTCR, HOCOWTCR.

9.3.5.8 ROM レジスタ

ROM （OFS など）内に存在し、コンパイル時にセットする必要のある MCU レジスタを定義します。レジスタはすべて bsp_cfg.h を使用してセットできます。

9.4 共通 BSP コード

すべての BSP に共通するコード。

すべての BSP に共通する関数を実装します。

9.4.1 Functions

- R_BSP_VersionGet
- R_SSP_VersionGet

9.4.2 定義

- #define BSP_IRQ_DISABLED

  初期値 :(0xFFFFFFFF)

  ELC イベントは割り込みとして使用できないことを伝達するために使用されます。
- #define BSP_CODE_VERSION_MAJOR

  初期値 :(1)
- #define BSP_CODE_VERSION_MINOR

  初期値 :(1)
- #define BSP_API_VERSION_MAJOR

  初期値 :(1)
- #define BSP_API_VERSION_MINOR

  初期値 :(0)
- #define SF_CONTEXT_SAVE

  初期値 :
- #define SF_CONTEXT_RESTORE

  初期値 :
- #define SSP_ASSERT_FAIL

  初期値 :

  アサーション エラーを返す前に挿入する関数呼び出し。

- #define SSP_ERROR_LOG

  初期値 :

  この関数はエラー コードを返す前に呼び出されます。ランタイム エラーで停止するには、ユーザー コード内で ssp_error_log を定義して、この関数で必要なデバッグ （ブレークポイント、スタック ダンプなど）を実行します。

- #define SSP_ERROR_RETURN

  初期値 :{ \ if ((a)) \ { \ (void) 0; /* Do nothing */ \ } \ else \ { \ `#define SSP_ERROR_LOG`((err), (module), (version)); \ return (err); \ } \ }

  すべての SSP エラー コードがこのマクロを使用して返されます。条件 "a" が false の場合に、#define SSP_ERROR_LOG 関数を呼び出します。SSP 関数でランタイム エラーを特定するために使用されます。

9.4.3 R_BSP_VersionGet

`ssp_err_t` R_BSP_VersionGet (`ssp_version_t` * p_version)

9.4.3.1 概要説明

コンパイル時マクロに基づいて BSP バージョンをセットします。

9.4.3.2 詳細説明

表 1056: パラメータ

名前	方向	説明
p_version	out	バージョン情報を返す先のメモリ アドレス。

表 1057: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報が保存されました。

9.4.4 R_SSP_VersionGet

`ssp_err_t` R_SSP_VersionGet (`ssp_pack_version_t` *const p_version)

9.4.4.1 概要説明

コンパイル時マクロに基づいて SSP バージョンをセットします。

9.4.4.2 詳細説明

表 1058: パラメータ

名前	方向	説明
p_version	out	バージョン情報を返す先のメモリ アドレス。

表 1059: 戻り値

名前	説明
SSP_SUCCESS	バージョン情報が保存されました。

9.4.5 共通 BSP LED コードとタイプ

ボード LED に対する共通サポート。

ボード上の LED の共通使用を可能にするタイプと関数が含まれます。

9.4.5.1 データ構造体

- bsp_leds_t

9.4.5.2 Functions

- R_BSP_LedsGet

9.4.5.3 R_BSP_LedsGet

```
ssp_err_t  R_BSP_LedsGet ( bsp_leds_t  * p_leds )
```

概要説明

現在のボード上の LED に関する情報を返します。

詳細説明

LED 情報を含む構造体。

表 1060: パラメータ

名前	方向	説明
p_leds	out	LED 情報が保存された構造体へのポインタ。

9.4.5.4 bsp_leds_t

bsp_leds_t

詳細説明

LED の数およびそれらが割り当てられているピンに関する情報。

変数

- uint16_t led_count

  このボード上の LED の数。

- ioport_port_pin_t const * p_leds

  LED を制御するための IOPORT ピンの配列へのポインタ。

9.4.6 コンパイラ サポート

このファイル内のマクロは使用されているコンパイラに基づいて定義されます。マクロは、共通セクション名を抽象化して、特定のセクションにコードを配置するための共通の方法を提供します。

マクロの説明 :

- BSP_SECTION_STACK - スタックが保存されたセクションの名前
- BSP_SECTION_HEAP - ヒープが保存されたセクションの名前
- BSP_SECTION_VECTOR - ベクター テーブルが保存されたセクションの名前
- BSP_SECTION_ROM_REGISTERS - ROM レジスタが配置されたセクションの名前
- BSP_PLACE_IN_SECTION - 特定のセクションにコードを配置するためのマクロ
- BSP_DONT_REMOVE - リンカー / コンパイラに変数または関数を最適化しないように指示するためのキーワード

I : 現在サポートされているコンパイラは GCC と IAR です。

9.4.7 ソフトウェア遅延

ソフトウェア遅延を実装するための共通関数。

すべての BSP 用のソフトウェア遅延関数を実装します。

9.4.7.1 Functions

- R_BSP_SoftwareDelay

9.4.7.2 R_BSP_SoftwareDelay

R_BSP_SoftwareDelay (uint32_t delay , bsp_delay_units_t units)

概要説明

指定された単位当たりの持続時間を遅らせて復帰します。

詳細説明

表 1061: パラメータ

名前	方向	説明
delay	複数のビットを書き換えることもできます。	遅らせる単位数。
units	複数のビットを書き換えることもできます。	指定された単位のベース (bsp_delay_units_t)。有効な値は、BSP_DELAY_UNITS_SECONDS、BSP_DELAY_UNITS_MILLISECONDS、BSP_DELAY_UNITS_MICROSECONDS です。 例 : 1 MHz では、1 サイクルに 1 マイクロ秒 （.000001 秒）かかります。 12 MHz では、1 サイクルに 1/12 マイクロ秒または 83 ナノ秒かかります。 そのため、software_delay_loop() 経由の 1 実行に ~ (83 * DELAY_LOOP_CYCLES) または 332 ns かかります。そのため、software_delay_loop() 経由の 1 実行に ~ (83 * DELAY_LOOP_CYCLES) または 332 ns かかります。

I：この関数は、最終的に R_CGC_SystemClockFreqGet を呼び出す bsp_cpu_clock_get() を呼び出すため、BSP がすでに CGC を初期化している （Sysinit の一部として実行する）必要があります。将来的にこの関数を BSP 初期化の一部として呼び出さなければならないケースがあり得ることに注意する必要があります。

表 1062: 戻り値

名前	説明
なし。	

関数のステップ

- 要求された時間をマイクロ秒に変換します。
- システム クロック周波数を Hz 単位で取得します。
- この関数は、BSP_SUB_CLOCK_HZ と同じくらい遅いクロックを使用して小時間単位 BSP_DELAY_UNITS_MICROSECONDS だけ正確に遅らせることができないため、1 ループだけ遅らせて復帰します。
- ナノ秒 / サイクルの数を取得します。
- すべての計算が 32 ビットに収まることが想定されており、整数の計算しか実行されません。そのため、最初に total_us をマイクロ秒にスケールします。その後で、ナノ秒にスケールします。2 マイクロ秒のように遅延が小さい場合は、最初にナノ秒に変換するため、ns_per_cycle による除算が正確で意味のある結果になります。一方、秒単位の大きい遅延の場合は、最初にナノ秒にスケールすると、32 ビットをオーバーフローすることになります。
- 除算の後に乗算してマイクロ秒 > ナノ秒 /（ns/ サイクル）を取得します。
- 乗算の後に除算してマイクロ秒 > ナノ秒 /（ns/ サイクル）を取得します。
- 指定されたパラメータで遅延が必要な場合にのみ遅らせます。指定された遅延が満たせない場合は、何もせずに復帰します。

9.4.8 エラー チェック

このファイルは、可能な場合にビルド時エラー チェックを実行します。

9.4.8.1 定義

- #define BSP_STACK_ALIGNMENT

  初期値 :(8)

  スタック （およびヒープ）はこの数字の倍数までサイズを調整し、揃える必要があります。

第 10 章 API リファレンス：共通

You can find error codes and version data structures that are common to the entire Synergy Software Package here.

10.1 Error codes

10.1.1 一般的なエラーコード

すべてのレイヤーの全 SSP コードはこれらの一般的なエラーコードを共有しています。

10.1.1.1 ユーザー定義列挙

- ssp_err_t
- ssp_command_t

10.1.1.2 ユーザー定義マクロ

- #define SSP_PARAMETER_NOT_USED

  初期値 :(void) ((p))

  この marco は関数で使用されていないパラメータについてのコンパイラ メッセージを抑えるのに使用されます。この実装を使用する利点は、余分な RAM や ROM を必要としないということです。

10.1.1.3 API データ

ssp_err_t

ssp_err_t

詳細説明

一般的なエラーコード

列挙値

名前	説明
SSP_SUCCESS	
p_ctrl の可能性があります。	重要なアサーションが失敗しました。
Null ポインタが指定されました。	ポインタが無効なメモリの場所をポイントしています。
SSP_ERR_INVALID_POINTER	無効な入力パラメータです。
SSP_ERR_INVALID_CHANNEL	選択したチャネルは存在しません。

名前	説明
未サポートまたは不正確なモードです。	SSP_ERR_INVALID_ARGUMENT
データ構造体が割り当てられませんでした。	選択したモードはこの API ではサポートされていません。
タッチ パネルが設定されていません。	リクエストしたチャネルは構成されていないか、API が開かれていません。
チャネルは現在動作中です。	チャネル / 周辺機器が実行中 / ビジーです。
SSP_ERR_OUT_OF_MEMORY	ドライバの cfg.h により多くのメモリを割り当ててください。
SSP_ERR_INVALID_ARGUMENT	ハードウェアはロックされています。
SSP_ERR_IRQ_BSP_DISABLED	IRQ が BSP で有効になっていません。
SSP_ERR_OVERFLOW	ハードウェアオーバフロー。
SSP_ERR_UNDERFLOW	ハードウェアアンダーフロー。
SSP_ERR_ALREADY_OPEN	リクエストしたチャネルは異なる構成ですでに開かれています。
SSP_ERR_APPROXIMATION	値を正確な結果に設定できませんでした。
SSP_ERR_CLAMPED	値は何らかの理由で制限される必要がありました。
SSP_ERR_INVALID_RATE	選択したレートを満たすことができませんでした。
SSP_ERR_ABORTED	操作が回避されました。
SSP_ERR_NOT_ENABLED	リクエストされた操作は有効にされていません。
SSP_ERR_TIMEOUT	タイムアウトエラー。
SSP_ERR_INVALID_BLOCKS	無効な数のブロックが提供されました。
SSP_ERR_INVALID_ADDRESS	無効なアドレスが提供されました。
SSP_ERR_INVALID_SIZE	無効なサイズ / 長さが操作で提供されました。
SSP_ERR_WRITE_FAILED	書き込み操作が失敗しました。
SSP_ERR_ERASE_FAILED	削除操作が失敗しました。
SSP_ERR_INVALID_CALL	無効な関数呼び出しが行われます。

名前	説明
SSP_ERR_INVALID_HW_CONDITION	検出されたハードウェアは無効な条件です。
内部エラーが発生しました。	内部エラー。RTOS のみのエラーコードの始まりです。
SSP_ERR_WAIT_ABORTED	待機。
SSP_ERR_FRAMING	フレーミングエラーが発生しました。UART 固有の始まりです。
SSP_ERR_BREAK_DETECT	ブレイク信号の検出。
SSP_ERR_PARITY	パリティエラーが発生しました。
SSP_ERR_RXBUF_OVERFLOW	キューオーバーフローを受け取ります。
SSP_ERR_QUEUE_UNAVAILABLE	s/w キューを開けません。
SSP_ERR_INSUFFICIENT_SPACE	伝送循環バッファに十分なスペースがありません。
SSP_ERR_INSUFFICIENT_DATA	受信循環バッファに十分なデータがありません。
SSP_ERR_TRANSFER_ABORTED	データ転送が中止されました。SPI 固有の始まりです
SSP_ERR_MODE_FAULT	モード障害エラー。
SSP_ERR_READ_OVERFLOW	読み取りオーバーフロー。
SSP_ERR_SPI_PARITY	パリティ エラー。
SSP_ERR_OVERRUN	オーバーラン エラー。
SSP_ERR_CLOCK_INACTIVE	非アクティブのクロックがシステムクロックとして指定されました。CGC 固有の始まりです
SSP_ERR_CLOCK_ACTIVE	アクティブなクロックソースは、まず停止しないと変更できません。
SSP_ERR_STABILIZED	クロックをオン / オフにした後に安定しました。
SSP_ERR_NOT_STABILIZED	クロックをオン / オフにした後に安定していません。
SSP_ERR_MAIN_OSC_INACTIVE	メイン OSC をオフにしたときに PLL 初期化が試みられました。
SSP_ERR_OSC_STOP_DET_ENABLED	オシレーション停止を有効にしたときに、LOCO を不正に停止しようとしました。

名前	説明
SSP_ERR_OSC_STOP_DETECTED	オシレーション停止検出ステータスフラグが設定されます。
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	PLL/MAIN_OSC がアクティブのときに、オシレーション停止検出ステータスをクリアしようとしました。
SSP_ERR_CLKOUT_EXCEEDED	ターゲット出力クロックピンの出力が、サポートされている上限を超えています。
SSP_ERR_USB_MODULE_ENABLED	USB モジュールが有効な場合の USB クロック構成リクエスト。
SSP_ERR_HARDWARE_TIMEOUT	レジスタ読み取りまたは書き込みがタイムアウトしました。
SSP_ERR_PE_FAILURE	プログラミングモードにすることができませんでした。FLASH 固有の始まりです
SSP_ERR_CMD_LOCKED	周辺機器がコマンドロック状態です。
SSP_ERR_FCLK	FCLK は >= 4 MHz となる必要があります。
SSP_ERR_INVALID_CAC_REF_CLOCK	測定クロックレート < 参照クロックレート。CAC 固有の始まりです
SSP_ERR_CLOCK_GENERATION	クロックをシステムクロックとして指定できません。GLCD 固有の始まりです
SSP_ERR_INVALID_TIMING_SETTING	無効なタイミングパラメータです。
SSP_ERR_INVALID_LAYER_SETTING	無効なレイヤーパラメータです。
SSP_ERR_INVALID_ALIGNMENT	無効なメモリ配列が見つかりました。
SSP_ERR_INVALID_GAMMA_SETTING	無効なガンマ修正パラメータです。
SSP_ERR_INVALID_LAYER_FORMAT	レイヤーでの無効なカラーフォーマットです。
SSP_ERR_INVALID_UPDATE_TIMING	レジスタ更新での無効なタイミングです。
SSP_ERR_INVALID_CLUT_ACCESS	CLUT エントリへの無効なアクセスです。
SSP_ERR_INVALID_FADE_SETTING	無効なフェードイン / フェードアウトです。
SSP_ERR_JPEG_ERR	JPEG エラー。JPEB 固有の始まりです
SSP_ERR_JPEG_SOI_NOT_DETECTED	EOI が検出されるまで SOI は検出されません。

名前	説明
SSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED	SOF1 から SOFF が検出されました。
SSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT	未提供のピクセルフォーマットが検出されました。
SSP_ERR_JPEG_SOF_ACCURACY_ERROR	SOF 精度エラー : 8 以外が検出されました。
SSP_ERR_JPEG_DQT_ACCURACY_ERROR	DQT 精度エラー : 0 以外が検出されました。
SSP_ERR_JPEG_COMPONENT_ERROR1	コンポーネントエラー 1: 検出された SOF0 ヘッダコンポーネントの数が 1、3、または 4 以外でした。
SSP_ERR_JPEG_COMPONENT_ERROR2	コンポーネントエラー 2: コンポーネントの数が SOF0 ヘッダと SOS で異なります。
SSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED	SOS が検出されると SOF0、DQT、および DHT は検出されません。
SSP_ERR_JPEG_SOS_NOT_DETECTED	SOS が検出されませんでした : EOI が検出されるまで SOS は検出されません。
SSP_ERR_JPEG_EOI_NOT_DETECTED	EOI が検出されませんでした（デフォルト）
SSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR	リスタート間隔データ数エラーが検出されました。
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	イメージサイズエラーが検出されました。
SSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR	最後の MCU データ番号エラーが検出されました。
SSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR	ブロックデータ数エラーが検出されました。
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	ユーザが提供したバッファサイズが十分ではありません。
SSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE	JPEG イメージサイズが MCU とアラインされていません。
SSP_ERR_CALIBRATE_FAILED	カリブレーションに失敗しました。タッチパネルフレームワーク固有の始まり
SSP_ERR_NO_MORE_BUFFER	メモリブロックプールにバッファがこれ以上見つかりません。メッセージフレームワーク固有の始まり
SSP_ERR_ILLEGAL_BUFFER_ADDRESS	バッファアドレスがブロックメモリプールの外にあります。

名前	説明
SSP_ERR_INVALID_WORKBUFFER_SIZE	ワークバッファサイズが無効です。
SSP_ERR_INVALID_MSG_BUFFER_SIZE	メッセージバッファサイズが無効です。
SSP_ERR_TOO_MANY_BUFFERS	バッファの数が多すぎます。
SSP_ERR_NO_SUBSCRIBER_FOUND	メッセージサブスクライバが見つかりませんでした。
SSP_ERR_MESSAGE_QUEUE_EMPTY	メッセージキューにメッセージが見つかりませんでした。
SSP_ERR_MESSAGE_QUEUE_FULL	メッセージキューに新しいメッセージを入れる余裕がありません。
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	不正なメッセージサブスクライバリストです。
SSP_ERR_BUFFER_RELEASED	バッファがリリースされました。
SSP_ERR_D2D_ERROR_INIT	Dave/2d は初期化にエラーがあります。2DG ドライバ固有の始まり
SSP_ERR_D2D_ERROR_DEINIT	Dave/2d は初期化にエラーがあります。
SSP_ERR_D2D_ERROR_RENDERING	Dave/2d はレンダリングにエラーがあります。
SSP_ERR_D2D_ERROR_SIZE	Dave/2d はレンダリングにエラーがあります。
SSP_ERR_QUEUE_FULL	キューが一杯で、別のデータをキューに入れられません。BYTEQ ライブラリ固有の始まり
SSP_ERR_QUEUE_EMPTY	キューが空で、キューを解除するデータがありません。
SSP_ERR_CTSU_SC_OVERFLOW	CTSU スキャン実行中にセンサー カウントのオーバーフローが発生しました。ユーザーは手動で CTSUSCOVF ビットをクリアする必要があります。
SSP_ERR_CTSU_RC_OVERFLOW	CTSU スキャン実行中に参照カウントのオーバーフローが発生しました。ユーザーは手動で CTSURCOVF ビットをクリアする必要があります。
SSP_ERR_CTSU_ICOMP	異常な TSCAP 電圧です。ユーザーは手動で CTSUICOMP ビットをクリアする必要があります。
NULL ポインタが必要なパラメータに引数として受け渡されました。	自動調整アルゴリズムに失敗しました。
SSP_ERR_CTSU_SAFETY_CHECK_FAILED	セーフティ チェックに失敗しました。

名前	説明
SSP_ERR_CARD_INIT_FAILED	SD カードまたは eMMC デバイスの初期化に失敗しました。SDMMC 固有の始まりです
SSP_ERR_CARD_NOT_INSERTED	SD カードをインストールできません。
SSP_ERR_SDHI_FAILED	SD 周辺機器が正常に応答しませんでした。
SSP_ERR_READ_FAILED	データの読み取りに失敗しました。
SSP_ERR_CARD_NOT_READY	SD カードが取り除かれました。
SSP_ERR_CARD_WRITE_PROTECTED	メディアは書き込み保護されています。
SSP_ERR_TRANSFER_BUSY	転送が進行中です。
SSP_ERR_MEDIA_FORMAT_FAILED	メディアのフォーマットに失敗しました。FX_IO 固有の始まりです
SSP_ERR_MEDIA_OPEN_FAILED	メディアのオープンに失敗しました。
SSP_ERR_CAN_DATA_UNAVAILABLE	利用可能なデータがありません。CAN 固有の始まりです。
SSP_ERR_CAN_MODE_SWITCH_FAILED	動作モードの切り替えに失敗しました。
SSP_ERR_CAN_INIT_FAILED	ハードウェアの初期化に失敗しました。
SSP_ERR_CAN_TRANSMIT_NOT_READY	送信が進行中です。
SSP_ERR_CAN_RECEIVE_MAILBOX	メールボックスは受信メールボックスとして設定されています。
SSP_ERR_CAN_TRANSMIT_MAILBOX	メールボックスは送信メールボックスとして設定されています。
SSP_ERR_CAN_MESSAGE_LOST	受信メッセージが上書きまたはオーバーランされました。

ssp_command_t

ssp_command_t

詳細説明

ioctl commands.

列挙値

参考資料

名前	説明
SSP_COMMAND_GET_SECTOR_COUNT	メディア セクター カウントを取得します。
SSP_COMMAND_GET_SECTOR_SIZE	セクター サイズを取得します。
SSP_COMMAND_GET_BLOCK_SIZE	消去ブロック サイズを取得します。
SSP_COMMAND_CTRL_ERASE_SECTOR	セクターを消去します。
SSP_COMMAND_GET_WRITE_PROTECTED	書き込み保護ステータスを取得します。

参考資料

10.2 Version control

10.2.1 ssp_version_t

10.2.1.1 詳細説明

共通バージョン構造体

10.2.1.2 変数

version_id

code_version_minor

code_version_major

api_version_minor

api_version_major

10.2.2 version_id

uint32_t ::version_id

10.2.2.1 詳細説明

バージョン ID

10.2.3 code_version_minor

uint8_t ::code_version_minor

10.2.3.1 概要説明

コードのマイナー バージョン。

10.2.4 code_version_major

uint8_t ::code_version_major

10.2.4.1 概要説明

コードのメジャー バージョン。

10.2.5 api_version_minor

uint8_t ::api_version_minor

10.2.5.1 概要説明

API のマイナー バージョン。

10.2.6 api_version_major

uint8_t ::api_version_major

10.2.6.1 概要説明

API のメジャー バージョン。

10.2.7 ssp_pack_version_t

10.2.7.1 詳細説明

SSP パック バージョン構造体

10.2.7.2 変数

version_id

build

patch

minor

major

10.2.8 version_id

uint32_t ::version_id

10.2.8.1 詳細説明

バージョン ID

10.2.9 build

uint8_t ::build

10.2.9.1 概要説明

SSP パックのビルド バージョン

10.2.10 patch

uint8_t ::patch

10.2.10.1 概要説明

SSP パックのパッチ バージョン

10.2.11 minor

uint8_t ::minor

10.2.11.1 概要説明

SSP パックのマイナー バージョン。

10.2.12 major

uint8_t ::major

10.2.12.1 概要説明

SSP パックのメジャー バージョン

参考資料

第 11 章 API リファレンス：構造体のインデックス

11.1 Structures

11.1.1 adc_api_t

```
typedef struct{
    ssp_err_t(* open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
    ssp_err_t(* scanCfg)(adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
    ssp_err_t(* scanStart)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(* scanStop)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(* scanStatusGet)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(* read)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, adc_data_size_t *const p_data)
    ssp_err_t(* sampleStateCountSet)(adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)
    ssp_err_t(* close)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(* infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} adc_api_t
```

11.1.2 adc_callback_args_t

```
typedef struct{
    uint16_t unit
    adc_cb_event_t event
    void const * p_context
} adc_callback_args_t
```

11.1.2.1 unit

uint16_t adc_callback_args_t::unit

概要説明

使用中の ADC デバイス。

11.1.2.2 event

adc_cb_event_t::event

概要説明

ADC コールバック イベント。

11.1.2.3 p_context

void const* adc_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.3 adc_cfg_t

```
typedef struct{
    uint16_t unit
    adc_mode_t mode
    adc_resolution_t resolution
    adc_alignment_t alignment
    adc_add_t add_average_count
    adc_clear_t clearing
    adc_trigger_t trigger
    adc_trigger_t trigger_group_b
    void(* p_callback)(adc_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} adc_cfg_t
```

11.1.3.1 unit

uint16_t adc_cfg_t::unit

概要説明

使用される ADC ユニット。

11.1.3.2 mode

adc_mode_t::mode

概要説明

8、10、12 ビットの ADC 分解能。

11.1.3.3 resolution

adc_resolution_t::resolution

概要説明

追加が使用される場合は無視されます。

11.1.3.4 alignment

adc_alignment_t::alignment

概要説明

追加が使用される場合は無視されます。

11.1.3.5 add_average_count

adc_add_t::add_average_count

概要説明

サンプルを追加または平均化します。

11.1.3.6 clearing

adc_clear_t::clearing

概要説明

読み取り後クリアします。

11.1.3.7 trigger

adc_trigger_t::trigger

概要説明

デフォルトおよびグループ A のトリガー ソース。

11.1.3.8 trigger_group_b

adc_trigger_t::trigger_group_b

概要説明

グループ モードでのみ有効です。

11.1.3.9 p_callback

void(* adc_cfg_t::p_callback)(adc_callback_args_t *p_args)

概要説明

使用しない場合は NULL に設定します。

11.1.3.10 p_context

void const* adc_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。adc_api_t::adc_callback_args_t 内のユーザー コールバックに渡されます。

11.1.3.11 p_extend

void const* adc_cfg_t::p_extend

概要説明

ハードウェア固有の設定値に対応するための拡張パラメータです。

11.1.4 adc_channel_cfg_t

```
typedef struct{
    uint32_t scan_mask
    uint32_t scan_mask_group_b
    adc_group_a_t priority_group_a
    uint32_t add_mask
    uint8_t sample_hold_mask
    uint8_t sample_hold_states
} adc_channel_cfg_t
```

11.1.4.1 scan_mask

uint32_t adc_channel_cfg_t::scan_mask

詳細説明

ビット 0 は ch0、ビット 15 は ch15 です。r_adc.h の #define ADC_MASK_xxx を使用します。

11.1.4.2 scan_mask_group_b

uint32_t adc_channel_cfg_t::scan_mask_group_b

概要説明

グループ モードで有効です。r_adc.h の #define ADC_MASK_xxx を使用します。

11.1.4.3 priority_group_a

adc_group_a_t::priority_group_a

概要説明

グループ モードで有効です。

11.1.4.4 add_mask

uint32_t adc_channel_cfg_t::add_mask

概要説明

Open() で追加が有効な場合に有効です。r_adc.h の #define ADC_MASK_xxx を使用します。

11.1.4.5 sample_hold_mask

uint8_t adc_channel_cfg_t::sample_hold_mask

概要説明

チャネル / ビット 0-2。r_adc.h の #define ADC_MASK_xxx を使用します。

11.1.4.6 sample_hold_states

uint8_t adc_channel_cfg_t::sample_hold_states

概要説明

サンプル アンド ホールドに使用される状態数。チャネル 0-2 に影響を与えます。

11.1.5 adc_ctrl_t

```
typedef struct{
    uint16_t unit
    void const * p_context
} adc_ctrl_t
```

11.1.5.1 unit

uint16_t adc_ctrl_t::unit

概要説明

使用中の ADC 単位。

11.1.5.2 p_context

void const* adc_ctrl_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.6 adc_info_t

```
typedef struct{
   uint16_t * p_address
   uint32_t  length
   elc_peripheral_t elc_peripheral
   elc_event_t elc_event
} adc_info_t
```

11.1.6.1 p_address

volatile uint16_t* adc_info_t::p_address

概要説明

データの読み取りを開始するアドレス。

11.1.6.2 length

uint32_t adc_info_t::length

概要説明

読み取る合計バイト数。

11.1.6.3 elc_peripheral

elc_peripheral_t::elc_peripheral

概要説明

ELC リストの周辺機能名。

11.1.6.4 elc_event

elc_event_t::elc_event

概要説明

周辺機能の ELC イベント名。

11.1.7 adc_instance_t

```
typedef struct{
    adc_ctrl_t * p_ctrl
    adc_cfg_t const * p_cfg
    adc_channel_cfg_t const * p_channel_cfg
    adc_api_t const * p_api
} adc_instance_t
```

11.1.7.1 p_ctrl

adc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.7.2 p_cfg

adc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.7.3 p_channel_cfg

adc_channel_cfg_t::p_channel_cfg

概要説明

このインスタンスのチャネル設定構造体へのポインタ。

11.1.7.4 p_api

adc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.8 adc_sample_state_t

```
typedef struct{
    adc_sample_state_reg_t reg_id
    uint8_t num_states
} adc_sample_state_t
```

11.1.8.1 reg_id

adc_sample_state_reg_t::reg_id

概要説明

サンプル状態レジスタ ID。

11.1.8.2 num_states

uint8_t adc_sample_state_t::num_states

概要説明

変換のサンプリング状態数。Ch16-20/21 は、同じ値を使用します。

11.1.9 aes_api_t

```
typedef struct{
   uint32_t(* open)(aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)
   uint32_t(* createKey)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key)
   uint32_t(* encrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t
*p_dest)
   uint32_t(* addAdditionalAuthenticationData)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t
num_words, uint32_t *p_source)
   uint32_t(* encryptFinal)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t input_num_words, uint32_t
*p_source, uint32_t output_num_words, uint32_t *p_dest)
   uint32_t(* decrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t
*p_dest)
   uint32_t(* setGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source)
   uint32_t(* getGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest)
   uint32_t(* close)(aes_ctrl_t *const p_ctrl)
   uint32_t(* zeroPaddingEncrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t
*p_source, uint32_t *p_dest)
   uint32_t(* zeroPaddingDecrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t
*p_source, uint32_t *p_dest)
   uint32_t(* versionGet)(ssp_version_t *const p_version)
} aes_api_t
```

11.1.10 aes_cfg_t

```
typedef struct{
   crypto_api_t const * p_crypto_api
} aes_cfg_t
```

11.1.10.1 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.11 aes_ctrl_t

```
typedef struct{
   crypto_ctrl_t * p_crypto_ctrl
   crypto_api_t const * p_crypto_api
   uint32_t work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]
} aes_ctrl_t
```

11.1.11.1 p_crypto_ctrl

crypto_ctrl_t::p_crypto_ctrl

概要説明

暗号化エンジン制御構造体へのポインタ

11.1.11.2 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.11.3 work_buffer

uint32_t aes_ctrl_t::work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]

概要説明

例 : AES-GCM モードは、認証タグなどの格納にこれを使用します。

詳細説明

暗号化のコンテキスト / 状態の格納に使用します

11.1.12 aes_instance_t

```
typedef struct{
    aes_ctrl_t * p_ctrl
    aes_cfg_t const * p_cfg
    aes_api_t const * p_api
} aes_instance_t
```

11.1.12.1 p_ctrl

aes_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.12.2 p_cfg

aes_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.12.3 p_api

aes_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.13 bsp_leds_t

```
typedef struct{
    uint16_t  led_count
    ioport_port_pin_t const * p_leds
} bsp_leds_t
```

11.1.13.1 led_count

uint16_t ::led_count

概要説明

このボード上の LED の数。

11.1.13.2 p_leds

const* ::p_leds

概要説明

LED を制御するための IOPORT ピンの配列へのポインタ。

11.1.14 bsp_lock_t

```
typedef struct{
   uint8_t  lock
} bsp_lock_t
```

11.1.14.1 lock

uint8_t ::lock

概要説明

サイズは 8 ビットである必要があるので、enum の代わりに uint8_t が使用されます。

11.1.15 cac_api_t

```
typedef struct{
   ssp_err_t(*  open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
   ssp_err_t(*  read)(cac_ctrl_t *const p_ctrl, uint8_t *const p_status, uint16_t *const p_counter)
   ssp_err_t(*  close)(cac_ctrl_t *const p_ctrl)
   ssp_err_t(*  stopMeasurement)(cac_ctrl_t *const p_ctrl)
   ssp_err_t(*  startMeasurement)(cac_ctrl_t *const p_ctrl)
   ssp_err_t(*  reset)(cac_ctrl_t *const p_ctrl)
   ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} cac_api_t
```

11.1.16 cac_callback_args_t

```
typedef struct{
   cac_event_t  event
   void const *  p_context
} cac_callback_args_t
```

11.1.16.1 event

cac_event_t::event

概要説明

このイベントを使用して、コールバックの原因 （cac レディまたはエラー）を特定できます。

11.1.16.2 p_context

void const* cac_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。cac_cfg_t の open 関数で設定されます。

11.1.17 cac_cfg_t

```
typedef struct{
   cac_ref_clock_config_t cac_ref_clock
   cac_meas_clock_config_t cac_meas_clock
   uint16_t cac_upper_limit
   uint16_t cac_lower_limit
   bool mei_interrupt_enabled
   bool ovf_interrupt_enabled
   bool ferr_interrupt_enabled
   bool continuous_mode
   void(* p_callback)(cac_callback_args_t *p_args)
   void const * p_extend
   void const * p_context
} cac_cfg_t
```

11.1.17.1 cac_ref_clock

cac_ref_clock_config_t::cac_ref_clock

概要説明

参照クロック固有の設定

11.1.17.2 cac_meas_clock

cac_meas_clock_config_t::cac_meas_clock

概要説明

測定クロック固有の設定

11.1.17.3 cac_upper_limit

uint16_t cac_cfg_t::cac_upper_limit

概要説明

上限カウンタしきい値

11.1.17.4 cac_lower_limit

uint16_t cac_cfg_t::cac_lower_limit

概要説明

下限カウンタしきい値

11.1.17.5 mei_interrupt_enabled

bool cac_cfg_t::mei_interrupt_enabled

概要説明

Measurement Complete 割り込みが有効な場合は True。

11.1.17.6 ovf_interrupt_enabled

bool cac_cfg_t::ovf_interrupt_enabled

概要説明

Overflow 割り込みが有効な場合は True。

11.1.17.7 ferr_interrupt_enabled

bool cac_cfg_t::ferr_interrupt_enabled

概要説明

Frequency Error 割り込みが有効な場合は True。

11.1.17.8 continuous_mode

bool cac_cfg_t::continuous_mode

概要説明

測定が完了した後、連続的に再開始する場合は True。

11.1.17.9 p_callback

void(* cac_cfg_t::p_callback)(cac_callback_args_t *p_args)

概要説明

CAC 割り込み ISR の発生時に提供されるコールバック。

11.1.17.10 p_extend

void const* cac_cfg_t::p_extend

概要説明

CAC ハードウェアに依存する設定 */。

11.1.17.11 p_context

void const* cac_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。cac_callback_args_t 内のユーザー コールバックに渡されます。

11.1.18 cac_ctrl_t

```
typedef struct{
    void const * p_extend
} cac_ctrl_t
```

11.1.18.1 p_extend

void const* cac_ctrl_t::p_extend

11.1.19 cac_instance_t

```
typedef struct{
    cac_ctrl_t * p_ctrl
    cac_cfg_t const * p_cfg
    cac_api_t const * p_api
} cac_instance_t
```

11.1.19.1 p_ctrl

cac_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.19.2 p_cfg

cac_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.19.3 p_api

cac_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.20 cac_meas_clock_config_t

```
typedef struct{
    cac_meas_divider_t divider
    cac_clock_source_t clock
} cac_meas_clock_config_t
```

11.1.20.1 divider

cac_meas_divider_t::divider

概要説明

測定クロック用の除算値の指定。

11.1.20.2 clock

cac_clock_source_t::clock

概要説明

測定クロック用のクロック ソース。

11.1.21 cac_ref_clock_config_t

```
typedef struct{
    cac_ref_divider_t divider
    cac_clock_source_t clock
    cac_ref_digfilter_t digfilter
    cac_ref_edge_t edge
} cac_ref_clock_config_t
```

11.1.21.1 divider

cac_ref_divider_t::divider

概要説明

参照クロック用の除算値の指定。

11.1.21.2 clock

cac_clock_source_t::clock

概要説明

参照クロック用のクロック ソース。

11.1.21.3 digfilter

cac_ref_digfilter_t::digfilter

概要説明

CACREF 拡張クロック用のデジタル フィルタ選択。

11.1.21.4 edge

cac_ref_edge_t::edge

概要説明

参照クロック用のエッジ検出。

11.1.22 can_api_t

```
typedef struct{
    ssp_err_t(*  open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
    ssp_err_t(*  write)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
    ssp_err_t(*  close)(can_ctrl_t *const p_ctrl)
    ssp_err_t(*  control)(can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data)
    ssp_err_t(*  infoGet)(can_ctrl_t *const p_ctrl, can_info_t *const p_info)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} can_api_t
```

11.1.23 can_bit_timing_cfg_t

```
typedef struct{
   uint32_t baud_rate_prescaler
   can_time_segment1_t time_segment_1
   can_time_segment2_t time_segment_2
   can_sync_jump_width_t synchronization_jump_width
} can_bit_timing_cfg_t
```

11.1.23.1 baud_rate_prescaler

uint32_t can_bit_timing_cfg_t::baud_rate_prescaler

概要説明

CAN チャネルのビット レート。

11.1.23.2 time_segment_1

can_time_segment1_t::time_segment_1

概要説明

タイム セグメント 1 コントローラ。

11.1.23.3 time_segment_2

can_time_segment2_t::time_segment_2

概要説明

タイム セグメント 2 コントローラ。

11.1.23.4 synchronization_jump_width

can_sync_jump_width_t::synchronization_jump_width

概要説明

同期ジャンプ幅。

11.1.24 can_callback_args_t

```
typedef struct{
   uint32_t channel
   can_event_t event
   uint32_t mailbox
   void const * p_context
} can_callback_args_t
```

11.1.24.1 channel

uint32_t can_callback_args_t::channel

概要説明

デバイス チャネル番号。

11.1.24.2 event

can_event_t::event

概要説明

イベント コード。

11.1.24.3 mailbox

uint32_t can_callback_args_t::mailbox

概要説明

割り込みソースのメールボックス番号。

11.1.24.4 p_context

void const* can_callback_args_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.25 can_cfg_t

```
typedef struct{
    uint32_t channel
    can_bit_timing_cfg_t * p_bit_timing
    can_id_mode_t id_mode
    uint32_t mailbox_count
    can_mailbox_t * p_mailbox
    can_message_mode_t message_mode
    void(* p_callback)(can_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} can_cfg_t
```

11.1.25.1 channel

uint32_t can_cfg_t::channel

概要説明

CAN チャネル。

11.1.25.2 p_bit_timing

can_bit_timing_cfg_t::p_bit_timing

概要説明

CAN ビット タイミング。

11.1.25.3 id_mode

can_id_mode_t::id_mode

概要説明

標準または拡張 ID モード。

11.1.25.4 mailbox_count

uint32_t can_cfg_t::mailbox_count

概要説明

メールボックスの数。

11.1.25.5 p_mailbox

can_mailbox_t::p_mailbox

概要説明

メールボックスへのポインタ。

11.1.25.6 message_mode

can_message_mode_t::message_mode

概要説明

上書きメッセージまたはオーバーラン。

11.1.25.7 p_callback

void(* can_cfg_t::p_callback)(can_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.25.8 p_context

void const* can_cfg_t::p_context

概要説明

ユーザー定義のコールバック コンテキスト。

11.1.25.9 p_extend

void const* can_cfg_t::p_extend

概要説明

CAN ハードウェアに依存する設定。

11.1.26 can_ctrl_t

```
typedef struct{
    void * p_instance_ctrl
} can_ctrl_t
```

11.1.26.1 p_instance_ctrl

void* can_ctrl_t::p_instance_ctrl

概要説明

インスタンス制御情報へのポインタ。

11.1.27 can_error_t

```
typedef struct{
   uint32_t error
   struct can_error_t::st_error_b error_b
} can_error_t
```

11.1.27.1 error

uint32_t can_error_t::error

11.1.27.2 error_b

struct can_error_t::st_error_b can_error_t::error_b

11.1.28 can_extended_cfg_t

```
typedef struct{
   can_clock_source_t clock_source
   uint32_t * p_mailbox_mask
} can_extended_cfg_t
```

11.1.28.1 clock_source

can_clock_source_t::clock_source

概要説明

CAN クロックのソース。

11.1.28.2 p_mailbox_mask

uint32_t* ::p_mailbox_mask

概要説明

メールボックス マスク、4 つすべてのメールボックスのうちの 1 つ。

11.1.29 can_frame_t

```
typedef struct{
    can_id_t id
    uint8_t data_length_code
    uint8_t data[8]
    can_frame_type_t type
} can_frame_t
```

11.1.29.1 id

can_id_t::id

概要説明

CAN ID。

11.1.29.2 data_length_code

uint8_t can_frame_t::data_length_code

概要説明

CAN データ長コード、メッセージ内のバイト数。

11.1.29.3 data

uint8_t can_frame_t::data[8]

概要説明

CAN データ、最大 8 バイト。

11.1.29.4 type

can_frame_type_t::type

概要説明

フレーム タイプ、データまたはリモート フレーム。

11.1.30 can_info_t

```
typedef struct{
    can_mode_t operation_mode
    can_status_t status
    uint32_t bit_rate
    uint8_t error_count_transmit
    uint8_t error_count_receive
    can_error_t error_code
} can_info_t
```

11.1.30.1 operation_mode

can_mode_t::operation_mode

概要説明

CAN 動作モード。

11.1.30.2 status

can_status_t::status

概要説明

CAN のステータス。

11.1.30.3 bit_rate

uint32_t can_info_t::bit_rate

概要説明

CAN ビット レート。

11.1.30.4 error_count_transmit

uint8_t can_info_t::error_count_transmit

概要説明

送信エラー カウント。

11.1.30.5 error_count_receive

uint8_t can_info_t::error_count_receive

概要説明

受信エラー カウント。

11.1.30.6 error_code

can_error_t::error_code

概要説明

エラー コード、リード後削除されます。

11.1.31 can_instance_t

```
typedef struct{
    can_ctrl_t * p_ctrl
    can_cfg_t const * p_cfg
    can_api_t const * p_api
} can_instance_t
```

11.1.31.1 p_ctrl

can_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.31.2 p_cfg

can_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.31.3 p_api

can_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.32 can_mailbox_t

```
typedef struct{
   can_id_t mailbox_id
   can_mailbox_send_receive_t mailbox_type
   can_frame_type_t frame_type
} can_mailbox_t
```

11.1.32.1 mailbox_id

can_id_t::mailbox_id

概要説明

メールボックス ID。

11.1.32.2 mailbox_type

can_mailbox_send_receive_t::mailbox_type

概要説明

受信または送信メールボックスのタイプ。

11.1.32.3 frame_type

can_frame_type_t::frame_type

概要説明

受信メールボックスのフレーム タイプ。

11.1.33 can_status_t

```
typedef struct{
   uint32_t status
   struct can_status_t::st_status_b status_b
} can_status_t
```

11.1.33.1 status

uint32_t can_status_t::status

11.1.33.2 status_b

struct can_status_t::st_status_b can_status_t::status_b

11.1.34 cgc_api_t

```
typedef struct{
   ssp_err_t(* init)(void)
   ssp_err_t(* clockStart)(cgc_clock_t clock_source, cgc_clock_cfg_t *p_clock_cfg)
   ssp_err_t(* clockStop)(cgc_clock_t clock_source)
   ssp_err_t(* systemClockSet)(cgc_clock_t clock_source, cgc_system_clock_cfg_t *p_clock_cfg)
   ssp_err_t(* systemClockGet)(cgc_clock_t *p_clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg)
   ssp_err_t(* systemClockFreqGet)(cgc_system_clocks_t clock, uint32_t *p_freq_hz)
   ssp_err_t(* clockCheck)(cgc_clock_t clock_source)
   ssp_err_t(* oscStopDetect)(void(*p_callback)(cgc_callback_args_t *p_args), bool enable)
   ssp_err_t(* oscStopStatusClear)(void)
   ssp_err_t(* busClockOutCfg)(cgc_bclockout_dividers_t divider)
   ssp_err_t(* busClockOutEnable)(void)
   ssp_err_t(* busClockOutDisable)(void)
   ssp_err_t(* clockOutCfg)(cgc_clock_t clock, cgc_clockout_dividers_t divider)
   ssp_err_t(* clockOutEnable)(void)
   ssp_err_t(* clockOutDisable)(void)
   ssp_err_t(* lcdClockCfg)(cgc_clock_t clock)
   ssp_err_t(* lcdClockEnable)(void)
   ssp_err_t(* lcdClockDisable)(void)
   ssp_err_t(* sdramClockOutEnable)(void)
   ssp_err_t(* sdramClockOutDisable)(void)
   ssp_err_t(* usbClockCfg)(cgc_usb_clock_div_t divider)
   ssp_err_t(* systickUpdate)(uint32_t period_count, cgc_systick_period_units_t units)
   ssp_err_t(* versionGet)(ssp_version_t *p_version)
} cgc_api_t
```

11.1.35 cgc_callback_args_t

```
typedef struct{
   cgc_event_t  event
   void const *  p_context
} cgc_callback_args_t
```

11.1.35.1 event

cgc_event_t::event

概要説明

このイベントは、コールバックの原因を特定するために使用できます。

11.1.35.2 p_context

void const* cgc_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.36 cgc_clock_cfg_t

```
typedef struct{
   cgc_clock_t source_clock
   cgc_pll_div_t divider
   float multiplier
} cgc_clock_cfg_t
```

11.1.36.1 source_clock

cgc_clock_t::source_clock

概要説明

PLL ソース クロック （S7G2 のみ）。

11.1.36.2 divider

cgc_pll_div_t::divider

概要説明

PLL 除算値。

11.1.36.3 multiplier

float cgc_clock_cfg_t::multiplier

概要説明

PLL 乗算値。

11.1.37 cgc_instance_t

```
typedef struct{
   cgc_clock_cfg_t const * p_cfg
   cgc_api_t const * p_api
} cgc_instance_t
```

11.1.37.1 p_cfg

cgc_clock_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.37.2 p_api

cgc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.38 cgc_system_clock_cfg_t

```
typedef struct{
    cgc_sys_clock_div_t pclka_div
    cgc_sys_clock_div_t pclkb_div
    cgc_sys_clock_div_t pclkc_div
    cgc_sys_clock_div_t pclkd_div
    cgc_sys_clock_div_t bclk_div
    cgc_sys_clock_div_t fclk_div
    cgc_sys_clock_div_t iclk_div
} cgc_system_clock_cfg_t
```

11.1.38.1 pclka_div

cgc_sys_clock_div_t::pclka_div

概要説明

PCLKA の除算値。

11.1.38.2 pclkb_div

cgc_sys_clock_div_t::pclkb_div

概要説明

PCLKB の除算値。

11.1.38.3 pclkc_div

cgc_sys_clock_div_t::pclkc_div

概要説明

PCLKC の除算値。

11.1.38.4 pclkd_div

cgc_sys_clock_div_t::pclkd_div

概要説明

PCLKD の除算値。

11.1.38.5 bclk_div

cgc_sys_clock_div_t::bclk_div

概要説明

BCLK の除算値。

11.1.38.6 fclk_div

cgc_sys_clock_div_t::fclk_div

概要説明

FCLK の除算値。

11.1.38.7 iclk_div

cgc_sys_clock_div_t::iclk_div

概要説明

ICLK の除算値。

11.1.39 crc_api_t

```
typedef struct{
    ssp_err_t(* open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
    ssp_err_t(* close)(crc_ctrl_t *const p_ctrl)
    ssp_err_t(* crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
    ssp_err_t(* snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
    ssp_err_t(* snoopDisable)(crc_ctrl_t *const p_ctrl)
    ssp_err_t(* snoopCfg)(crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const p_snoop_cfg)
    ssp_err_t(* calculate)(crc_ctrl_t *const p_ctrl, void *p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)
    ssp_err_t(* versionGet)(ssp_version_t *version)
} crc_api_t
```

11.1.40 crc_cfg_t

```
typedef struct{
   crc_polynomial_t polynomial
   crc_bit_order_t bit_order
   void const * p_extend
} crc_cfg_t
```

11.1.40.1 polynomial

crc_polynomial_t::polynomial

概要説明

CRC 生成多項式切り替え（GPS）

11.1.40.2 bit_order

crc_bit_order_t::bit_order

概要説明

CRC 演算切り替え （LMS）

11.1.40.3 p_extend

void const* crc_cfg_t::p_extend

概要説明

CRC のハードウェアに依存する設定。

11.1.41 crc_ctrl_t

```
typedef struct{
   crc_polynomial_t polynomial
   crc_bit_order_t bit_order
} crc_ctrl_t
```

11.1.41.1 polynomial

crc_polynomial_t::polynomial

概要説明

CRC 生成多項式切り替え （GPS）

11.1.41.2 bit_order

crc_bit_order_t::bit_order

概要説明

CRC 演算切り替え （LMS）

11.1.42 crc_instance_t

```
typedef struct{
    crc_ctrl_t * p_ctrl
    crc_cfg_t const * p_cfg
    crc_api_t const * p_api
} crc_instance_t
```

11.1.42.1 p_ctrl

crc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.42.2 p_cfg

crc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.42.3 p_api

crc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.43 crc_snoop_cfg_t

```
typedef struct{
    uint32_t snoop_channel
    crc_snoop_direction_t snoop_direction
} crc_snoop_cfg_t
```

11.1.43.1 snoop_channel

uint32_t crc_snoop_cfg_t::snoop_channel

概要説明

レジスタ スヌープ アドレス （CRCSA）

11.1.43.2 snoop_direction

crc_snoop_direction_t::snoop_direction

概要説明

スヌープオンライト / リード切り替え （CRCSWR）

11.1.44 crypto_api_t

```
typedef struct{
   uint32_t(*  open)(crypto_ctrl_t *const p_ctrl, crypto_cfg_t const *const p_cfg)
   uint32_t(*  close)(crypto_ctrl_t *const p_ctrl)
   uint32_t(*  statusGet)(uint32_t *p_status)
   uint32_t(*  versionGet)(ssp_version_t *const p_version)
} crypto_api_t
```

11.1.45 crypto_cfg_t

```
typedef struct{
   void(*  p_sce_long_plg_start_callback)(void)
   void(*  p_sce_long_plg_end_callback)(void)
} crypto_cfg_t
```

11.1.45.1 p_sce_long_plg_start_callback

void(* crypto_cfg_t::p_sce_long_plg_start_callback)(void)

11.1.45.2 p_sce_long_plg_end_callback

void(* crypto_cfg_t::p_sce_long_plg_end_callback)(void)

詳細説明

ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

11.1.46 crypto_ctrl_t

```
typedef struct{
   uint32_t state
   uint32_t cb_data
   void(* p_sce_long_plg_start_callback)(void)
   void(* p_sce_long_plg_end_callback)(void)
} crypto_ctrl_t
```

11.1.46.1 state

uint32_t crypto_ctrl_t::state

概要説明

SCE/SCE-Lite ドライバの状態 （初期化済みかどうか）を示します

11.1.46.2 cb_data

uint32_t crypto_ctrl_t::cb_data

11.1.46.3 p_sce_long_plg_start_callback

void(* crypto_ctrl_t::p_sce_long_plg_start_callback)(void)

11.1.46.4 p_sce_long_plg_end_callback

void(* crypto_ctrl_t::p_sce_long_plg_end_callback)(void)

11.1.47 crypto_instance_t

```
typedef struct{
   crypto_ctrl_t * p_ctrl
   crypto_cfg_t const * p_cfg
   crypto_api_t const * p_api
} crypto_instance_t
```

11.1.47.1 p_ctrl

crypto_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.47.2 p_cfg

crypto_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.47.3 p_api

crypto_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.48 ctsu_api_t

```
typedef struct{
    ssp_err_t(* open)(ctsu_ctrl_t *p_ctrl, ctsu_cfg_t *p_cfg)
    ssp_err_t(* close)(ctsu_ctrl_t *p_ctrl, ctsu_close_option_t opts)
    ssp_err_t(* scan)(ctsu_ctrl_t *p_ctrl)
    ssp_err_t(* update)(ctsu_ctrl_t *p_ctrl)
    ssp_err_t(* read)(ctsu_ctrl_t *p_ctrl, void *p_dest, ctsu_read_t opts, const ctsu_channel_pair_t *channels, const uint16_t count)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} ctsu_api_t
```

11.1.49 ctsu_callback_args_t

```
typedef struct{
    ctsu_event_t event
    void const * p_context
} ctsu_callback_args_t
```

11.1.49.1 event

ctsu_event_t::event

概要説明

CTSU コールバック イベント。

11.1.49.2 p_context

void const* ctsu_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.50 ctsu_cfg_t

```
typedef struct{
    transfer_instance_t const *const  p_lower_lvl_transfer_read
    transfer_instance_t const *const  p_lower_lvl_transfer_write
    ctsu_hw_cfg_t *  p_ctsu_hw_cfg
    ctsu_functions_t *  p_ctsu_functions
    void(*  p_callback)(ctsu_callback_args_t *p_args)
    void *  p_context
    ctsu_process_option_t  ctsu_soft_option
    ctsu_close_option_t  ctsu_close_option
} ctsu_cfg_t
```

11.1.50.1 p_lower_lvl_transfer_read

transfer_instance_t::p_lower_lvl_transfer_read

概要説明

リード結果への転送インスタンスへのポインタ。

11.1.50.2 p_lower_lvl_transfer_write

transfer_instance_t::p_lower_lvl_transfer_write

概要説明

write cfg への転送インスタンスへのポインタ。

11.1.50.3 p_ctsu_hw_cfg

ctsu_hw_cfg_t::p_ctsu_hw_cfg

概要説明

CTSU 構成へのポインタ。

11.1.50.4 p_ctsu_functions

ctsu_functions_t::p_ctsu_functions

概要説明

カスタム データ関数のプレース ホルダーへのポインタ。

11.1.50.5 p_callback

void(* ctsu_cfg_t::p_callback)(ctsu_callback_args_t *p_args)

概要説明

スキャン完了時に使用する関数へのコールバック。

11.1.50.6 p_context

void* ctsu_cfg_t::p_context

概要説明

update_complete 通知に渡すデータへのポインタ。

11.1.50.7 ctsu_soft_option

ctsu_process_option_t::ctsu_soft_option

概要説明

Open および Process の実行時に使用するソフトウェア オプション。

11.1.50.8 ctsu_close_option

ctsu_close_option_t::ctsu_close_option

概要説明

タッチ動作終了時に使用するソフトウェア オプション。

11.1.51 ctsu_channel_data_mutual_t

```
typedef struct{
    uint16_t sen_cnt_1
    uint16_t ref_cnt_1
    uint16_t sen_cnt_2
    uint16_t ref_cnt_2
} ctsu_channel_data_mutual_t
```

11.1.51.1 sen_cnt_1

uint16_t ctsu_channel_data_mutual_t::sen_cnt_1

概要説明

未加工センサー カウントのプライマリ リード。

11.1.51.2 ref_cnt_1

uint16_t ctsu_channel_data_mutual_t::ref_cnt_1

概要説明

未加工参照 ICO カウントのプライマリ リード。

11.1.51.3 sen_cnt_2

uint16_t ctsu_channel_data_mutual_t::sen_cnt_2

概要説明

未加工センサー ICO カウントのセカンダリ リード。

11.1.51.4 ref_cnt_2

uint16_t ctsu_channel_data_mutual_t::ref_cnt_2

概要説明

未加工参照 ICO カウントのセカンダリ リード。

11.1.52 ctsu_channel_data_self_t

```
typedef struct{
    uint16_t sensor_count
    uint16_t reference_count
} ctsu_channel_data_self_t
```

11.1.52.1 sensor_count

uint16_t ctsu_channel_data_self_t::sensor_count

概要説明

未加工センサー カウント。

11.1.52.2 reference_count

uint16_t ctsu_channel_data_self_t::reference_count

概要説明

未加工参照カウント。

11.1.53 ctsu_channel_pair_t

```
typedef struct{
   int8_t rx
   int8_t tx
} ctsu_channel_pair_t
```

11.1.53.1 rx

int8_t ctsu_channel_pair_t::rx

概要説明

プライマリ チャネルを示します。

11.1.53.2 tx

int8_t ctsu_channel_pair_t::tx

概要説明

セカンダリ チャネルを示します（相互容量モードでのみ使用されます）

11.1.54 ctsu_channel_setting_t

```
typedef struct{
   uint16_t ctsussc
   uint16_t ctsuso0
   uint16_t ctsuso1
} ctsu_channel_setting_t
```

11.1.54.1 ctsussc

volatile uint16_t ctsu_channel_setting_t::ctsussc

概要説明

CTSUSSC レジスタ用の値を保持します。

11.1.54.2 ctsuso0

volatile uint16_t ctsu_channel_setting_t::ctsuso0

概要説明

CTSUSO0 レジスタ用の値を保持します。

11.1.54.3 ctsuso1

volatile uint16_t ctsu_channel_setting_t::ctsuso1

概要説明

CTSUSO1 レジスタ用の値を保持します。

11.1.55 ctsu_ctrl_t

```
typedef struct{
    transfer_api_t const * p_api_transfer
    transfer_ctrl_t * p_lowerl_lvl_transfer_read_ctrl
    transfer_ctrl_t * p_lowerl_lvl_transfer_write_ctrl
    bool ctsu_opened
    uint8_t ctsu_unit
    ctsu_hw_cfg_t * p_ctsu_hw_cfg
    ctsu_process_option_t ctsu_open_option
    ctsu_process_option_t ctsu_update_option
    ctsu_close_option_t ctsu_close_option
    ctsu_action_t ctsu_process_state
    void(* p_callback)(ctsu_callback_args_t *p_args)
    void * p_context
} ctsu_ctrl_t
```

11.1.55.1 p_api_transfer

transfer_api_t::p_api_transfer

概要説明

ローレベル転送ドライバ関数ポインタへのポインタ。

11.1.55.2 p_lowerl_lvl_transfer_read_ctrl

transfer_ctrl_t::p_lowerl_lvl_transfer_read_ctrl

概要説明

転送リード制御へのポインタ。

11.1.55.3 p_lowerl_lvl_transfer_write_ctrl

transfer_ctrl_t::p_lowerl_lvl_transfer_write_ctrl

概要説明

転送書き込み制御へのポインタ。

11.1.55.4 ctsu_opened

bool ctsu_ctrl_t::ctsu_opened

概要説明

初期化状態を格納します。

11.1.55.5 ctsu_unit

uint8_t ctsu_ctrl_t::ctsu_unit

概要説明

使用中の CTSU 単位。

11.1.55.6 p_ctsu_hw_cfg

ctsu_hw_cfg_t::p_ctsu_hw_cfg

概要説明

CTSU 構成へのポインタ。

11.1.55.7 ctsu_open_option

ctsu_process_option_t::ctsu_open_option

概要説明

Open および Process の実行時に使用するソフトウェア オプション。

11.1.55.8 ctsu_update_option

ctsu_process_option_t::ctsu_update_option

概要説明

パラメータ更新プロセスの実行時に使用するソフトウェア オプション。

11.1.55.9 ctsu_close_option

ctsu_close_option_t::ctsu_close_option

概要説明

タッチ動作終了時に使用するソフトウェア オプション。

11.1.55.10 ctsu_process_state

ctsu_action_t::ctsu_process_state

概要説明

CTSU 処理状態のマシン動作を観察するための変数。

11.1.55.11 p_callback

void(* ctsu_ctrl_t::p_callback)(ctsu_callback_args_t *p_args)

概要説明

従属パラメータの更新完了時に使用する関数へのコールバック。

11.1.55.12 p_context

void* ctsu_ctrl_t::p_context

概要説明

update_complete 通知に渡すデータへのポインタ。

11.1.56 ctsu_functions_t

```
typedef struct{
    int32_t(*  preFilter)(void *p_args)
    int32_t(*  filter)(volatile uint16_t *output, volatile uint16_t *input)
    int32_t(*  postFilter)(void *p_args)
    int32_t(*  ctsuDecode)(void *p_args)
    int32_t(*  otDriftComp)(void *p_args)
    int32_t(*  rtDriftComp)(void *p_args)
    int32_t(*  otAutoTune)(void *p_args)
    int32_t(*  rtAutoTune)(void *p_args)
} ctsu_functions_t
```

11.1.56.1 preFilter

int32_t(* ctsu_functions_t::preFilter)(void *p_args)

概要説明

データのフィルタリング前に、未加工データ SNR を計算するために使用されます。

11.1.56.2 filter

int32_t(* ctsu_functions_t::filter)(volatile uint16_t *output, volatile uint16_t *input)

概要説明

CTSU_CFG_FILTER_DEPTH を使用した加重平均。

11.1.56.3 postFilter

int32_t(* ctsu_functions_t::postFilter)(void *p_args)

概要説明

フィルタリング結果の処理。

11.1.56.4 ctsuDecode

int32_t(* ctsu_functions_t::ctsuDecode)(void *p_args)

概要説明

チャネルがタッチされたかどうかを判定するアルゴリズム。

11.1.56.5 otDriftComp

int32_t(* ctsu_functions_t::otDriftComp)(void *p_args)

概要説明

初期化時にベースライン、エンベロープ、およびしきい値への操作を行うアルゴリズム。

11.1.56.6 rtDriftComp

int32_t(* ctsu_functions_t::rtDriftComp)(void *p_args)

概要説明

実行時にベースライン、エンベロープ、およびしきい値への操作を行うアルゴリズム。

11.1.56.7 otAutoTune

int32_t(* ctsu_functions_t::otAutoTune)(void *p_args)

概要説明

システム初期化時に一度呼び出される関数。

11.1.56.8 rtAutoTune

int32_t(* ctsu_functions_t::rtAutoTune)(void *p_args)

概要説明

システムの実行中にセンサーの自動調整のために呼び出される関数。

11.1.57 ctsu_hw_cfg_t

```
typedef struct{
  R_CTSU_Type ctsu_settings
  ctsu_channel_setting_t * write_settings
  uint16_t * threshold
  uint16_t * hysteresis
  uint16_t * baseline
  void * raw_result
  void * filter_output
  void * binary_result
  ctsu_channel_pair_t * excluded
  int8_t num_excluded
  const uint16_t * series_resistance
} ctsu_hw_cfg_t
```

11.1.57.1 ctsu_settings

R_CTSU_Type ctsu_hw_cfg_t::ctsu_settings

概要説明

CR0、CR1、SDPRS、SST、CHACn、CHTRCn、DCLKC に対するユーザー定義の SFR 設定。

11.1.57.2 write_settings

ctsu_channel_setting_t::write_settings

概要説明

各アクティブ チャネルのしきい値に対するユーザー定義の初期設定。しきい値は、センサー カウントの実行時ベースラインとフィルタリングされた出力の差です。

詳細説明

各アクティブ チャネルの SSC、SO0、SO1 に対するユーザー定義の設定。

11.1.57.3 threshold

uint16_t* ctsu_hw_cfg_t::threshold

11.1.57.4 hysteresis

uint16_t* ctsu_hw_cfg_t::hysteresis

概要説明

カウント値の許容誤差に対するユーザー定義の設定。

11.1.57.5 baseline

uint16_t* ctsu_hw_cfg_t::baseline

概要説明

タッチしていない場合に、各アクティブ チャネルに対して想定されるカウントのベースライン。

11.1.57.6 raw_result

void* ctsu_hw_cfg_t::raw_result

概要説明

CTSU 測定の未加工の結果を保持するバッファへのポインタ。

11.1.57.7 filter_output

void* ctsu_hw_cfg_t::filter_output

概要説明

未加工の結果をフィルタリングした後の出力を保持するバッファへのポインタ。

11.1.57.8 binary_result

void* ctsu_hw_cfg_t::binary_result

概要説明

バイナリ データを保存できる場所へのポインタ。

11.1.57.9 excluded

ctsu_channel_pair_t::excluded

概要説明

無視する必要があるチャネル ペアを rx、tx の順に昇順に並べたリストが含まれている配列へのポインタ。

11.1.57.10 num_excluded

int8_t ctsu_hw_cfg_t::num_excluded

概要説明

除外された配列内の要素数。

11.1.57.11 series_resistance

const uint16_t* ctsu_hw_cfg_t::series_resistance

概要説明

チャネルの抵抗 （調整時に RC 定数の判定に使用されます）

11.1.58 ctsu_instance_t

```
typedef struct{
    ctsu_ctrl_t *  p_ctrl
    ctsu_cfg_t *  p_cfg
    ctsu_api_t const *  p_api
} ctsu_instance_t
```

11.1.58.1 p_ctrl

ctsu_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.58.2 p_cfg

ctsu_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.58.3 p_api

ctsu_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.59 dac_api_t

```
typedef struct{
    ssp_err_t(*  open)(dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(dac_ctrl_t *p_ctrl)
    ssp_err_t(*  write)(dac_ctrl_t *p_ctrl, dac_size_t value)
    ssp_err_t(*  start)(dac_ctrl_t *p_ctrl)
    ssp_err_t(*  stop)(dac_ctrl_t *p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} dac_api_t
```

11.1.60 dac_cfg_t

```
typedef struct{
    uint8_t  channel
    bool  ad_da_synchronized
    dac_data_format_t  data_format
    bool  output_amplifier_enabled
    void const *  p_extend
} dac_cfg_t
```

11.1.60.1 channel

uint8_t dac_cfg_t::channel

概要説明

この DAC チャネルに関連付けられた ID。

11.1.60.2 ad_da_synchronized

bool dac_cfg_t::ad_da_synchronized

概要説明

AD/DA 同期。

11.1.60.3 data_format

dac_data_format_t::data_format

概要説明

データ形式。

11.1.60.4 output_amplifier_enabled

bool dac_cfg_t::output_amplifier_enabled

概要説明

出力増幅器有効化。

11.1.60.5 p_extend

void const* dac_cfg_t::p_extend

11.1.61 dac_ctrl_t

```
typedef struct{
    uint8_t channel
    uint8_t channel_started
    uint8_t channel_opened
    uint8_t reserved1
} dac_ctrl_t
```

11.1.61.1 channel

uint8_t dac_ctrl_t::channel

概要説明

この DAC チャネルに関連付けられた ID。

11.1.61.2 channel_started

uint8_t dac_ctrl_t::channel_started

概要説明

開始されたチャネル上の DAC 動作。

11.1.61.3 channel_opened

uint8_t dac_ctrl_t::channel_opened

概要説明

開いている DAC チャネル。

11.1.61.4 reserved1

uint8_t dac_ctrl_t::reserved1

11.1.62 dac_instance_t

```
typedef struct{
    dac_ctrl_t *  p_ctrl
    dac_cfg_t const *  p_cfg
    dac_api_t const *  p_api
} dac_instance_t
```

11.1.62.1 p_ctrl

dac_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.62.2 p_cfg

dac_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.62.3 p_api

dac_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.63 display_api_t

```
typedef struct{
    ssp_err_t(*  open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(display_ctrl_t *const p_ctrl)
    ssp_err_t(*  start)(display_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(display_ctrl_t *const p_ctrl)
    ssp_err_t(*  layerChange)(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
    ssp_err_t(*  correction)(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)
    ssp_err_t(*  clut)(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)
    ssp_err_t(*  statusGet)(display_ctrl_t const *const p_ctrl, display_status_t *const p_status)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} display_api_t
```

11.1.64 display_brightness_t

```
typedef struct{
    bool  enable
    uint16_t  r
    uint16_t  g
    uint16_t  b
} display_brightness_t
```

11.1.64.1 enable

bool display_brightness_t::enable

概要説明

明るさ補正オン / オフ。

11.1.64.2 r

uint16_t display_brightness_t::r

概要説明

R チャネルの明るさ （DC）調整。

11.1.64.3 g

uint16_t display_brightness_t::g

概要説明

G チャネルの明るさ （DC）調整。

11.1.64.4 b

uint16_t display_brightness_t::b

概要説明

B チャネルの明るさ （DC）調整。

11.1.65 display_callback_args_t

```
typedef struct{
   display_event_t event
   void const * p_context
} display_callback_args_t
```

11.1.65.1 event

display_event_t::event

概要説明

イベント コード。

11.1.65.2 p_context

void const* display_callback_args_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.66 display_cfg_t

```
typedef struct{
   display_input_cfg_t input[DISPLAY_FRAME_LAYER_2+1]
   display_output_cfg_t output
   display_layer_t layer[DISPLAY_FRAME_LAYER_2+1]
   void(* p_callback)(display_callback_args_t *p_args)
   void const * p_context
   void const * p_extend
} display_cfg_t
```

11.1.66.1 input

display_input_cfg_t::input

概要説明

グラフィックス入力フレーム設定値。

詳細説明

表示デバイス用の汎用設定

11.1.66.2 output

`display_output_cfg_t`::output

概要説明

グラフィックス出力フレーム設定値。

11.1.66.3 layer

`display_layer_t`::layer

概要説明

グラフィクス レイヤーのブレンド設定値。

11.1.66.4 p_callback

void(* `display_cfg_t`::p_callback)(`display_callback_args_t` *p_args)

概要説明

コールバック関数へのポインタ。

詳細説明

表示イベント処理用の設定

11.1.66.5 p_context

void const* `display_cfg_t`::p_context

概要説明

コールバック関数に渡されるユーザー定義のコンテキスト。

11.1.66.6 p_extend

void const* `display_cfg_t`::p_extend

概要説明

表示ハードウェアに依存する設定。

詳細説明

表示ペリフェラルに固有の設定へのポインタ

11.1.67 display_clut_cfg_t

```
typedef struct{
   uint32_t * p_base
   uint16_t start
   uint16_t size
} display_clut_cfg_t
```

11.1.67.1 p_base

uint32_t* display_clut_cfg_t::p_base

概要説明

CLUT ソース データへのポインタ。

11.1.67.2 start

uint16_t display_clut_cfg_t::start

概要説明

更新される CLUT エントリの開始。

11.1.67.3 size

uint16_t display_clut_cfg_t::size

概要説明

更新される CLUT エントリのサイズ。

11.1.68 display_clut_t

```
typedef struct{
   uint32_t color_num
   const uint32_t * p_clut
} display_clut_t
```

11.1.68.1 color_num

uint32_t display_clut_cfg_t::color_num

概要説明

CLUT 内の色数。

11.1.68.2 p_clut

const uint32_t* display_clut_cfg_t::p_clut

概要説明

CLUT データを格納する領域のアドレス （ARGB8888 形式）

11.1.69 display_color_t

```
typedef struct{
   uint32_t argb
   uint8_t b
   uint8_t g
   uint8_t r
   uint8_t a
   struct{}          byte
   union{}           union{}
} display_color_t
```

11.1.69.1 argb

uint32_t display_color_t::argb

11.1.69.2 b

uint8_t display_color_t::b

概要説明

青

11.1.69.3 g

uint8_t display_color_t::g

概要説明

緑

11.1.69.4 r

uint8_t display_color_t::r

概要説明

赤

11.1.69.5 a

uint8_t display_color_t::a

概要説明

a

11.1.69.6 byte

このメンバーの定義については、ソース コードを参照してください。

11.1.69.7 union{}

このメンバーの定義については、ソース コードを参照してください。

11.1.70 display_contrast_t

```
typedef struct{
   bool enable
   uint8_t r
   uint8_t g
   uint8_t b
} display_contrast_t
```

11.1.70.1 enable

bool display_contrast_t::enable

概要説明

コントラスト補正オン / オフ。

11.1.70.2 r

uint8_t display_contrast_t::r

概要説明

R チャネルのコントラスト （ゲイン）調整。

11.1.70.3 g

uint8_t display_contrast_t::g

概要説明

G チャネルのコントラスト （ゲイン）調整。

11.1.70.4 b

uint8_t display_contrast_t::b

概要説明

B チャネルのコントラスト （ゲイン）調整。

11.1.71 display_coordinate_t

```
typedef struct{
    int16_t x
    int16_t y
} display_coordinate_t
```

11.1.71.1 x

int16_t display_coordinate_t::x

概要説明

X 座標、単一の値を設定できます。

11.1.71.2 y

int16_t display_coordinate_t::y

概要説明

Y 座標、単一の値を設定できます。

11.1.72 display_correction_t

```
typedef struct{
   display_brightness_t brightness
   display_contrast_t contrast
} display_correction_t
```

11.1.72.1 brightness

display_brightness_t::brightness

概要説明

明るさ。

11.1.72.2 contrast

display_contrast_t::contrast

概要説明

コントラスト。

11.1.73 display_ctrl_t

```
typedef struct{
   display_state_t state
   void(* p_callback)(display_callback_args_t *p_args)
   void const * p_context
} display_ctrl_t
```

11.1.73.1 state

display_state_t::state

概要説明

GLCD モジュールのステータス。

11.1.73.2 p_callback

void(* display_ctrl_t::p_callback)(display_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.73.3 p_context

void const* display_ctrl_t::p_context

概要説明

ハイレベルのデバイス コンテキストへのポインタ。

11.1.74 display_gamma_correction_t

```
typedef struct{
    gamma_correction_t r
    gamma_correction_t g
    gamma_correction_t b
} display_gamma_correction_t
```

11.1.74.1 r

gamma_correction_t::r

概要説明

R チャネルのガンマ補正。

11.1.74.2 g

gamma_correction_t::g

概要説明

G チャネルのガンマ補正。

11.1.74.3 b

gamma_correction_t::b

概要説明

B チャネルのガンマ補正。

11.1.75 display_input_cfg_t

```
typedef struct{
    uint32_t * p_base
    uint16_t hsize
    uint16_t vsize
    uint32_t hstride
    display_in_format_t format
    bool line_descending_enable
    bool lines_repeat_enable
    uint16_t lines_repeat_times
} display_input_cfg_t
```

11.1.75.1 p_base

uint32_t* display_input_cfg_t::p_base

概要説明

フレーム バッファへのベース アドレス。

11.1.75.2 hsize

uint16_t display_input_cfg_t::hsize

概要説明

1 ラインの水平ピクセル サイズ。

11.1.75.3 vsize

uint16_t display_input_cfg_t::vsize

概要説明

1 フレームの垂直ピクセル サイズ。

11.1.75.4 hstride

uint32_t display_input_cfg_t::hstride

概要説明

1 ラインのメモリ ストライド （バイト単位）。

11.1.75.5 format

display_in_format_t::format

概要説明

入力形式設定。

11.1.75.6 line_descending_enable

bool display_input_cfg_t::line_descending_enable

概要説明

ライン降順有効化。

11.1.75.7 lines_repeat_enable

bool display_input_cfg_t::lines_repeat_enable

概要説明

ライン繰り返し有効化。

11.1.75.8 lines_repeat_times

uint16_t display_input_cfg_t::lines_repeat_times

概要説明

期待されるライン繰り返し回数。

11.1.76 display_instance_t

```
typedef struct{
    display_ctrl_t * p_ctrl
    display_cfg_t const * p_cfg
    display_api_t const * p_api
} display_instance_t
```

11.1.76.1 p_ctrl

display_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.76.2 p_cfg

display_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.76.3 p_api

display_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.77 display_layer_t

```
typedef struct{
    display_coordinate_t coordinate
    display_color_t bg_color
    display_fade_control_t fade_control
    uint8_t fade_speed
} display_layer_t
```

11.1.77.1 coordinate

display_coordinate_t::coordinate

概要説明

ブレンド位置 （画像の開始ポイント）

11.1.77.2 bg_color

display_color_t::bg_color

概要説明

領域外の色。

11.1.77.3 fade_control

display_fade_control_t::fade_control

概要説明

レイヤー フェードイン / アウト制御オン / オフ。

11.1.77.4 fade_speed

uint8_t display_layer_t::fade_speed

概要説明

レイヤー フェードイン / アウト フレーム レート。

11.1.78 display_output_cfg_t

```
typedef struct{
   display_timing_t htiming
   display_timing_t vtiming
   display_out_format_t format
   display_endian_t endian
   display_color_order_t color_order
   display_signal_polarity_t data_enable_polarity
   display_sync_edge_t sync_edge
   display_color_t bg_color
   display_brightness_t brightness
   display_contrast_t contrast
   display_gamma_correction_t * p_gamma_correction
   bool dithering_on
} display_output_cfg_t
```

11.1.78.1 htiming

display_timing_t::htiming

概要説明

水平表示のサイクル設定値。

11.1.78.2 vtiming

display_timing_t::vtiming

概要説明

垂直表示のサイクル設定値。

11.1.78.3 format

display_out_format_t::format

概要説明

出力形式設定。

11.1.78.4 endian

display_endian_t::endian

概要説明

出力データのビット順序。

11.1.78.5 color_order

display_color_order_t::color_order

概要説明

カラー オーダー （ピクセル単位）。

11.1.78.6 data_enable_polarity

display_signal_polarity_t::data_enable_polarity

概要説明

データ有効信号極性。

11.1.78.7 sync_edge

display_sync_edge_t::sync_edge

概要説明

信号同期エッジ選択。

11.1.78.8 bg_color

display_color_t::bg_color

概要説明

バックグラウンド カラー。

11.1.78.9 brightness

display_brightness_t::brightness

概要説明

明るさの設定値。

11.1.78.10 contrast

display_contrast_t::contrast

概要説明

コントラスト設定値。

11.1.78.11 p_gamma_correction

`display_gamma_correction_t`::p_gamma_correction

概要説明

ガンマ補正設定値へのポインタ。

11.1.78.12 dithering_on

bool `display_output_cfg_t`::dithering_on

概要説明

ディザリングのオン / オフ。

11.1.79 display_runtime_cfg_t

```
typedef struct{
    display_input_cfg_t input
    display_layer_t layer
} display_runtime_cfg_t
```

11.1.79.1 input

`display_input_cfg_t`::input

概要説明

グラフィックス入力フレーム設定値。

詳細説明

表示デバイス用の汎用設定

11.1.79.2 layer

`display_layer_t`::layer

概要説明

グラフィクス レイヤーのアルファ ブレンド設定値。

11.1.80 display_status_t

```
typedef struct{
   display_state_t state
   display_fade_status_t fade_status[DISPLAY_FRAME_LAYER_2+1]
} display_status_t
```

11.1.80.1 state

display_state_t::state

概要説明

GLCD モジュールのステータス。

11.1.80.2 fade_status

display_fade_status_t::fade_status

概要説明

フェードイン / フェードアウトのステータス。

11.1.81 display_timing_t

```
typedef struct{
   uint16_t total_cyc
   uint16_t display_cyc
   uint16_t back_porch
   uint16_t sync_width
   display_signal_polarity_t sync_polarity
} display_timing_t
```

11.1.81.1 total_cyc

uint16_t display_timing_t::total_cyc

概要説明

1 ラインの合計サイクルまたは 1 フレームの合計ライン。

11.1.81.2 display_cyc

uint16_t display_timing_t::display_cyc

概要説明

アクティブ ビデオ サイクルまたはライン。

11.1.81.3 back_porch

uint16_t display_timing_t::back_porch

概要説明

バック ポーチ サイクルまたはライン。

11.1.81.4 sync_width

uint16_t display_timing_t::sync_width

概要説明

同期信号アサーション幅。

11.1.81.5 sync_polarity

display_signal_polarity_t::sync_polarity

概要説明

同期信号極性。

11.1.82 doc_api_t

```
typedef struct{
    ssp_err_t(*  open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(doc_ctrl_t *const p_ctrl)
    ssp_err_t(*  statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
    ssp_err_t(*  statusClear)(doc_ctrl_t *const p_ctrl)
    ssp_err_t(*  write)(doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)
    ssp_err_t(*  inputRegisterWrite)(doc_ctrl_t *const p_ctrl, doc_size_t data)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} doc_api_t
```

11.1.83 doc_callback_args_t

```
typedef struct{
    doc_event_t  event
    void const *  p_context
} doc_callback_args_t
```

11.1.83.1 event

doc_event_t::event

概要説明

このイベントは、コールバックの原因を特定するために使用されます。

11.1.83.2 p_context

void const* doc_callback_args_t::p_context

詳細説明

ユーザー データのプレースホルダー。doc_cfg_t の open 関数で設定されます。

11.1.84 doc_cfg_t

```
typedef struct{
    doc_event_t  event
    void(*  p_callback)(doc_callback_args_t *p_args)
    void const *  p_context
} doc_cfg_t
```

11.1.84.1 event

doc_event_t::event

概要説明

doc_event_t から列挙値を選択します。

11.1.84.2 p_callback

void(* doc_cfg_t::p_callback)(doc_callback_args_t *p_args)

詳細説明

DOC ISR の発生時に提供されるコールバック。

11.1.84.3 p_context

void const* doc_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。doc_callback_args_t 内のユーザー コールバックに渡されます。

11.1.85 doc_ctrl_t

```
typedef struct{
   uint32_t open
   void(* p_callback)(doc_callback_args_t *p_args)
   doc_event_t event
   void const * p_context
} doc_ctrl_t
```

11.1.85.1 open

uint32_t doc_ctrl_t::open

概要説明

ドライバによって使用され、制御構造体が有効かどうかを確認します。

11.1.85.2 p_callback

void(* doc_ctrl_t::p_callback)(doc_callback_args_t *p_args)

詳細説明

DOC ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL を使用します。

11.1.85.3 event

doc_event_t::event

概要説明

イベント DOC の設定対象です。ISR コールバックに渡されます。

11.1.85.4 p_context

void const* doc_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。doc_callback_args_t 内のユーザー コールバックに渡されます。

11.1.86 doc_data_t

```
typedef struct{
   doc_size_t dodir
   doc_size_t dodsr
} doc_data_t
```

11.1.86.1 dodir

doc_size_t::dodir

概要説明

DOC DODIR に書き込まれる値。

11.1.86.2 dodsr

doc_size_t::dodsr

概要説明

DOC DODSR に書き込まれる値。

11.1.87 doc_instance_t

```
typedef struct{
    doc_ctrl_t *  p_ctrl
    doc_cfg_t const *  p_cfg
    doc_api_t const *  p_api
} doc_instance_t
```

11.1.87.1 p_ctrl

doc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.87.2 p_cfg

doc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.87.3 p_api

doc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.88 dsa_api_t

```
typedef struct{
   uint32_t(* open)(dsa_ctrl_t *const p_ctrl, dsa_cfg_t const *const p_cfg)
   uint32_t(* verify)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t
*p_paddedHash)
   uint32_t(* hashVerify)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_signature, uint32_t *p_paddedHash)
   uint32_t(* sign)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)
   uint32_t(* hashSign)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_paddedHash, uint32_t *p_dest)
   uint32_t(* close)(dsa_ctrl_t *const p_ctrl)
   uint32_t(* versionGet)(ssp_version_t *const p_version)
} dsa_api_t
```

11.1.89 dsa_cfg_t

```
typedef struct{
   crypto_api_t const * p_crypto_api
} dsa_cfg_t
```

11.1.89.1 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.90 dsa_ctrl_t

```
typedef struct{
   crypto_ctrl_t * p_crypto_ctrl
   crypto_api_t const * p_crypto_api
} dsa_ctrl_t
```

11.1.90.1 p_crypto_ctrl

crypto_ctrl_t::p_crypto_ctrl

概要説明

暗号化エンジン制御構造体へのポインタ

11.1.90.2 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.91 dsa_domain_1024_160_t

```
typedef struct{
   uint32_t  q[(160/32)]
   uint32_t  p[(1024/32)]
   uint32_t  g[(1024/32)]
} dsa_domain_1024_160_t
```

11.1.91.1 q

uint32_t ::q[(160/32)]

概要説明

DSA (1024,160) ドメイン パラメータ Q。

11.1.91.2 p

uint32_t ::p[(1024/32)]

概要説明

DSA (1024,160) ドメイン パラメータ P。

11.1.91.3 g

uint32_t ::g[(1024/32)]

概要説明

DSA (1024,160) ドメイン パラメータ G。

11.1.92 dsa_domain_2048_224_t

```
typedef struct{
   uint32_t  q[(224/32)]
   uint32_t  p[(2048/32)]
   uint32_t  g[(2048/32)]
} dsa_domain_2048_224_t
```

11.1.92.1 q

uint32_t ::q[(224/32)]

概要説明

DSA (2048,224) ドメイン パラメータ Q。

11.1.92.2 p

uint32_t ::p[(2048/32)]

概要説明

DSA (2048,224) ドメイン パラメータ P。

11.1.92.3 g

uint32_t ::g[(2048/32)]

概要説明

DSA (2048,224) ドメイン パラメータ G。

11.1.93 dsa_domain_2048_256_t

```
typedef struct{
    uint32_t  q[(256/32)]
    uint32_t  p[(2048/32)]
    uint32_t  g[(2048/32)]
} dsa_domain_2048_256_t
```

11.1.93.1 q

uint32_t ::q[(256/32)]

概要説明

DSA (2048,256) ドメイン パラメータ Q。

11.1.93.2 p

uint32_t ::p[(2048/32)]

概要説明

DSA (2048,256) ドメイン パラメータ P。

11.1.93.3 g

uint32_t ::g[(2048/32)]

概要説明

DSA (2048,256) ドメイン パラメータ G。

11.1.94 dsa_instance_t

```
typedef struct{
    dsa_ctrl_t * p_ctrl
    dsa_cfg_t const * p_cfg
    dsa_api_t const * p_api
} dsa_instance_t
```

11.1.94.1 p_ctrl

dsa_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.94.2 p_cfg

dsa_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.94.3 p_api

dsa_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.95 dsa_signature_1024_160_t

```
typedef struct{
    uint32_t r[(160/32)]
    uint32_t s[(160/32)]
} dsa_signature_1024_160_t
```

11.1.95.1 r

uint32_t ::r[(160/32)]

概要説明

DSA (1024,160) 署名コンポーネント R。

11.1.95.2 s

uint32_t ::s[(160/32)]

概要説明

DSA (1024,160) 署名コンポーネント S。

11.1.96 dsa_signature_2048_224_t

```
typedef struct{
   uint32_t  r[(224/32)]
   uint32_t  s[(224/32)]
} dsa_signature_2048_224_t
```

11.1.96.1 r

uint32_t ::r[(224/32)]

概要説明

DSA (2048,224) 署名コンポーネント R。

11.1.96.2 s

uint32_t ::s[(224/32)]

概要説明

DSA (2048,224) 署名コンポーネント S。

11.1.97 dsa_signature_2048_256_t

```
typedef struct{
   uint32_t  r[(256/32)]
   uint32_t  s[(256/32)]
} dsa_signature_2048_256_t
```

11.1.97.1 r

uint32_t ::r[(256/32)]

概要説明

DSA (2048,256) 署名コンポーネント R。

11.1.97.2 s

uint32_t ::s[(256/32)]

概要説明

DSA (2048,256) 署名コンポーネント S。

11.1.98 dtc_reg_t

```
typedef struct{
  uint32_t __pad0__
  uint8_t MRB
  uint8_t __pad0__
  uint8_t DM
  uint8_t DTS
  uint8_t DISEL
  uint8_t CHNS
  uint8_t CHNE
  struct{}        MRB_b
  uint8_t MRA
  uint8_t SM
  uint8_t SZ
  uint8_t MD
  struct{}        MRA_b
  struct{}        struct{}
  void *volatile SAR
  void *volatile DAR
  uint16_t CRB
  uint16_t CRA
  uint8_t CRAL
  uint8_t CRAH
  struct{}        CRA_b
  struct{}        struct{}
} dtc_reg_t
```

11.1.98.1 __pad0__

uint32_t ::__pad0__

11.1.98.2 MRB

uint8_t ::MRB

詳細説明

モード レジスタ B

11.1.98.3 __pad0__

uint8_t ::__pad0__

11.1.98.4 DM

uint8_t ::DM

概要説明

転送宛先アドレス モード。

11.1.98.5 DTS

uint8_t ::DTS

概要説明

DTC 転送モード選択。

11.1.98.6 DISEL

uint8_t ::DISEL

概要説明

DTC 割り込み選択。

11.1.98.7 CHNS

uint8_t ::CHNS

概要説明

DTC チェーン転送選択。

11.1.98.8 CHNE

uint8_t ::CHNE

概要説明

DTC チェーン転送の有効化。

11.1.98.9 MRB_b

このメンバーの定義については、ソース コードを参照してください。

詳細説明

- MRB ビット */

11.1.98.10 MRA

uint8_t ::MRA

詳細説明

モード レジスタ A

11.1.98.11 SM

uint8_t ::SM

概要説明

転送ソース アドレス モード。

11.1.98.12 SZ

uint8_t ::SZ

概要説明

DTC データ転送サイズ。

11.1.98.13 MD

uint8_t ::MD

概要説明

DTC 転送モード選択。

11.1.98.14 MRA_b

このメンバーの定義については、ソース コードを参照してください。

詳細説明

- MRA ビット */

11.1.98.15 struct{}

このメンバーの定義については、ソース コードを参照してください。

詳細説明

- モード レジスタ */

11.1.98.16 SAR

void* volatile ::SAR

概要説明

ソース アドレス レジスタ。

11.1.98.17 DAR

void* volatile ::DAR

詳細説明

宛先アドレス レジスタ

11.1.98.18 CRB

volatile uint16_t ::CRB

詳細説明

転送カウント レジスタ B

11.1.98.19 CRA

uint16_t ::CRA

詳細説明

転送カウント レジスタ A

11.1.98.20 CRAL

uint8_t ::CRAL

概要説明

転送カウンタ A 下位レジスタ。

11.1.98.21 CRAH

uint8_t ::CRAH

概要説明

転送カウンタ A 上位レジスタ。

11.1.98.22 CRA_b

このメンバーの定義については、ソース コードを参照してください。

詳細説明

- ビット */

11.1.98.23 struct{}

このメンバーの定義については、ソース コードを参照してください。

詳細説明

- 転送カウント レジスタ */

11.1.99 elc_api_t

```
typedef struct{
   ssp_err_t(* init)(elc_cfg_t const *const p_cfg)
   ssp_err_t(* softwareEventGenerate)(elc_software_event_t event_num)
   ssp_err_t(* linkSet)(elc_peripheral_t peripheral, elc_event_t signal)
   ssp_err_t(* linkBreak)(elc_peripheral_t peripheral)
   ssp_err_t(* enable)(void)
   ssp_err_t(* disable)(void)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} elc_api_t
```

11.1.100 elc_cfg_t

```
typedef struct{
   bool autostart
   uint32_t link_count
   elc_link_t const * link_list
} elc_cfg_t
```

11.1.100.1 autostart

bool elc_cfg_t::autostart

概要説明

open() の実行中に操作を開始し、割り込みを有効にします。

11.1.100.2 link_count

uint32_t elc_cfg_t::link_count

概要説明

イベント リンク数。

11.1.100.3 link_list

elc_link_t::link_list

概要説明

イベント リンク。

11.1.101 elc_instance_t

```
typedef struct{
    elc_cfg_t const *  p_cfg
    elc_api_t const *  p_api
} elc_instance_t
```

11.1.101.1 p_cfg

elc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.101.2 p_api

elc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.102 elc_link_t

```
typedef struct{
   elc_peripheral_t peripheral
   elc_event_t event
} elc_link_t
```

11.1.102.1 peripheral

elc_peripheral_t::peripheral

概要説明

信号を受信するペリフェラル。

11.1.102.2 event

elc_event_t::event

概要説明

ペリフェラルに送信される信号。

11.1.103 external_irq_api_t

```
typedef struct{
   ssp_err_t(* open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
   ssp_err_t(* enable)(external_irq_ctrl_t *const p_ctrl)
   ssp_err_t(* disable)(external_irq_ctrl_t *const p_ctrl)
   ssp_err_t(* triggerSet)(external_irq_ctrl_t *const p_ctrl, external_irq_trigger_t const trigger)
   ssp_err_t(* filterEnable)(external_irq_ctrl_t *const p_ctrl)
   ssp_err_t(* filterDisable)(external_irq_ctrl_t *const p_ctrl)
   ssp_err_t(* close)(external_irq_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} external_irq_api_t
```

11.1.104 external_irq_callback_args_t

```
typedef struct{
   void const * p_context
   uint32_t channel
} external_irq_callback_args_t
```

11.1.104.1 p_context

void const* external_irq_callback_args_t::p_context

詳細説明

ユーザー データのプレースホルダー。external_irq_cfg_t の open 関数で設定されます。

11.1.104.2 channel

uint32_t external_irq_callback_args_t::channel

概要説明

割り込みを発生させた物理ハードウェア チャネル。

11.1.105 external_irq_cfg_t

```
typedef struct{
    uint32_t channel
    external_irq_trigger_t trigger
    bool filter_enable
    external_irq_pclk_div_t pclk_div
    bool autostart
    void(* p_callback)(external_irq_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} external_irq_cfg_t
```

11.1.105.1 channel

uint32_t external_irq_cfg_t::channel

概要説明

使用されるハードウェア チャネル。

11.1.105.2 trigger

external_irq_trigger_t::trigger

概要説明

トリガー設定値。

11.1.105.3 filter_enable

bool external_irq_cfg_t::filter_enable

概要説明

デジタル フィルタ有効 / 無効設定。

11.1.105.4 pclk_div

external_irq_pclk_div_t::pclk_div

概要説明

デジタル フィルタ クロック除算値設定。

11.1.105.5 autostart

bool external_irq_cfg_t::autostart

概要説明

open() の実行中に操作を開始し、割り込みを有効にします。

11.1.105.6 p_callback

void(* external_irq_cfg_t::p_callback)(external_irq_callback_args_t *p_args)

詳細説明

外部入力トリガーの発生時に提供されるコールバック。

11.1.105.7 p_context

void const* external_irq_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。external_irq_callback_args_t 内のユーザー コールバックに渡されます。

11.1.105.8 p_extend

void const* external_irq_cfg_t::p_extend

概要説明

外部 IRQ ハードウェアに依存する設定。

11.1.106 external_irq_ctrl_t

```
typedef struct{
    uint32_t channel
    void(* p_callback)(external_irq_callback_args_t *p_args)
    void const * p_context
} external_irq_ctrl_t
```

11.1.106.1 channel

uint32_t external_irq_ctrl_t::channel

概要説明

チャネル。

11.1.106.2 p_callback

void(* external_irq_ctrl_t::p_callback)(external_irq_callback_args_t *p_args)

詳細説明

外部 IRQ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

11.1.106.3 p_context

void const* external_irq_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。external_irq_callback_args_t 内のユーザー コールバックに渡されます。

11.1.107 external_irq_instance_t

```
typedef struct{
    external_irq_ctrl_t *  p_ctrl
    external_irq_cfg_t const *  p_cfg
    external_irq_api_t const *  p_api
} external_irq_instance_t
```

11.1.107.1 p_ctrl

external_irq_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.107.2 p_cfg

external_irq_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.107.3 p_api

external_irq_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.108 flash_api_t

```
typedef struct{
   ssp_err_t(* open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
   ssp_err_t(* write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
   ssp_err_t(* read)(flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)
   ssp_err_t(* erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)
   ssp_err_t(* blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const
p_blank_check_result)
   ssp_err_t(* close)(flash_ctrl_t *const p_ctrl)
   ssp_err_t(* statusGet)(flash_ctrl_t *const p_ctrl)
   ssp_err_t(* accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)
   ssp_err_t(* accessWindowClear)(flash_ctrl_t *const p_ctrl)
   ssp_err_t(* reset)(flash_ctrl_t *const p_ctrl)
   ssp_err_t(* updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)
   ssp_err_t(* startupAreaSelect)(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
   ssp_err_t(* versionGet)(ssp_version_t *p_version)
} flash_api_t
```

11.1.109 flash_callback_args_t

```
typedef struct{
   flash_event_t event
   void const * p_context
} flash_callback_args_t
```

11.1.109.1 event

flash_event_t::event

概要説明

イベントを使用して、コールバックの原因 （フラッシュ レディまたはエラー）を特定できます。

11.1.109.2 p_context

void const* flash_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。::flash_cfg_t 内の open 関数で設定されます。

11.1.110 flash_cfg_t

```
typedef struct{
    bool data_flash_bgo
    void(* p_callback)(flash_callback_args_t *p_args)
    void const * p_extend
    void const * p_context
} flash_cfg_t
```

11.1.110.1 data_flash_bgo

bool flash_cfg_t::data_flash_bgo

概要説明

データ フラッシュで BGO （バックグラウンド操作）で有効な場合は true。

11.1.110.2 p_callback

void(* flash_cfg_t::p_callback)(flash_callback_args_t *p_args)

概要説明

フラッシュ割り込み ISR の発生時に提供されるコールバック。

11.1.110.3 p_extend

void const* flash_cfg_t::p_extend

概要説明

FLASH ハードウェアに依存する設定。

11.1.110.4 p_context

void const* flash_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。flash_callback_args_t 内のユーザー コールバックに渡されます。

11.1.111 flash_ctrl_t

```
typedef struct{
   bool  opened
} flash_ctrl_t
```

11.1.111.1 opened

bool flash_ctrl_t::opened

11.1.112 flash_instance_t

```
typedef struct{
   flash_ctrl_t *  p_ctrl
   flash_cfg_t const *  p_cfg
   flash_api_t const *  p_api
} flash_instance_t
```

11.1.112.1 p_ctrl

flash_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.112.2 p_cfg

flash_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.112.3 p_api

flash_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.113 fmi_api_t

```
typedef struct{
   ssp_err_t(* productInfoGet)(fmi_product_info_t **pp_product_info)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} fmi_api_t
```

11.1.114 fmi_header_t

```
typedef struct{
   uint32_t contents
   uint32_t variant
   uint32_t count
   uint32_t minor
   uint32_t major
} fmi_header_t
```

11.1.114.1 contents

uint32_t fmi_header_t::contents

11.1.114.2 variant

uint32_t fmi_header_t::variant

11.1.114.3 count

uint32_t fmi_header_t::count

11.1.114.4 minor

uint32_t fmi_header_t::minor

11.1.114.5 major

uint32_t fmi_header_t::major

11.1.115 fmi_instance_t

```
typedef struct{
   fmi_api_t const * p_api
} fmi_instance_t
```

11.1.115.1 p_api

fmi_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.116 fmi_product_info_t

```
typedef struct{
   fmi_header_t header
   uint8_t unique_id[16]
   uint8_t product_name[16]
   uint8_t product_marking[16]
   uint32_t mask_revision
   uint32_t pin_count
   uint32_t pkg_type
   uint32_t temp_range
   uint32_t quality_code
   uint32_t reserved
   struct{}        struct{}
   uint32_t max_freq
   uint32_t reserved1
   struct{}        struct{}
} fmi_product_info_t
```

11.1.116.1 header

fmi_header_t fmi_product_info_t::header

11.1.116.2 unique_id

uint8_t fmi_product_info_t::unique_id[16]

11.1.116.3 product_name

uint8_t fmi_product_info_t::product_name[16]

11.1.116.4 product_marking

uint8_t fmi_product_info_t::product_marking[16]

11.1.116.5 mask_revision

uint32_t fmi_product_info_t::mask_revision

11.1.116.6 pin_count

uint32_t fmi_product_info_t::pin_count

11.1.116.7 pkg_type

uint32_t fmi_product_info_t::pkg_type

11.1.116.8 temp_range

uint32_t fmi_product_info_t::temp_range

11.1.116.9 quality_code

uint32_t fmi_product_info_t::quality_code

11.1.116.10 reserved

uint32_t fmi_product_info_t::reserved

11.1.116.11 struct{}

このメンバーの定義については、ソース コードを参照してください。

11.1.116.12 max_freq

uint32_t fmi_product_info_t::max_freq

11.1.116.13 reserved1

uint32_t fmi_product_info_t::reserved1

11.1.116.14 struct{}

このメンバーの定義については、ソース コードを参照してください。

11.1.117 gamma_correction_t

```
typedef struct{
   bool enable
   uint16_t gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]
   uint16_t threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]
} gamma_correction_t
```

11.1.117.1 enable

bool gamma_correction_t::enable

概要説明

ガンマ補正オン / オフ。

11.1.117.2 gain

uint16_t gamma_correction_t::gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]

概要説明

ゲイン調整。

11.1.117.3 threshold

uint16_t gamma_correction_t::threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]

概要説明

開始しきい値。

11.1.118 glcd_cfg_t

```
typedef struct{
    glcd_tcon_pin_t tcon_hsync
    glcd_tcon_pin_t tcon_vsync
    glcd_tcon_pin_t tcon_de
    glcd_correction_proc_order_t correction_proc_order
    glcd_clk_src_t clksrc
    glcd_panel_clk_div_t clock_div_ratio
    glcd_dithering_mode_t dithering_mode
    glcd_dithering_pattern_t dithering_pattern_A
    glcd_dithering_pattern_t dithering_pattern_B
    glcd_dithering_pattern_t dithering_pattern_C
    glcd_dithering_pattern_t dithering_pattern_D
} glcd_cfg_t
```

11.1.118.1 tcon_hsync

glcd_tcon_pin_t::tcon_hsync

概要説明

GLCD TCON 出力ピン選択。

11.1.118.2 tcon_vsync

glcd_tcon_pin_t::tcon_vsync

概要説明

GLCD TCON 出力ピン選択。

11.1.118.3 tcon_de

glcd_tcon_pin_t::tcon_de

概要説明

GLCD TCON 出力ピン選択。

11.1.118.4 correction_proc_order

glcd_correction_proc_order_t::correction_proc_order

概要説明

補正制御ルート選択。

11.1.118.5 clksrc

glcd_clk_src_t::clksrc

概要説明

クロック ソース選択。

11.1.118.6 clock_div_ratio

glcd_panel_clk_div_t::clock_div_ratio

概要説明

ドット クロックのクロック分割比。

11.1.118.7 dithering_mode

glcd_dithering_mode_t::dithering_mode

概要説明

ディザリング モード。

11.1.118.8 dithering_pattern_A

glcd_dithering_pattern_t::dithering_pattern_A

概要説明

ディザリング パターン A。

11.1.118.9 dithering_pattern_B

glcd_dithering_pattern_t::dithering_pattern_B

概要説明

ディザリング パターン B。

11.1.118.10 dithering_pattern_C

glcd_dithering_pattern_t::dithering_pattern_C

概要説明

ディザリング パターン C。

11.1.118.11 dithering_pattern_D

glcd_dithering_pattern_t::dithering_pattern_D

概要説明

ディザリング パターン D。

11.1.119 glcd_ctrl_t

```
typedef struct{
    display_coordinate_t back_porch
    uint16_t hsize
    uint16_t vsize
    bsp_lock_t resource_lock
    void * p_context
} glcd_ctrl_t
```

11.1.119.1 back_porch

display_coordinate_t::back_porch

概要説明

グラフィックス プレーンのゼロ座標 （バック ポーチ終点）

11.1.119.2 hsize

uint16_t ::hsize

概要説明

1 ラインの水平ピクセル サイズ。

11.1.119.3 vsize

uint16_t ::vsize

概要説明

1 フレームの垂直ピクセル サイズ。

11.1.119.4 resource_lock

bsp_lock_t::resource_lock

概要説明

リソース ロック。

11.1.119.5 p_context

void* ::p_context

詳細説明

関数レベルのデバイス コンテキスト （例 : display_ctrl_t 型のデータ）へのポインタ

11.1.120 gpt_input_capture_extend_t

```
typedef struct{
    gpt_input_capture_signal_t signal
    gpt_input_capture_signal_filter_t signal_filter
    gpt_input_capture_clock_divider_t clock_divider
    input_capture_signal_level_t enable_level
    gpt_input_capture_signal_filter_t enable_filter
} gpt_input_capture_extend_t
```

11.1.120.1 signal

gpt_input_capture_signal_t::signal

概要説明

gpt_input_capture_signal_t のいずれかです。

11.1.120.2 signal_filter

gpt_input_capture_signal_filter_t::signal_filter

概要説明

gpt_input_capture_signal_filter_t のいずれかです。

11.1.120.3 clock_divider

gpt_input_capture_clock_divider_t::clock_divider

概要説明

gpt_input_capture_clock_divider_t のいずれかです。

11.1.120.4 enable_level

input_capture_signal_level_t::enable_level

詳細説明

未使用の GTIOCx ピンは、キャプチャを有効化するためのイネーブル信号として用いることができます。入力キャプチャ信号ピンが GTIOCA の場合、イネーブル ピンは GTIOCB です。使用されている場合、有効レベルはここで設定されます。

11.1.120.5 enable_filter

gpt_input_capture_signal_filter_t::enable_filter

概要説明

gpt_input_capture_signal_filter_t のいずれかです。

11.1.121 gpt_output_pin_t

```
typedef struct{
   bool output_enabled
   gpt_pin_level_t stop_level
} gpt_output_pin_t
```

11.1.121.1 output_enabled

bool ::output_enabled

概要説明

出力を有効にするには true、無効にするには false に設定します。

11.1.121.2 stop_level

gpt_pin_level_t::stop_level

概要説明

gpt_pin_level_t から停止レベルを選択します。

11.1.122 hash_api_t

```
typedef struct{
    uint32_t(*  open)(hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)
    uint32_t(*  updateHash)(const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
    uint32_t(*  hashUpdate)(hash_ctrl_t *const p_ctrl, const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
    uint32_t(*  close)(hash_ctrl_t *const p_ctrl)
    uint32_t(*  versionGet)(ssp_version_t *const p_version)
} hash_api_t
```

11.1.123 hash_cfg_t

```
typedef struct{
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const *  p_crypto_api
} hash_cfg_t
```

11.1.123.1 p_crypto_ctrl

crypto_ctrl_t::p_crypto_ctrl

概要説明

暗号化エンジン制御構造体へのポインタ

11.1.123.2 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API 構造体へのポインタ

11.1.124 hash_ctrl_t

```
typedef struct{
   uint32_t msgbuf[HASH_MESSAGE_BLOCK_SIZE_WORDS]
   uint32_t hash[HASH_MAX_DIGEST_SIZE_WORDS]
   uint64_t length
} hash_ctrl_t
```

11.1.124.1 msgbuf

uint32_t hash_ctrl_t::msgbuf[HASH_MESSAGE_BLOCK_SIZE_WORDS]

概要説明

ハッシュされるメッセージ バッファ

11.1.124.2 hash

uint32_t hash_ctrl_t::hash[HASH_MAX_DIGEST_SIZE_WORDS]

概要説明

現在のハッシュ値

11.1.124.3 length

uint64_t hash_ctrl_t::length

概要説明

64 ビット メッセージ長 （ビット数）

11.1.125 hash_instance_t

```
typedef struct{
   hash_ctrl_t * p_ctrl
   hash_cfg_t const * p_cfg
   hash_api_t const * p_api
} hash_instance_t
```

11.1.125.1 p_ctrl

hash_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.125.2 p_cfg

hash_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.125.3 p_api

hash_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.126 i2c_api_master_t

```
typedef struct{
    ssp_err_t(*  open)(i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  read)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
    ssp_err_t(*  write)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
    ssp_err_t(*  reset)(i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} i2c_api_master_t
```

11.1.127 i2c_callback_args_t

```
typedef struct{
    void const *const  p_context
    uint32_t const  bytes
    i2c_event_t const  event
} i2c_callback_args_t
```

11.1.127.1 p_context

void const* const i2c_callback_args_t::p_context

概要説明

ユーザー定義のコンテキストへのポインタ。

11.1.127.2 bytes

uint32_t const i2c_callback_args_t::bytes

概要説明

バッファ内の受信 / 送信バイト数。

11.1.127.3 event

i2c_event_t::event

概要説明

イベント コード。

11.1.128 i2c_cfg_t

```
typedef struct{
    uint32_t channel
    i2c_rate_t rate
    uint16_t slave
    i2c_addr_mode_t addr_mode
    void(* p_callback)(i2c_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} i2c_cfg_t
```

11.1.128.1 channel

uint32_t i2c_cfg_t::channel

概要説明

uint32_t

詳細説明

実装によって設定可能な識別子。

11.1.128.2 rate

i2c_rate_t::rate

概要説明

enum i2c_rate_t のデバイスの最大クロック レート。

11.1.128.3 slave

uint16_t i2c_cfg_t::slave

概要説明

スレーブ デバイスのアドレス。

11.1.128.4 addr_mode

i2c_addr_mode_t::addr_mode

概要説明

スレーブ フィールドの解釈方法を示します。

11.1.128.5 p_callback

void(* i2c_cfg_t::p_callback)(i2c_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

詳細説明

ソフトウェアの動作を制御するパラメータ

11.1.128.6 p_context

void const* i2c_cfg_t::p_context

概要説明

ユーザー定義のコンテキストへのポインタ。

11.1.128.7 p_extend

void const* i2c_cfg_t::p_extend

概要説明

ハードウェアで必要な任意の設定データ。

詳細説明

実装固有の拡張設定

11.1.129 i2c_ctrl_t

```
typedef struct{
   i2c_cfg_t info
   uint32_t open
} i2c_ctrl_t
```

11.1.129.1 info

i2c_cfg_t::info

概要説明

I2C デバイスに関する情報。

11.1.129.2 open

uint32_t i2c_ctrl_t::open

概要説明

デバイスが開いているかどうかを示すフラグ。

11.1.130 i2c_master_instance_t

```
typedef struct{
   i2c_ctrl_t * p_ctrl
   i2c_cfg_t const * p_cfg
   i2c_api_master_t const * p_api
} i2c_master_instance_t
```

11.1.130.1 p_ctrl

i2c_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.130.2 p_cfg

i2c_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.130.3 p_api

i2c_api_master_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.131 i2s_api_t

```
typedef struct{
   ssp_err_t(* open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
   ssp_err_t(* stop)(i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)
   ssp_err_t(* mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
   ssp_err_t(* write)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)
   ssp_err_t(* read)(i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint16_t const bytes)
   ssp_err_t(* writeRead)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes)
   ssp_err_t(* infoGet)(i2s_ctrl_t *const p_ctrl, i2s_info_t *const p_info)
   ssp_err_t(* close)(i2s_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} i2s_api_t
```

11.1.132 i2s_callback_args_t

```
typedef struct{
   void const * p_context
   i2s_event_t event
} i2s_callback_args_t
```

11.1.132.1 p_context

void const* i2s_callback_args_t::p_context

詳細説明

ユーザー データのプレースホルダー。i2s_cfg_t の open 関数で設定されます。

11.1.132.2 event

i2s_event_t::event

概要説明

このイベントを使用して、コールバックの原因 （オーバーフローまたはエラー）を特定できます。

11.1.133 i2s_cfg_t

```
typedef struct{
    uint8_t channel
    i2s_pcm_width_t pcm_width
    i2s_word_length_t word_length
    i2s_ws_continue_t ws_continue
    uint32_t sampling_freq_hz
    uint32_t audio_clk_freq_hz
    timer_instance_t const * p_timer
    transfer_instance_t const * p_transfer_tx
    transfer_instance_t const * p_transfer_rx
    void(* p_callback)(i2s_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} i2s_cfg_t
```

11.1.133.1 channel

uint8_t i2s_cfg_t::channel

詳細説明

ハードウェアのチャネル番号に対応するチャネルを選択します。

11.1.133.2 pcm_width

i2s_pcm_width_t::pcm_width

概要説明

オーディオ PCM のデータ幅。

11.1.133.3 word_length

i2s_word_length_t::word_length

概要説明

オーディオのワード長、>= pcm_width ビットである必要があります。

11.1.133.4 ws_continue

i2s_ws_continue_t::ws_continue

概要説明

アイドル状態において、WS の送信を続行するかどうかを示します。

11.1.133.5 sampling_freq_hz

uint32_t i2s_cfg_t::sampling_freq_hz

概要説明

サンプリング周波数 （ヘルツ単位)。

11.1.133.6 audio_clk_freq_hz

uint32_t i2s_cfg_t::audio_clk_freq_hz

詳細説明

オーディオのクロック周波数 （ヘルツ単位)。(16 * sampling_freq_hz * (word_length <enum_value> + 1) の 1 ～ 128 の倍数である必要があります。

11.1.133.7 p_timer

timer_instance_t::p_timer

詳細説明

GPT でオーディオ クロックを作成する場合、ここにタイマ インスタンスをリンクします。使用しない場合は、NULL に設定します。

11.1.133.8 p_transfer_tx

transfer_instance_t::p_transfer_tx

詳細説明

書き込み時に DTC を使用する場合、ここに DTC インスタンスをリンクします。使用しない場合は、NULL に設定します。

11.1.133.9 p_transfer_rx

transfer_instance_t::p_transfer_rx

詳細説明

読み取り時に DTC を使用する場合、ここに DTC インスタンスをリンクします。使用しない場合は、NULL に設定します。

11.1.133.10 p_callback

void(* i2s_cfg_t::p_callback)(i2s_callback_args_t *p_args)

詳細説明

I2S ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

11.1.133.11 p_context

void const* i2s_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。i2s_callback_args_t 内のユーザー コールバックに渡されます。

11.1.133.12 p_extend

void const* i2s_cfg_t::p_extend

概要説明

ハードウェア固有の設定値に対応するための拡張パラメータです。

11.1.134 i2s_ctrl_t

```
typedef struct{
   void(* p_callback)(i2s_callback_args_t *p_args)
   void const * p_context
   timer_instance_t const * p_timer
   transfer_instance_t const * p_transfer_tx
   transfer_instance_t const * p_transfer_rx
   uint32_t const * p_tx_src
   uint32_t tx_src_bytes
   uint32_t * p_rx_dest
   uint32_t rx_dest_bytes
   uint32_t sampling_freq_hz
   uint8_t channel
   bool stop_requested_tx
   bool stop_requested_rx
   bool tx_in_progress
   bool zeros_written
} i2s_ctrl_t
```

11.1.134.1 p_callback

void(* i2s_ctrl_t::p_callback)(i2s_callback_args_t *p_args)

詳細説明

I2S ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL を使用します。

11.1.134.2 p_context

void const* i2s_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。i2s_callback_args_t 内のユーザー コールバックに渡されます。

11.1.134.3 p_timer

timer_instance_t::p_timer

概要説明

オーディオ クロックの作成に使用されるタイマ。

11.1.134.4 p_transfer_tx

transfer_instance_t::p_transfer_tx

概要説明

書き込み時のハードウェア加速に使用される転送。

11.1.134.5 p_transfer_rx

transfer_instance_t::p_transfer_rx

概要説明

読み取り時のハードウェア加速に使用される転送。

11.1.134.6 p_tx_src

uint32_t const* i2s_ctrl_t::p_tx_src

詳細説明

送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファ ポインタ。

11.1.134.7 tx_src_bytes

uint32_t i2s_ctrl_t::tx_src_bytes

詳細説明

送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファのサイズ。

11.1.134.8 p_rx_dest

uint32_t* i2s_ctrl_t::p_rx_dest

詳細説明

受信 ISR からハードウェア FIFO を指定するために使用される宛先バッファ ポインタ。

11.1.134.9 rx_dest_bytes

uint32_t i2s_ctrl_t::rx_dest_bytes

詳細説明

受信 ISR からハードウェア FIFO を指定するために使用される宛先バッファのサイズ。

11.1.134.10 sampling_freq_hz

uint32_t i2s_ctrl_t::sampling_freq_hz

概要説明

サンプリング周波数 （ヘルツ単位）。

11.1.134.11 channel

uint8_t i2s_ctrl_t::channel

概要説明

チャネル番号。

11.1.134.12 stop_requested_tx

bool i2s_ctrl_t::stop_requested_tx

概要説明

送信の完了時に I2S を停止します。

11.1.134.13 stop_requested_rx

bool i2s_ctrl_t::stop_requested_rx

概要説明

受信の完了時に I2S を停止します。

11.1.134.14 tx_in_progress

bool i2s_ctrl_t::tx_in_progress

概要説明

送信転送が実行中の場合は true。

11.1.134.15 zeros_written

bool i2s_ctrl_t::zeros_written

概要説明

送信転送が実行中の場合は true。

11.1.135 i2s_info_t

```
typedef struct{
    i2s_status_t  status
    uint32_t  sampling_freq_hz
} i2s_info_t
```

11.1.135.1 status

i2s_status_t::status

11.1.135.2 sampling_freq_hz

uint32_t i2s_info_t::sampling_freq_hz

概要説明

サンプリング周波数 （ヘルツ単位）。

11.1.136 i2s_instance_t

```
typedef struct{
    i2s_ctrl_t *  p_ctrl
    i2s_cfg_t const *  p_cfg
    i2s_api_t const *  p_api
} i2s_instance_t
```

11.1.136.1 p_ctrl

i2s_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.136.2 p_cfg

i2s_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.136.3 p_api

i2s_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.137 input_capture_api_t

```
typedef struct{
    ssp_err_t(* open)(input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t const *const p_cfg)
    ssp_err_t(* disable)(input_capture_ctrl_t const *const p_ctrl)
    ssp_err_t(* enable)(input_capture_ctrl_t const *const p_ctrl)
    ssp_err_t(* infoGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_info_t *const p_info)
    ssp_err_t(* lastCaptureGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_capture_t *const p_counter)
    ssp_err_t(* close)(input_capture_ctrl_t *const p_ctrl)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} input_capture_api_t
```

11.1.138 input_capture_callback_args_t

```
typedef struct{
    uint8_t channel
    input_capture_event_t event
    uint32_t counter
    uint32_t overflows
    void const * p_context
} input_capture_callback_args_t
```

11.1.138.1 channel

uint8_t input_capture_callback_args_t::channel

概要説明

使用中のチャネル。

11.1.138.2 event

input_capture_event_t::event

概要説明

割り込みとコールバックを発生させたイベント。

11.1.138.3 counter

uint32_t input_capture_callback_args_t::counter

概要説明

割り込み時にキャプチャされたタイマの値。

11.1.138.4 overflows

uint32_t input_capture_callback_args_t::overflows

概要説明

この測定中に発生したカウンタ オーバーフローの回数。

11.1.138.5 p_context

void const* input_capture_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。p_context で設定されます。

11.1.139 input_capture_capture_t

```
typedef struct{
    uint32_t counter
    uint32_t overflows
} input_capture_capture_t
```

11.1.139.1 counter

uint32_t input_capture_capture_t::counter

概要説明

割り込み時にキャプチャされたタイマの値。

11.1.139.2 overflows

uint32_t input_capture_capture_t::overflows

概要説明

この測定中に発生したカウンタ オーバーフローの回数。

11.1.140 input_capture_cfg_t

```
typedef struct{
    uint8_t channel
    input_capture_mode_t mode
    input_capture_signal_edge_t edge
    input_capture_repetition_t repetition
    bool autostart
    void const * p_extend
    void(* p_callback)(input_capture_callback_args_t *p_args)
    void const * p_context
} input_capture_cfg_t
```

11.1.140.1 channel

uint8_t input_capture_cfg_t::channel

概要説明

使用中のチャネル。

11.1.140.2 mode

input_capture_mode_t::mode

概要説明

実行される測定モード。

11.1.140.3 edge

input_capture_signal_edge_t::edge

概要説明

測定を開始するトリガー エッジ （立ち上がりエッジ、立ち下がりエッジ、両エッジ）。

11.1.140.4 repetition

input_capture_repetition_t::repetition

概要説明

ユーザーのコールバック関数へのポインタ。

11.1.140.5 autostart

bool input_capture_cfg_t::autostart

概要説明

開いた後に、割り込みを有効にするかどうかの指定。

11.1.140.6 p_extend

void const* input_capture_cfg_t::p_extend

詳細説明

必須です。ペリフェラルに固有の拡張パラメータへのポインタ。GPT については gpt_input_capture_extend_t を参照してください。

11.1.140.7 p_callback

void(* input_capture_cfg_t::p_callback)(input_capture_callback_args_t *p_args)

詳細説明

割り込みを行わない場合は、NULL です。

11.1.140.8 p_context

void const* input_capture_cfg_t::p_context

概要説明

コールバックに渡される、ユーザーのコンテキスト データへのポインタ。

11.1.141 input_capture_ctrl_t

```
typedef struct{
    uint8_t channel
    input_capture_mode_t mode
    input_capture_repetition_t repetition
    uint32_t overflows_last
    uint32_t overflows_current
    void(* p_callback)(input_capture_callback_args_t *p_args)
    void const * p_context
} input_capture_ctrl_t
```

11.1.141.1 channel

uint8_t input_capture_ctrl_t::channel

概要説明

使用中のチャネル。

11.1.141.2 mode

input_capture_mode_t::mode

概要説明

実行されている測定モード。

11.1.141.3 repetition

input_capture_repetition_t::repetition

概要説明

ユーザーのコールバック関数へのポインタ。

11.1.141.4 overflows_last

uint32_t input_capture_ctrl_t::overflows_last

概要説明

最後の測定時に発生したオーバーフロー カウント。

11.1.141.5 overflows_current

uint32_t input_capture_ctrl_t::overflows_current

概要説明

現在の測定中に発生しているオーバーフローの累積カウント。

11.1.141.6 p_callback

void(* input_capture_ctrl_t::p_callback)(input_capture_callback_args_t *p_args)

概要説明

ユーザー コールバックへのポインタ。

11.1.141.7 p_context

void const* input_capture_ctrl_t::p_context

概要説明

コールバック関数に渡される、ユーザーのコンテキスト データへのポインタ。

11.1.142 input_capture_info_t

```
typedef struct{
    info_capture_status_t status
} input_capture_info_t
```

11.1.142.1 status

info_capture_status_t::status

概要説明

このキャプチャが実行中かどうかを示します。

11.1.143 input_capture_instance_t

```
typedef struct{
    input_capture_ctrl_t * p_ctrl
    input_capture_cfg_t const * p_cfg
    input_capture_api_t const * p_api
} input_capture_instance_t
```

11.1.143.1 p_ctrl

input_capture_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.143.2 p_cfg

input_capture_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.143.3 p_api

input_capture_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.144 ioport_api_t

```
typedef struct{
   ssp_err_t(* init)(const ioport_cfg_t *p_cfg)
   ssp_err_t(* pinCfg)(ioport_port_pin_t pin, uint32_t cfg)
   ssp_err_t(* pinDirectionSet)(ioport_port_pin_t pin, ioport_direction_t direction)
   ssp_err_t(* pinEventInputRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_event)
   ssp_err_t(* pinEventOutputWrite)(ioport_port_pin_t pin, ioport_level_t pin_value)
   ssp_err_t(* pinEthernetModeCfg)(ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)
   ssp_err_t(* pinRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_value)
   ssp_err_t(* pinWrite)(ioport_port_pin_t pin, ioport_level_t level)
   ssp_err_t(* portDirectionSet)(ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask)
   ssp_err_t(* portEventInputRead)(ioport_port_t port, ioport_size_t *p_event_data)
   ssp_err_t(* portEventOutputWrite)(ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
   ssp_err_t(* portRead)(ioport_port_t port, ioport_size_t *p_port_value)
   ssp_err_t(* portWrite)(ioport_port_t port, ioport_size_t value, ioport_size_t mask)
   ssp_err_t(* versionGet)(ssp_version_t *p_data)
} ioport_api_t
```

11.1.145 ioport_cfg_t

```
typedef struct{
   uint16_t number_of_pins
   ioport_pin_cfg_t const * p_pin_cfg_data
} ioport_cfg_t
```

11.1.145.1 number_of_pins

uint16_t ioport_cfg_t::number_of_pins

概要説明

設定データが存在するピンの数。

11.1.145.2 p_pin_cfg_data

ioport_pin_cfg_t::p_pin_cfg_data

概要説明

ピンの設定データ。

11.1.146 ioport_instance_t

```
typedef struct{
    ioport_cfg_t const *  p_cfg
    ioport_api_t const *  p_api
} ioport_instance_t
```

11.1.146.1 p_cfg

ioport_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.146.2 p_api

ioport_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.147 ioport_pin_cfg_t

```
typedef struct{
    uint32_t  pin_cfg
    ioport_port_pin_t  pin
} ioport_pin_cfg_t
```

11.1.147.1 pin_cfg

uint32_t ioport_pin_cfg_t::pin_cfg

概要説明

ioport_cfg_options_t パラメータを使用して設定します。

11.1.147.2 pin

ioport_port_pin_t ioport_pin_cfg_t::pin

概要説明

ピンの識別子。

11.1.148 jpeg_decode_api_t

```
typedef struct{
   ssp_err_t(* open)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)
   ssp_err_t(* outputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
   ssp_err_t(* horizontalStrideSet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
   ssp_err_t(* imageSubsampleSet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample,
jpeg_decode_subsample_t vertical_subsample)
   ssp_err_t(* inputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
   ssp_err_t(* linesDecodedGet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
   ssp_err_t(* imageSizeGet)(jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
   ssp_err_t(* statusGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
   ssp_err_t(* close)(jpeg_decode_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *p_version)
   ssp_err_t(* pixelFormatGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)
} jpeg_decode_api_t
```

11.1.149 jpeg_decode_callback_args_t

```
typedef struct{
   jpeg_decode_status_t status
   void const * p_context
} jpeg_decode_callback_args_t
```

11.1.149.1 status

jpeg_decode_status_t::status

概要説明

JPEG のステータス。

11.1.149.2 p_context

void const* jpeg_decode_callback_args_t::p_context

概要説明

ユーザー定義のコンテキストへのポインタ。

11.1.150 jpeg_decode_cfg_t

```
typedef struct{
   jpeg_decode_color_space_t color_space
   jpeg_decode_data_format_t input_data_format
   jpeg_decode_data_format_t output_data_format
   jpeg_decode_pixel_format_t pixel_format
   uint8_t alpha_value
   void(* p_callback)(jpeg_decode_callback_args_t *p_args)
   void const * p_context
} jpeg_decode_cfg_t
```

11.1.150.1 color_space

jpeg_decode_color_space_t::color_space

概要説明

色空間。

11.1.150.2 input_data_format

jpeg_decode_data_format_t::input_data_format

概要説明

入力データ ストリームのバイト順序。

11.1.150.3 output_data_format

jpeg_decode_data_format_t::output_data_format

概要説明

出力データ ストリームのバイト順序。

11.1.150.4 pixel_format

jpeg_decode_pixel_format_t::pixel_format

概要説明

ピクセル フォーマット。

11.1.150.5 alpha_value

uint8_t jpeg_decode_cfg_t::alpha_value

概要説明

デコードされたピクセル データに適用されるアルファ値。これのみです。

11.1.150.6 p_callback

void(* jpeg_decode_cfg_t::p_callback)(jpeg_decode_callback_args_t *p_args)

概要説明

ユーザー定義のコールバック関数。

11.1.150.7 p_context

void const* jpeg_decode_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。jpeg_decode_callback_args_t 内のユーザー コールバックに渡されます。

11.1.151 jpeg_decode_ctrl_t

```
typedef struct{
  jpeg_decode_status_t status
  ssp_err_t error_code
  void(* p_callback)(jpeg_decode_callback_args_t *p_args)
  void const * p_extend
  void const * p_context
  jpeg_decode_pixel_format_t pixel_format
  uint32_t horizontal_stride
  uint32_t outbuffer_size
  uint32_t total_lines_decoded
} jpeg_decode_ctrl_t
```

11.1.151.1 status

jpeg_decode_status_t::status

概要説明

JPEG コーデック モジュールのステータス。

11.1.151.2 error_code

ssp_err_t jpeg_decode_ctrl_t::error_code

概要説明

JPEG コーデックのエラー コード （ある場合）。

11.1.151.3 p_callback

void(* jpeg_decode_ctrl_t::p_callback)(jpeg_decode_callback_args_t *p_args)

概要説明

ユーザー定義のコールバック関数。

11.1.151.4 p_extend

void const* jpeg_decode_ctrl_t::p_extend

概要説明

JPEG コーデックのハードウェアに依存する設定 */。

11.1.151.5 p_context

void const* jpeg_decode_ctrl_t::p_context

概要説明

ユーザー データのプレースホルダー。jpeg_decode_callback_args_t 内のユーザー コールバックに渡されます。

11.1.151.6 pixel_format

jpeg_decode_pixel_format_t::pixel_format

概要説明

ピクセル フォーマット。

11.1.151.7 horizontal_stride

uint32_t jpeg_decode_ctrl_t::horizontal_stride

概要説明

水平ストライドの設定値。

11.1.151.8 outbuffer_size

uint32_t jpeg_decode_ctrl_t::outbuffer_size

概要説明

出力バッファのサイズ

11.1.151.9 total_lines_decoded

uint32_t jpeg_decode_ctrl_t::total_lines_decoded

概要説明

それまでにデコードされたライン数を追跡します。

11.1.152 jpeg_decode_instance_t

```
typedef struct{
    jpeg_decode_ctrl_t * p_ctrl
    jpeg_decode_cfg_t const * p_cfg
    jpeg_decode_api_t const * p_api
} jpeg_decode_instance_t
```

11.1.152.1 p_ctrl

jpeg_decode_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.152.2 p_cfg

jpeg_decode_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.152.3 p_api

jpeg_decode_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.153 keymatrix_api_t

```
typedef struct{
    ssp_err_t(* open)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)
    ssp_err_t(* enable)(keymatrix_ctrl_t *const p_ctrl)
    ssp_err_t(* disable)(keymatrix_ctrl_t *const p_ctrl)
    ssp_err_t(* triggerSet)(keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t const trigger)
    ssp_err_t(* close)(keymatrix_ctrl_t *const p_ctrl)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} keymatrix_api_t
```

11.1.154 keymatrix_callback_args_t

```
typedef struct{
    void const * p_context
    keymatrix_channels_t channels
} keymatrix_callback_args_t
```

11.1.154.1 p_context

void const* keymatrix_callback_args_t::p_context

概要説明

ユーザー データのホルダー。keymatrix_cfg_t の open 関数で設定されます。

11.1.154.2 channels

keymatrix_channels_t::channels

詳細説明

割り込みを発生させた物理ハードウェア チャネルを表すビット ベクター。このビット ベクターは、同時に複数の入力がアクティブになるマトリクス設計との互換性のために使用されます。
I :HAL ドライバがすべてマトリクス モードに対応しているわけではありません。r_kint.h を参照してください。

11.1.155 keymatrix_cfg_t

```
typedef struct{
    keymatrix_channels_t channels
    keymatrix_trigger_t trigger
    bool autostart
    void(* p_callback)(keymatrix_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} keymatrix_cfg_t
```

11.1.155.1 channels

keymatrix_channels_t::channels

概要説明

キー入力チャネル。開くチャネルを指定するためのビット マスクです。

11.1.155.2 trigger

keymatrix_trigger_t::trigger

概要説明

キー入力のトリガー設定値。

11.1.155.3 autostart

bool keymatrix_cfg_t::autostart

概要説明

open() の実行中に操作を開始し、割り込みを有効にします。

11.1.155.4 p_callback

void(* keymatrix_cfg_t::p_callback)(keymatrix_callback_args_t *p_args)

概要説明

キー割り込み ISR のコールバック。

11.1.155.5 p_context

void const* keymatrix_cfg_t::p_context

概要説明

ユーザー データのホルダー。keymatrix_user_cb_data_t 内のコールバックに渡されます。

11.1.155.6 p_extend

void const* keymatrix_cfg_t::p_extend

概要説明

ハードウェア固有の設定値に対応するための拡張パラメータです。

11.1.156 keymatrix_ctrl_t

```
typedef struct{
    keymatrix_channels_t channels
} keymatrix_ctrl_t
```

11.1.156.1 channels

keymatrix_channels_t::channels

11.1.157 keymatrix_instance_t

```
typedef struct{
    keymatrix_ctrl_t * p_ctrl
    keymatrix_cfg_t const * p_cfg
    keymatrix_api_t const * p_api
} keymatrix_instance_t
```

11.1.157.1 p_ctrl

keymatrix_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.157.2 p_cfg

keymatrix_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.157.3 p_api

keymatrix_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.158 lpm_api_t

```
typedef struct{
   ssp_err_t(* init)(lpm_cfg_t const *const p_cfg)
   ssp_err_t(* mstpcrSet)(uint32_t mstpcra_value, uint32_t mstpcrb_value, uint32_t mstpcrc_value, uint32_t mstpcrd_value)
   ssp_err_t(* mstpcrGet)(uint32_t *mstpcra_value, uint32_t *mstpcrb_value, uint32_t *mstpcrc_value, uint32_t *mstpcrd_value)
   ssp_err_t(* moduleStop)(lpm_mstp_t module)
   ssp_err_t(* moduleStart)(lpm_mstp_t module)
   ssp_err_t(* operatingPowerModeSet)(lpm_operating_power_t power_mode, lpm_subosc_t subosc)
   ssp_err_t(* snoozeEnable)(lpm_snooze_rxd0_t rdx0_mode, lpm_snooze_dtc_t dtc_mode, lpm_snooze_request_t requests,
lpm_snooze_end_t triggers)
   ssp_err_t(* snoozeDisable)(void)
   ssp_err_t(* lowPowerCfg)(lpm_low_power_mode_t power_mode, lpm_output_port_enable_t output_port_enable,
lpm_power_supply_t power_supply, lpm_io_port_t io_port_state)
   ssp_err_t(* wupenSet)(uint32_t wupen_value)
   ssp_err_t(* wupenGet)(uint32_t *wupen_value)
   ssp_err_t(* deepStandbyCancelRequestEnable)(lpm_deep_standby_t pin_signal, lpm_cancel_request_edge_t rising_falling)
   ssp_err_t(* deepStandbyCancelRequestDisable)(lpm_deep_standby_t pin_signal)
   ssp_err_t(* lowPowerModeEnter)(void)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} lpm_api_t
```

11.1.159 lpm_cfg_t

```
typedef struct{
   lpm_operating_power_t operating_power
   lpm_subosc_t sub_oscillator
   lpm_code_flash_t code_flash
} lpm_cfg_t
```

11.1.159.1 operating_power

lpm_operating_power_t::operating_power

概要説明

動作電力モード。

11.1.159.2 sub_oscillator

lpm_subosc_t::sub_oscillator

概要説明

サブオシレーター。

11.1.159.3 code_flash

lpm_code_flash_t::code_flash

概要説明

コード フラッシュを有効にします。

11.1.160 lpm_instance_t

```
typedef struct{
    lpm_cfg_t const *  p_cfg
    lpm_api_t const *  p_api
} lpm_instance_t
```

11.1.160.1 p_cfg

lpm_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.160.2 p_api

lpm_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.161 lvd_api_t

```
typedef struct{
    ssp_err_t(*  open)(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)
    ssp_err_t(*  statusGet)(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
    ssp_err_t(*  statusClear)(lvd_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(lvd_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} lvd_api_t
```

11.1.162 lvd_callback_args_t

```
typedef struct{
   const uint32_t monitor_number
   lvd_status_t status
   void const * p_context
} lvd_callback_args_t
```

11.1.162.1 monitor_number

const uint32_t lvd_callback_args_t::monitor_number

概要説明

監視番号。

11.1.162.2 status

lvd_status_t::status

概要説明

監視のステータス。

11.1.162.3 p_context

void const* lvd_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.163 lvd_cfg_t

```
typedef struct{
   const uint32_t monitor_number
   lvd_threshold_t voltage_threshold
   lvd_response_t detection_response
   lvd_voltage_slope_t voltage_slope
   void(* p_callback)(lvd_callback_args_t *p_args)
   void const * p_context
   void const * p_extend
} lvd_cfg_t
```

11.1.163.1 monitor_number

const uint32_t lvd_cfg_t::monitor_number

詳細説明

監視番号 1、2、…

11.1.163.2 voltage_threshold

lvd_threshold_t::voltage_threshold

詳細説明

範囲外の電圧検出しきい値

11.1.163.3 detection_response

lvd_response_t::detection_response

詳細説明

しきい値調査検出に対するレスポンス

11.1.163.4 voltage_slope

lvd_voltage_slope_t::voltage_slope

詳細説明

電圧の上昇または下降が検出されます

11.1.163.5 p_callback

void(* lvd_cfg_t::p_callback)(lvd_callback_args_t *p_args)

詳細説明

割り込みから呼び出されるユーザー関数

11.1.163.6 p_context

void const* lvd_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。以下に含まれるユーザー コールバックに渡されます :

11.1.163.7 p_extend

void const* lvd_cfg_t::p_extend

詳細説明

ハードウェア固有の設定値に対応するための拡張パラメータ

11.1.164 lvd_ctrl_t

```
typedef struct{
   uint32_t monitor_number
} lvd_ctrl_t
```

11.1.164.1 monitor_number

uint32_t lvd_ctrl_t::monitor_number

詳細説明

監視番号 1、2、…

11.1.165 lvd_extend_t

```
typedef struct{
   lvd_negation_delay_t negation_delay
   lvd_sample_clock_t sample_clock_divisor
} lvd_extend_t
```

11.1.165.1 negation_delay

lvd_negation_delay_t::negation_delay

詳細説明

LVD 信号のネゲートは、リセットに続いて、または電圧が範囲内になった後で行われます

11.1.165.2 sample_clock_divisor

lvd_sample_clock_t::sample_clock_divisor

詳細説明

サンプル クロック分周器、デジタル フィルタリングを無効化するには LVD_SAMPLE_CLOCK_DISABLED を使用します

11.1.166 lvd_instance_t

```
typedef struct{
    lvd_ctrl_t * p_ctrl
    lvd_cfg_t const * p_cfg
    lvd_api_t const * p_api
} lvd_instance_t
```

11.1.166.1 p_ctrl

lvd_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.166.2 p_cfg

lvd_cfg_t::p_cfg

概要説明

インタフェース インスタンスの構成構造体へのポインタ。

11.1.166.3 p_api

lvd_api_t::p_api

概要説明

インタフェース インスタンスの API 構造体へのポインタ。

11.1.167 lvd_status_t

```
typedef struct{
    lvd_threshold_crossing_t crossing_detected
    lvd_current_state_t current_state
} lvd_status_t
```

11.1.167.1 crossing_detected

lvd_threshold_crossing_t::crossing_detected

詳細説明

しきい値超過検出 （ラッチ）

11.1.167.2 current_state

lvd_current_state_t::current_state

詳細説明

監視対象となる電圧の瞬間的なステータス （しきい値を上回るまたは下回る）

11.1.168 pdc_api_t

```
typedef struct{
   ssp_err_t(*  open)(pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)
   ssp_err_t(*  close)(pdc_ctrl_t *const p_ctrl)
   ssp_err_t(*  captureStart)(pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)
   ssp_err_t(*  stateGet)(pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)
   ssp_err_t(*  versionGet)(ssp_version_t *const p_data)
} pdc_api_t
```

11.1.169 pdc_callback_args_t

```
typedef struct{
   pdc_event_t  event
   uint8_t *  p_buffer
   void const *  p_context
} pdc_callback_args_t
```

11.1.169.1 event

pdc_event_t::event

概要説明

コールバックを生じさせるイベント。

11.1.169.2 p_buffer

uint8_t* pdc_callback_args_t::p_buffer

概要説明

キャプチャしたデータを含むバッファへのポインタ。

11.1.169.3 p_context

void const* pdc_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。pdc_cfg_t の open 関数で設定されます。

11.1.170 pdc_cfg_t

```
typedef struct{
  uint8_t bytes_per_pixel
  uint16_t x_capture_start_pixel
  uint16_t x_capture_pixels
  uint16_t y_capture_start_pixel
  uint16_t y_capture_pixels
  pdc_clock_division_t clock_division
  pdc_endian_t endian
  pdc_hsync_polarity_t hsync_polarity
  pdc_vsync_polarity_t vsync_polarity
  uint8_t * p_buffer
  transfer_instance_t const *const p_lower_lvl_transfer
  void(* p_callback)(pdc_callback_args_t *p_args)
  void const * p_context
  void const * p_extend
} pdc_cfg_t
```

11.1.170.1 bytes_per_pixel

uint8_t pdc_cfg_t::bytes_per_pixel

概要説明

ピクセルごとのバイト数。

11.1.170.2 x_capture_start_pixel

uint16_t pdc_cfg_t::x_capture_start_pixel

概要説明

キャプチャを開始する水平位置。

11.1.170.3 x_capture_pixels

uint16_t pdc_cfg_t::x_capture_pixels

概要説明

キャプチャする水平ピクセル数。

11.1.170.4 y_capture_start_pixel

uint16_t pdc_cfg_t::y_capture_start_pixel

概要説明

キャプチャを開始する垂直位置。

11.1.170.5 y_capture_pixels

uint16_t pdc_cfg_t::y_capture_pixels

概要説明

キャプチャする垂直ライン / ピクセルの数。

11.1.170.6 clock_division

pdc_clock_division_t::clock_division

概要説明

クロック分周器。

11.1.170.7 endian

pdc_endian_t::endian

概要説明

キャプチャするデータのエンディアン。

11.1.170.8 hsync_polarity

pdc_hsync_polarity_t::hsync_polarity

概要説明

HSYNC 入力の極性。

11.1.170.9 vsync_polarity

pdc_vsync_polarity_t::vsync_polarity

概要説明

VSYNC 入力の極性。

11.1.170.10 p_buffer

uint8_t* pdc_cfg_t::p_buffer

概要説明

画像書き込み先バッファへのポインタ。

11.1.170.11 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

PDC が使用する転送インスタンスへのポインタ。

11.1.170.12 p_callback

void(* pdc_cfg_t::p_callback)(pdc_callback_args_t *p_args)

概要説明

PDC 転送 ISR の発生時に提供されるコールバック。

11.1.170.13 p_context

void const* pdc_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。pdc_callback_args_t 内のユーザー コールバックに渡されます。

11.1.170.14 p_extend

void const* pdc_cfg_t::p_extend

11.1.171 pdc_ctrl_t

```
typedef struct{
    bool pdc_open
    uint8_t bytes_per_pixel
    uint16_t x_resolution_pixels
    uint16_t y_resolution_pixels
    uint16_t x_capture_start_pixel
    uint16_t x_capture_pixels
    uint16_t y_capture_start_pixel
    uint16_t y_capture_pixels
    pdc_endian_t endian
    pdc_hsync_polarity_t hsync_polarity
    pdc_vsync_polarity_t vsync_polarity
    uint8_t * p_current_buffer
    bool transfer_in_progress
    transfer_instance_t const * p_lower_lvl_transfer
    transfer_info_t info_transfer
    void const * p_context
    void(* p_callback)(pdc_callback_args_t *p_args)
} pdc_ctrl_t
```

11.1.171.1 pdc_open

bool pdc_ctrl_t::pdc_open

詳細説明

open() API が正常に呼び出されたかどうかを示します。

11.1.171.2 bytes_per_pixel

uint8_t pdc_ctrl_t::bytes_per_pixel

概要説明

ピクセルごとのバイト数。

11.1.171.3 x_resolution_pixels

uint16_t pdc_ctrl_t::x_resolution_pixels

概要説明

PDC への水平ピクセル入力の合計数。

11.1.171.4 y_resolution_pixels

uint16_t pdc_ctrl_t::y_resolution_pixels

概要説明

PDC へのライン入力の合計数。

11.1.171.5 x_capture_start_pixel

uint16_t pdc_ctrl_t::x_capture_start_pixel

概要説明

キャプチャを開始する水平位置。

11.1.171.6 x_capture_pixels

uint16_t pdc_ctrl_t::x_capture_pixels

概要説明

キャプチャする水平ピクセル数。

11.1.171.7 y_capture_start_pixel

uint16_t pdc_ctrl_t::y_capture_start_pixel

概要説明

キャプチャを開始する垂直位置。

11.1.171.8 y_capture_pixels

uint16_t pdc_ctrl_t::y_capture_pixels

概要説明

キャプチャする垂直ライン / ピクセルの数。

11.1.171.9 endian

pdc_endian_t::endian

概要説明

キャプチャするデータのエンディアン。

11.1.171.10 hsync_polarity

pdc_hsync_polarity_t::hsync_polarity

概要説明

HSYNC 入力の極性。

11.1.171.11 vsync_polarity

pdc_vsync_polarity_t::vsync_polarity

概要説明

VSYNC 入力の極性。

11.1.171.12 p_current_buffer

uint8_t* pdc_ctrl_t::p_current_buffer

概要説明

現在使用中のバッファへのポインタ。

11.1.171.13 transfer_in_progress

bool pdc_ctrl_t::transfer_in_progress

概要説明

PDC 転送がすでに実行中であるかどうかを示します。

11.1.171.14 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

PDC が使用する転送インスタンスへのポインタ。

11.1.171.15 info_transfer

transfer_info_t::info_transfer

概要説明

ローレベル転送インタフェースに対する転送情報構造体。

11.1.171.16 p_context

void const* pdc_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。pdc_callback_args_t 内のユーザー コールバックに渡されます。

11.1.171.17 p_callback

void(* pdc_ctrl_t::p_callback)(pdc_callback_args_t *p_args)

概要説明

PDC 転送 ISR の発生時に提供されるコールバック。

11.1.172 pdc_instance_t

```
typedef struct{
    pdc_ctrl_t *  p_ctrl
    pdc_cfg_t const *  p_cfg
    pdc_api_t const *  p_api
} pdc_instance_t
```

11.1.172.1 p_ctrl

pdc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.172.2 p_cfg

pdc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.172.3 p_api

pdc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.173 pdc_state_t

```
typedef struct{
   pdc_vsync_state_t vsync
   pdc_hsync_state_t hsync
} pdc_state_t
```

11.1.173.1 vsync

pdc_vsync_state_t::vsync

概要説明

VSYNC の信号状態。

11.1.173.2 hsync

pdc_hsync_state_t::hsync

概要説明

HSYNC の信号状態。

11.1.174 qspi_api_t

```
typedef struct{
   ssp_err_t(* open)(qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)
   ssp_err_t(* close)(qspi_ctrl_t *p_ctrl)
   ssp_err_t(* read)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
   ssp_err_t(* pageProgram)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
   ssp_err_t(* sectorErase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)
   ssp_err_t(* statusGet)(qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)
   ssp_err_t(* bankSelect)(uint32_t bank)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} qspi_api_t
```

11.1.175 qspi_cfg_t

```
typedef struct{
   void * p_extend
} qspi_cfg_t
```

11.1.175.1 p_extend

void* qspi_cfg_t::p_extend

概要説明

将来の開発用のプレース ホルダー

11.1.176 qspi_ctrl_t

```
typedef struct{
  uint8_t manufacturer_id
  uint8_t memory_type
  uint8_t memory_capacity
  uint32_t max_eraseable_size
  uint32_t num_address_bytes
  uint32_t spi_mode
  uint32_t page_size
  bool xip_mode
} qspi_ctrl_t
```

11.1.176.1 manufacturer_id

uint8_t qspi_ctrl_t::manufacturer_id

概要説明

メーカー ID。

11.1.176.2 memory_type

uint8_t qspi_ctrl_t::memory_type

概要説明

メモリ タイプ。

11.1.176.3 memory_capacity

uint8_t qspi_ctrl_t::memory_capacity

概要説明

メモリ容量 （MB 単位）

11.1.176.4 max_eraseable_size

uint32_t qspi_ctrl_t::max_eraseable_size

概要説明

消去可能な最大セクター （KB 単位）バッファ サイズの判定に使用されます。消去可能な最大セクター（KB 単位）バッファ サイズの判定に使用されます。

11.1.176.5 num_address_bytes

uint32_t qspi_ctrl_t::num_address_bytes

概要説明

アドレスの表現に使用するバイト数。

11.1.176.6 spi_mode

uint32_t qspi_ctrl_t::spi_mode

概要説明

0 = 拡張、1 = デュアル、2 = クワッド。

11.1.176.7 page_size

uint32_t qspi_ctrl_t::page_size

概要説明

プログラミング可能なページのバイト数。

11.1.176.8 xip_mode

bool qspi_ctrl_t::xip_mode

概要説明

0 = 読み取りモードで実行、1 = XIP モードで実行

11.1.177 qspi_instance_t

```
typedef struct{
    qspi_ctrl_t * p_ctrl
    qspi_cfg_t const * p_cfg
    qspi_api_t const * p_api
} qspi_instance_t
```

11.1.177.1 p_ctrl

qspi_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.177.2 p_cfg

qspi_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.177.3 p_api

qspi_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.178 rsa_api_t

```
typedef struct{
    uint32_t(* open)(rsa_ctrl_t *const p_ctrl, rsa_cfg_t const *const p_cfg)
    uint32_t(* encrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_source, uint32_t *p_dest)
    uint32_t(* decrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_source, uint32_t *p_dest)
    uint32_t(* decryptCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_source, uint32_t *p_dest)
    uint32_t(* verify)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_signature, uint32_t *p_padded_hash)
    uint32_t(* sign)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_padded_hash, uint32_t *p_dest)
    uint32_t(* signCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t
*p_padded_hash, uint32_t *p_dest)
    uint32_t(* close)(rsa_ctrl_t *const p_ctrl)
    uint32_t(* versionGet)(ssp_version_t *const p_version)
} rsa_api_t
```

11.1.179 rsa_cfg_t

```
typedef struct{
    crypto_api_t const * p_crypto_api
} rsa_cfg_t
```

11.1.179.1 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.180 rsa_ctrl_t

```
typedef struct{
   crypto_ctrl_t * p_crypto_ctrl
   crypto_api_t const * p_crypto_api
   uint32_t stage_num
} rsa_ctrl_t
```

11.1.180.1 p_crypto_ctrl

crypto_ctrl_t::p_crypto_ctrl

概要説明

暗号化エンジン制御構造体へのポインタ

11.1.180.2 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.180.3 stage_num

uint32_t rsa_ctrl_t::stage_num

概要説明

処理段階

11.1.181 rsa_instance_t

```
typedef struct{
   rsa_ctrl_t * p_ctrl
   rsa_cfg_t const * p_cfg
   rsa_api_t const * p_api
} rsa_instance_t
```

11.1.181.1 p_ctrl

rsa_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.181.2 p_cfg

rsa_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.181.3 p_api

rsa_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.182 rspi_access_delay_t

```
typedef struct{
   rspi_next_access_delay_count_t rspi_next_access_delay_count
   rspi_next_access_delay_state_t rspi_next_access_delay_state
} rspi_access_delay_t
```

11.1.182.1 rspi_next_access_delay_count

rspi_next_access_delay_count_t::rspi_next_access_delay_count

11.1.182.2 rspi_next_access_delay_state

rspi_next_access_delay_state_t::rspi_next_access_delay_state

11.1.183 rspi_clock_delay_t

```
typedef struct{
   rspi_clock_delay_count_t rspi_clock_delay_count
   rspi_clock_delay_state_t rspi_clock_delay_state
} rspi_clock_delay_t
```

11.1.183.1 rspi_clock_delay_count

rspi_clock_delay_count_t::rspi_clock_delay_count

11.1.183.2 rspi_clock_delay_state

rspi_clock_delay_state_t::rspi_clock_delay_state

11.1.184 rspi_loopback_t

```
typedef struct{
   rspi_loopback1_t rspi_loopback1
   rspi_loopback2_t rspi_loopback2
} rspi_loopback_t
```

11.1.184.1 rspi_loopback1

rspi_loopback1_t::rspi_loopback1

11.1.184.2 rspi_loopback2

rspi_loopback2_t::rspi_loopback2

11.1.185 rspi_mosi_idle_t

```
typedef struct{
   rspi_mosi_idle_fixed_val_t rspi_mosi_idle_fixed_val
   rspi_mosi_idle_val_fixing_t rspi_mosi_idle_val_fixing
} rspi_mosi_idle_t
```

11.1.185.1 rspi_mosi_idle_fixed_val

rspi_mosi_idle_fixed_val_t::rspi_mosi_idle_fixed_val

11.1.185.2 rspi_mosi_idle_val_fixing

rspi_mosi_idle_val_fixing_t::rspi_mosi_idle_val_fixing

11.1.186 rspi_parity_t

```
typedef struct{
   rspi_parity_state_t rspi_parity
   rspi_parity_mode_t rspi_parity_mode
} rspi_parity_t
```

11.1.186.1 rspi_parity

rspi_parity_state_t::rspi_parity

11.1.186.2 rspi_parity_mode

rspi_parity_mode_t::rspi_parity_mode

11.1.187 rspi_ssl_negation_delay_t

```
typedef struct{
    rspi_ssl_negation_delay_count_t rspi_ssl_neg_delay_count
    rspi_ssl_negation_delay_state_t rspi_ssl_neg_delay_state
} rspi_ssl_negation_delay_t
```

11.1.187.1 rspi_ssl_neg_delay_count

rspi_ssl_negation_delay_count_t::rspi_ssl_neg_delay_count

11.1.187.2 rspi_ssl_neg_delay_state

rspi_ssl_negation_delay_state_t::rspi_ssl_neg_delay_state

11.1.188 rspi_ssl_polarity_t

```
typedef struct{
    rspi_sslp_t rspi_ssl2
    rspi_sslp_t rspi_ssl3
    rspi_sslp_t rspi_ssl0
    rspi_sslp_t rspi_ssl1
} rspi_ssl_polarity_t
```

11.1.188.1 rspi_ssl2

rspi_sslp_t::rspi_ssl2

11.1.188.2 rspi_ssl3

rspi_sslp_t::rspi_ssl3

11.1.188.3 rspi_ssl0

rspi_sslp_t::rspi_ssl0

11.1.188.4 rspi_ssl1

rspi_sslp_t::rspi_ssl1

11.1.189 rtc_alarm_time_t

```
typedef struct{
   rtc_time_t time
   bool sec_match
   bool min_match
   bool hour_match
   bool mday_match
   bool mon_match
   bool year_match
   bool dayofweek_match
} rtc_alarm_time_t
```

11.1.189.1 time

rtc_time_t::time

概要説明

時間構造体。

11.1.189.2 sec_match

bool rtc_alarm_time_t::sec_match

概要説明

秒フィールドが一致した場合、アラームを有効にします。

11.1.189.3 min_match

bool rtc_alarm_time_t::min_match

概要説明

分フィールドが一致した場合、アラームを有効にします。

11.1.189.4 hour_match

bool rtc_alarm_time_t::hour_match

概要説明

時刻フィールドが一致した場合、アラームを有効にします。

11.1.189.5 mday_match

bool rtc_alarm_time_t::mday_match

概要説明

日付フィールドが一致した場合、アラームを有効にします。

11.1.189.6 mon_match

bool rtc_alarm_time_t::mon_match

概要説明

月フィールドが一致した場合、アラームを有効にします。

11.1.189.7 year_match

bool rtc_alarm_time_t::year_match

概要説明

年フィールドが一致した場合、アラームを有効にします。

11.1.189.8 dayofweek_match

bool rtc_alarm_time_t::dayofweek_match

概要説明

曜日フィールドが一致した場合、アラームを有効にします。

11.1.190 rtc_api_t

```
typedef struct{
    ssp_err_t(* open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
    ssp_err_t(* close)(rtc_ctrl_t *const p_ctrl)
    ssp_err_t(* calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time, bool clock_start)
    ssp_err_t(* calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)
    ssp_err_t(* calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm, bool irq_enable_flag)
    ssp_err_t(* calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm)
    ssp_err_t(* calendarCounterStart)(rtc_ctrl_t *const p_ctrl)
    ssp_err_t(* calendarCounterStop)(rtc_ctrl_t *const p_ctrl)
    ssp_err_t(* irqEnable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
    ssp_err_t(* irqDisable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
    ssp_err_t(* periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t rate)
    ssp_err_t(* infoGet)(rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)
    ssp_err_t(* versionGet)(ssp_version_t *version)
} rtc_api_t
```

11.1.191 rtc_callback_args_t

```
typedef struct{
   rtc_event_t event
   void const * p_context
} rtc_callback_args_t
```

11.1.191.1 event

rtc_event_t::event

概要説明

このイベントを使用して、コールバックの原因 （コンペア マッチまたはエラー）を特定できます。

11.1.191.2 p_context

void const* rtc_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。Set in r_timer_t::open function in timer_cfg_t 内の r_timer_t::open 関数で設定されます。

11.1.192 rtc_cfg_t

```
typedef struct{
   rtc_clock_source_t clock_source
   uint32_t error_adjustment_value
   rtc_error_adjustment_t error_adjustment_type
   void(* p_callback)(rtc_callback_args_t *p_args)
   void const * p_context
   void const * p_extend
} rtc_cfg_t
```

11.1.192.1 clock_source

rtc_clock_source_t::clock_source

概要説明

RTC ブロックのクロック ソース。

11.1.192.2 error_adjustment_value

uint32_t rtc_cfg_t::error_adjustment_value

概要説明

エラー調整用のプリスケーラーの値。

11.1.192.3 error_adjustment_type

rtc_error_adjustment_t::error_adjustment_type

概要説明

プリスケーラの値の適用方法。

11.1.192.4 p_callback

void(* rtc_cfg_t::p_callback)(rtc_callback_args_t *p_args)

概要説明

ISR から呼び出されます。

11.1.192.5 p_context

void const* rtc_cfg_t::p_context

概要説明

コールバックに渡されます。

11.1.192.6 p_extend

void const* rtc_cfg_t::p_extend

概要説明

RTC ハードウェアに依存する設定。

11.1.193 rtc_ctrl_t

```
typedef struct{
    rtc_clock_source_t clock_source
} rtc_ctrl_t
```

11.1.193.1 clock_source

rtc_clock_source_t::clock_source

概要説明

RTC ブロックのクロック ソース。

11.1.194 rtc_info_t

```
typedef struct{
    rtc_clock_source_t clock_source
} rtc_info_t
```

11.1.194.1 clock_source

rtc_clock_source_t::clock_source

概要説明

RTC ブロックのクロック ソース。

11.1.195 rtc_instance_t

```
typedef struct{
    rtc_ctrl_t * p_ctrl
    rtc_cfg_t const * p_cfg
    rtc_api_t const * p_api
} rtc_instance_t
```

11.1.195.1 p_ctrl

rtc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.195.2 p_cfg

rtc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.195.3 p_api

rtc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.196 sci_ctrl_t

```
typedef struct{
   sci_mode_t mode
   bool  tx_busy
   void * p_context
   void(* p_extpin_ctrl)(uint32_t channel, uint32_t level)
   bsp_lock_t resource_lock_tx
   bsp_lock_t resource_lock_rx
} sci_ctrl_t
```

11.1.196.1 mode

sci_mode_t::mode

詳細説明

動作モード

11.1.196.2 tx_busy

bool ::tx_busy

詳細説明

送信ビジー フラグ

11.1.196.3 p_context

void* ::p_context

詳細説明

関数レベルのデバイス コンテキスト （例 : uart_ctrl_t 型のデータ）へのポインタ

11.1.196.4 p_extpin_ctrl

void(* ::p_extpin_ctrl)(uint32_t channel, uint32_t level)

詳細説明

外部ピン制御

11.1.196.5 resource_lock_tx

bsp_lock_t::resource_lock_tx

詳細説明

送信用のリソース ロック

11.1.196.6 resource_lock_rx

bsp_lock_t::resource_lock_rx

詳細説明

受信用のリソース ロック

11.1.197 sdmmc_api_t

```
typedef struct{
   ssp_err_t(* open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)
   ssp_err_t(* close)(sdmmc_ctrl_t *const p_ctrl)
   ssp_err_t(* read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
   ssp_err_t(* write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
   ssp_err_t(* control)(sdmmc_ctrl_t *const p_ctrl, ssp_command_t const command, void *p_data)
   ssp_err_t(* readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
   ssp_err_t(* writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address,
sdmmc_io_write_mode_t const read_after_write)
   ssp_err_t(* readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t
*const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
   ssp_err_t(* writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address,
uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
   ssp_err_t(* IoIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
   ssp_err_t(* infoGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)
   ssp_err_t(* erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)
} sdmmc_api_t
```

11.1.198 sdmmc_callback_args_t

```
typedef struct{
   sdmmc_event_t event
   void const * p_context
} sdmmc_callback_args_t
```

11.1.198.1 event

sdmmc_event_t::event

概要説明

このイベントは、コールバックの原因を特定するために使用できます。

11.1.198.2 p_context

void const* sdmmc_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.199 sdmmc_cfg_t

```
typedef struct{
    sdmmc_hw_t hw
    transfer_instance_t const * p_lower_lvl_transfer
    void(* p_callback)(sdmmc_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} sdmmc_cfg_t
```

11.1.199.1 hw

sdmmc_hw_t::hw

概要説明

ハードウェアごとに定義されているチャネル、メディア タイプ、バス幅。

11.1.199.2 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

DMA または DTC を使用してデータを移動するための転送インスタンス。

11.1.199.3 p_callback

void(* sdmmc_cfg_t::p_callback)(sdmmc_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.199.4 p_context

void const* sdmmc_cfg_t::p_context

概要説明

コールバック関数に渡されるユーザー定義のコンテキスト。

11.1.199.5 p_extend

void const* sdmmc_cfg_t::p_extend

概要説明

SDMMC ハードウェアに依存する設定。

11.1.200 sdmmc_ctrl_t

```
typedef struct{
   sdmmc_hw_t hw
   transfer_instance_t const * p_lower_lvl_transfer
   sdmmc_info_t status
   bool transfer_in_progress
   void(* p_callback)(sdmmc_callback_args_t *p_args)
   void const * p_context
} sdmmc_ctrl_t
```

11.1.200.1 hw

sdmmc_hw_t::hw

概要説明

ハードウェアごとに定義されているチャネル、メディア タイプ、バス幅。

11.1.200.2 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

DMA または DTC を使用してデータを移動するための転送インスタンス。

11.1.200.3 status

sdmmc_info_t::status

概要説明

メディア ステータス情報。

11.1.200.4 transfer_in_progress

bool sdmmc_ctrl_t::transfer_in_progress

概要説明

DMA または DTC 転送ステータス。

11.1.200.5 p_callback

void(* sdmmc_ctrl_t::p_callback)(sdmmc_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.200.6 p_context

void const* sdmmc_ctrl_t::p_context

概要説明

ハイレベルのデバイス コンテキストへのポインタ。

11.1.201 sdmmc_hw_t

```
typedef struct{
    uint8_t channel
    sdmmc_media_type_t media_type
    sdmmc_bus_width_t bus_width
} sdmmc_hw_t
```

11.1.201.1 channel

uint8_t sdmmc_hw_t::channel

概要説明

SD/MMC ホスト インタフェースのチャネル。

11.1.201.2 media_type

sdmmc_media_type_t::media_type

概要説明

組み込みカードまたはプラグ可能カード。

11.1.201.3 bus_width

sdmmc_bus_width_t::bus_width

概要説明

デバイスのバス幅は、1、4、8 ビット幅のいずれかです。

11.1.202 sdmmc_info_t

```
typedef struct{
    sdmmc_card_type_t card_type
    bool ready
    bool hc
    bool sdio
    bool write_protected
    bool transfer_in_progress
    uint8_t csd_version
    uint8_t device_type
    sdmmc_bus_width_t bus_width
    uint8_t hs_timing
    uint32_t sdhi_rca
    uint32_t max_clock_rate
    uint32_t clock_rate
    uint32_t sector_size
    uint32_t sector_count
    uint32_t erase_sector_count
} sdmmc_info_t
```

11.1.202.1 card_type

sdmmc_card_type_t::card_type

概要説明

SD または MMC。

11.1.202.2 ready

bool sdmmc_info_t::ready

概要説明

準備完了または未完了

11.1.202.3 hc

bool sdmmc_info_t::hc

概要説明

true = カードが大容量カード

11.1.202.4 sdio

bool sdmmc_info_t::sdio

概要説明

true = SDIO が存在

11.1.202.5 write_protected

bool sdmmc_info_t::write_protected

概要説明

カードが書き込み保護されています。

11.1.202.6 transfer_in_progress

bool sdmmc_info_t::transfer_in_progress

概要説明

DMA または DTC 転送ステータス。

11.1.202.7 csd_version

uint8_t sdmmc_info_t::csd_version

概要説明

CSD のバージョン。

11.1.202.8 device_type

uint8_t sdmmc_info_t::device_type

概要説明

速度とデータ レート。

11.1.202.9 bus_width

sdmmc_bus_width_t::bus_width

概要説明

現在のメディア バス幅。

11.1.202.10 hs_timing

uint8_t sdmmc_info_t::hs_timing

概要説明

高速ステータス。

11.1.202.11 sdhi_rca

uint32_t sdmmc_info_t::sdhi_rca

概要説明

相対カード アドレス。

11.1.202.12 max_clock_rate

uint32_t sdmmc_info_t::max_clock_rate

概要説明

メディア カードの最大クロック レート。

11.1.202.13 clock_rate

uint32_t sdmmc_info_t::clock_rate

概要説明

現在のクロック レート

11.1.202.14 sector_size

uint32_t sdmmc_info_t::sector_size

概要説明

セクター サイズ

11.1.202.15 sector_count

uint32_t sdmmc_info_t::sector_count

概要説明

セクター数

11.1.202.16 erase_sector_count

uint32_t sdmmc_info_t::erase_sector_count

概要説明

一度に消去する最大セクター数。

11.1.203 sdmmc_instance_t

```
typedef struct{
    sdmmc_ctrl_t * p_ctrl
    sdmmc_cfg_t const * p_cfg
    sdmmc_api_t const * p_api
} sdmmc_instance_t
```

11.1.203.1 p_ctrl

sdmmc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.203.2 p_cfg

sdmmc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.203.3 p_api

sdmmc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.204 sdmmc_io_mode_t

```
typedef struct{
   sdmmc_io_command_t command
   sdmmc_io_transfer_mode_t transfer_mode
   sdmmc_io_address_mode_t address_mode
   sdmmc_io_write_mode_t write_mode
} sdmmc_io_mode_t
```

11.1.204.1 command

sdmmc_io_command_t::command

概要説明

SDIO コマンド。

11.1.204.2 transfer_mode

sdmmc_io_transfer_mode_t sdmmc_io_mode_t::transfer_mode

概要説明

SDIO 転送タイプ。

11.1.204.3 address_mode

sdmmc_io_address_mode_t sdmmc_io_mode_t::address_mode

概要説明

SDIO アドレス モード。

11.1.204.4 write_mode

sdmmc_io_write_mode_t sdmmc_io_mode_t::write_mode

概要説明

SDIO 書き込みモード。

11.1.205 sf_adc_periodic_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_adc_periodic_ctrl_t *const p_ctrl, sf_adc_periodic_cfg_t const *const p_cfg)
    ssp_err_t(*  start)(sf_adc_periodic_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(sf_adc_periodic_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(sf_adc_periodic_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_adc_periodic_api_t
```

11.1.206 sf_adc_periodic_callback_args_t

```
typedef struct{
    sf_adc_periodic_event_t  event
    uint32_t  buffer_index
    void const *  p_context
} sf_adc_periodic_callback_args_t
```

11.1.206.1 event

sf_adc_periodic_event_t::event

概要説明

周期 ADC コールバック イベント。

11.1.206.2 buffer_index

uint32_t sf_adc_periodic_callback_args_t::buffer_index

概要説明

新しいデータが格納されているバッファへのインデックス。

11.1.206.3 p_context

void const* sf_adc_periodic_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.207 sf_adc_periodic_cfg_t

```
typedef struct{
    adc_instance_t const *const  p_lower_lvl_adc
    timer_instance_t const *const  p_lower_lvl_timer
    transfer_instance_t const *const  p_lower_lvl_transfer
    adc_data_size_t *  p_data_buffer
    uint32_t  data_buffer_length
    uint32_t  sample_count
    elc_event_t  scan_trigger
    void(*  p_callback)(sf_adc_periodic_callback_args_t *p_args)
    void const *  p_context
    void const *  p_extend
} sf_adc_periodic_cfg_t
```

11.1.207.1 p_lower_lvl_adc

adc_instance_t::p_lower_lvl_adc

概要説明

ADC インスタンスへのポインタ。

11.1.207.2 p_lower_lvl_timer

timer_instance_t::p_lower_lvl_timer

概要説明

タイマ インスタンスへのポインタ。

11.1.207.3 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

転送インスタンスへのポインタ。

11.1.207.4 p_data_buffer

adc_data_size_t::p_data_buffer

概要説明

サンプルを格納するバッファへのポインタ。

11.1.207.5 data_buffer_length

uint32_t sf_adc_periodic_cfg_t::data_buffer_length

概要説明

サンプルを格納するデータ バッファの長さ。

11.1.207.6 sample_count

uint32_t sf_adc_periodic_cfg_t::sample_count

概要説明

アプリケーションを通知する前にバッファされるチャネルごとのサンプル。

11.1.207.7 scan_trigger

elc_event_t::scan_trigger

概要説明

ADC スキャンを開始するハードウェア トリガ。

11.1.207.8 p_callback

void(* sf_adc_periodic_cfg_t::p_callback)(sf_adc_periodic_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.207.9 p_context

void const* sf_adc_periodic_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.207.10 p_extend

void const* sf_adc_periodic_cfg_t::p_extend

概要説明

ハードウェア固有の設定値に対応するための拡張パラメータです。

11.1.208 sf_adc_periodic_ctrl_t

```
typedef struct{
    uint32_t open
    TX_MUTEX mutex
    adc_api_t const * p_api_adc
    timer_api_t const * p_api_timer
    transfer_api_t const * p_api_transfer
    adc_ctrl_t ctrl_adc
    timer_ctrl_t ctrl_timer
    transfer_ctrl_t ctrl_transfer
    void const *volatile p_src_transfer
    adc_data_size_t * p_data_buffer
    uint32_t data_buffer_length
    uint32_t data_buffer_index
    uint32_t sample_count
    uint32_t dtc_transfer_length
    void(* p_callback)(sf_adc_periodic_callback_args_t *p_args)
    void const * p_context
} sf_adc_periodic_ctrl_t
```

11.1.208.1 open

uint32_t sf_adc_periodic_ctrl_t::open

概要説明

ドライバによって使用され、制御ブロックへのポインタが有効かどうかを確認します。

11.1.208.2 mutex

TX_MUTEX sf_adc_periodic_ctrl_t::mutex

概要説明

ローレベル ドライバ ハードウェア レジスタへのアクセスを保護するためのミューテックス。

11.1.208.3 p_api_adc

adc_api_t::p_api_adc

概要説明

ローレベル ADC ドライバ関数ポインタへのポインタ。

11.1.208.4 p_api_timer

timer_api_t::p_api_timer

概要説明

ローレベル タイマ ドライバ関数ポインタへのポインタ。

11.1.208.5 p_api_transfer

transfer_api_t::p_api_transfer

概要説明

ローレベル転送ドライバ関数ポインタへのポインタ。

11.1.208.6 ctrl_adc

adc_ctrl_t::ctrl_adc

概要説明

ローレベル ADC ドライバ制御ブロック。

11.1.208.7 ctrl_timer

timer_ctrl_t::ctrl_timer

概要説明

ローレベル タイマ ドライバ制御ブロック。

11.1.208.8 ctrl_transfer

transfer_ctrl_t::ctrl_transfer

概要説明

ローレベル転送ドライバ制御ブロック。

11.1.208.9 p_src_transfer

void const* volatile sf_adc_periodic_ctrl_t::p_src_transfer

概要説明

ローレベル転送メソッドのソース ポインタ。

11.1.208.10 p_data_buffer

adc_data_size_t::p_data_buffer

概要説明

サンプルを格納するバッファへのポインタ。

11.1.208.11 data_buffer_length

uint32_t sf_adc_periodic_ctrl_t::data_buffer_length

概要説明

サンプルを格納するデータ バッファの長さ。

11.1.208.12 data_buffer_index

uint32_t sf_adc_periodic_ctrl_t::data_buffer_index

概要説明

データが次に書き込まれるデータ バッファのインデックス。

11.1.208.13 sample_count

uint32_t sf_adc_periodic_ctrl_t::sample_count

概要説明

アプリケーションを通知する前にバッファされるチャネルごとのサンプル。

11.1.208.14 dtc_transfer_length

uint32_t sf_adc_periodic_ctrl_t::dtc_transfer_length

概要説明

要求された数のサンプルに対する DTC 転送の合計長。

11.1.208.15 p_callback

void(* sf_adc_periodic_ctrl_t::p_callback)(sf_adc_periodic_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.208.16 p_context

void const* sf_adc_periodic_ctrl_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.209 sf_adc_periodic_instance_t

```
typedef struct{
    sf_adc_periodic_ctrl_t * p_ctrl
    sf_adc_periodic_cfg_t const * p_cfg
    sf_adc_periodic_api_t const * p_api
} sf_adc_periodic_instance_t
```

11.1.209.1 p_ctrl

sf_adc_periodic_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.209.2 p_cfg

sf_adc_periodic_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.209.3 p_api

sf_adc_periodic_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.210 sf_audio_playback_api_t

```
typedef struct{
    ssp_err_t(* open)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_cfg_t const *const p_cfg)
    ssp_err_t(* close)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(* start)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)
    ssp_err_t(* pause)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(* stop)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(* resume)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(* volumeSet)(sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const volume)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_audio_playback_api_t
```

11.1.211 sf_audio_playback_cfg_t

```
typedef struct{
   void(* p_callback)(sf_message_callback_args_t *p_args)
   sf_audio_playback_common_ctrl_t * p_common_ctrl
   sf_audio_playback_common_cfg_t const * p_common_cfg
   uint8_t class_instance
} sf_audio_playback_cfg_t
```

11.1.211.1 p_callback

void(* sf_audio_playback_cfg_t::p_callback)(sf_message_callback_args_t *p_args)

詳細説明

start に渡されたバッファ再生の完了時に呼び出されるコールバック。コールバックを行わない場合は、NULL に設定します。

11.1.211.2 p_common_ctrl

sf_audio_playback_common_ctrl_t::p_common_ctrl

詳細説明

このストリームが使用するハードウェア制御ブロックへのポインタ。

11.1.211.3 p_common_cfg

sf_audio_playback_common_cfg_t::p_common_cfg

詳細説明

同じハードウェアを使用するすべてのストリームによって共有される共通の設定へのポインタ。

11.1.211.4 class_instance

uint8_t sf_audio_playback_cfg_t::class_instance

概要説明

メッセージング フレームワークへのストリーム指定に使用されるクラス インスタンス。

11.1.212 sf_audio_playback_common_cfg_t

```
typedef struct{
    UINT priority
    sf_audio_playback_hw_instance_t const * p_lower_lvl_hw
    sf_message_instance_t const * p_message
    TX_QUEUE * p_queue
    void const * p_extend
} sf_audio_playback_common_cfg_t
```

11.1.212.1 priority

UINT sf_audio_playback_common_cfg_t::priority

概要説明

オーディオ再生スレッドの優先順位。

11.1.212.2 p_lower_lvl_hw

sf_audio_playback_hw_instance_t::p_lower_lvl_hw

概要説明

ハードウェア インスタンス。

11.1.212.3 p_message

sf_message_instance_t::p_message

詳細説明

オーディオ メッセージをポストするために使用するメッセージング フレームワーク インスタンスへのポインタ。

11.1.212.4 p_queue

TX_QUEUE* sf_audio_playback_common_cfg_t::p_queue

詳細説明

このオーディオ ストリームに対して指定されたメッセージング フレームワーク キューへのポインタ。
SF_MESSAGE_EVENT_CLASS_AUDIO イベント クラスにサブスクライブする必要があります。

11.1.212.5 p_extend

void const* sf_audio_playback_common_cfg_t::p_extend

詳細説明

実装固有の拡張設定。

11.1.213 sf_audio_playback_common_ctrl_t

```
typedef struct{
  uint32_t open
  void const * p_next_buffer
  uint32_t next_length
  sf_message_instance_t const * p_message
  TX_QUEUE * p_queue
  sf_audio_playback_hw_instance_t const * p_lower_lvl_hw
  sf_audio_playback_ctrl_t * p_stream[SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS]
  TX_THREAD thread
  TX_EVENT_FLAGS_GROUP flags
  sf_audio_playback_data_type_t data_type
  uint8_t volume
  uint8_t buffer_index
  uint8_t stack[SF_AUDIO_PLAYBACK_STACK_SIZE]
  int16_t samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]
  bool playing
} sf_audio_playback_common_ctrl_t
```

11.1.213.1 open

uint32_t sf_audio_playback_common_ctrl_t::open

概要説明

ドライバが初期化済みかどうかの判定に使用されます。

11.1.213.2 p_next_buffer

void const* sf_audio_playback_common_ctrl_t::p_next_buffer

概要説明

次のバッファ （現在のバッファが完了したときに再生されるバッファ）へのポインタ。

11.1.213.3 next_length

uint32_t sf_audio_playback_common_ctrl_t::next_length

概要説明

次のバッファ （現在のバッファが完了したときに再生されるバッファ）の長さ。

11.1.213.4 p_message

sf_message_instance_t::p_message

概要説明

メッセージ制御ブロックへのポインタ。

11.1.213.5 p_queue

TX_QUEUE* sf_audio_playback_common_ctrl_t::p_queue

概要説明

SF_MESSAGE_EVENT_CLASS_AUDIO イベントにサブスクライブされるキュー。

11.1.213.6 p_lower_lvl_hw

sf_audio_playback_hw_instance_t::p_lower_lvl_hw

概要説明

ハードウェア API が使用されています。

11.1.213.7 p_stream

sf_audio_playback_ctrl_t*
sf_audio_playback_common_ctrl_t::p_stream[SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS]

概要説明

特定のデータをストリーミングします。

11.1.213.8 thread

TX_THREAD sf_audio_playback_common_ctrl_t::thread

概要説明

メインのオーディオ スレッド。

11.1.213.9 flags

TX_EVENT_FLAGS_GROUP sf_audio_playback_common_ctrl_t::flags

概要説明

オーディオ スレッドでの待機を解除するためのイベント フラグ。

11.1.213.10 data_type

sf_audio_playback_data_type_t::data_type

概要説明

ハードウェアで要求されるサンプル フォーマット。

11.1.213.11 volume

uint8_t sf_audio_playback_common_ctrl_t::volume

概要説明

範囲は 0 （消音）～ 255 （最大、開いた時点でのデフォルト）です。

11.1.213.12 buffer_index

uint8_t sf_audio_playback_common_ctrl_t::buffer_index

概要説明

使用されるピンポン バッファ。

11.1.213.13 stack

uint8_t sf_audio_playback_common_ctrl_t::stack[SF_AUDIO_PLAYBACK_STACK_SIZE]

概要説明

オーディオ スレッド用のスタック。

11.1.213.14 samples

int16_t
sf_audio_playback_common_ctrl_t::samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]

詳細説明

転送時に変換されたデータの格納に使用するピンポン バッファ。

11.1.213.15 playing

bool sf_audio_playback_common_ctrl_t::playing

概要説明

オーディオ インスタンスの状態 （true の場合は、現在再生中）

11.1.214 sf_audio_playback_data_t

```
typedef struct{
   sf_message_header_t header
   sf_audio_playback_data_type_t type
   uint32_t size_bytes
   void const * p_data
   UINT loop_timeout
   bool stream_end
} sf_audio_playback_data_t
```

11.1.214.1 header

sf_message_header_t::header

概要説明

メッセージング フレームワーク ペイロードの必須の共通メンバー。

11.1.214.2 type

sf_audio_playback_data_type_t::type

概要説明

データ型。非圧縮である必要があります。

11.1.214.3 size_bytes

uint32_t sf_audio_playback_data_t::size_bytes

概要説明

データ サイズ （バイト単位）。

11.1.214.4 p_data

void const* sf_audio_playback_data_t::p_data

概要説明

データへのポインタ。データの開始アドレスは、4 バイトでアラインする必要があります。

11.1.214.5 loop_timeout

UINT sf_audio_playback_data_t::loop_timeout

詳細説明

タイムアウト値を ThreadX ティック カウント （0x00000001 ～ 0xFFFFFFFE）で指定すると、ティック カウントが期限切れになるまでループ再生されます。

11.1.214.6 stream_end

bool sf_audio_playback_data_t::stream_end

詳細説明

これにより、他のスレッドがデータをポストできるようになります。この論理ビットストリームで送信するデータが他に存在しない場合は、true に設定します。他のパケットが準備されている場合は、false に設定します。

11.1.215 sf_audio_playback_data_type_t

```
typedef struct{
   uint8_t scale_bits_max
   bool is_signed
} sf_audio_playback_data_type_t
```

11.1.215.1 scale_bits_max

uint8_t sf_audio_playback_data_type_t::scale_bits_max

概要説明

ビット単位の最大分解能。

11.1.215.2 is_signed

bool sf_audio_playback_data_type_t::is_signed

概要説明

符号付きサンプルの場合は 1、符号なしサンプルの場合は 0 に設定します。

11.1.216 sf_audio_playback_hw_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)
   ssp_err_t(* start)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
   ssp_err_t(* stop)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
   ssp_err_t(* play)(sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const *const p_buffer, uint32_t length)
   ssp_err_t(* dataTypeGet)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)
   ssp_err_t(* close)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_audio_playback_hw_api_t
```

11.1.217 sf_audio_playback_hw_callback_args_t

```
typedef struct{
   void * p_context
} sf_audio_playback_hw_callback_args_t
```

11.1.217.1 p_context

void* sf_audio_playback_hw_callback_args_t::p_context

詳細説明

ユーザー データのプレースホルダー。sf_audio_playback_hw_cfg_t の open 関数で設定されます。

11.1.218 sf_audio_playback_hw_cfg_t

```
typedef struct{
   void(* p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
   void * p_context
   void const * p_extend
} sf_audio_playback_hw_cfg_t
```

11.1.218.1 p_callback

void(* sf_audio_playback_hw_cfg_t::p_callback)(sf_audio_playback_hw_callback_args_t *p_args)

詳細説明

再生完了時に呼び出されるコールバック。コールバックを行わない場合は、NULL に設定します。

11.1.218.2 p_context

void* sf_audio_playback_hw_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。sf_audio_playback_hw_callback_args_t 内のユーザー コールバックに渡されます。

11.1.218.3 p_extend

void const* sf_audio_playback_hw_cfg_t::p_extend

概要説明

ハードウェアに依存する設定。

11.1.219 sf_audio_playback_hw_ctrl_t

```
typedef struct{
   void(*  p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
   void *  p_context
   void *  p_extend
} sf_audio_playback_hw_ctrl_t
```

11.1.219.1 p_callback

void(* sf_audio_playback_hw_ctrl_t::p_callback)(sf_audio_playback_hw_callback_args_t *p_args)

詳細説明

再生完了時に呼び出されるコールバック。

11.1.219.2 p_context

void* sf_audio_playback_hw_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。sf_audio_playback_hw_callback_args_t 内のユーザー コールバックに渡されます。

11.1.219.3 p_extend

void* sf_audio_playback_hw_ctrl_t::p_extend

概要説明

ハードウェアに依存する制御ブロック。

11.1.220 sf_audio_playback_hw_dac_cfg_t

```
typedef struct{
    dac_instance_t const * p_lower_lvl_dac
    timer_instance_t const * p_lower_lvl_timer
    transfer_instance_t const * p_lower_lvl_transfer
    sf_audio_playback_hw_dac_ctrl_t * p_ctrl
} sf_audio_playback_hw_dac_cfg_t
```

11.1.220.1 p_lower_lvl_dac

dac_instance_t::p_lower_lvl_dac

概要説明

DAC ハードウェアへのアクセスに使用される DAC API。

11.1.220.2 p_lower_lvl_timer

timer_instance_t::p_lower_lvl_timer

概要説明

サンプリング周波数の生成に使用されるタイマ API。

11.1.220.3 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

各サンプリング周波数でデータを転送するために使用される転送 API。

11.1.220.4 p_ctrl

sf_audio_playback_hw_dac_ctrl_t::p_ctrl

詳細説明

ハードウェア制御ブロックに割り当てられたメモリへのポインタ。初期化は必要ありません。

11.1.221 sf_audio_playback_hw_dac_ctrl_t

```
typedef struct{
    dac_instance_t const * p_lower_lvl_dac
    timer_instance_t const * p_lower_lvl_timer
    transfer_instance_t const * p_lower_lvl_transfer
} sf_audio_playback_hw_dac_ctrl_t
```

11.1.221.1 p_lower_lvl_dac

dac_instance_t::p_lower_lvl_dac

概要説明

DAC ハードウェアへのアクセスに使用される DAC API。

11.1.221.2 p_lower_lvl_timer

timer_instance_t::p_lower_lvl_timer

概要説明

サンプリング周波数の生成に使用されるタイマ API。

11.1.221.3 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

概要説明

各サンプリング周波数でデータを転送するために使用される転送 API。

11.1.222 sf_audio_playback_hw_i2s_cfg_t

```
typedef struct{
    i2s_instance_t const * p_lower_lvl_i2s
    sf_audio_playback_hw_i2s_ctrl_t * p_ctrl
} sf_audio_playback_hw_i2s_cfg_t
```

11.1.222.1 p_lower_lvl_i2s

i2s_instance_t::p_lower_lvl_i2s

概要説明

I2S ハードウェアへのアクセスに使用される I2S API。

11.1.222.2 p_ctrl

sf_audio_playback_hw_i2s_ctrl_t::p_ctrl

詳細説明

ハードウェア制御ブロックに割り当てられたメモリへのポインタ。初期化は必要ありません。

11.1.223 sf_audio_playback_hw_i2s_ctrl_t

```
typedef struct{
    i2s_instance_t const * p_lower_lvl_i2s
} sf_audio_playback_hw_i2s_ctrl_t
```

11.1.223.1 p_lower_lvl_i2s

i2s_instance_t::p_lower_lvl_i2s

概要説明

I2S ハードウェアへのアクセスに使用される I2S API。

11.1.224 sf_audio_playback_hw_instance_t

```
typedef struct{
    sf_audio_playback_hw_ctrl_t * p_ctrl
    sf_audio_playback_hw_cfg_t const * p_cfg
    sf_audio_playback_hw_api_t const * p_api
} sf_audio_playback_hw_instance_t
```

11.1.224.1 p_ctrl

sf_audio_playback_hw_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.224.2 p_cfg

sf_audio_playback_hw_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.224.3 p_api

sf_audio_playback_hw_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.225 sf_audio_playback_instance_t

```
typedef struct{
    sf_audio_playback_ctrl_t *  p_ctrl
    sf_audio_playback_cfg_t const *  p_cfg
    sf_audio_playback_api_t const *  p_api
} sf_audio_playback_instance_t
```

11.1.225.1 p_ctrl

sf_audio_playback_ctrl_t* sf_audio_playback_instance_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.225.2 p_cfg

sf_audio_playback_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.225.3 p_api

sf_audio_playback_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.226 sf_block_media_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
    ssp_err_t(*  write)(sf_block_media_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count)
    ssp_err_t(*  ioctl)(sf_block_media_ctrl_t *p_ctrl, ssp_command_t const command, void *p_data)
    ssp_err_t(*  close)(sf_block_media_ctrl_t *p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_block_media_api_t
```

11.1.227 sf_block_media_cfg_t

```
typedef struct{
   uint32_t block_size
   void const * p_extend
} sf_block_media_cfg_t
```

11.1.227.1 block_size

uint32_t sf_block_media_cfg_t::block_size

概要説明

バイト単位のブロック サイズ。

11.1.227.2 p_extend

void const* sf_block_media_cfg_t::p_extend

概要説明

インスタンスに依存する設定。

11.1.228 sf_block_media_ctrl_t

```
typedef struct{
   uint32_t block_size
   void const * p_extend
} sf_block_media_ctrl_t
```

11.1.228.1 block_size

uint32_t sf_block_media_ctrl_t::block_size

概要説明

バイト単位のブロック サイズ。

11.1.228.2 p_extend

void const* sf_block_media_ctrl_t::p_extend

概要説明

インスタンスのチャネル情報。

11.1.229 sf_block_media_instance_t

```
typedef struct{
    sf_block_media_ctrl_t *  p_ctrl
    sf_block_media_cfg_t const *  p_cfg
    sf_block_media_api_t const *  p_api
} sf_block_media_instance_t
```

11.1.229.1 p_ctrl

sf_block_media_ctrl_t::p_ctrl

概要説明

デバイス ドライバ制御構造体へのブロック メディア ポインタ。

11.1.229.2 p_cfg

sf_block_media_cfg_t::p_cfg

概要説明

デバイス ドライバ設定構造体への Block Media ポインタ。

11.1.229.3 p_api

sf_block_media_api_t::p_api

概要説明

デバイス ドライバ API 構造体へのブロック メディア ポインタ。

11.1.230 sf_block_media_on_sdmmc_cfg_t

```
typedef struct{
    sdmmc_instance_t const *const  p_lower_lvl_sdmmc
} sf_block_media_on_sdmmc_cfg_t
```

11.1.230.1 p_lower_lvl_sdmmc

sdmmc_instance_t::p_lower_lvl_sdmmc

概要説明

SDMMC インスタンス構造体へのポインタ。

11.1.231 sf_block_media_on_sdmmc_ctrl_t

```
typedef struct{
   sdmmc_instance_t * p_lower_lvl_sdmmc
   TX_EVENT_FLAGS_GROUP eventflag
} sf_block_media_on_sdmmc_ctrl_t
```

11.1.231.1 p_lower_lvl_sdmmc

sdmmc_instance_t::p_lower_lvl_sdmmc

概要説明

SDMMC インスタンス構造体へのポインタ。

11.1.231.2 eventflag

TX_EVENT_FLAGS_GROUP ::eventflag

概要説明

SDMMC データ転送用のイベント フラグ オブジェクトへのポインタ。

11.1.232 sf_comms_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t const *const p_cfg)
   ssp_err_t(* close)(sf_comms_ctrl_t *const p_ctrl)
   ssp_err_t(* read)(sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout)
   ssp_err_t(* write)(sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)
   ssp_err_t(* lock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type, UINT timeout)
   ssp_err_t(* unlock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_comms_api_t
```

11.1.233 sf_comms_cfg_t

```
typedef struct{
   void const * p_extend
} sf_comms_cfg_t
```

11.1.233.1 p_extend

void const* sf_comms_cfg_t::p_extend

概要説明

ローレベル通信制御構造体へのポインタ。

11.1.234 sf_comms_ctrl_t

```
typedef struct{
    void const *  p_extend
} sf_comms_ctrl_t
```

11.1.234.1 p_extend

void const* sf_comms_ctrl_t::p_extend

概要説明

ローレベル通信制御構造体へのポインタ。

11.1.235 sf_comms_instance_t

```
typedef struct{
    sf_comms_ctrl_t *  p_ctrl
    sf_comms_cfg_t const *  p_cfg
    sf_comms_api_t const *  p_api
} sf_comms_instance_t
```

11.1.235.1 p_ctrl

sf_comms_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.235.2 p_cfg

sf_comms_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.235.3 p_api

sf_comms_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.236 sf_console_api_t

```
typedef struct{
    ssp_err_t(* open)(sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const *const p_cfg)
    ssp_err_t(* close)(sf_console_ctrl_t *const p_ctrl)
    ssp_err_t(* prompt)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_menu, UINT const timeout)
    ssp_err_t(* parse)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_cmd_list, uint8_t const *const p_input, uint32_t
const bytes)
    ssp_err_t(* read)(sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout)
    ssp_err_t(* write)(sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout)
    ssp_err_t(* argumentFind)(uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_console_api_t
```

11.1.237 sf_console_callback_args_t

```
typedef struct{
    sf_console_ctrl_t * p_ctrl
    uint8_t const * p_remaining_string
    uint8_t const * context
    uint32_t bytes
} sf_console_callback_args_t
```

11.1.237.1 p_ctrl

sf_console_ctrl_t::p_ctrl

概要説明

このコールバックを発生させたコマンドを受け取ったコンソールへのポインタ。

11.1.237.2 p_remaining_string

uint8_t const* sf_console_callback_args_t::p_remaining_string

概要説明

解析コマンドの後に残る文字列。

11.1.237.3 context

uint8_t const* sf_console_callback_args_t::context

概要説明

ユーザー定義のデータへのポインタ。

11.1.237.4 bytes

uint32_t sf_console_callback_args_t::bytes

概要説明

入力文字列の残りのバイト数。

11.1.238 sf_console_cfg_t

```
typedef struct{
    sf_comms_instance_t const *  p_comms
    sf_console_menu_t const *  p_initial_menu
    bool  echo
    bool  autostart
} sf_console_cfg_t
```

11.1.238.1 p_comms

sf_comms_instance_t::p_comms

概要説明

通信ドライバ インスタンスへのポインタ。

11.1.238.2 p_initial_menu

const* sf_console_cfg_t::p_initial_menu

概要説明

Open を実行すると表示される最初のメニュー。

11.1.238.3 echo

bool sf_console_cfg_t::echo

概要説明

入力コマンドをトランスミッターにエコーするかどうかの指定。

11.1.238.4 autostart

bool sf_console_cfg_t::autostart

概要説明

true の場合は、初期化の後に p_initial_menu を使用したプロンプトが発生します。

11.1.239 sf_console_command_t

```
typedef struct{
    uint8_t * command
    uint8_t * help
    void(* callback)(sf_console_callback_args_t *p_args)
    void const * context
} sf_console_command_t
```

11.1.239.1 command

uint8_t* sf_console_command_t::command

概要説明

コマンド文字列。

11.1.239.2 help

uint8_t* sf_console_command_t::help

概要説明

コマンドの説明。

11.1.239.3 callback

void(* sf_console_command_t::callback)(sf_console_callback_args_t *p_args)

概要説明

コマンドを選択したときに呼び出されるコールバック。

11.1.239.4 context

void const* sf_console_command_t::context

概要説明

コールバックに渡されるユーザー定義のコンテキスト。

11.1.240 sf_console_ctrl_t

```
typedef struct{
    sf_console_menu_t const *  p_current_menu
    sf_comms_instance_t const *  p_comms
    uint8_t  new_line
    bool  echo
    uint8_t  input[SF_CONSOLE_MAX_INPUT_LENGTH]
} sf_console_ctrl_t
```

11.1.240.1 p_current_menu

const* sf_console_ctrl_t::p_current_menu

概要説明

現在のメニューがここに格納されます。

11.1.240.2 p_comms

sf_comms_instance_t::p_comms

概要説明

通信ドライバ インスタンスへのポインタ。

11.1.240.3 new_line

uint8_t sf_console_ctrl_t::new_line

概要説明

入力コマンドをトランスミッターにエコーするかどうかの指定。

11.1.240.4 echo

bool sf_console_ctrl_t::echo

概要説明

入力コマンドをトランスミッターにエコーするかどうかの指定。

11.1.240.5 input

uint8_t sf_console_ctrl_t::input[SF_CONSOLE_MAX_INPUT_LENGTH]

概要説明

ユーザー入力の格納に使用される入力バッファ。

11.1.241 sf_console_instance_t

```
typedef struct{
    sf_console_ctrl_t * p_ctrl
    sf_console_cfg_t const * p_cfg
    sf_console_api_t const * p_api
} sf_console_instance_t
```

11.1.241.1 p_ctrl

sf_console_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.241.2 p_cfg

sf_console_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.241.3 p_api

sf_console_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.242 sf_el_fx_t

```
typedef struct{
    sf_block_media_instance_t * p_lower_lvl_block_media
    void * p_extend
} sf_el_fx_t
```

11.1.242.1 p_lower_lvl_block_media

sf_block_media_instance_t::p_lower_lvl_block_media

概要説明

ローレベル ブロック メディア ポインタ。

11.1.242.2 p_extend

void* ::p_extend

11.1.243 sf_el_gx_api_t

```
typedef struct{
    ssp_err_t(* open)(sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const p_cfg)
    ssp_err_t(* close)(sf_el_gx_ctrl_t *const p_ctrl)
    ssp_err_t(* versionGet)(ssp_version_t *p_version)
    UINT(* setup)(GX_DISPLAY *p_display)
    ssp_err_t(* canvasInit)(sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT *p_window_root)
} sf_el_gx_api_t
```

11.1.244 sf_el_gx_callback_args_t

```
typedef struct{
    sf_el_gx_device_t device
    sf_el_gx_event_t event
    uint32_t error
} sf_el_gx_callback_args_t
```

11.1.244.1 device

sf_el_gx_device_t::device

概要説明

デバイス コード。

11.1.244.2 event

sf_el_gx_event_t::event

概要説明

ローレベル ハードウェアのイベント コード。

11.1.244.3 error

uint32_t sf_el_gx_callback_args_t::error

概要説明

SF_EL_GX_EVENT_ERROR の場合のエラー コード。

11.1.245 sf_el_gx_cfg_t

```
typedef struct{
    display_instance_t * p_display_instance
    display_runtime_cfg_t * p_display_runtime_cfg
    void * p_canvas
    void * p_framebuffer_a
    void * p_framebuffer_b
    void(* p_callback)(sf_el_gx_callback_args_t *p_args)
    void * p_context
    void * p_jpegbuffer
    uint32_t jpegbuffer_size
    uint16_t rotation_angle
} sf_el_gx_cfg_t
```

11.1.245.1 p_display_instance

display_instance_t::p_display_instance

概要説明

表示インスタンスへのポインタ。

11.1.245.2 p_display_runtime_cfg

display_runtime_cfg_t::p_display_runtime_cfg

概要説明

実行時表示構成へのポインタ。

11.1.245.3 p_canvas

void* sf_el_gx_cfg_t::p_canvas

概要説明

キャンバスへのポインタ （予約済み）

11.1.245.4 p_framebuffer_a

void* sf_el_gx_cfg_t::p_framebuffer_a

概要説明

フレーム バッファ （A）へのポインタ

11.1.245.5 p_framebuffer_b

void* sf_el_gx_cfg_t::p_framebuffer_b

概要説明

フレーム バッファ （B）へのポインタ

11.1.245.6 p_callback

void(* sf_el_gx_cfg_t::p_callback)(sf_el_gx_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.245.7 p_context

void* sf_el_gx_cfg_t::p_context

概要説明

コンテキストへのポインタ。

11.1.245.8 p_jpegbuffer

void* sf_el_gx_cfg_t::p_jpegbuffer

概要説明

JPEG 作業バッファへのポインタ。

11.1.245.9 jpegbuffer_size

uint32_t sf_el_gx_cfg_t::jpegbuffer_size

概要説明

JPEG 作業バッファのサイズ。

11.1.245.10 rotation_angle

uint16_t sf_el_gx_cfg_t::rotation_angle

概要説明

画面のローテーション角度 （0/90/270）

11.1.246 sf_el_gx_ctrl_t

```
typedef struct{
   GX_DISPLAY * p_display
   display_instance_t * p_display_instance
   display_runtime_cfg_t * p_display_runtime_cfg
   void * p_canvas
   void * p_framebuffer_read
   void * p_framebuffer_write
   void(* p_callback)(sf_el_gx_callback_args_t *p_args)
   void * p_context
   TX_SEMAPHORE semaphore
   bool rendering_enable
   bool display_list_flushed
   sf_el_gx_state_t state
   void * p_jpegbuffer
   uint32_t jpegbuffer_size
   uint16_t rotation_angle
} sf_el_gx_ctrl_t
```

11.1.246.1 p_display

GX_DISPLAY* sf_el_gx_ctrl_t::p_display

概要説明

GUIX™ 表示コンテキストへのポインタ。

11.1.246.2 p_display_instance

display_instance_t::p_display_instance

概要説明

表示インスタンスへのポインタ。

11.1.246.3 p_display_runtime_cfg

display_runtime_cfg_t::p_display_runtime_cfg

概要説明

実行時表示構成へのポインタ。

11.1.246.4 p_canvas

void* sf_el_gx_ctrl_t::p_canvas

概要説明

キャンバスへのポインタ （予約済み）

11.1.246.5 p_framebuffer_read

void* sf_el_gx_ctrl_t::p_framebuffer_read

概要説明

フレーム バッファ （表示用）へのポインタ

11.1.246.6 p_framebuffer_write

void* sf_el_gx_ctrl_t::p_framebuffer_write

概要説明

フレーム バッファ （レンダリング用）へのポインタ

11.1.246.7 p_callback

void(* sf_el_gx_ctrl_t::p_callback)(sf_el_gx_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.246.8 p_context

void* sf_el_gx_ctrl_t::p_context

概要説明

コンテキストへのポインタ。

11.1.246.9 semaphore

TX_SEMAPHORE sf_el_gx_ctrl_t::semaphore

概要説明

フレーム バッファのフリップ同期のセマフォ。

11.1.246.10 rendering_enable

bool sf_el_gx_ctrl_t::rendering_enable

概要説明

レンダリングと表示の間の同期フラグ。

11.1.246.11 display_list_flushed

bool sf_el_gx_ctrl_t::display_list_flushed

概要説明

表示リストがフラッシュされたことを示すフラグ。

11.1.246.12 state

sf_el_gx_state_t::state

概要説明

このモジュールの状態。

11.1.246.13 p_jpegbuffer

void* sf_el_gx_ctrl_t::p_jpegbuffer

概要説明

JPEG 作業バッファへのポインタ。

11.1.246.14 jpegbuffer_size

uint32_t sf_el_gx_ctrl_t::jpegbuffer_size

概要説明

JPEG 作業バッファのサイズ。

11.1.246.15 rotation_angle

uint16_t sf_el_gx_ctrl_t::rotation_angle

概要説明

画面のローテーション角度 （0/90/270）

11.1.247 sf_el_gx_instance_t

```
typedef struct{
   sf_el_gx_ctrl_t * p_ctrl
   sf_el_gx_cfg_t const * p_cfg
   sf_el_gx_api_t const * p_api
} sf_el_gx_instance_t
```

11.1.247.1 p_ctrl

sf_el_gx_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.247.2 p_cfg

sf_el_gx_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.247.3 p_api

sf_el_gx_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.248 sf_el_nx_comms_on_comms_cfg_t

```
typedef struct{
   sf_el_nx_comms_on_comms_ctrl_t * p_ctrl
   uint32_t ip_address
   uint32_t subnet_mask
   VOID(* driver)(NX_IP_DRIVER *driver_req_ptr)
} sf_el_nx_comms_on_comms_cfg_t
```

11.1.248.1 p_ctrl

sf_el_nx_comms_on_comms_ctrl_t::p_ctrl

概要説明

NetX™ Telnet サーバー デバイス制御ブロックに割り当てられたメモリ。

11.1.248.2 ip_address

uint32_t sf_::ip_address

11.1.248.3 subnet_mask

uint32_t sf_::subnet_mask

11.1.248.4 driver

VOID(* sf_::driver)(NX_IP_DRIVER *driver_req_ptr)

11.1.249 sf_el_nx_comms_on_comms_ctrl_t

```
typedef struct{
  uint32_t open
  NX_PACKET * p_current_packet
  uint32_t packet_index
  TX_MUTEX mutex[2]
  NX_PACKET_POOL pool
  uint8_t pool_memory[SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES]
  NX_IP ip
  uint8_t ip_memory[SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES]
  uint8_t arp_memory[SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES]
  NX_TELNET_SERVER telnet_server
  uint8_t telnet_server_memory[SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES]
  UINT logical_connection
  TX_EVENT_FLAGS_GROUP available
  TX_QUEUE queue
  uint8_t queue_memory[SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES]
} sf_el_nx_comms_on_comms_ctrl_t
```

11.1.249.1 open

uint32_t sf_::open

11.1.249.2 p_current_packet

NX_PACKET* sf_::p_current_packet

11.1.249.3 packet_index

uint32_t sf_::packet_index

11.1.249.4 mutex

TX_MUTEX sf_::mutex[2]

11.1.249.5 pool

NX_PACKET_POOL sf_::pool

11.1.249.6 pool_memory

uint8_t sf_::pool_memory[SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES]

11.1.249.7 ip

NX_IP sf_::ip

11.1.249.8 ip_memory

uint8_t sf_::ip_memory[SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES]

11.1.249.9 arp_memory

uint8_t sf_::arp_memory[SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES]

11.1.249.10 telnet_server

NX_TELNET_SERVER sf_::telnet_server

11.1.249.11 telnet_server_memory

uint8_t sf_::telnet_server_memory[SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES]

11.1.249.12 logical_connection

UINT sf_::logical_connection

11.1.249.13 available

TX_EVENT_FLAGS_GROUP sf_::available

概要説明

この接続が使用可能かどうかを示すフラグ。

11.1.249.14 queue

TX_QUEUE sf_::queue

概要説明

受信バイトのキュー。

11.1.249.15 queue_memory

uint8_t sf_::queue_memory[SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES]

11.1.250 sf_el_ux_comms_on_comms_cfg_t

```
typedef struct{
   sf_el_ux_comms_on_comms_ctrl_t * p_ctrl
} sf_el_ux_comms_on_comms_cfg_t
```

11.1.250.1 p_ctrl

sf_el_ux_comms_on_comms_ctrl_t::p_ctrl

概要説明

USBX™ CDC ACM デバイス制御ブロックに割り当てられたメモリ。

11.1.251 sf_el_ux_comms_on_comms_ctrl_t

```
typedef struct{
   uint32_t open
   TX_MUTEX mutex[2]
   UX_SLAVE_CLASS_CDC_ACM * p_cdc
   uint32_t leftover_length
   uint32_t index
   uint8_t memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]
   uint8_t rx_memory[SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH]
} sf_el_ux_comms_on_comms_ctrl_t
```

11.1.251.1 open

uint32_t sf_::open

11.1.251.2 mutex

TX_MUTEX sf_::mutex[2]

11.1.251.3 p_cdc

UX_SLAVE_CLASS_CDC_ACM* sf_::p_cdc

11.1.251.4 leftover_length

uint32_t sf_::leftover_length

11.1.251.5 index

uint32_t sf_::index

11.1.251.6 memory

uint8_t sf_::memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]

11.1.251.7 rx_memory

uint8_t sf_::rx_memory[SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH]

11.1.252 sf_external_irq_api_t

```
typedef struct{
    ssp_err_t(* open)(sf_external_irq_ctrl_t *const p_ctrl, sf_external_irq_cfg_t const *const p_cfg)
    ssp_err_t(* wait)(sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
    ssp_err_t(* close)(sf_external_irq_ctrl_t *const p_ctrl)
} sf_external_irq_api_t
```

11.1.253 sf_external_irq_cfg_t

```
typedef struct{
    external_irq_instance_t const * p_lower_lvl_irq
    sf_external_irq_event_t event
} sf_external_irq_cfg_t
```

11.1.253.1 p_lower_lvl_irq

external_irq_instance_t::p_lower_lvl_irq

詳細説明

下位レイヤーとの連携に必要なすべての情報

11.1.253.2 event

sf_external_irq_event_t::event

概要説明

外部 IRQ の発生時に行われる処理を選択します。

11.1.254 sf_external_irq_ctrl_t

```
typedef struct{
    uint32_t open
    TX_MUTEX mutex
    TX_SEMAPHORE semaphore
    external_irq_api_t const * p_api
    external_irq_ctrl_t ctrl
    bool callback_used
} sf_external_irq_ctrl_t
```

11.1.254.1 open

uint32_t sf_external_irq_ctrl_t::open

概要説明

ドライバによって使用され、制御ブロックが有効かどうかを確認します。

11.1.254.2 mutex

TX_MUTEX sf_external_irq_ctrl_t::mutex

概要説明

ローレベル ドライバ ハードウェア レジスタへのアクセスを保護するためのミューテックス。

11.1.254.3 semaphore

TX_SEMAPHORE sf_external_irq_ctrl_t::semaphore

概要説明

SF_EXTERNAL_IRQ_Wait に対して使用されるセマフォ。

11.1.254.4 p_api

external_irq_api_t::p_api

概要説明

ローレベル ドライバ関数ポインタへのポインタ。

11.1.254.5 ctrl

external_irq_ctrl_t::ctrl

概要説明

ローレベル ドライバ制御ブロック。

11.1.254.6 callback_used

bool sf_external_irq_ctrl_t::callback_used

概要説明

ドライバによって使用され、待機が使用可能かどうかを確認します。

11.1.255 sf_external_irq_instance_t

```
typedef struct{
    sf_external_irq_ctrl_t *  p_ctrl
    sf_external_irq_cfg_t const *  p_cfg
    sf_external_irq_api_t const *  p_api
} sf_external_irq_instance_t
```

11.1.255.1 p_ctrl

sf_external_irq_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.255.2 p_cfg

sf_external_irq_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.255.3 p_api

sf_external_irq_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.256 sf_i2c_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)
   ssp_err_t(* read)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout)
   ssp_err_t(* write)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout)
   ssp_err_t(* reset)(sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)
   ssp_err_t(* close)(sf_i2c_ctrl_t *const p_ctrl)
   ssp_err_t(* lock)(sf_i2c_ctrl_t *const p_ctrl)
   ssp_err_t(* unlock)(sf_i2c_ctrl_t *const p_ctrl)
   ssp_err_t(* version)(ssp_version_t *const p_version)
} sf_i2c_api_t
```

11.1.257 sf_i2c_bus_t

```
typedef struct{
   uint8_t channel
   TX_MUTEX * p_lock_mutex
   TX_EVENT_FLAGS_GROUP * p_sync_eventflag
   sf_i2c_ctrl_t ** pp_curr_ctrl
   uint8_t * p_bus_name
   i2c_api_master_t * p_lower_lvl_api
   uint8_t device_count
} sf_i2c_bus_t
```

11.1.257.1 channel

uint8_t sf_i2c_bus_t::channel

概要説明

チャネル。

11.1.257.2 p_lock_mutex

TX_MUTEX* sf_i2c_bus_t::p_lock_mutex

概要説明

このチャネルのロック ミューテックス ハンドル。

11.1.257.3 p_sync_eventflag

TX_EVENT_FLAGS_GROUP* sf_i2c_bus_t::p_sync_eventflag

概要説明

I2C データ転送用のイベント フラグ オブジェクトへのポインタ。

11.1.257.4 pp_curr_ctrl

sf_i2c_ctrl_t** sf_i2c_bus_t::pp_curr_ctrl

概要説明

バスを使用している現在のデバイス。

11.1.257.5 p_bus_name

uint8_t* sf_i2c_bus_t::p_bus_name

概要説明

バスを識別するためのユーザー定義の名前。デバッグに役立ちます。

11.1.257.6 p_lower_lvl_api

i2c_api_master_t::p_lower_lvl_api

概要説明

フレームワークで使用される I2C HAL インタフェースへのポインタ。

11.1.257.7 device_count

uint8_t sf_i2c_bus_t::device_count

概要説明

初期値は 0 です。

11.1.258 sf_i2c_cfg_t

```
typedef struct{
    sf_i2c_bus_t *  p_bus
    i2c_cfg_t const *  p_lower_lvl_cfg
} sf_i2c_cfg_t
```

11.1.258.1 p_bus

sf_i2c_bus_t::p_bus

概要説明

デバイスが使用するバス。

11.1.258.2 p_lower_lvl_cfg

i2c_cfg_t::p_lower_lvl_cfg

概要説明

I2C HAL 設定へのポインタ。

11.1.259 sf_i2c_instance_t

```
typedef struct{
    sf_i2c_ctrl_t *  p_ctrl
    sf_i2c_cfg_t const *  p_cfg
    sf_i2c_api_t const *  p_api
} sf_i2c_instance_t
```

11.1.259.1 p_ctrl

sf_i2c_ctrl_t* sf_i2c_instance_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.259.2 p_cfg

sf_i2c_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.259.3 p_api

sf_i2c_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.260 sf_jpeg_decode_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg)
   ssp_err_t(* inputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const buffer_size)
   ssp_err_t(* outputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
   ssp_err_t(* linesDecodedGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
   ssp_err_t(* horizontalStrideSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
   ssp_err_t(* imageSubsampleSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample,
jpeg_decode_subsample_t vertical_subsample)
   ssp_err_t(* wait)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status, uint32_t timeout)
   ssp_err_t(* statusGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
   ssp_err_t(* imageSizeGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
   ssp_err_t(* pixelFormatGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)
   ssp_err_t(* close)(sf_jpeg_decode_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_jpeg_decode_api_t
```

11.1.261 sf_jpeg_decode_cfg_t

```
typedef struct{
   jpeg_decode_instance_t const * p_lower_lvl_jpeg_decode
} sf_jpeg_decode_cfg_t
```

11.1.261.1 p_lower_lvl_jpeg_decode

jpeg_decode_instance_t::p_lower_lvl_jpeg_decode

詳細説明

このインタフェースを実装するドライバ構造体へのポインタ。r_jpeg_decode.c に設定済みのドライバ構造体が用意されており、r_jpeg_decode.h に extern されています。

11.1.262 sf_jpeg_decode_ctrl_t

```
typedef struct{
   uint32_t open
   uint32_t state
   TX_MUTEX mutex
   TX_EVENT_FLAGS_GROUP events
   jpeg_decode_api_t const * p_api
   jpeg_decode_ctrl_t ctrl
} sf_jpeg_decode_ctrl_t
```

11.1.262.1 open

uint32_t sf_jpeg_decode_ctrl_t::open

概要説明

ドライバが開いているかどうかを示します。

11.1.262.2 state

uint32_t sf_jpeg_decode_ctrl_t::state

概要説明

ドライバによって使用され、制御ブロックへのポインタが有効かどうかを確認します。

11.1.262.3 mutex

TX_MUTEX sf_jpeg_decode_ctrl_t::mutex

概要説明

ローレベル ドライバ ハードウェアへのアクセスを保護するためのミューテックス。

11.1.262.4 events

TX_EVENT_FLAGS_GROUP sf_jpeg_decode_ctrl_t::events

概要説明

HAL ドライバがフレームワーク ドライバへの通知に使用するイベント フラグ。

11.1.262.5 p_api

jpeg_decode_api_t::p_api

概要説明

ローレベル ドライバ関数ポインタへのポインタ。

11.1.262.6 ctrl

jpeg_decode_ctrl_t::ctrl

概要説明

ローレベル ドライバ制御ブロック。

11.1.263 sf_jpeg_decode_instance_t

```
typedef struct{
    sf_jpeg_decode_ctrl_t * p_ctrl
    sf_jpeg_decode_cfg_t const * p_cfg
    sf_jpeg_decode_api_t const * p_api
} sf_jpeg_decode_instance_t
```

11.1.263.1 p_ctrl

sf_jpeg_decode_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.263.2 p_cfg

sf_jpeg_decode_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.263.3 p_api

sf_jpeg_decode_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.264 sf_message_acquire_cfg_t

```
typedef struct{
    bool buffer_keep
} sf_message_acquire_cfg_t
```

11.1.264.1 buffer_keep

bool sf_message_acquire_cfg_t::buffer_keep

概要説明

バッファ キープ オプション。

11.1.265 sf_message_api_t

```
typedef struct{
   ssp_err_t(*  open)(sf_message_ctrl_t *const p_ctrl, sf_message_cfg_t const *const p_cfg)
   ssp_err_t(*  close)(sf_message_ctrl_t *const p_ctrl)
   ssp_err_t(*  bufferAcquire)(sf_message_ctrl_t const *const p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t
const *const p_acquire_cfg, uint32_t const wait_option)
   ssp_err_t(*  bufferRelease)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t
const option)
   ssp_err_t(*  post)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const
p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)
   ssp_err_t(*  pend)(sf_message_ctrl_t const *const p_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer,
uint32_t const wait_option)
   ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_message_api_t
```

11.1.266 sf_message_buffer_ctrl_t

```
typedef struct{
   void(*  p_callback)(sf_message_callback_args_t *)
   void const *  p_context
   struct sf_message_buffer_ctrl_t::st_buffer_ctrl_flag  flag_b
} sf_message_buffer_ctrl_t
```

11.1.266.1 p_callback

void(* sf_message_buffer_ctrl_t::p_callback)(sf_message_callback_args_t *)

概要説明

オプションのユーザー コールバック関数。

11.1.266.2 p_context

void const* sf_message_buffer_ctrl_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.266.3 flag_b

sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::flag_b

11.1.267 sf_message_buffer_ctrl_t::st_buffer_ctrl_flag

```
typedef struct{
   uint32_t semaphore
   uint32_t buffer_keep
   uint32_t nak_response
   uint32_t reserved
   uint32_t in_use
} sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
```

11.1.267.1 semaphore

uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::semaphore

概要説明

バッファの解放を防止するカウンティング セマフォ。

11.1.267.2 buffer_keep

uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::buffer_keep

概要説明

バッファ キープ要求。

11.1.267.3 nak_response

uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::nak_response

概要説明

NAK （サブスクライバーが複数存在する場合の論理和 （OR）ロジック）

11.1.267.4 reserved

uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::reserved

概要説明

予備ビット。

11.1.267.5 in_use

uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::in_use

概要説明

使用中のバッファ。

11.1.268 sf_message_callback_args_t

```
typedef struct{
   sf_message_callback_event_t event
   void const * p_context
} sf_message_callback_args_t
```

11.1.268.1 event

sf_message_callback_event_t::event

概要説明

イベント コード。

11.1.268.2 p_context

void const* sf_message_callback_args_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.269 sf_message_cfg_t

```
typedef struct{
   void * p_work_memory_start
   uint32_t work_memory_size_bytes
   uint32_t buffer_size
   sf_message_subscriber_list_t ** pp_subscriber_lists
   uint8_t * p_block_pool_name
} sf_message_cfg_t
```

11.1.269.1 p_work_memory_start

void* sf_message_cfg_t::p_work_memory_start

概要説明

メモリ領域の開始アドレス。

11.1.269.2 work_memory_size_bytes

uint32_t sf_message_cfg_t::work_memory_size_bytes

概要説明

作業メモリ領域のサイズ （バイト単位）。

11.1.269.3 buffer_size

uint32_t sf_message_cfg_t::buffer_size

概要説明

メッセージ ブロックのバイト数。

11.1.269.4 pp_subscriber_lists

sf_message_subscriber_list_t::pp_subscriber_lists

概要説明

サブスクライバー リストへのポインタ配列。

11.1.269.5 p_block_pool_name

uint8_t* sf_message_cfg_t::p_block_pool_name

概要説明

ブロック プール名へのポインタ。

11.1.270 sf_message_ctrl_t

```
typedef struct{
   TX_BLOCK_POOL block_pool
   sf_message_subscriber_list_t ** pp_subscriber_lists
   uint32_t buffer_size
   uint32_t number_of_buffers
   uint16_t number_of_subscriber_groups
   sf_message_state_t state
} sf_message_ctrl_t
```

11.1.270.1 block_pool

TX_BLOCK_POOL sf_message_buffer_ctrl_t::block_pool

概要説明

メモリ ブロック プール制御へのポインタ。

11.1.270.2 pp_subscriber_lists

sf_message_subscriber_list_t::pp_subscriber_lists

概要説明

サブスクライバー リストへのポインタ配列。

11.1.270.3 buffer_size

uint32_t sf_message_buffer_ctrl_t::buffer_size

概要説明

メッセージ バッファのバイト数。

11.1.270.4 number_of_buffers

uint32_t sf_message_buffer_ctrl_t::number_of_buffers

概要説明

割り当てられたバッファの数。

11.1.270.5 number_of_subscriber_groups

uint16_t sf_message_buffer_ctrl_t::number_of_subscriber_groups

概要説明

サブスクライバー グループの数。

11.1.270.6 state

sf_message_state_t::state

概要説明

メッセージ フレームワークのステータス。

11.1.271 sf_message_header_t

```
typedef struct{
   uint32_t event
   uint32_t class_instance
   uint32_t code
   struct{}          event_b
   union{}           union{}
} sf_message_header_t
```

11.1.271.1 event

uint32_t sf_message_header_t::event

11.1.271.2 class_instance

uint32_t sf_message_header_t::class_instance

概要説明

< イベント クラス コード

詳細説明

イベント クラス インスタンス番号

11.1.271.3 code

uint32_t sf_message_header_t::code

概要説明

イベント コード。

11.1.271.4 event_b

このメンバーの定義については、ソース コードを参照してください。

11.1.271.5 union{}

このメンバーの定義については、ソース コードを参照してください。

11.1.272 sf_message_instance_range_t

```
typedef struct{
   uint8_t start
   uint8_t end
} sf_message_instance_range_t
```

11.1.272.1 start

uint8_t sf_message_instance_range_t::start

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.272.2 end

uint8_t sf_message_instance_range_t::end

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.273 sf_message_instance_t

```
typedef struct{
   sf_message_ctrl_t * p_ctrl
   sf_message_cfg_t const * p_cfg
   sf_message_api_t const * p_api
} sf_message_instance_t
```

11.1.273.1 p_ctrl

sf_message_buffer_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.273.2 p_cfg

sf_message_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.273.3 p_api

sf_message_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.274 sf_message_post_cfg_t

```
typedef struct{
   sf_message_priority_t priority
   void(* p_callback)(sf_message_callback_args_t *)
   void const * p_context
} sf_message_post_cfg_t
```

11.1.274.1 priority

sf_message_priority_t::priority

概要説明

メッセージの優先度。

11.1.274.2 p_callback

void(* sf_message_post_cfg_t::p_callback)(sf_message_callback_args_t *)

概要説明

ユーザー コールバック関数。

11.1.274.3 p_context

void const* sf_message_post_cfg_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.275 sf_message_post_err_t

```
typedef struct{
   TX_QUEUE * p_queue
} sf_message_post_err_t
```

11.1.275.1 p_queue

TX_QUEUE* sf_message_post_err_t::p_queue

概要説明

キュー。

11.1.276 sf_message_subscriber_list_t

```
typedef struct{
   sf_message_event_class_t event_class
   uint16_t number_of_nodes
   sf_message_subscriber_t ** pp_subscriber_group
} sf_message_subscriber_list_t
```

11.1.276.1 event_class

sf_message_event_class_t sf_message_subscriber_list_t::event_class

概要説明

イベント クラス コード。

11.1.276.2 number_of_nodes

uint16_t sf_message_subscriber_list_t::number_of_nodes

概要説明

サブスクライバー グループ内のノード数。

11.1.276.3 pp_subscriber_group

sf_message_subscriber_t::pp_subscriber_group

概要説明

イベント クラスのサブスクライバー グループ。

11.1.277 sf_message_subscriber_list_t

```
typedef struct{
   TX_QUEUE * p_queue
   sf_message_instance_range_t instance_range
} sf_message_subscriber_t
```

11.1.277.1 p_queue

TX_QUEUE* sf_message_subscriber_t::p_queue

概要説明

サブスクライバー スレッド用のメッセージ キューへのポインタ。

11.1.277.2 instance_range

sf_message_instance_range_t::instance_range

概要説明

メッセージを受信するイベント クラス インスタンスの範囲。

11.1.278 sf_power_profiles_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_power_profiles_ctrl_t *const p_ctrl, sf_power_profiles_cfg_t const *const p_cfg)
    ssp_err_t(*  sleep)(sf_power_profiles_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(sf_power_profiles_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_power_profiles_api_t
```

11.1.279 sf_power_profiles_callback_args_t

```
typedef struct{
    sf_power_profiles_event_t  event
    void *  p_context
} sf_power_profiles_callback_args_t
```

11.1.279.1 event

sf_power_profiles_event_t::event

概要説明

電力プロファイル コールバック イベント。

11.1.279.2 p_context

void* sf_power_profiles_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.280 sf_power_profiles_cfg_t

```
typedef struct{
    ioport_cfg_t const *const  p_wake_ioport_pin_tbl
    ioport_cfg_t const *const  p_sleep_ioport_pin_tbl
    sf_power_profiles_mode_t  operating_mode
    bool  retain_output_signals
    lpm_instance_t const *const  p_lower_lvl_lpm
    rtc_instance_t const *const  p_lower_lvl_rtc
    void(*  p_callback)(sf_power_profiles_callback_args_t *p_args)
    void *  p_context
} sf_power_profiles_cfg_t
```

11.1.280.1 p_wake_ioport_pin_tbl

ioport_cfg_t::p_wake_ioport_pin_tbl

概要説明

ウェイクアップの I/O ポート設定へのポインタ。

11.1.280.2 p_sleep_ioport_pin_tbl

ioport_cfg_t::p_sleep_ioport_pin_tbl

概要説明

スリープの I/O ポート設定へのポインタ。

11.1.280.3 operating_mode

sf_power_profiles_mode_t::operating_mode

概要説明

使用する電力プロファイル モード。

11.1.280.4 retain_output_signals

bool sf_power_profiles_cfg_t::retain_output_signals

詳細説明

（実装予定 :）True （デフォルト）== > アドレス バスとバスコントロール信号が、出力状態を保持します
False == > 信号が高インピーダンス状態に設定されます

11.1.280.5 p_lower_lvl_lpm

lpm_instance_t::p_lower_lvl_lpm

概要説明

LPM インスタンスへのポインタ。

11.1.280.6 p_lower_lvl_rtc

rtc_instance_t::p_lower_lvl_rtc

概要説明

RTC インスタンスへのポインタ （存在する場合）。

11.1.280.7 p_callback

void(* sf_power_profiles_cfg_t::p_callback)(sf_power_profiles_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.280.8 p_context

void* sf_power_profiles_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.281 sf_power_profiles_ctrl_t

```
typedef struct{
    uint32_t  open
    ioport_cfg_t const *  p_wake_ioport_pin_tbl
    ioport_cfg_t const *  p_sleep_ioport_pin_tbl
    sf_power_profiles_mode_t  operating_mode
    lpm_api_t const *  p_api_lpm
    rtc_api_t const *  p_api_rtc
    rtc_ctrl_t *  p_ctrl_rtc
    void(*  p_callback)(sf_power_profiles_callback_args_t *p_args)
    void *  p_context
} sf_power_profiles_ctrl_t
```

11.1.281.1 open

uint32_t sf_power_profiles_ctrl_t::open

概要説明

ドライバによって使用され、制御ブロックへのポインタが有効かどうかを確認します。

11.1.281.2 p_wake_ioport_pin_tbl

ioport_cfg_t::p_wake_ioport_pin_tbl

概要説明

ウェイクアップの I/O ポート設定へのポインタ。

11.1.281.3 p_sleep_ioport_pin_tbl

ioport_cfg_t::p_sleep_ioport_pin_tbl

概要説明

スリープの I/O ポート設定へのポインタ。

11.1.281.4 operating_mode

sf_power_profiles_mode_t::operating_mode

概要説明

使用する電力プロファイル モード。

11.1.281.5 p_api_lpm

lpm_api_t::p_api_lpm

概要説明

ローレベル ローパワー ドライバ関数ポインタへのポインタ。

11.1.281.6 p_api_rtc

rtc_api_t::p_api_rtc

概要説明

ローレベル RTC ドライバ関数ポインタへのポインタ。

11.1.281.7 p_ctrl_rtc

rtc_ctrl_t::p_ctrl_rtc

概要説明

ローレベル RTC ドライバ制御ブロックへのポインタ。

11.1.281.8 p_callback

void(* sf_power_profiles_ctrl_t::p_callback)(sf_power_profiles_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.281.9 p_context

void* sf_power_profiles_ctrl_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.282 sf_power_profiles_instance_t

```
typedef struct{
    sf_power_profiles_ctrl_t *  p_ctrl
    sf_power_profiles_cfg_t const *  p_cfg
    sf_power_profiles_api_t const *  p_api
} sf_power_profiles_instance_t
```

11.1.282.1 p_ctrl

sf_power_profiles_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.282.2 p_cfg

sf_power_profiles_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.282.3 p_api

sf_power_profiles_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.283 sf_slider_on_ctsu_cfg_t

```
typedef struct{
    const sf_slider_type_t type
    uint32_t num_slider_channels
    ctsu_channel_pair_t const * p_slider_channels
    int32_t const * p_normalization
    int32_t * p_channel_average
    uint32_t * p_offset
    uint16_t *const p_slider_scount
    uint16_t *const p_slider_baseline
    int32_t *const p_slider_delta
    sf_touch_ctsu_slider_id_t id
    int32_t max_slider_value
    const uint16_t slider_norm_max
    const int32_t slider_threshold
    const int32_t channel_average_weight
    const int32_t prev_sum_weight
    const int32_t cutoff
} sf_slider_on_ctsu_cfg_t
```

11.1.283.1 type

sf_slider_type_t::type

詳細説明

線形または円形 （ホイール）

11.1.283.2 num_slider_channels

uint32_t ::num_slider_channels

11.1.283.3 p_slider_channels

ctsu_channel_pair_t::p_slider_channels

詳細説明

スライダを構成するチャネル / チャネル ペアを定義します。

11.1.283.4 p_normalization

int32_t const* ::p_normalization

11.1.283.5 p_channel_average

int32_t* ::p_channel_average

11.1.283.6 p_offset

uint32_t* ::p_offset

11.1.283.7 p_slider_scount

uint16_t* const ::p_slider_scount

11.1.283.8 p_slider_baseline

uint16_t* const ::p_slider_baseline

11.1.283.9 p_slider_delta

int32_t* const ::p_slider_delta

11.1.283.10 id

sf_touch_ctsu_slider_id_t::id

詳細説明

スライダの一意の識別子

11.1.283.11 max_slider_value

int32_t ::max_slider_value

詳細説明

スライダの最大値、0 より大きい値でなければならず、最小値は常に 0

11.1.283.12 slider_norm_max

const uint16_t ::slider_norm_max

詳細説明

ユーザーにより変更可能な個々のスライダ設定。
st_sf_touch_ctsu_slider_hdlTOUCH_SLIDER_CFG_NORM_MAX と同一でなければなりません

11.1.283.13 slider_threshold

const int32_t ::slider_threshold

詳細説明

タッチのしきい値。値が 0 より大きいことを確認してください

11.1.283.14 channel_average_weight

const int32_t ::channel_average_weight

詳細説明

各チャネルに対するカウントの加重移動平均

11.1.283.15 prev_sum_weight

const int32_t ::prev_sum_weight

詳細説明

位置計算における前回の合計の加重移動平均 （0 より大きい値でなければなりません）

11.1.283.16 cutoff

const int32_t ::cutoff

詳細説明

prev_sum 移動平均をどの程度下回った場合に「SF_TOUCH_SLIDER_STATE_RELEASED」を検出するか定義します

11.1.284 sf_spi_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)
   ssp_err_t(* read)(sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
   ssp_err_t(* write)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
   ssp_err_t(* writeRead)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
   ssp_err_t(* close)(sf_spi_ctrl_t *const p_ctrl)
   ssp_err_t(* lock)(sf_spi_ctrl_t *const p_ctrl)
   ssp_err_t(* unlock)(sf_spi_ctrl_t *const p_ctrl)
   ssp_err_t(* version)(ssp_version_t *const p_version)
} sf_spi_api_t
```

11.1.285 sf_spi_bus_t

```
typedef struct{
   uint8_t channel
   uint32_t freq_hz_max
   uint32_t freq_hz_min
   TX_MUTEX * p_lock_mutex
   TX_EVENT_FLAGS_GROUP * p_sync_eventflag
   sf_spi_ctrl_t ** pp_curr_ctrl
   uint8_t * p_bus_name
   spi_api_t * p_lower_lvl_api
   uint8_t device_count
} sf_spi_bus_t
```

11.1.285.1 channel

uint8_t sf_spi_bus_t::channel

概要説明

チャネル。

11.1.285.2 freq_hz_max

uint32_t sf_spi_bus_t::freq_hz_max

概要説明

サポートされている最大バス周波数。

11.1.285.3 freq_hz_min

uint32_t sf_spi_bus_t::freq_hz_min

概要説明

サポートされている最小バス周波数。

11.1.285.4 p_lock_mutex

TX_MUTEX* sf_spi_bus_t::p_lock_mutex

概要説明

このチャネルのロック ミューテックス ハンドル。

11.1.285.5 p_sync_eventflag

TX_EVENT_FLAGS_GROUP* sf_spi_bus_t::p_sync_eventflag

概要説明

SPI データ転送用のイベント フラグ オブジェクトへのポインタ。

11.1.285.6 pp_curr_ctrl

sf_spi_ctrl_t** sf_spi_bus_t::pp_curr_ctrl

概要説明

バスを使用している現在のデバイス。

11.1.285.7 p_bus_name

uint8_t* sf_spi_bus_t::p_bus_name

概要説明

ペリフェラル名 SSPI/RSPI

11.1.285.8 p_lower_lvl_api

spi_api_t::p_lower_lvl_api

概要説明

フレームワークで使用される SPI HAL インタフェースへのポインタ。

11.1.285.9 device_count

uint8_t sf_spi_bus_t::device_count

概要説明

初期値は 0 です。

11.1.286 sf_spi_cfg_t

```
typedef struct{
    sf_spi_bus_t * p_bus
    ioport_port_pin_t chip_select
    ioport_level_t chip_select_level_active
    spi_cfg_t const * p_lower_lvl_cfg
} sf_spi_cfg_t
```

11.1.286.1 p_bus

sf_spi_bus_t::p_bus

概要説明

デバイスが使用するバス。

11.1.286.2 chip_select

ioport_port_pin_t sf_spi_cfg_t::chip_select

概要説明

このデバイスに対するチップの選択。

11.1.286.3 chip_select_level_active

ioport_level_t::chip_select_level_active

概要説明

CS 極性、アクティブ High または Low。

11.1.286.4 p_lower_lvl_cfg

spi_cfg_t::p_lower_lvl_cfg

概要説明

SPI HAL 設定へのポインタ。

11.1.287 sf_spi_instance_t

```
typedef struct{
   sf_spi_ctrl_t * p_ctrl
   sf_spi_cfg_t const * p_cfg
   sf_spi_api_t const * p_api
} sf_spi_instance_t
```

11.1.287.1 p_ctrl

sf_spi_ctrl_t* sf_spi_instance_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.287.2 p_cfg

sf_spi_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.287.3 p_api

sf_spi_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.288 sf_thread_monitor_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_cfg_t const *const p_cfg)
   ssp_err_t(* close)(sf_thread_monitor_ctrl_t *const p_ctrl)
   ssp_err_t(* threadRegister)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_counter_min_max_t const
*p_counter_min_max)
   ssp_err_t(* threadUnregister)(sf_thread_monitor_ctrl_t *const p_ctrl)
   ssp_err_t(* countIncrement)(sf_thread_monitor_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_thread_monitor_api_t
```

11.1.289 sf_thread_monitor_cfg_t

```
typedef struct{
   wdt_instance_t const * p_lower_lvl_wdt
   bool profiling_mode_enabled
   UINT priority
} sf_thread_monitor_cfg_t
```

11.1.289.1 p_lower_lvl_wdt

wdt_instance_t::p_lower_lvl_wdt

概要説明

ローレベル ウォッチドッグ インスタンスへのポインタ。

11.1.289.2 profiling_mode_enabled

bool sf_thread_monitor_cfg_t::profiling_mode_enabled

概要説明

プロファイリング モードを有効化または無効化します。

11.1.289.3 priority

UINT sf_thread_monitor_cfg_t::priority

概要説明

スレッド監視スレッドのプライオリティ。

11.1.290 sf_thread_monitor_counter_min_max_t

```
typedef struct{
   uint32_t minimum_count
   uint32_t maximum_count
} sf_thread_monitor_counter_min_max_t
```

11.1.290.1 minimum_count

uint32_t sf_thread_monitor_counter_min_max_t::minimum_count

詳細説明

最小期待カウント値。現在のカウントがこの値よりも小さい場合、ウォッチドッグがリセットされます。

11.1.290.2 maximum_count

uint32_t sf_thread_monitor_counter_min_max_t::maximum_count

詳細説明

最大期待カウント値。現在のカウントがこの値よりも大きい場合、ウォッチドッグがリセットされます

11.1.291 sf_thread_monitor_ctrl_t

```
typedef struct{
    uint32_t open
    wdt_instance_t const * p_lower_lvl_wdt
    uint32_t timeout_period_msec
    uint32_t timeout_period_watchdog_clocks
    bool profiling_mode_enabled
    TX_MUTEX mutex
    uint32_t profiling_mode_check
    sf_thread_monitor_thread_counter_t thread_counters[THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS]
    TX_THREAD thread
    void const * p_extend
    uint8_t stack[THREAD_MONITOR_THREAD_STACK_SIZE]
} sf_thread_monitor_ctrl_t
```

11.1.291.1 open

uint32_t sf_thread_monitor_ctrl_t::open

詳細説明

ドライバによって使用され、制御構造体が有効かどうかを確認します

11.1.291.2 p_lower_lvl_wdt

wdt_instance_t::p_lower_lvl_wdt

詳細説明

ウォッチドッグ周辺機能のインタフェース構造体へのポインタ

11.1.291.3 timeout_period_msec

uint32_t sf_thread_monitor_ctrl_t::timeout_period_msec

詳細説明

ウォッチドッグ タイムアウト期間の時間 （ミリ秒単位)。スレッドを監視する期間を計算するために使用します。

11.1.291.4 timeout_period_watchdog_clocks

uint32_t sf_thread_monitor_ctrl_t::timeout_period_watchdog_clocks

詳細説明

ウォッチドッグの最大カウント値。カウントへの同期に使用されます。

11.1.291.5 profiling_mode_enabled

bool sf_thread_monitor_ctrl_t::profiling_mode_enabled

詳細説明

ドライバによって使用され、プロファイリング モードが有効かどうかを確認します。

11.1.291.6 mutex

TX_MUTEX sf_thread_monitor_ctrl_t::mutex

概要説明

スレッド カウンタへのアクセスを保護するためのミューテックス。

11.1.291.7 profiling_mode_check

uint32_t sf_thread_monitor_ctrl_t::profiling_mode_check

詳細説明

prfiling_mode_enabled == true の場合に、プロファイリング モードが有効であることを確認するために使用される値。

11.1.291.8 thread_counters

sf_thread_monitor_thread_counter_t::thread_counters

詳細説明

スレッド カウンタ情報用のデータ ストレージ。

11.1.291.9 thread

TX_THREAD sf_thread_monitor_ctrl_t::thread

概要説明

スレッド監視スレッド。

11.1.291.10 p_extend

void const* sf_thread_monitor_ctrl_t::p_extend

概要説明

拡張構成データ。

11.1.291.11 stack

uint8_t sf_thread_monitor_ctrl_t::stack[THREAD_MONITOR_THREAD_STACK_SIZE]

詳細説明

スレッド監視スレッドのスタック。

11.1.292 sf_thread_monitor_instance_t

```
typedef struct{
    sf_thread_monitor_ctrl_t *  p_ctrl
    sf_thread_monitor_cfg_t const *  p_cfg
    sf_thread_monitor_api_t const *  p_api
} sf_thread_monitor_instance_t
```

11.1.292.1 p_ctrl

sf_thread_monitor_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.292.2 p_cfg

sf_thread_monitor_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.292.3 p_api

sf_thread_monitor_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.293 sf_thread_monitor_thread_counter_t

```
typedef struct{
   uint32_t current_count
   uint32_t minimum_count
   uint32_t maximum_count
   bool active
   TX_THREAD * p_thread
} sf_thread_monitor_thread_counter_t
```

11.1.293.1 current_count

uint32_t sf_thread_monitor_thread_counter_t::current_count

概要説明

スレッドの現在のカウント値。

11.1.293.2 minimum_count

uint32_t sf_thread_monitor_thread_counter_t::minimum_count

詳細説明

最小期待カウント値。現在のカウントがこの値よりも小さい場合、ウォッチドッグがリセットされます。

11.1.293.3 maximum_count

uint32_t sf_thread_monitor_thread_counter_t::maximum_count

詳細説明

最大期待カウント値。現在のカウントがこの値よりも大きい場合、ウォッチドッグがリセットされます

11.1.293.4 active

bool sf_thread_monitor_thread_counter_t::active

詳細説明

監視スレッドに対し、このカウント データが現在アクティブかどうかを示します。スレッドを登録した時点では、カウントが途中からになり、監視対象とすべきでないため、この値が false に設定されます。スレッド監視スレッドによってすべてのカウントがゼロにクリアされると、この値が true に変わります。

11.1.293.5 p_thread

TX_THREAD* sf_thread_monitor_thread_counter_t::p_thread

概要説明

このカウンタ データのスレッドへのポインタ。

11.1.294 sf_touch_ctsu_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)
   ssp_err_t(* read)(sf_touch_ctsu_ctrl_t *const p_ctrl, void *p_dest, ctsu_read_t opts, const ctsu_channel_pair_t *channels, const
uint16_t count)
   ssp_err_t(* close)(sf_touch_ctsu_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_touch_ctsu_api_t
```

11.1.295 sf_touch_ctsu_button_api_t

```
typedef struct{
   ssp_err_t(* open)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_cfg_t const *const p_cfg)
   ssp_err_t(* enable)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_id const button_id)
   ssp_err_t(* disable)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_id const button_id)
   ssp_err_t(* close)(sf_touch_ctsu_button_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_touch_ctsu_button_api_t
```

11.1.296 sf_touch_ctsu_button_callback_args_t

```
typedef struct{
   sf_touch_ctsu_button_id id
   sf_touch_button_state_t event
   void const * p_context
} sf_touch_ctsu_button_callback_args_t
```

11.1.296.1 id

sf_touch_ctsu_button_id::id

概要説明

ボタンの一意の識別子。

11.1.296.2 event

sf_touch_button_state_t::event

概要説明

ボタン コールバック イベント。

11.1.296.3 p_context

void const* sf_touch_ctsu_button_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.297 sf_touch_ctsu_button_cfg_t

```
typedef struct{
    sf_touch_ctsu_instance_t const *const  p_lower_lvl_touch_framework
    uint32_t  button_count
    sf_touch_ctsu_button_individual_t **  pp_button_cfgs
    void(*  p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)
    void const *  p_context
    void const *  p_extend
} sf_touch_ctsu_button_cfg_t
```

11.1.297.1 p_lower_lvl_touch_framework

sf_touch_ctsu_button_instance_t::p_lower_lvl_touch_framework

概要説明

タッチ フレームワーク インスタンスへのポインタ。

11.1.297.2 button_count

uint32_t sf_touch_ctsu_button_cfg_t::button_count

概要説明

構成内のボタンの数。

11.1.297.3 pp_button_cfgs

sf_touch_ctsu_button_individual_t::pp_button_cfgs

概要説明

ボタン構成へのポインタ。

11.1.297.4 p_callback

void(* sf_touch_ctsu_button_cfg_t::p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.297.5 p_context

void const* sf_touch_ctsu_button_cfg_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.297.6 p_extend

void const* sf_touch_ctsu_button_cfg_t::p_extend

概要説明

インスタンス固有の設定値に対応するための拡張パラメータ。

11.1.298 sf_touch_ctsu_button_ctrl_t

```
typedef struct{
    uint32_t  opened
    sf_touch_ctsu_button_hdl *  p_button_hdl
    uint32_t  button_count
    sf_touch_ctsu_api_t const *  p_lower_lvl_api
    sf_touch_ctsu_button_ctrl_t  lower_lvl_ctrl
    void const *  p_context
    void(*  p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)
} sf_touch_ctsu_button_ctrl_t
```

11.1.298.1 opened

uint32_t sf_touch_ctsu_button_ctrl_t::opened

概要説明

初期化状態を保存します。

11.1.298.2 p_button_hdl

sf_touch_ctsu_button_hdl::p_button_hdl

概要説明

ボタン ハンドルへのポインタ。

11.1.298.3 button_count

uint32_t sf_touch_ctsu_button_ctrl_t::button_count

概要説明

ボタン カウント。

11.1.298.4 p_lower_lvl_api

sf_touch_ctsu_api_t::p_lower_lvl_api

概要説明

ローレベル インスタンスの API 構造体へのポインタ。

11.1.298.5 lower_lvl_ctrl

sf_touch_ctsu_ctrl_t::lower_lvl_ctrl

概要説明

ローレベル インスタンスの制御構造体。

11.1.298.6 p_context

void const* sf_touch_ctsu_button_ctrl_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.298.7 p_callback

void(* sf_touch_ctsu_button_ctrl_t::p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.299 sf_touch_ctsu_button_hdl

```
typedef struct{
    sf_touch_ctsu_button_individual_t button_cfg
    sf_touch_button_state_t state
    int16_t offset
    sf_touch_button_state_t prev_state
    uint32_t debounce_counter
    uint32_t active_event_counter
    uint32_t press_down_counter
    uint32_t open
} sf_touch_ctsu_button_hdl
```

11.1.299.1 button_cfg

sf_touch_ctsu_button_individual_t::button_cfg

概要説明

個別のボタン構成。

11.1.299.2 state

sf_touch_button_state_t::state

概要説明

ボタンの現在の状態を表します。

11.1.299.3 offset

int16_t sf_touch_ctsu_button_hdl::offset

概要説明

結果配列でのオフセットおよびバイナリでのビット オフセット。

11.1.299.4 prev_state

sf_touch_button_state_t::prev_state

概要説明

ボタンの以前の状態を保持します。

11.1.299.5 debounce_counter

uint32_t sf_touch_ctsu_button_hdl::debounce_counter

概要説明

カウンタが復帰しないようにします。

11.1.299.6 active_event_counter

uint32_t sf_touch_ctsu_button_hdl::active_event_counter

概要説明

ボタンが 2 つの状態の間にある時間の長さ。

11.1.299.7 press_down_counter

uint32_t sf_touch_ctsu_button_hdl::press_down_counter

概要説明

ボタンが押されたままになる時間の長さ。

11.1.299.8 open

uint32_t sf_touch_ctsu_button_hdl::open

概要説明

ボタンが開いていることを示します。

11.1.300 sf_touch_ctsu_button_individual_t

```
typedef struct{
  ctsu_channel_pair_t button_channel
  uint8_t release_enable
  uint8_t press_enable
  uint8_t repeat_enable
  uint8_t shorthold_enable
  uint8_t longhold_enable
  uint8_t byte
  union{}         event_enable
  uint16_t debounce_threshold
  sf_touch_ctsu_button_id id
} sf_touch_ctsu_button_individual_t
```

11.1.300.1 button_channel

ctsu_channel_pair_t::button_channel

概要説明

ボタンを構成するチャネル / チャネル ペアを定義します。

11.1.300.2 release_enable

uint8_t sf_touch_ctsu_button_individual_t::release_enable

概要説明

ボタン離しイベントを有効化します

11.1.300.3 press_enable

uint8_t sf_touch_ctsu_button_individual_t::press_enable

概要説明

ボタン押しイベントを有効化します

11.1.300.4 repeat_enable

uint8_t sf_touch_ctsu_button_individual_t::repeat_enable

概要説明

繰り返しイベントを有効化します

11.1.300.5 shorthold_enable

uint8_t sf_touch_ctsu_button_individual_t::shorthold_enable

概要説明

短押しイベントを有効化します

11.1.300.6 longhold_enable

uint8_t sf_touch_ctsu_button_individual_t::longhold_enable

概要説明

長押しイベントを有効化します

11.1.300.7 byte

uint8_t sf_touch_ctsu_button_individual_t::byte

概要説明

イベントのバイト表現を有効化します。

11.1.300.8 event_enable

このメンバーの定義については、ソース コードを参照してください。

11.1.300.9 debounce_threshold

uint16_t sf_touch_ctsu_button_individual_t::debounce_threshold

概要説明

ボタンがタッチされたと判定されて状態が変化するまでの連続した時間の長さ。

11.1.300.10 id

sf_touch_ctsu_button_id::id

概要説明

ボタンの一意の識別子。

11.1.301 sf_touch_ctsu_button_instance_t

```
typedef struct{
    sf_touch_ctsu_button_ctrl_t * p_ctrl
    sf_touch_ctsu_button_cfg_t const * p_cfg
    sf_touch_ctsu_button_api_t const * p_api
} sf_touch_ctsu_button_instance_t
```

11.1.301.1 p_ctrl

sf_touch_ctsu_button_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.301.2 p_cfg

sf_touch_ctsu_button_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.301.3 p_api

sf_touch_ctsu_button_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.302 sf_touch_ctsu_callback_args_t

```
typedef struct{
   void * p_context
} sf_touch_ctsu_callback_args_t
```

11.1.302.1 p_context

void* sf_touch_ctsu_callback_args_t::p_context

11.1.303 sf_touch_ctsu_cfg_t

```
typedef struct{
   UINT priority
   uint16_t update_hz
   void(* p_callback)(sf_touch_ctsu_callback_args_t *p_args)
   void * p_context
   ctsu_instance_t * p_ctsu_instance
} sf_touch_ctsu_cfg_t
```

11.1.303.1 priority

UINT sf_touch_ctsu_cfg_t::priority

概要説明

タッチ パネル スレッドの優先順位。

11.1.303.2 update_hz

uint16_t sf_touch_ctsu_cfg_t::update_hz

概要説明

スキャンの実行周波数。これは最新の open() 呼び出しにより設定されます。

11.1.303.3 p_callback

void(* sf_touch_ctsu_cfg_t::p_callback)(sf_touch_ctsu_callback_args_t *p_args)

概要説明

コールバック関数。

11.1.303.4 p_context

void* sf_touch_ctsu_cfg_t::p_context

概要説明

コールバックの引数。

11.1.303.5 p_ctsu_instance

ctsu_instance_t::p_ctsu_instance

概要説明

CTSU インスタンス。

11.1.304 sf_touch_ctsu_ctrl_t

```
typedef struct{
    uint32_t open
    uint16_t update_hz
    ctsu_instance_t * p_lower_lvl_instance
    uint32_t cb_index
} sf_touch_ctsu_ctrl_t
```

11.1.304.1 open

uint32_t sf_touch_ctsu_ctrl_t::open

概要説明

ドライバによって使用され、制御ブロックが有効かどうかを確認します。

11.1.304.2 update_hz

uint16_t sf_touch_ctsu_ctrl_t::update_hz

概要説明

スキャンの実行周波数。

11.1.304.3 p_lower_lvl_instance

ctsu_instance_t::p_lower_lvl_instance

概要説明

CTSU インスタンスへのポインタ。

11.1.304.4 cb_index

uint32_t sf_touch_ctsu_ctrl_t::cb_index

概要説明

コールバック レジストリ テーブルのインデックスを示します。

11.1.305 sf_touch_ctsu_instance_t

```
typedef struct{
    sf_touch_ctsu_ctrl_t * p_ctrl
    sf_touch_ctsu_cfg_t const * p_cfg
    sf_touch_ctsu_api_t const * p_api
} sf_touch_ctsu_instance_t
```

11.1.305.1 p_ctrl

sf_touch_ctsu_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.305.2 p_cfg

sf_touch_ctsu_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.305.3 p_api

sf_touch_ctsu_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.306 sf_touch_ctsu_slider_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_cfg_t const *const p_cfg)
    ssp_err_t(*  enable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
    ssp_err_t(*  disable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
    ssp_err_t(*  close)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_touch_ctsu_slider_api_t
```

11.1.307 sf_touch_ctsu_slider_callback_args_t

```
typedef struct{
    sf_touch_ctsu_slider_id_t  id
    uint32_t  event
    uint32_t  current_position
    void const *  p_context
} sf_touch_ctsu_slider_callback_args_t
```

11.1.307.1 id

sf_touch_ctsu_slider_id_t::id

概要説明

スライダの一意の識別子。

11.1.307.2 event

uint32_t sf_touch_ctsu_slider_callback_args_t::event

概要説明

スライダ コールバック イベント。

11.1.307.3 current_position

uint32_t sf_touch_ctsu_slider_callback_args_t::current_position

概要説明

現在のスライダ位置。

11.1.307.4 p_context

void const* sf_touch_ctsu_slider_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.308 sf_touch_ctsu_slider_cfg_t

```
typedef struct{
   sf_touch_ctsu_instance_t * p_lower_lvl_touch_framework
   uint32_t slider_count
   void(* p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)
   void const * p_context
   void const * p_extend
} sf_touch_ctsu_slider_cfg_t
```

11.1.308.1 p_lower_lvl_touch_framework

sf_touch_ctsu_button_instance_t::p_lower_lvl_touch_framework

詳細説明

タッチ フレームワーク インスタンスへのポインタ

11.1.308.2 slider_count

uint32_t sf_touch_ctsu_slider_cfg_t::slider_count

詳細説明

構成内のスライダの数

11.1.308.3 p_callback

void(* sf_touch_ctsu_slider_cfg_t::p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)

詳細説明

コールバック関数

11.1.308.4 p_context

void const* sf_touch_ctsu_slider_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー

11.1.308.5 p_extend

void const* sf_touch_ctsu_slider_cfg_t::p_extend

詳細説明

インスタンス固有の設定値に対応するための拡張パラメータ。

11.1.309 sf_touch_ctsu_slider_ctrl_t

```
typedef struct{
    uint32_t opened
    uint32_t slider_count
    sf_touch_ctsu_api_t const * p_lower_lvl_api
    sf_touch_ctsu_button_ctrl_t lower_lvl_ctrl
    void const * p_context
    void(* p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)
} sf_touch_ctsu_slider_ctrl_t
```

11.1.309.1 opened

uint32_t sf_touch_ctsu_slider_ctrl_t::opened

概要説明

フレームワークの初期化状態を保存します。

11.1.309.2 slider_count

uint32_t sf_touch_ctsu_slider_ctrl_t::slider_count

概要説明

スライダ カウント。

11.1.309.3 p_lower_lvl_api

sf_touch_ctsu_api_t::p_lower_lvl_api

概要説明

ローレベル インスタンスの API 構造体へのポインタ。

11.1.309.4 lower_lvl_ctrl

sf_touch_ctsu_ctrl_t::lower_lvl_ctrl

概要説明

ローレベル インスタンスの制御構造体。

11.1.309.5 p_context

void const* sf_touch_ctsu_slider_ctrl_t::p_context

概要説明

ユーザー データのプレースホルダー。

11.1.309.6 p_callback

void(* sf_touch_ctsu_slider_ctrl_t::p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)

詳細説明

イベントの発生時に呼び出す関数

11.1.310 sf_touch_ctsu_slider_instance_t

```
typedef struct{
    sf_touch_ctsu_slider_ctrl_t *  p_ctrl
    sf_touch_ctsu_slider_cfg_t const *  p_cfg
    sf_touch_ctsu_slider_api_t const *  p_api
} sf_touch_ctsu_slider_instance_t
```

11.1.310.1 p_ctrl

sf_touch_ctsu_slider_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.310.2 p_cfg

sf_touch_ctsu_slider_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.310.3 p_api

sf_touch_ctsu_slider_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.311 sf_touch_panel_api_t

```
typedef struct{
    ssp_err_t(* open)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_cfg_t const *const p_cfg)
    ssp_err_t(* calibrate)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_calibrate_t const *const p_expected,
sf_touch_panel_payload_t const *const p_actual, ULONG timeout)
    ssp_err_t(* start)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(* stop)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(* reset)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(* close)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} sf_touch_panel_api_t
```

11.1.312 sf_touch_panel_calibrate_t

```
typedef struct{
    uint16_t x
    uint16_t y
    uint16_t tolerance_pixels
    void const * p_extend
} sf_touch_panel_calibrate_t
```

11.1.312.1 x

uint16_t sf_touch_panel_calibrate_t::x

概要説明

期待 x 座標。

11.1.312.2 y

uint16_t sf_touch_panel_calibrate_t::y

概要説明

期待 y 座標。

11.1.312.3 tolerance_pixels

uint16_t sf_touch_panel_calibrate_t::tolerance_pixels

詳細説明

期待座標からの許容される直線偏差 （ピクセル単位）。

11.1.312.4 p_extend

void const* sf_touch_panel_calibrate_t::p_extend

詳細説明

ハードウェアに固有の拡張機能へのポインタ。sf_touch_panel_<instance>.h の sf_touch_panel_<instance>_cfg_t を参照してください。

11.1.313 sf_touch_panel_cfg_t

```
typedef struct{
    uint16_t hsize_pixels
    uint16_t vsize_pixels
    UINT priority
    sf_message_instance_t const * p_message
    uint8_t event_class_instance
    uint16_t update_hz
    void const * p_extend
} sf_touch_panel_cfg_t
```

11.1.313.1 hsize_pixels

uint16_t sf_touch_panel_cfg_t::hsize_pixels

概要説明

画面の幅 （ピクセル単位）

11.1.313.2 vsize_pixels

uint16_t sf_touch_panel_cfg_t::vsize_pixels

概要説明

画面の高さ （ピクセル単位）

11.1.313.3 priority

UINT sf_touch_panel_cfg_t::priority

概要説明

タッチ パネル スレッドの優先順位。

11.1.313.4 p_message

sf_message_instance_t::p_message

概要説明

メッセージング フレームワーク制御ブロックへのポインタ。

11.1.313.5 event_class_instance

uint8_t sf_touch_panel_cfg_t::event_class_instance

概要説明

タッチ イベント クラス メッセージ ポスト用のイベント クラス インスタンス番号。

11.1.313.6 update_hz

uint16_t sf_touch_panel_cfg_t::update_hz

詳細説明

反復的な （SF_TOUCH_PANEL_EVENT_DOWN または SF_TOUCH_PANEL_EVENT_HOLD) タッチ イベントをレポートする周期 （ヘルツ単位）。
I：この値は、ドライバで RTOS ティックに変換され、近似整数値の RTOS ティックに丸められます。

11.1.313.7 p_extend

void const* sf_touch_panel_cfg_t::p_extend

詳細説明

ハードウェアに固有の拡張機能へのポインタ。sf_touch_panel_<instance>.h を参照してください。

11.1.314 sf_touch_panel_ctrl_t

```
typedef struct{
    uint32_t open
    uint16_t hsize_pixels
    uint16_t vsize_pixels
    sf_message_instance_t const * p_message
    uint8_t event_class_instance
    sf_touch_panel_payload_t * p_payload
    sf_touch_panel_payload_t last_payload
    TX_MUTEX mutex
    TX_EVENT_FLAGS_GROUP flags
    TX_THREAD thread
    void * p_lower_lvl_ctrl
    uint16_t update_hz
} sf_touch_panel_ctrl_t
```

11.1.314.1 open

uint32_t sf_touch_panel_ctrl_t::open

概要説明

ドライバによって使用され、制御ブロックが有効かどうかを確認します。

11.1.314.2 hsize_pixels

uint16_t sf_touch_panel_ctrl_t::hsize_pixels

概要説明

画面の幅 （ピクセル単位）

11.1.314.3 vsize_pixels

uint16_t sf_touch_panel_ctrl_t::vsize_pixels

概要説明

画面の高さ （ピクセル単位）

11.1.314.4 p_message

sf_message_instance_t::p_message

概要説明

メッセージング フレームワーク制御ブロックへのポインタ。

11.1.314.5 event_class_instance

uint8_t sf_touch_panel_ctrl_t::event_class_instance

概要説明

タッチ イベント クラス メッセージ ポスト用のイベント クラス インスタンス番号。

11.1.314.6 p_payload

sf_touch_panel_payload_t::p_payload

概要説明

ペイロードの格納に使用されたバッファへのポインタ。

11.1.314.7 last_payload

sf_touch_panel_payload_t::last_payload

概要説明

比較のために、キューに課された最後のペイロードを格納します。

11.1.314.8 mutex

TX_MUTEX sf_touch_panel_ctrl_t::mutex

概要説明

共有リソースへのアクセスを保護するためのミューテックス。

11.1.314.9 flags

TX_EVENT_FLAGS_GROUP sf_touch_panel_ctrl_t::flags

概要説明

内部通信用のイベント フラグ。

11.1.314.10 thread

TX_THREAD sf_touch_panel_ctrl_t::thread

概要説明

メインのタッチ パネル スレッド。

11.1.314.11 p_lower_lvl_ctrl

void* sf_touch_panel_ctrl_t::p_lower_lvl_ctrl

概要説明

ローレベル制御ブロックへのポインタ。

11.1.314.12 update_hz

uint16_t sf_touch_panel_ctrl_t::update_hz

詳細説明

反復的な （SF_TOUCH_PANEL_EVENT_DOWN または SF_TOUCH_PANEL_EVENT_HOLD）タッチ イベントをレポートする周期 （ヘルツ単位）。
I：この値は、ドライバで RTOS ティックに変換され、近似整数値の RTOS ティックに丸められます。

11.1.315 sf_touch_panel_i2c_cfg_t

```
typedef struct{
    i2c_master_instance_t const *  p_lower_lvl_i2c
    sf_external_irq_instance_t const *  p_lower_lvl_irq
    sf_touch_panel_i2c_ctrl_t *const  p_lower_lvl_ctrl
    ioport_port_pin_t  pin
    sf_touch_panel_i2c_chip_t const *const  p_chip
} sf_touch_panel_i2c_cfg_t
```

11.1.315.1 p_lower_lvl_i2c

i2c_master_instance_t::p_lower_lvl_i2c

詳細説明

ローレベル I^2C へのポインタ。

11.1.315.2 p_lower_lvl_irq

sf_external_irq_instance_t::p_lower_lvl_irq

詳細説明

ローレベル外部 IRQ へのポインタ。

11.1.315.3 p_lower_lvl_ctrl

sf_touch_panel_i2c_ctrl_t::p_lower_lvl_ctrl

詳細説明

ローレベル制御ブロックに対して割り当てられたメモリへのポインタ。初期化しないでください。初期化は、SF_TOUCH_PANEL_I2C_Open の呼び出し時に実行されます。

11.1.315.4 pin

::pin

詳細説明

タッチ コントローラ チップ上のラインをリセットするために接続されたポート ピン。使用しない場合は、SF_TOUCH_PANEL_I2C_RESET_PIN_UNUSED に設定されます。

11.1.315.5 p_chip

sf_touch_panel_i2c_chip_t::p_chip

詳細説明

選択されたタッチ コントローラ チップ。

11.1.316 sf_touch_panel_i2c_chip_t

```
typedef struct{
    ssp_err_t(*  payloadGet)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_payload_t *const p_payload)
    ssp_err_t(*  reset)(sf_touch_panel_ctrl_t *const p_ctrl)
} sf_touch_panel_i2c_chip_t
```

11.1.316.1 payloadGet

(* ::payloadGet)(*const p_ctrl, *const p_payload)

詳細説明

タッチ チップを読み取り、タッチ ペイロード データを書き込みます。

表 1063: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。この制御構造体は、この関数で初期化されます。
p_payload	out	データ構造体に対するペイロードへのポインタ。提供されたタッチ データは、論理ピクセル値に変換する必要があります。

パラメータ **p_payload**

11.1.316.2 reset

(* ::reset)(*const p_ctrl)

詳細説明

タッチ チップをリセットします。

表 1064: パラメータ

名前	方向	説明
p_ctrl	入力 / 出力	ユーザーによって割り当てられた構造体へのポインタ。この制御構造体は、この関数で初期化されます。

11.1.317 sf_touch_panel_i2c_ctrl_t

```
typedef struct{
  ioport_port_pin_t pin
  i2c_master_instance_t const * p_lower_lvl_i2c
  sf_external_irq_instance_t const * p_lower_lvl_irq
  uint8_t stack[SF_TOUCH_PANEL_I2C_STACK_SIZE]
  sf_touch_panel_i2c_chip_t const * p_chip
} sf_touch_panel_i2c_ctrl_t
```

11.1.317.1 pin

::pin

詳細説明

タッチ コントローラ チップ上のラインをリセットするために接続されたポート ピン。

11.1.317.2 p_lower_lvl_i2c

i2c_master_instance_t::p_lower_lvl_i2c

詳細説明

ローレベル I^2C 。

11.1.317.3 p_lower_lvl_irq

sf_external_irq_instance_t::p_lower_lvl_irq

詳細説明

ローレベル外部 IRQ。

11.1.317.4 stack

uint8_t ::stack[SF_TOUCH_PANEL_I2C_STACK_SIZE]

詳細説明

タッチ パネル スレッド用のスタック。

11.1.317.5 p_chip

sf_touch_panel_i2c_chip_t::p_chip

詳細説明

チップ固有の関数および定義。

11.1.318 sf_touch_panel_instance_t

```
typedef struct{
    sf_touch_panel_ctrl_t * p_ctrl
    sf_touch_panel_cfg_t const * p_cfg
    sf_touch_panel_api_t const * p_api
} sf_touch_panel_instance_t
```

11.1.318.1 p_ctrl

sf_touch_panel_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.318.2 p_cfg

sf_touch_panel_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.318.3 p_api

sf_touch_panel_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.319 sf_touch_panel_payload_t

```
typedef struct{
    sf_message_header_t header
    int16_t x
    int16_t y
    sf_touch_panel_event_t event_type
} sf_touch_panel_payload_t
```

11.1.319.1 header

sf_message_header_t::header

概要説明

メッセージング フレームワークの必須のヘッダー。

11.1.319.2 x

int16_t sf_touch_panel_payload_t::x

概要説明

x 座標。

11.1.319.3 y

int16_t sf_touch_panel_payload_t::y

概要説明

Y 座標。

11.1.319.4 event_type

sf_touch_panel_event_t::event_type

概要説明

タッチ イベント タイプ。

11.1.320 sf_uart_comms_cfg_t

```
typedef struct{
    uart_instance_t const * p_lower_lvl_uart
    sf_uart_comms_ctrl_t * p_ctrl
} sf_uart_comms_cfg_t
```

11.1.320.1 p_lower_lvl_uart

uart_instance_t::p_lower_lvl_uart

詳細説明

UART ドライバ インスタンスへのポインタ

11.1.320.2 p_ctrl

sf_uart_comms_ctrl_t::p_ctrl

詳細説明

制御ブロックに割り当てられたメモリへのポインタ。制御構造体は初期化しないでください。

11.1.321 sf_uart_comms_ctrl_t

```
typedef struct{
    uint32_t state
    uart_api_t * p_lower_lvl_api
    uart_ctrl_t uart_ctrl
    TX_MUTEX mutex[2]
    TX_EVENT_FLAGS_GROUP eventflag[2]
    TX_QUEUE queue
    uint32_t queue_mem[SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS]
} sf_uart_comms_ctrl_t
```

11.1.321.1 state

uint32_t ::state

概要説明

UART のステータス。

11.1.321.2 p_lower_lvl_api

uart_api_t::p_lower_lvl_api

概要説明

UART インタフェースへのポインタ （cfg からコピー）。

11.1.321.3 uart_ctrl

uart_ctrl_t::uart_ctrl

概要説明

UART ペリフェラル制御ブロック。

11.1.321.4 mutex

TX_MUTEX ::mutex[2]

概要説明

UART リソース相互排除のためのミューテックス オブジェクトへのポインタ。

11.1.321.5 eventflag

TX_EVENT_FLAGS_GROUP ::eventflag[2]

概要説明

UART データ転送用のイベント フラグ オブジェクトへのポインタ。

11.1.321.6 queue

TX_QUEUE ::queue

概要説明

リード用のキュー。

11.1.321.7 queue_mem

uint32_t ::queue_mem[SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS]

概要説明

キューのメモリ。

11.1.322 slcdc_api_t

```
typedef struct{
    ssp_err_t(* open)(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)
    ssp_err_t(* write)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
    ssp_err_t(* modify)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment, slcdc_size_t const data_mask, slcdc_size_t const data)
    ssp_err_t(* start)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(* stop)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(* contrastIncrease)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(* contrastDecrease)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(* setdisplayArea)(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)
    ssp_err_t(* close)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(* versionGet)(ssp_version_t *p_version)
} slcdc_api_t
```

11.1.323 slcdc_cfg_t

```
typedef struct{
    slcdc_display_clock_t slcdc_clock
    slcdc_clk_div_t slcdc_clock_setting
    slcdc_bias_method_t bias_method
    slcdc_time_slice_t time_slice
    slcdc_wave_form_t wave_form
    slcdc_drive_volt_gen_t drive_volt_gen
} slcdc_cfg_t
```

11.1.323.1 slcdc_clock

slcdc_display_clock_t::slcdc_clock

概要説明

LCD クロック ソース （LCDSCKSEL）

11.1.323.2 slcdc_clock_setting

slcdc_clk_div_t::slcdc_clock_setting

概要説明

LCD クロック設定値 （LCDC0）

11.1.323.3 bias_method

slcdc_bias_method_t::bias_method

概要説明

LCD の表示バイアス法選択 （LBAS ビット）

11.1.323.4 time_slice

slcdc_time_slice_t::time_slice

概要説明

LCD 表示の時分割数選択 （LDTY ビット）

11.1.323.5 wave_form

slcdc_wave_form_t::wave_form

概要説明

LCD 表示波形選択 （LWAVE ビット）

11.1.323.6 drive_volt_gen

slcdc_drive_volt_gen_t::drive_volt_gen

概要説明

LCD 駆動電圧生成回路選択 （MDSTET ビット）

11.1.324 slcdc_ctrl_t

```
typedef struct{
   slcdc_display_state_t state
   slcdc_cfg_t info
   void const * p_context
} slcdc_ctrl_t
```

11.1.324.1 state

slcdc_display_state_t::state

概要説明

SLCD モジュールのステータス。

11.1.324.2 info

slcdc_cfg_t::info

概要説明

SLCDC 設定情報。

11.1.324.3 p_context

void const* slcdc_ctrl_t::p_context

概要説明

ハイレベルのデバイス コンテキストへのポインタ。

11.1.325 slcdc_instance_t

```
typedef struct{
    slcdc_ctrl_t * p_ctrl
    slcdc_cfg_t const * p_cfg
    slcdc_api_t const * p_api
} slcdc_instance_t
```

11.1.325.1 p_ctrl

slcdc_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.325.2 p_cfg

slcdc_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.325.3 p_api

slcdc_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.326 spi_api_t

```
typedef struct{
   ssp_err_t(* open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
   ssp_err_t(* read)(spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
   ssp_err_t(* write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
   ssp_err_t(* writeRead)(spi_ctrl_t *const p_ctrl, void const *p_src, void const *p_dest, uint32_t const length, spi_bit_width_t const
bit_width)
   ssp_err_t(* close)(spi_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *p_version)
} spi_api_t
```

11.1.327 spi_callback_args_t

```
typedef struct{
   uint32_t channel
   spi_event_t event
   void const * p_context
} spi_callback_args_t
```

11.1.327.1 channel

uint32_t spi_callback_args_t::channel

概要説明

デバイス チャネル番号。

11.1.327.2 event

spi_event_t::event

概要説明

イベント コード。

11.1.327.3 p_context

void const* spi_callback_args_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.328 spi_cfg_t

```
typedef struct{
    uint32_t channel
    spi_mode_t operating_mode
    spi_clk_phase_t clk_phase
    spi_clk_polarity_t clk_polarity
    spi_mode_fault_t mode_fault
    spi_bit_order_t bit_order
    uint32_t bitrate
    transfer_instance_t const * p_transfer_tx
    transfer_instance_t const * p_transfer_rx
    void(* p_callback)(spi_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} spi_cfg_t
```

11.1.328.1 channel

uint32_t spi_cfg_t::channel

概要説明

使用するチャネル番号。

11.1.328.2 operating_mode

spi_mode_t::operating_mode

概要説明

マスターまたはスレーブのいずれかの動作モードを選択します。

11.1.328.3 clk_phase

spi_clk_phase_t::clk_phase

概要説明

奇数または偶数クロック エッジのいずれかのデータ サンプリングを選択します。

11.1.328.4 clk_polarity

spi_clk_polarity_t::clk_polarity

概要説明

アイドル時のクロック レベル。

11.1.328.5 mode_fault

spi_mode_fault_t::mode_fault

概要説明

モード障害エラー （マスターまたはスレーブ）のフラグ。

11.1.328.6 bit_order

spi_bit_order_t::bit_order

概要説明

MSB ファーストと LSB ファーストのいずれかの送信順序を選択します。

11.1.328.7 bitrate

uint32_t spi_cfg_t::bitrate

概要説明

ビット / 秒。

11.1.328.8 p_transfer_tx

transfer_instance_t::p_transfer_tx

概要説明

SPI DTC/DMA 書き込み転送を使用する場合、ここに DTC/DMA インスタンスをリンクします。使用しない場合は、NULL に設定します。

11.1.328.9 p_transfer_rx

transfer_instance_t::p_transfer_rx

概要説明

SPI DTC/DMA リード転送を使用する場合、ここに DTC/DMA インスタンスをリンクします。使用しない場合は、NULL に設定します。

11.1.328.10 p_callback

void(* spi_cfg_t::p_callback)(spi_callback_args_t *p_args)

概要説明

ユーザー コールバック関数へのポインタ。

11.1.328.11 p_context

void const* spi_cfg_t::p_context

概要説明

コールバック関数に渡されるユーザー定義のコンテキスト。

11.1.328.12 p_extend

void const* spi_cfg_t::p_extend

概要説明

SPI ハードウェアに依存する拡張設定。

11.1.329 spi_ctrl_t

```
typedef struct{
    uint8_t channel
    uint8_t current_slave
    bool channel_opened
    transfer_instance_t const * p_transfer_tx
    transfer_instance_t const * p_transfer_rx
    void(* p_callback)(spi_callback_args_t *p_args)
    void const * p_context
} spi_ctrl_t
```

11.1.329.1 channel

uint8_t spi_ctrl_t::channel

概要説明

使用するチャネル番号。

11.1.329.2 current_slave

uint8_t spi_ctrl_t::current_slave

概要説明

現在割り当てられているスレーブ数。

11.1.329.3 channel_opened

bool spi_ctrl_t::channel_opened

概要説明

ペリフェラルが初期化されたことを示す内部フラグ。

11.1.329.4 p_transfer_tx

transfer_instance_t::p_transfer_tx

概要説明

SPI DTC/DMA 書き込み転送を使用します。

11.1.329.5 p_transfer_rx

transfer_instance_t::p_transfer_rx

概要説明

SPI DTC/DMA リード転送を使用します。

11.1.329.6 p_callback

void(* spi_ctrl_t::p_callback)(spi_callback_args_t *p_args)

概要説明

ユーザー コールバック関数へのポインタ。

11.1.329.7 p_context

void const* spi_ctrl_t::p_context

概要説明

ハイレベルのデバイス コンテキストへのポインタ。

11.1.330 spi_instance_t

```
typedef struct{
    spi_ctrl_t * p_ctrl
    spi_cfg_t const * p_cfg
    spi_api_t const * p_api
} spi_instance_t
```

11.1.330.1 p_ctrl

spi_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.330.2 p_cfg

spi_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.330.3 p_api

spi_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.331 spi_on_rspi_cfg_t

```
typedef struct{
    rspi_operation_t rspi_clksyn
    rspi_communication_t rspi_comm
    rspi_ssl_polarity_t ssl_polarity
    rspi_loopback_t loopback
    rspi_mosi_idle_t mosi_idle
    rspi_parity_t parity
    rspi_ssl_select_t ssl_select
    rspi_ssl_level_keep_t ssl_level_keep
    rspi_clock_delay_t clock_delay
    rspi_ssl_negation_delay_t ssl_neg_delay
    rspi_access_delay_t access_delay
} spi_on_rspi_cfg_t
```

11.1.331.1 rspi_clksyn

rspi_operation_t::rspi_clksyn

詳細説明

動作モードを選択します （SPI またはクロック同期）

11.1.331.2 rspi_comm

rspi_communication_t::rspi_comm

詳細説明

通信方式を選択します （全二重または送信のみ）

11.1.331.3 ssl_polarity

rspi_ssl_polarity_t::ssl_polarity

詳細説明

SSLn の信号極性を選択します

11.1.331.4 loopback

rspi_loopback_t::loopback

詳細説明

loopback1 または loopback2 を選択します

11.1.331.5 mosi_idle

rspi_mosi_idle_t::mosi_idle

詳細説明

MOSI アイドル固定値および選択肢を選択します

11.1.331.6 parity

rspi_parity_t::parity

詳細説明

パリティ値を選択し、パリティ値の有効 / 無効を設定します

11.1.331.7 ssl_select

rspi_ssl_select_t::ssl_select

詳細説明

使用するスレーブを選択します （0-SSL0、1-SSL1、2-SSL2、3-SSL3）

11.1.331.8 ssl_level_keep

rspi_ssl_level_keep_t::ssl_level_keep

詳細説明

転送完了後の SSL レベルを選択します （0- ネゲート、1- 維持）

11.1.331.9 clock_delay

rspi_clock_delay_t::clock_delay

詳細説明

クロック遅延を 0 ～ 7 の範囲で選択します

11.1.331.10 ssl_neg_delay

rspi_ssl_negation_delay_t::ssl_neg_delay

詳細説明

スレーブ起動ネゲート遅延を 0 ～ 7 の範囲で選択します

11.1.331.11 access_delay

rspi_access_delay_t::access_delay

詳細説明

次アクセス遅延を 0 ～ 7 の範囲で選択します

11.1.332 st_sf_audio_playback_ctrl

```
typedef struct{
   uint32_t open
   TX_THREAD * p_owner
   void(* p_callback)(sf_message_callback_args_t *p_args)
   uint8_t class_instance
   uint32_t samples_remaining
   uint32_t index
   uint32_t end
   sf_audio_playback_data_t * p_data[2]
   sf_audio_playback_status_t status
   sf_audio_playback_common_ctrl_t * p_common_ctrl
} st_sf_audio_playback_ctrl
```

11.1.332.1 open

uint32_t sf_audio_playback_ctrl_t::open

概要説明

ドライバが初期化済みかどうかの判定に使用されます。

11.1.332.2 p_owner

TX_THREAD* sf_audio_playback_ctrl_t::p_owner

詳細説明

このインデックスでストリームを開始したスレッドへのポインタ。複数のスレッドが同じストリームのデータをインターリーブすることを避けるために使用されます。

11.1.332.3 p_callback

void(* sf_audio_playback_ctrl_t::p_callback)(sf_message_callback_args_t *p_args)

詳細説明

start に渡されたバッファ再生の完了時に呼び出されるコールバック。

11.1.332.4 class_instance

uint8_t sf_audio_playback_ctrl_t::class_instance

概要説明

メッセージング フレームワークへのストリーム指定に使用されるクラス インスタンス。

11.1.332.5 samples_remaining

uint32_t sf_audio_playback_ctrl_t::samples_remaining

概要説明

このストリームの残りのデータ サンプルの内部状態。

11.1.332.6 index

uint32_t sf_audio_playback_ctrl_t::index

概要説明

このストリームの現在のデータ インデックスの内部状態。

11.1.332.7 end

uint32_t sf_audio_playback_ctrl_t::end

概要説明

ループ再生の完了を追跡するために使用されます。

11.1.332.8 p_data

sf_audio_playback_data_t::p_data

概要説明

キューから読み取られたオーディオ データ。

11.1.332.9 status

sf_audio_playback_status_t::status

概要説明

現在のストリームのステータス。

11.1.332.10 p_common_ctrl

sf_audio_playback_common_ctrl_t::p_common_ctrl

詳細説明

このストリームが使用するハードウェア制御ブロックへのポインタ。

11.1.333 st_sf_console_menu

```
typedef struct{
    struct st_sf_console_menu const * menu_prev
    uint8_t const * menu_name
    uint32_t num_commands
    sf_console_command_t const * command_list
} st_sf_console_menu
```

11.1.333.1 menu_prev

st_sf_console_menu::menu_prev

概要説明

前のメニュー。

11.1.333.2 menu_name

uint8_t const* ::menu_name

概要説明

プロンプトとして使用されるメニュー名。

11.1.333.3 num_commands

uint32_t ::num_commands

概要説明

このメニューに含まれるコマンドの数。

11.1.333.4 command_list

sf_console_command_t::command_list

概要説明

長さが num_commands のコマンド配列へのポインタ。

11.1.334 st_sf_i2c_ctrl

```
typedef struct{
   sf_i2c_bus_t * p_bus
   i2c_cfg_t lower_lvl_cfg
   i2c_ctrl_t lower_lvl_ctrl
   sf_i2c_dev_state_t dev_state
   bool locked
   bool restarted
} st_sf_i2c_ctrl
```

11.1.334.1 p_bus

sf_i2c_bus_t::p_bus

概要説明

このデバイスを使用しているバス。設定構造体からのコピー。

11.1.334.2 lower_lvl_cfg

i2c_cfg_t::lower_lvl_cfg

概要説明

I2C ペリフェラルの設定。バス設定のために使用します。

11.1.334.3 lower_lvl_ctrl

i2c_ctrl_t::lower_lvl_ctrl

概要説明

I2C ペリフェラル制御ブロック。

11.1.334.4 dev_state

sf_i2c_dev_state_t::dev_state

概要説明

デバイスのステータス。

11.1.334.5 locked

bool sf_i2c_ctrl_t::locked

概要説明

バスをデバイスに対してロックおよびロック解除します。

11.1.334.6 restarted

bool sf_i2c_ctrl_t::restarted

概要説明

デバイスがリスタートを発行したかどうかを示します。

11.1.335 st_sf_spi_ctrl

```
typedef struct{
   sf_spi_bus_t * p_bus
   ioport_port_pin_t chip_select
   ioport_level_t chip_select_level_active
   spi_cfg_t lower_lvl_cfg
   spi_ctrl_t lower_lvl_ctrl
   sf_spi_dev_state_t dev_state
   bool locked
} st_sf_spi_ctrl
```

11.1.335.1 p_bus

sf_spi_bus_t::p_bus

概要説明

このデバイスを使用しているバス （cfg からコピー）

11.1.335.2 chip_select

ioport_port_pin_t sf_spi_ctrl_t::chip_select

概要説明

このデバイスに対するチップの選択 （cfg からコピー）

11.1.335.3 chip_select_level_active

ioport_level_t::chip_select_level_active

概要説明

CS 極性、アクティブ High または Low （cfg からコピー）

11.1.335.4 lower_lvl_cfg

spi_cfg_t::lower_lvl_cfg

概要説明

SPI ペリフェラルの設定、バス設定に使用されます。

11.1.335.5 lower_lvl_ctrl

spi_ctrl_t::lower_lvl_ctrl

概要説明

SPI ペリフェラル制御ブロック。

11.1.335.6 dev_state

sf_spi_dev_state_t::dev_state

概要説明

デバイスのステータス。

11.1.335.7 locked

bool sf_spi_ctrl_t::locked

概要説明

バスをデバイスに対してロックおよびロック解除します。

11.1.336 st_sf_touch_ctsu_slider_hdl

```
typedef struct{
  uint32_t open
  sf_touch_ctsu_slider_id_t id
  sf_touch_ctsu_slider_state_t state
  sf_slider_type_t type
  uint32_t num_slider_channels
  ctsu_channel_pair_t const * p_slider_channels
  int32_t const * p_normalization
  int32_t * p_channel_average
  uint32_t * p_offset
  uint16_t * p_slider_scount
  uint16_t * p_slider_baseline
  int32_t * p_slider_delta
  uint32_t position
  int32_t prev_sum
  int32_t max_slider_value
  ssp_err_t(* p_update)(sf_touch_ctsu_slider_hdl_t *const hdl, sf_touch_ctsu_instance_t const *const p_lower_lvl_touch_framework,
uint32_t *p_pos, sf_touch_ctsu_slider_state_t *const p_state)
  uint64_t bit_mask[SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE]
  uint16_t slider_norm_max
  int32_t slider_threshold
  int32_t channel_average_weight
  int32_t prev_sum_weight
  int32_t cutoff
} st_sf_touch_ctsu_slider_hdl
```

11.1.336.1 open

uint32_t ::open

詳細説明

スライダが開かれていることを示します

11.1.336.2 id

sf_touch_ctsu_slider_id_t::id

詳細説明

スライダの一意の識別子。

11.1.336.3 state

sf_touch_ctsu_slider_state_t::state

詳細説明

スライダの現在の状態を表します。

11.1.336.4 type

sf_slider_type_t::type

詳細説明

線形または円形 （ホイール）

11.1.336.5 num_slider_channels

uint32_t ::num_slider_channels

11.1.336.6 p_slider_channels

ctsu_channel_pair_t::p_slider_channels

詳細説明

スライダを構成するチャネル / チャネル ペアを定義します。

11.1.336.7 p_normalization

int32_t const* ::p_normalization

11.1.336.8 p_channel_average

int32_t* ::p_channel_average

11.1.336.9 p_offset

uint32_t* ::p_offset

11.1.336.10 p_slider_scount

uint16_t* ::p_slider_scount

11.1.336.11 p_slider_baseline

uint16_t* ::p_slider_baseline

11.1.336.12 p_slider_delta

int32_t* ::p_slider_delta

11.1.336.13 position

uint32_t ::position

詳細説明

計算済みの位置。

11.1.336.14 prev_sum

int32_t ::prev_sum

詳細説明

位置計算における前回の合計の移動平均の保存に使用します

11.1.336.15 max_slider_value

int32_t ::max_slider_value

詳細説明

スライダの最大値、0 より大きい値でなければならず、最小値は常に 0

11.1.336.16 p_update

(* ::p_update)(*const hdl, const *const p_lower_lvl_touch_framework, uint32_t *p_pos, *const p_state)

詳細説明

タッチの位置を計算する関数。

11.1.336.17 bit_mask

uint64_t ::bit_mask[SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE]

詳細説明

関数の更新に使用するビット マスク

11.1.336.18 slider_norm_max

uint16_t ::slider_norm_max

詳細説明

ユーザーにより変更可能な個々のスライダ設定。
sf_slider_on_ctsu_cfg_tTOUCH_SLIDER_CFG_NORM_MAX と同一でなければなりません

11.1.336.19 slider_threshold

int32_t ::slider_threshold

詳細説明

タッチのしきい値。値が 0 より大きいことを確認してください

11.1.336.20 channel_average_weight

int32_t ::channel_average_weight

詳細説明

各チャネルに対するカウントの加重移動平均

11.1.336.21 prev_sum_weight

int32_t ::prev_sum_weight

詳細説明

位置計算における前回の合計の加重移動平均 （0 より大きい値でなければなりません）

11.1.336.22 cutoff

int32_t ::cutoff

詳細説明

prev_sum 移動平均をどの程度下回った場合に「SF_TOUCH_SLIDER_STATE_RELEASED」を検出するか定義します

11.1.337 tdes_api_t

```
typedef struct{
   uint32_t(* open)(tdes_ctrl_t *const p_ctrl, tdes_cfg_t const *const p_cfg)
   uint32_t(* encrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t
*p_dest)
   uint32_t(* decrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t
*p_dest)
   uint32_t(* close)(tdes_ctrl_t *const p_ctrl)
   uint32_t(* versionGet)(ssp_version_t *const p_version)
} tdes_api_t
```

11.1.338 tdes_cfg_t

```
typedef struct{
   crypto_api_t const * p_crypto_api
} tdes_cfg_t
```

11.1.338.1 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.339 tdes_ctrl_t

```
typedef struct{
   crypto_ctrl_t  crypto_ctrl
   crypto_api_t const * p_crypto_api
} tdes_ctrl_t
```

11.1.339.1 crypto_ctrl

crypto_ctrl_t::crypto_ctrl

概要説明

暗号化制御構造体へのポインタ

11.1.339.2 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.340 tdes_instance_t

```
typedef struct{
    tdes_ctrl_t *  p_ctrl
    tdes_cfg_t const *  p_cfg
    tdes_api_t const *  p_api
} tdes_instance_t
```

11.1.340.1 p_ctrl

tdes_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.340.2 p_cfg

tdes_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.340.3 p_api

tdes_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.341 timer_api_t

```
typedef struct{
   ssp_err_t(* open)(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
   ssp_err_t(* stop)(timer_ctrl_t *const p_ctrl)
   ssp_err_t(* start)(timer_ctrl_t *const p_ctrl)
   ssp_err_t(* reset)(timer_ctrl_t *const p_ctrl)
   ssp_err_t(* counterGet)(timer_ctrl_t *const p_ctrl, timer_size_t *const p_value)
   ssp_err_t(* periodSet)(timer_ctrl_t *const p_ctrl, timer_size_t const period, timer_unit_t const unit)
   ssp_err_t(* dutyCycleSet)(timer_ctrl_t *const p_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t
const pin)
   ssp_err_t(* infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
   ssp_err_t(* close)(timer_ctrl_t *const p_ctrl)
   ssp_err_t(* versionGet)(ssp_version_t *const p_version)
} timer_api_t
```

11.1.342 timer_callback_args_t

```
typedef struct{
   void const * p_context
   timer_event_t event
} timer_callback_args_t
```

11.1.342.1 p_context

void const* timer_callback_args_t::p_context

詳細説明

ユーザー データのプレースホルダー。timer_cfg_t の open 関数で設定されます。

11.1.342.2 event

timer_event_t::event

概要説明

このイベントを使用して、コールバックの原因 （オーバーフローまたはエラー）を特定できます。

11.1.343 timer_cfg_t

```
typedef struct{
    timer_mode_t mode
    uint32_t period
    timer_unit_t unit
    timer_size_t duty_cycle
    timer_pwm_unit_t duty_cycle_unit
    uint8_t channel
    bool autostart
    void(* p_callback)(timer_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} timer_cfg_t
```

11.1.343.1 mode

timer_mode_t::mode

概要説明

timer_mode_t から列挙値を選択します。

11.1.343.2 period

uint32_t timer_cfg_t::period

詳細説明

タイマが期限切れになるタイミングを定義します。フリーラン カウンタの場合は、TIMER_MAX_CLOCK に設定し、以下の単位を使用してください :TIMER_UNIT_PERIOD_RAW_COUNTS

11.1.343.3 unit

timer_unit_t::unit

概要説明

period の単位。

11.1.343.4 duty_cycle

timer_size_t::duty_cycle

概要説明

duty_cycle_unit 単位のデューティ サイクル。

11.1.343.5 duty_cycle_unit

timer_pwm_unit_t::duty_cycle_unit

概要説明

duty_cycle の単位。

11.1.343.6 channel

uint8_t timer_cfg_t::channel

詳細説明

ハードウェアのチャネル番号に対応するチャネルを選択します。

11.1.343.7 autostart

bool timer_cfg_t::autostart

詳細説明

Open 呼び出しの間に開始するかどうかを指定します。True の場合は、Open 呼び出しの間に開始します。
False の場合は、Open 呼び出しの間に開始しません。

11.1.343.8 p_callback

void(* timer_cfg_t::p_callback)(timer_callback_args_t *p_args)

詳細説明

タイマ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL に設定します。

11.1.343.9 p_context

void const* timer_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。timer_callback_args_t 内のユーザー コールバックに渡されます。

11.1.343.10 p_extend

void const* timer_cfg_t::p_extend

概要説明

ハードウェア固有の設定値に対応するための拡張パラメータです。

11.1.344 timer_ctrl_t

```
typedef struct{
   void(*  p_callback)(timer_callback_args_t *p_args)
   void const *  p_context
   uint8_t  channel
} timer_ctrl_t
```

11.1.344.1 p_callback

void(* timer_ctrl_t::p_callback)(timer_callback_args_t *p_args)

詳細説明

タイマ ISR の発生時に提供されるコールバック。CPU 割り込みを行わない場合は、NULL を使用します。

11.1.344.2 p_context

void const* timer_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。timer_callback_args_t 内のユーザー コールバックに渡されます。

11.1.344.3 channel

uint8_t timer_ctrl_t::channel

概要説明

チャネル番号。

11.1.345 timer_info_t

```
typedef struct{
   timer_direction_t  count_direction
   uint32_t  clock_frequency
   timer_size_t  period_counts
   timer_status_t  status
} timer_info_t
```

11.1.345.1 count_direction

timer_direction_t::count_direction

概要説明

タイマ リソースのクロックのカウント方向。

11.1.345.2 clock_frequency

uint32_t timer_info_t::clock_frequency

概要説明

タイマ リソースのクロック周波数。

11.1.345.3 period_counts

timer_size_t::period_counts

概要説明

タイマが期限切れになるまでの時間 （単位はクロック カウント）。

11.1.345.4 status

timer_status_t::status

11.1.346 timer_instance_t

```
typedef struct{
    timer_ctrl_t * p_ctrl
    timer_cfg_t const * p_cfg
    timer_api_t const * p_api
} timer_instance_t
```

11.1.346.1 p_ctrl

timer_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.346.2 p_cfg

timer_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.346.3 p_api

timer_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.347 timer_on_agt_cfg_t

```
typedef struct{
    agt_count_source_t count_source
    bool agto_output_enabled
    bool agtio_output_enabled
    bool output_inverted
} timer_on_agt_cfg_t
```

11.1.347.1 count_source

::count_source

概要説明

AGT チャネルのクロック ソース。設定可能な値 : AGT_CLOCK_PCLKB、AGT_CLOCK_LOCO、AGT_CLOCK_FSUB。

11.1.347.2 agto_output_enabled

bool ::agto_output_enabled

概要説明

AGTO ピンで、出力コンペアが有効 （true、false）

11.1.347.3 agtio_output_enabled

bool ::agtio_output_enabled

概要説明

AGTIO ピンで、出力コンペアが有効 （true、false）

11.1.347.4 output_inverted

bool ::output_inverted

概要説明

出力の反転 （true、false）

11.1.348 timer_on_gpt_cfg_t

```
typedef struct{
    gpt_output_pin_t  gtioca
    gpt_output_pin_t  gtiocb
} timer_on_gpt_cfg_t
```

11.1.348.1 gtioca

gpt_output_pin_t::gtioca

概要説明

GPT I/O ピン A の設定。

11.1.348.2 gtiocb

gpt_output_pin_t::gtiocb

概要説明

GPT I/O ピン B の設定。

11.1.349 transfer_api_t

```
typedef struct{
    ssp_err_t(*  open)(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
    ssp_err_t(*  reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)
    ssp_err_t(*  enable)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  disable)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  start)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)
    ssp_err_t(*  stop)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_info)
    ssp_err_t(*  close)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} transfer_api_t
```

11.1.350 transfer_callback_args_t

```
typedef struct{
    void const *  p_context
} transfer_callback_args_t
```

11.1.350.1 p_context

void const* transfer_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。transfer_cfg_t の r_transfer_t::open 関数で設定されます。

11.1.351 transfer_cfg_t

```
typedef struct{
   transfer_info_t * p_info
   elc_event_t activation_source
   bool auto_enable
   void(* p_callback)(transfer_callback_args_t *p_args)
   void const * p_context
   void const * p_extend
} transfer_cfg_t
```

11.1.351.1 p_info

transfer_info_t::p_info

詳細説明

転送設定オプションへのポインタ。チェーン転送を使用する場合 （DTC のみ)、チェーンする転送の配列を作成し、その配列へのポインタを指定して、転送を順に実行することができます。

11.1.351.2 activation_source

elc_event_t::activation_source

詳細説明

転送をトリガーするイベントを選択します。
I：また ELC_EVENT_ELC_SOFTWARE_EVENT_0 または ELC_EVENT_ELC_SOFTWARE_EVENT_0 を選択して、ソフトウェア アクティベーションを指定します。DTC を使用する場合、これらのイベントをそれぞれ一度限りしか使用できないことがあります。これらのイベントのいずれかを選択すると、DMAC の内部ソフトウェア開始機能が使用されます。

11.1.351.3 auto_enable

bool transfer_cfg_t::auto_enable

詳細説明

オープン後に転送を有効にするかどうかを選択します。

11.1.351.4 p_callback

void(* transfer_cfg_t::p_callback)(transfer_callback_args_t *p_args)

詳細説明

転送終了割り込みのコールバック。CPU 割り込みを行わない場合は、NULL に設定します。
I a:DTC ではサポートされていません。DTC 転送は、アクティベーション ソースに関連付けられた割り込みをトリガーします。

11.1.351.5 p_context

void const* transfer_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。transfer_callback_args_t のユーザー p_callback に渡されます。

11.1.351.6 p_extend

void const* transfer_cfg_t::p_extend

概要説明

ハードウェア固有の設定値に対応するための拡張パラメータです。

11.1.352 transfer_ctrl_t

```
typedef struct{
    uint32_t id
    elc_event_t trigger
    IRQn_Type irq
    uint8_t channel
} transfer_ctrl_t
```

11.1.352.1 id

uint32_t transfer_ctrl_t::id

概要説明

ドライバ ID。

11.1.352.2 trigger

elc_event_t::trigger

概要説明

転送アクティベーション イベント。infoGet によって返されたイベントと照合されます。

11.1.352.3 irq

IRQn_Type transfer_ctrl_t::irq

概要説明

一部の HAL ドライバではサポートされていません。

11.1.352.4 channel

uint8_t transfer_ctrl_t::channel

概要説明

一部の HAL ドライバではサポートされていません。

11.1.353 transfer_info_t

```
typedef struct{
    uint32_t __pad0__
    uint32_t __pad1__
    transfer_addr_mode_t dest_addr_mode
    transfer_repeat_area_t repeat_area
    transfer_irq_t irq
    transfer_chain_mode_t chain_mode
    uint32_t __pad2__
    transfer_addr_mode_t src_addr_mode
    transfer_size_t size
    transfer_mode_t mode
    void const *volatile p_src
    void *volatile p_dest
    uint16_t num_blocks
    uint16_t length
} transfer_info_t
```

11.1.353.1 __pad0__

uint32_t transfer_info_t::__pad0__

11.1.353.2 __pad1__

uint32_t transfer_info_t::__pad1__

11.1.353.3 dest_addr_mode

transfer_addr_mode_t::dest_addr_mode

詳細説明

各転送が終了した後の、宛先ポインタの扱いを選択します。

11.1.353.4 repeat_area

transfer_repeat_area_t::repeat_area

詳細説明

ソース領域または宛先領域を繰り返す場合に選択します。TRANSFER_MODE_NORMAL では使用されません。

11.1.353.5 irq

transfer_irq_t::irq

詳細説明

1 つの転送が完了するごとに、または予定したすべての転送が完了した後、割り込みを行う場合に選択します。

11.1.353.6 chain_mode

transfer_chain_mode_t::chain_mode

詳細説明

チェーン転送が完了するタイミングを選択します。

11.1.353.7 __pad2__

uint32_t transfer_info_t::__pad2__

11.1.353.8 src_addr_mode

transfer_addr_mode_t::src_addr_mode

詳細説明

各転送が終了した後の、ソース ポインタの扱いを選択します。

11.1.353.9 size

`transfer_size_t`::size

詳細説明

一度に転送するバイト数を選択します。length.

11.1.353.10 mode

`transfer_mode_t`::mode

詳細説明

transfer_mode_t からモードを選択します。

11.1.353.11 p_src

void const* volatile `transfer_info_t`::p_src

概要説明

ソースポインタ。

11.1.353.12 p_dest

void* volatile `transfer_info_t`::p_dest

概要説明

宛先ポインタ。

11.1.353.13 num_blocks

volatile uint16_t `transfer_info_t`::num_blocks

詳細説明

TRANSFER_MODE_BLOCK（DTC と DMAC の両方）および TRANSFER_MODE_REPEAT（DMAC のみ）を使用する場合の転送ブロック数。他のモードでは使用されません。

11.1.353.14 length

volatile uint16_t `transfer_info_t`::length

詳細説明

各転送の長さ。TRANSFER_MODE_BLOCK と TRANSFER_MODE_REPEAT では範囲が制限されています。詳細については、HAL ドライバを参照してください。

11.1.354 transfer_instance_t

```
typedef struct{
   transfer_ctrl_t * p_ctrl
   transfer_cfg_t const * p_cfg
   transfer_api_t const * p_api
} transfer_instance_t
```

11.1.354.1 p_ctrl

transfer_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.354.2 p_cfg

transfer_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.354.3 p_api

transfer_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.355 transfer_properties_t

```
typedef struct{
   uint32_t transfer_length_max
   uint16_t transfer_length_remaining
   bool in_progress
} transfer_properties_t
```

11.1.355.1 transfer_length_max

uint32_t transfer_properties_t::transfer_length_max

概要説明

転送の最大数。

11.1.355.2 transfer_length_remaining

uint16_t transfer_properties_t::transfer_length_remaining

概要説明

残りの転送数。

11.1.355.3 in_progress

bool transfer_properties_t::in_progress

概要説明

この転送が実行中かどうかを示します。

11.1.356 trng_api_t

```
typedef struct{
    uint32_t(*  open)(trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
    uint32_t(*  read)(trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)
    uint32_t(*  close)(trng_ctrl_t *const p_ctrl)
    uint32_t(*  versionGet)(ssp_version_t *const p_version)
} trng_api_t
```

11.1.357 trng_cfg_t

```
typedef struct{
    crypto_api_t const *  p_crypto_api
    uint32_t  nattempts
} trng_cfg_t
```

11.1.357.1 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化 API へのポインタ

11.1.357.2 nattempts

uint32_t trng_cfg_t::nattempts

概要説明

連続してテストに失敗した場合のリトライ回数

11.1.358 trng_ctrl_t

```
typedef struct{
    uint32_t  nattempts
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const *  p_crypto_api
    uint32_t  prevbuf[TRNG_REGISTER_SIZE_WORDS]
    uint32_t  currbuf[TRNG_REGISTER_SIZE_WORDS]
} trng_ctrl_t
```

11.1.358.1 nattempts

uint32_t trng_ctrl_t::nattempts

概要説明

リトライ回数

11.1.358.2 p_crypto_ctrl

crypto_ctrl_t::p_crypto_ctrl

概要説明

暗号化制御構造体へのポインタ

11.1.358.3 p_crypto_api

crypto_api_t::p_crypto_api

概要説明

暗号化エンジン API へのポインタ

11.1.358.4 prevbuf

uint32_t trng_ctrl_t::prevbuf[TRNG_REGISTER_SIZE_WORDS]

概要説明

前の乱数データ

11.1.358.5 currbuf

uint32_t trng_ctrl_t::currbuf[TRNG_REGISTER_SIZE_WORDS]

概要説明

現在の乱数データ

11.1.359 trng_instance_t

```
typedef struct{
    trng_ctrl_t *  p_ctrl
    trng_cfg_t const *  p_cfg
    trng_api_t const *  p_api
} trng_instance_t
```

11.1.359.1 p_ctrl

trng_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.359.2 p_cfg

trng_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.359.3 p_api

trng_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.360 uart_api_t

```
typedef struct{
    ssp_err_t(* open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
    ssp_err_t(* read)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest, uint32_t const bytes)
    ssp_err_t(* write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)
    ssp_err_t(* baudSet)(uart_ctrl_t const *const p_ctrl, uint32_t const baudrate)
    ssp_err_t(* infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
    ssp_err_t(* close)(uart_ctrl_t *const p_ctrl)
    ssp_err_t(* versionGet)(ssp_version_t *p_version)
} uart_api_t
```

11.1.361 uart_callback_args_t

```
typedef struct{
    uint32_t channel
    uart_event_t event
    uint32_t data
    void const * p_context
} uart_callback_args_t
```

11.1.361.1 channel

uint32_t uart_callback_args_t::channel

概要説明

デバイス チャネル番号。

11.1.361.2 event

uart_event_t::event

概要説明

イベント コード。

11.1.361.3 data

uint32_t uart_callback_args_t::data

概要説明

汎用のデータ ストレージ。

11.1.361.4 p_context

void const* uart_callback_args_t::p_context

概要説明

コールバック時にユーザーに提供されるコンテキスト。

11.1.362 uart_cfg_t

```
typedef struct{
    uint32_t channel
    uint32_t baud_rate
    uart_data_bits_t data_bits
    uart_parity_t parity
    uart_stop_bits_t stop_bits
    bool ctsrts_en
    transfer_instance_t const * p_transfer_rx
    transfer_instance_t const * p_transfer_tx
    void(* p_callback)(uart_callback_args_t *p_args)
    void const * p_context
    void const * p_extend
} uart_cfg_t
```

11.1.362.1 channel

uint32_t uart_cfg_t::channel

概要説明

ハードウェアのチャネル番号に対応するチャネルを選択します。

11.1.362.2 baud_rate

uint32_t uart_cfg_t::baud_rate

概要説明

ボー レート （9600、19200、115200）

11.1.362.3 data_bits

uart_data_bits_t::data_bits

概要説明

データのビット長 （8、7、9 のいずれか）

11.1.362.4 parity

uart_parity_t::parity

概要説明

パリティ タイプ （パリティなし、奇数、偶数のいずれか）

11.1.362.5 stop_bits

uart_stop_bits_t::stop_bits

概要説明

ストップ ビット長 （1 または 2）

11.1.362.6 ctsrts_en

bool uart_cfg_t::ctsrts_en

概要説明

CTS/RTS ハードウェア フロー制御有効化。

11.1.362.7 p_transfer_rx

transfer_instance_t::p_transfer_rx

詳細説明

複数バイトを割り込みなく受信するために使用されるオプションの転送インスタンス。使用しない場合は、NULL に設定します。NULL の場合、読み取り API で許可されるバイト数は、一度に 1 バイトのみです。

11.1.362.8 p_transfer_tx

transfer_instance_t::p_transfer_tx

詳細説明

複数バイトを割り込みなく送信するために使用されるオプションの転送インスタンス。使用しない場合は、NULL に設定します。NULL の場合、書き込み API で許可されるバイト数は、一度に 1 バイトのみです。

11.1.362.9 p_callback

void(* uart_cfg_t::p_callback)(uart_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.362.10 p_context

void const* uart_cfg_t::p_context

概要説明

コールバック関数に渡されるユーザー定義のコンテキスト。

11.1.362.11 p_extend

void const* uart_cfg_t::p_extend

概要説明

UART ハードウェアに依存する設定。

11.1.363 uart_ctrl_t

```
typedef struct{
    uint32_t channel
    transfer_instance_t const * p_transfer_rx
    transfer_instance_t const * p_transfer_tx
    uint8_t const * p_tx_src
    uint32_t tx_src_bytes
    bool rx_transfer_in_progress
    void(* p_callback)(uart_callback_args_t *p_args)
    void const * p_context
} uart_ctrl_t
```

11.1.363.1 channel

uint32_t uart_ctrl_t::channel

概要説明

チャネル番号。

11.1.363.2 p_transfer_rx

transfer_instance_t::p_transfer_rx

詳細説明

複数バイトを割り込みなく送信または受信するために使用されるオプションの転送インスタンス。

11.1.363.3 p_transfer_tx

transfer_instance_t::p_transfer_tx

詳細説明

複数バイトを割り込みなく送信または受信するために使用されるオプションの転送インスタンス。

11.1.363.4 p_tx_src

uint8_t const* uart_ctrl_t::p_tx_src

詳細説明

送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファ ポインタ。

11.1.363.5 tx_src_bytes

uint32_t uart_ctrl_t::tx_src_bytes

詳細説明

送信 ISR からハードウェア FIFO を指定するために使用されるソース バッファ ポインタのサイズ。

11.1.363.6 rx_transfer_in_progress

bool uart_ctrl_t::rx_transfer_in_progress

詳細説明

この受信転送が実行中かどうかを示します。

11.1.363.7 p_callback

void(* uart_ctrl_t::p_callback)(uart_callback_args_t *p_args)

概要説明

コールバック関数へのポインタ。

11.1.363.8 p_context

void const* uart_ctrl_t::p_context

概要説明

ハイレベルのデバイス コンテキストへのポインタ。

11.1.364 uart_info_t

```
typedef struct{
   uint32_t write_bytes_max
   uint32_t read_bytes_max
} uart_info_t
```

11.1.364.1 write_bytes_max

uint32_t uart_info_t::write_bytes_max

詳細説明

一度に書き込むことができる最大バイト数。p_transfer_tx が NULL でない場合のみ適用されます。

11.1.364.2 read_bytes_max

uint32_t uart_info_t::read_bytes_max

詳細説明

一度に読み取ることができる最大バイト数。p_transfer_rx が NULL でない場合のみ適用されます。

11.1.365 uart_instance_t

```
typedef struct{
   uart_ctrl_t * p_ctrl
   uart_cfg_t const * p_cfg
   uart_api_t const * p_api
} uart_instance_t
```

11.1.365.1 p_ctrl

uart_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.365.2 p_cfg

uart_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.365.3 p_api

uart_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.366 uart_on_sci_cfg_t

```
typedef struct{
   sci_clk_src_t clk_src
   bool baudclk_out
   bool rx_edge_start
   bool noisecancel_en
   void(* p_extpin_ctrl)(uint32_t channel, uint32_t level)
} uart_on_sci_cfg_t
```

11.1.366.1 clk_src

sci_clk_src_t::clk_src

詳細説明

SCI_CLK_SRC_INT/EXT8X/EXT16X を使用します

11.1.366.2 baudclk_out

bool ::baudclk_out

詳細説明

ボー レート クロック出力を有効化します

11.1.366.3 rx_edge_start

bool ::rx_edge_start

詳細説明

立ち下がりエッジで受信を開始します

11.1.366.4 noisecancel_en

bool ::noisecancel_en

詳細説明

ノイズ キャンセルを有効化します

11.1.366.5 p_extpin_ctrl

void(* ::p_extpin_ctrl)(uint32_t channel, uint32_t level)

詳細説明

RTS 信号として使用される外部 GPIO ピン制御に対するユーザー コールバック関数へのポインタ

11.1.367 wdt_api_t

```
typedef struct{
    ssp_err_t(*  cfgGet)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)
    ssp_err_t(*  open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
    ssp_err_t(*  refresh)(wdt_ctrl_t *const p_ctrl)
    ssp_err_t(*  statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
    ssp_err_t(*  statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
    ssp_err_t(*  counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
    ssp_err_t(*  timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_data)
} wdt_api_t
```

11.1.368 wdt_callback_args_t

```
typedef struct{
    void const *  p_context
} wdt_callback_args_t
```

11.1.368.1 p_context

void const* wdt_callback_args_t::p_context

概要説明

ユーザー データのプレースホルダー。wdt_cfg_t の open 関数で設定されます。

11.1.369 wdt_cfg_t

```
typedef struct{
   wdt_start_mode_t start_mode
   bool autostart
   wdt_timeout_t timeout
   wdt_clock_division_t clock_division
   wdt_window_start_t window_start
   wdt_window_end_t window_end
   wdt_reset_control_t reset_control
   wdt_stop_control_t stop_control
   void(* p_callback)(wdt_callback_args_t *p_args)
   void const * p_context
   void const * p_extend
} wdt_cfg_t
```

11.1.369.1 start_mode

wdt_start_mode_t::start_mode

概要説明

WDT に対して指定するスタート モード。

11.1.369.2 autostart

bool wdt_cfg_t::autostart

詳細説明

true の場合、WDT がこの設定の一部として開始されます （レジスタ スタート モード)。false の場合、リフレッシュ API を呼び出して、手動で WDT を開始する必要があります。

11.1.369.3 timeout

wdt_timeout_t::timeout

概要説明

タイムアウト期間。

11.1.369.4 clock_division

wdt_clock_division_t::clock_division

概要説明

クロック分周器。

11.1.369.5 window_start

wdt_window_start_t::window_start

概要説明

許可ウィンドウの開始位置をリフレッシュします。

11.1.369.6 window_end

wdt_window_end_t::window_end

概要説明

許可ウィンドウの終了位置をリフレッシュします。

11.1.369.7 reset_control

wdt_reset_control_t::reset_control

概要説明

NMI を選択するか、アンダーフロー時にリセットを生成します。

11.1.369.8 stop_control

wdt_stop_control_t::stop_control

概要説明

カウンタがスリープ モードで動作するかどうかを選択します。

11.1.369.9 p_callback

void(* wdt_cfg_t::p_callback)(wdt_callback_args_t *p_args)

概要説明

WDT NMI ISR の発生時に提供されるコールバック。

11.1.369.10 p_context

void const* wdt_cfg_t::p_context

詳細説明

ユーザー データのプレースホルダー。wdt_callback_args_t 内のユーザー コールバックに渡されます。

11.1.369.11 p_extend

void const* wdt_cfg_t::p_extend

概要説明

ユーザー拡張のプレースホルダー。

11.1.370 wdt_ctrl_t

```
typedef struct{
   bool  wdt_open
   void const *  p_context
   void(*  p_callback)(wdt_callback_args_t *p_args)
} wdt_ctrl_t
```

11.1.370.1 wdt_open

bool wdt_ctrl_t::wdt_open

詳細説明

open() API が正常に呼び出されたかどうかを示します。

11.1.370.2 p_context

void const* wdt_ctrl_t::p_context

詳細説明

ユーザー データのプレースホルダー。wdt_callback_args_t 内のユーザー コールバックに渡されます。

11.1.370.3 p_callback

void(* wdt_ctrl_t::p_callback)(wdt_callback_args_t *p_args)

概要説明

WDT NMI ISR の発生時に提供されるコールバック。

11.1.371 wdt_instance_t

```
typedef struct{
   wdt_ctrl_t *  p_ctrl
   wdt_cfg_t const *  p_cfg
   wdt_api_t const *  p_api
} wdt_instance_t
```

11.1.371.1 p_ctrl

wdt_ctrl_t::p_ctrl

概要説明

このインスタンスの制御構造体へのポインタ。

11.1.371.2 p_cfg

wdt_cfg_t::p_cfg

概要説明

イベント クラスのインスタンス範囲の始点。

11.1.371.3 p_api

wdt_api_t::p_api

概要説明

イベント クラスのインスタンス範囲の終点。

11.1.372 wdt_timeout_values_t

```
typedef struct{
    uint32_t clock_frequency_hz
    uint32_t timeout_clocks
} wdt_timeout_values_t
```

11.1.372.1 clock_frequency_hz

uint32_t wdt_timeout_values_t::clock_frequency_hz

概要説明

分周器後のウォッチドッグ クロックの周波数。

11.1.372.2 timeout_clocks

uint32_t wdt_timeout_values_t::timeout_clocks

概要説明

ウォッチドッグ クロック ティック単位のタイムアウト期間。

11.2 Functions

11.2.1 インタフェース関数

11.2.1.1 インタフェース： adc_api_t

説明： ADC インタフェース

関数名	定義
.open	ssp_err_t(* adc_api_t::open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
.scanCfg	ssp_err_t(* adc_api_t::scanCfg)(adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
.scanStart	ssp_err_t(* adc_api_t::scanStart)(adc_ctrl_t *const p_ctrl)
.scanStop	ssp_err_t(* adc_api_t::scanStop)(adc_ctrl_t *const p_ctrl)
.scanStatusGet	ssp_err_t(* adc_api_t::scanStatusGet)(adc_ctrl_t *const p_ctrl)
.read	ssp_err_t(* adc_api_t::read)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, adc_data_size_t *const p_data)
.sampleStateCountSet	ssp_err_t(* adc_api_t::sampleStateCountSet)(adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)
.close	ssp_err_t(* adc_api_t::close)(adc_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* adc_api_t::infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
.versionGet	ssp_err_t(* adc_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.2 インタフェース： aes_api_t

説明： AES インタフェース

関数名	定義
.open	uint32_t(* aes_api_t::open)(aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)
.createKey	uint32_t(* aes_api_t::createKey)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key)
.encrypt	uint32_t(* aes_api_t::encrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.addAdditionalAuthenticationData	uint32_t(* aes_api_t::addAdditionalAuthenticationData)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source)
.encryptFinal	uint32_t(* aes_api_t::encryptFinal)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t output_num_words, uint32_t *p_dest)
.decrypt	uint32_t(* aes_api_t::decrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)
.setGcmTag	uint32_t(* aes_api_t::setGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source)
.getGcmTag	uint32_t(* aes_api_t::getGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest)
.close	uint32_t(* aes_api_t::close)(aes_ctrl_t *const p_ctrl)
.zeroPaddingEncrypt	uint32_t(* aes_api_t::zeroPaddingEncrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)
.zeroPaddingDecrypt	uint32_t(* aes_api_t::zeroPaddingDecrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)
.versionGet	uint32_t(* aes_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.3 インタフェース： cac_api_t

説明： CAC インタフェース

関数名	定義
.open	ssp_err_t(* cac_api_t::open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
.read	ssp_err_t(* cac_api_t::read)(cac_ctrl_t *const p_ctrl, uint8_t *const p_status, uint16_t *const p_counter)
.close	ssp_err_t(* cac_api_t::close)(cac_ctrl_t *const p_ctrl)
.stopMeasurement	ssp_err_t(* cac_api_t::stopMeasurement)(cac_ctrl_t *const p_ctrl)
.startMeasurement	ssp_err_t(* cac_api_t::startMeasurement)(cac_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* cac_api_t::reset)(cac_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* cac_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.4 インタフェース： can_api_t

説明： CAN インタフェース

関数名	定義
.open	ssp_err_t(* can_api_t::open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
.read	ssp_err_t(* can_api_t::read)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
.write	ssp_err_t(* can_api_t::write)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
.close	ssp_err_t(* can_api_t::close)(can_ctrl_t *const p_ctrl)
.control	ssp_err_t(* can_api_t::control)(can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data)

関数名	定義
.infoGet	ssp_err_t(* can_api_t::infoGet)(can_ctrl_t *const p_ctrl, can_info_t *const p_info)
.versionGet	ssp_err_t(* can_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.5 インタフェース： cgc_api_t

説明： CGC インタフェース

関数名	定義
.init	ssp_err_t(* cgc_api_t::init)(void)
.clockStart	ssp_err_t(* cgc_api_t::clockStart)(cgc_clock_t clock_source, cgc_clock_cfg_t *p_clock_cfg)
.clockStop	ssp_err_t(* cgc_api_t::clockStop)(cgc_clock_t clock_source)
.systemClockSet	ssp_err_t(* cgc_api_t::systemClockSet)(cgc_clock_t clock_source, cgc_system_clock_cfg_t *p_clock_cfg)
.systemClockGet	ssp_err_t(* cgc_api_t::systemClockGet)(cgc_clock_t *p_clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg)
.systemClockFreqGet	ssp_err_t(* cgc_api_t::systemClockFreqGet)(cgc_system_clocks_t clock, uint32_t *p_freq_hz)
.clockCheck	ssp_err_t(* cgc_api_t::clockCheck)(cgc_clock_t clock_source)
.oscStopDetect	ssp_err_t(* cgc_api_t::oscStopDetect)(void(*p_callback)(cgc_callback_args_t *p_args), bool enable)
.oscStopStatusClear	ssp_err_t(* cgc_api_t::oscStopStatusClear)(void)
.busClockOutCfg	ssp_err_t(* cgc_api_t::busClockOutCfg)(cgc_bclockout_dividers_t divider)
.busClockOutEnable	ssp_err_t(* cgc_api_t::busClockOutEnable)(void)

関数名	定義
.busClockOutDisable	ssp_err_t(* cgc_api_t::busClockOutDisable)(void)
.clockOutCfg	ssp_err_t(* cgc_api_t::clockOutCfg)(cgc_clock_t clock, cgc_clockout_dividers_t divider)
.clockOutEnable	ssp_err_t(* cgc_api_t::clockOutEnable)(void)
.clockOutDisable	ssp_err_t(* cgc_api_t::clockOutDisable)(void)
.lcdClockCfg	ssp_err_t(* cgc_api_t::lcdClockCfg)(cgc_clock_t clock)
.lcdClockEnable	ssp_err_t(* cgc_api_t::lcdClockEnable)(void)
.lcdClockDisable	ssp_err_t(* cgc_api_t::lcdClockDisable)(void)
.sdramClockOutEnable	ssp_err_t(* cgc_api_t::sdramClockOutEnable)(void)
.sdramClockOutDisable	ssp_err_t(* cgc_api_t::sdramClockOutDisable)(void)
.usbClockCfg	ssp_err_t(* cgc_api_t::usbClockCfg)(cgc_usb_clock_div_t divider)
.systickUpdate	ssp_err_t(* cgc_api_t::systickUpdate)(uint32_t period_count, cgc_systick_period_units_t units)
.versionGet	ssp_err_t(* cgc_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.6 インタフェース： crc_api_t

説明： CRC インタフェース

関数名	定義
.open	ssp_err_t(* crc_api_t::open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
.close	ssp_err_t(* crc_api_t::close)(crc_ctrl_t *const p_ctrl)
.crcResultGet	ssp_err_t(* crc_api_t::crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
.snoopEnable	ssp_err_t(* crc_api_t::snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

関数名	定義
.snoopDisable	ssp_err_t(* crc_api_t::snoopDisable)(crc_ctrl_t *const p_ctrl)
.snoopCfg	ssp_err_t(* crc_api_t::snoopCfg)(crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const p_snoop_cfg)
.calculate	ssp_err_t(* crc_api_t::calculate)(crc_ctrl_t *const p_ctrl, void *p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)
.versionGet	ssp_err_t(* crc_api_t::versionGet)(ssp_version_t *version)

11.2.1.7 インタフェース： crypto_api_t

説明： 暗号インタフェース

関数名	定義
.open	uint32_t(* crypto_api_t::open)(crypto_ctrl_t *const p_ctrl, crypto_cfg_t const *const p_cfg)
.close	uint32_t(* crypto_api_t::close)(crypto_ctrl_t *const p_ctrl)
.statusGet	uint32_t(* crypto_api_t::statusGet)(uint32_t *p_status)
.versionGet	uint32_t(* crypto_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.8 インタフェース： ctsu_api_t

説明： CTSU インタフェース

関数名	定義
.open	ssp_err_t(* ctsu_api_t::open)(ctsu_ctrl_t *p_ctrl, ctsu_cfg_t *p_cfg)
.close	ssp_err_t(* ctsu_api_t::close)(ctsu_ctrl_t *p_ctrl, ctsu_close_option_t opts)
.scan	ssp_err_t(* ctsu_api_t::scan)(ctsu_ctrl_t *p_ctrl)

関数名	定義
.update	ssp_err_t(* ctsu_api_t::update)(ctsu_ctrl_t *p_ctrl)
.read	ssp_err_t(* ctsu_api_t::read)(ctsu_ctrl_t *p_ctrl, void *p_dest, ctsu_read_t opts, const ctsu_channel_pair_t *channels, const uint16_t count)
.versionGet	ssp_err_t(* ctsu_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.9 インタフェース： dac_api_t

説明： DAC インタフェース

関数名	定義
.open	ssp_err_t(* dac_api_t::open)(dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
.close	ssp_err_t(* dac_api_t::close)(dac_ctrl_t *p_ctrl)
.write	ssp_err_t(* dac_api_t::write)(dac_ctrl_t *p_ctrl, dac_size_t value)
.start	ssp_err_t(* dac_api_t::start)(dac_ctrl_t *p_ctrl)
.stop	ssp_err_t(* dac_api_t::stop)(dac_ctrl_t *p_ctrl)
.versionGet	ssp_err_t(* dac_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.10 インタフェース： display_api_t

説明： ディスプレイインタフェース

関数名	定義
.open	ssp_err_t(* display_api_t::open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)
.close	ssp_err_t(* display_api_t::close)(display_ctrl_t *const p_ctrl)

関数名	定義
.start	ssp_err_t(* display_api_t::start)(display_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* display_api_t::stop)(display_ctrl_t *const p_ctrl)
.layerChange	ssp_err_t(* display_api_t::layerChange)(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
.correction	ssp_err_t(* display_api_t::correction)(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)
.clut	ssp_err_t(* display_api_t::clut)(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)
.statusGet	ssp_err_t(* display_api_t::statusGet)(display_ctrl_t const *const p_ctrl, display_status_t *const p_status)
.versionGet	ssp_err_t(* display_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.11 インタフェース： doc_api_t

説明： DOC インタフェース

関数名	定義
.open	ssp_err_t(* doc_api_t::open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
.close	ssp_err_t(* doc_api_t::close)(doc_ctrl_t *const p_ctrl)
.statusGet	ssp_err_t(* doc_api_t::statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
.statusClear	ssp_err_t(* doc_api_t::statusClear)(doc_ctrl_t *const p_ctrl)
.write	ssp_err_t(* doc_api_t::write)(doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)
.inputRegisterWrite	ssp_err_t(* doc_api_t::inputRegisterWrite)(doc_ctrl_t *const p_ctrl, doc_size_t data)

関数名	定義
.versionGet	ssp_err_t(* doc_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.12 インタフェース： dsa_api_t

説明： DSA インタフェース

関数名	定義
.open	uint32_t(* dsa_api_t::open)(dsa_ctrl_t *const p_ctrl, dsa_cfg_t const *const p_cfg)
.verify	uint32_t(* dsa_api_t::verify)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash)
.hashVerify	uint32_t(* dsa_api_t::hashVerify)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_paddedHash)
.sign	uint32_t(* dsa_api_t::sign)(const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)
.hashSign	uint32_t(* dsa_api_t::hashSign)(dsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_paddedHash, uint32_t *p_dest)
.close	uint32_t(* dsa_api_t::close)(dsa_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* dsa_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.13 インタフェース： elc_api_t

説明： ELC インタフェース

関数名	定義
.init	ssp_err_t(* elc_api_t::init)(elc_cfg_t const *const p_cfg)

関数名	定義
.softwareEventGenerate	ssp_err_t(* elc_api_t::softwareEventGenerate)(elc_software_event_t event_num)
.linkSet	ssp_err_t(* elc_api_t::linkSet)(elc_peripheral_t peripheral, elc_event_t signal)
.linkBreak	ssp_err_t(* elc_api_t::linkBreak)(elc_peripheral_t peripheral)
.enable	ssp_err_t(* elc_api_t::enable)(void)
.disable	ssp_err_t(* elc_api_t::disable)(void)
.versionGet	ssp_err_t(* elc_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.14 インタフェース： external_irq_api_t

説明： 外部 IRQ インタフェース

関数名	定義
.open	ssp_err_t(* external_irq_api_t::open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
.enable	ssp_err_t(* external_irq_api_t::enable)(external_irq_ctrl_t *const p_ctrl)
.disable	ssp_err_t(* external_irq_api_t::disable)(external_irq_ctrl_t *const p_ctrl)
.triggerSet	ssp_err_t(* external_irq_api_t::triggerSet)(external_irq_ctrl_t *const p_ctrl, external_irq_trigger_t const trigger)
.filterEnable	ssp_err_t(* external_irq_api_t::filterEnable)(external_irq_ctrl_t *const p_ctrl)
.filterDisable	ssp_err_t(* external_irq_api_t::filterDisable)(external_irq_ctrl_t *const p_ctrl)
.close	ssp_err_t(* external_irq_api_t::close)(external_irq_ctrl_t *const p_ctrl)

関数名	定義
.versionGet	ssp_err_t(* external_irq_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.15 インタフェース： flash_api_t

説明： フラッシュインタフェース

関数名	定義
.open	ssp_err_t(* flash_api_t::open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
.write	ssp_err_t(* flash_api_t::write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
.read	ssp_err_t(* flash_api_t::read)(flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)
.erase	ssp_err_t(* flash_api_t::erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)
.blankCheck	ssp_err_t(* flash_api_t::blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
.close	ssp_err_t(* flash_api_t::close)(flash_ctrl_t *const p_ctrl)
.statusGet	ssp_err_t(* flash_api_t::statusGet)(flash_ctrl_t *const p_ctrl)
.accessWindowSet	ssp_err_t(* flash_api_t::accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)
.accessWindowClear	ssp_err_t(* flash_api_t::accessWindowClear)(flash_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* flash_api_t::reset)(flash_ctrl_t *const p_ctrl)
.updateFlashClockFreq	ssp_err_t(* flash_api_t::updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)

関数名	定義
.startupAreaSelect	ssp_err_t(* flash_api_t::startupAreaSelect)(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
.versionGet	ssp_err_t(* flash_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.16 インタフェース： fmi_api_t

説明： FMI インタフェース

関数名	定義
.productInfoGet	ssp_err_t(* fmi_api_t::productInfoGet)(fmi_product_info_t **pp_product_info)
.versionGet	ssp_err_t(* fmi_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.17 インタフェース： hash_api_t

説明： HASH アルゴリズムインタフェース

関数名	定義
.open	uint32_t(* hash_api_t::open)(hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)
.updateHash	uint32_t(* hash_api_t::updateHash)(const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
.hashUpdate	uint32_t(* hash_api_t::hashUpdate)(hash_ctrl_t *const p_ctrl, const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
.close	uint32_t(* hash_api_t::close)(hash_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* hash_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.18 インタフェース： i2s_api_t

説明： I2S インタフェース

関数名	定義
.open	ssp_err_t(* i2s_api_t::open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
.stop	ssp_err_t(* i2s_api_t::stop)(i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)
.mute	ssp_err_t(* i2s_api_t::mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
.write	ssp_err_t(* i2s_api_t::write)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)
.read	ssp_err_t(* i2s_api_t::read)(i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint16_t const bytes)
.writeRead	ssp_err_t(* i2s_api_t::writeRead)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes)
.infoGet	ssp_err_t(* i2s_api_t::infoGet)(i2s_ctrl_t *const p_ctrl, i2s_info_t *const p_info)
.close	ssp_err_t(* i2s_api_t::close)(i2s_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* i2s_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.19 インタフェース： input_capture_api_t

説明： 入力キャプチャインタフェース

関数名	定義
.open	ssp_err_t(* input_capture_api_t::open)(input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t const *const p_cfg)
.disable	ssp_err_t(* input_capture_api_t::disable)(input_capture_ctrl_t const *const p_ctrl)

関数名	定義
.enable	ssp_err_t(* input_capture_api_t::enable)(input_capture_ctrl_t const *const p_ctrl)
.infoGet	ssp_err_t(* input_capture_api_t::infoGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_info_t *const p_info)
.lastCaptureGet	ssp_err_t(* input_capture_api_t::lastCaptureGet)(input_capture_ctrl_t const *const p_ctrl, input_capture_capture_t *const p_counter)
.close	ssp_err_t(* input_capture_api_t::close)(input_capture_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* input_capture_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.20 インタフェース： ioport_api_t

説明： I/O ポートインタフェース

関数名	定義
.init	ssp_err_t(* ioport_api_t::init)(const ioport_cfg_t *p_cfg)
.pinCfg	ssp_err_t(* ioport_api_t::pinCfg)(ioport_port_pin_t pin, uint32_t cfg)
.pinDirectionSet	ssp_err_t(* ioport_api_t::pinDirectionSet)(ioport_port_pin_t pin, ioport_direction_t direction)
.pinEventInputRead	ssp_err_t(* ioport_api_t::pinEventInputRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_event)
.pinEventOutputWrite	ssp_err_t(* ioport_api_t::pinEventOutputWrite)(ioport_port_pin_t pin, ioport_level_t pin_value)

関数名	定義
.pinEthernetModeCfg	ssp_err_t(* ioport_api_t::pinEthernetModeCfg)(ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)
.pinRead	ssp_err_t(* ioport_api_t::pinRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_value)
.pinWrite	ssp_err_t(* ioport_api_t::pinWrite)(ioport_port_pin_t pin, ioport_level_t level)
.portDirectionSet	ssp_err_t(* ioport_api_t::portDirectionSet)(ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask)
.portEventInputRead	ssp_err_t(* ioport_api_t::portEventInputRead)(ioport_port_t port, ioport_size_t *p_event_data)
.portEventOutputWrite	ssp_err_t(* ioport_api_t::portEventOutputWrite)(ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
.portRead	ssp_err_t(* ioport_api_t::portRead)(ioport_port_t port, ioport_size_t *p_port_value)
.portWrite	ssp_err_t(* ioport_api_t::portWrite)(ioport_port_t port, ioport_size_t value, ioport_size_t mask)
.versionGet	ssp_err_t(* ioport_api_t::versionGet)(ssp_version_t *p_data)

11.2.1.21 インタフェース： jpeg_decode_api_t

説明： JPEG デコードインタフェース

関数名	定義
.open	ssp_err_t(* jpeg_decode_api_t::open)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)
.outputBufferSet	ssp_err_t(* jpeg_decode_api_t::outputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

関数名	定義
.horizontalStrideSet	ssp_err_t(* jpeg_decode_api_t::horizontalStrideSet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
.imageSubsampleSet	ssp_err_t(* jpeg_decode_api_t::imageSubsampleSet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
.inputBufferSet	ssp_err_t(* jpeg_decode_api_t::inputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
.linesDecodedGet	ssp_err_t(* jpeg_decode_api_t::linesDecodedGet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
.imageSizeGet	ssp_err_t(* jpeg_decode_api_t::imageSizeGet)(jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
.statusGet	ssp_err_t(* jpeg_decode_api_t::statusGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
.close	ssp_err_t(* jpeg_decode_api_t::close)(jpeg_decode_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* jpeg_decode_api_t::versionGet)(ssp_version_t *p_version)
.pixelFormatGet	ssp_err_t(* jpeg_decode_api_t::pixelFormatGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

11.2.1.22 インタフェース： keymatrix_api_t

説明： Key Matrix インタフェース

関数名	定義
.open	ssp_err_t(* keymatrix_api_t::open)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)
.enable	ssp_err_t(* keymatrix_api_t::enable)(keymatrix_ctrl_t *const p_ctrl)
.disable	ssp_err_t(* keymatrix_api_t::disable)(keymatrix_ctrl_t *const p_ctrl)
.triggerSet	ssp_err_t(* keymatrix_api_t::triggerSet)(keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t const trigger)
.close	ssp_err_t(* keymatrix_api_t::close)(keymatrix_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* keymatrix_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.23 インタフェース： lpm_api_t

説明： 低電力モードインタフェース

関数名	定義
.init	ssp_err_t(* lpm_api_t::init)(lpm_cfg_t const *const p_cfg)
.mstpcrSet	ssp_err_t(* lpm_api_t::mstpcrSet)(uint32_t mstpcra_value, uint32_t mstpcrb_value, uint32_t mstpcrc_value, uint32_t mstpcrd_value)
.mstpcrGet	ssp_err_t(* lpm_api_t::mstpcrGet)(uint32_t *mstpcra_value, uint32_t *mstpcrb_value, uint32_t *mstpcrc_value, uint32_t *mstpcrd_value)
.moduleStop	ssp_err_t(* lpm_api_t::moduleStop)(lpm_mstp_t module)
.moduleStart	ssp_err_t(* lpm_api_t::moduleStart)(lpm_mstp_t module)
.operatingPowerModeSet	ssp_err_t(* lpm_api_t::operatingPowerModeSet)(lpm_operating_power_t power_mode, lpm_subosc_t subosc)

関数名	定義
.snoozeEnable	ssp_err_t(* lpm_api_t::snoozeEnable)(lpm_snooze_rxd0_t rdx0_mode, lpm_snooze_dtc_t dtc_mode, lpm_snooze_request_t requests, lpm_snooze_end_t triggers)
.snoozeDisable	ssp_err_t(* lpm_api_t::snoozeDisable)(void)
.lowPowerCfg	ssp_err_t(* lpm_api_t::lowPowerCfg)(lpm_low_power_mode_t power_mode, lpm_output_port_enable_t output_port_enable, lpm_power_supply_t power_supply, lpm_io_port_t io_port_state)
.wupenSet	ssp_err_t(* lpm_api_t::wupenSet)(uint32_t wupen_value)
.wupenGet	ssp_err_t(* lpm_api_t::wupenGet)(uint32_t *wupen_value)
.deepStandbyCancelRequestEnable	ssp_err_t(* lpm_api_t::deepStandbyCancelRequestEnable)(lpm_deep_standby_t pin_signal, lpm_cancel_request_edge_t rising_falling)
.deepStandbyCancelRequestDisable	ssp_err_t(* lpm_api_t::deepStandbyCancelRequestDisable)(lpm_deep_standby_t pin_signal)
.lowPowerModeEnter	ssp_err_t(* lpm_api_t::lowPowerModeEnter)(void)
.versionGet	ssp_err_t(* lpm_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.24 インタフェース： lvd_api_t

説明： 低電圧検出ドライバインタフェース

関数名	定義
.open	ssp_err_t(* lvd_api_t::open)(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)
.statusGet	ssp_err_t(* lvd_api_t::statusGet)(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
.statusClear	ssp_err_t(* lvd_api_t::statusClear)(lvd_ctrl_t *const p_ctrl)

関数名	定義
.close	ssp_err_t(* lvd_api_t::close)(lvd_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* lvd_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.25 インタフェース： pdc_api_t

説明： PDC インタフェース

関数名	定義
.open	ssp_err_t(* pdc_api_t::open)(pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)
.close	ssp_err_t(* pdc_api_t::close)(pdc_ctrl_t *const p_ctrl)
.captureStart	ssp_err_t(* pdc_api_t::captureStart)(pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)
.stateGet	ssp_err_t(* pdc_api_t::stateGet)(pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)
.versionGet	ssp_err_t(* pdc_api_t::versionGet)(ssp_version_t *const p_data)

11.2.1.26 インタフェース： qspi_api_t

説明： クワッド SPI フラッシュインタフェース

関数名	定義
.open	ssp_err_t(* qspi_api_t::open)(qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)
.close	ssp_err_t(* qspi_api_t::close)(qspi_ctrl_t *p_ctrl)
.read	ssp_err_t(* qspi_api_t::read)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
.pageProgram	ssp_err_t(* qspi_api_t::pageProgram)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)

関数名	定義
.sectorErase	ssp_err_t(* qspi_api_t::sectorErase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)
.statusGet	ssp_err_t(* qspi_api_t::statusGet)(qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)
.bankSelect	ssp_err_t(* qspi_api_t::bankSelect)(uint32_t bank)
.versionGet	ssp_err_t(* qspi_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.27 インタフェース： rsa_api_t

説明： RSA インタフェース

関数名	定義
.open	uint32_t(* rsa_api_t::open)(rsa_ctrl_t *const p_ctrl, rsa_cfg_t const *const p_cfg)
.encrypt	uint32_t(* rsa_api_t::encrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.decrypt	uint32_t(* rsa_api_t::decrypt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.decryptCrt	uint32_t(* rsa_api_t::decryptCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.verify	uint32_t(* rsa_api_t::verify)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_signature, uint32_t *p_padded_hash)
.sign	uint32_t(* rsa_api_t::sign)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest)
.signCrt	uint32_t(* rsa_api_t::signCrt)(rsa_ctrl_t *const p_ctrl, const uint32_t *p_key, const uint32_t *p_domain, uint32_t num_words, uint32_t *p_padded_hash, uint32_t *p_dest)
.close	uint32_t(* rsa_api_t::close)(rsa_ctrl_t *const p_ctrl)

関数名	定義
.versionGet	uint32_t(* rsa_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.28 インタフェース： rtc_api_t

説明： RTC インタフェース

関数名	定義
.open	ssp_err_t(* rtc_api_t::open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
.close	ssp_err_t(* rtc_api_t::close)(rtc_ctrl_t *const p_ctrl)
.calendarTimeSet	ssp_err_t(* rtc_api_t::calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time, bool clock_start)
.calendarTimeGet	ssp_err_t(* rtc_api_t::calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)
.calendarAlarmSet	ssp_err_t(* rtc_api_t::calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm, bool irq_enable_flag)
.calendarAlarmGet	ssp_err_t(* rtc_api_t::calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm)
.calendarCounterStart	ssp_err_t(* rtc_api_t::calendarCounterStart)(rtc_ctrl_t *const p_ctrl)
.calendarCounterStop	ssp_err_t(* rtc_api_t::calendarCounterStop)(rtc_ctrl_t *const p_ctrl)
.irqEnable	ssp_err_t(* rtc_api_t::irqEnable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
.irqDisable	ssp_err_t(* rtc_api_t::irqDisable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
.periodicIrqRateSet	ssp_err_t(* rtc_api_t::periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t rate)
.infoGet	ssp_err_t(* rtc_api_t::infoGet)(rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)
.versionGet	ssp_err_t(* rtc_api_t::versionGet)(ssp_version_t *version)

11.2.1.29 インタフェース： sdmmc_api_t

説明： SDMMC インタフェース

関数名	定義
.open	ssp_err_t(* sdmmc_api_t::open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)
.close	ssp_err_t(* sdmmc_api_t::close)(sdmmc_ctrl_t *const p_ctrl)
.read	ssp_err_t(* sdmmc_api_t::read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
.write	ssp_err_t(* sdmmc_api_t::write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
.control	ssp_err_t(* sdmmc_api_t::control)(sdmmc_ctrl_t *const p_ctrl, ssp_command_t const command, void *p_data)
.readIo	ssp_err_t(* sdmmc_api_t::readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
.writeIo	ssp_err_t(* sdmmc_api_t::writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
.readIoExt	ssp_err_t(* sdmmc_api_t::readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
.writeIoExt	ssp_err_t(* sdmmc_api_t::writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
.ioIntEnable	ssp_err_t(* sdmmc_api_t::ioIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)
.versionGet	ssp_err_t(* sdmmc_api_t::versionGet)(ssp_version_t *const p_version)

関数名	定義
.infoGet	ssp_err_t(* sdmmc_api_t::infoGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)
.erase	ssp_err_t(* sdmmc_api_t::erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

11.2.1.30 インタフェース： sf_adc_periodic_api_t

説明： ADC 周期フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_adc_periodic_api_t::open)(sf_adc_periodic_ctrl_t *const p_ctrl, sf_adc_periodic_cfg_t const *const p_cfg)
.start	ssp_err_t(* sf_adc_periodic_api_t::start)(sf_adc_periodic_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_adc_periodic_api_t::stop)(sf_adc_periodic_ctrl_t *const p_ctrl)
.close	ssp_err_t(* sf_adc_periodic_api_t::close)(sf_adc_periodic_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_adc_periodic_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.31 インタフェース： sf_audio_playback_api_t

説明： オーディオフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_audio_playback_api_t::open)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_cfg_t const *const p_cfg)

関数名	定義
.close	ssp_err_t(* sf_audio_playback_api_t::close)(sf_audio_playback_ctrl_t *const p_ctrl)
.start	ssp_err_t(* sf_audio_playback_api_t::start)(sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)
.pause	ssp_err_t(* sf_audio_playback_api_t::pause)(sf_audio_playback_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_audio_playback_api_t::stop)(sf_audio_playback_ctrl_t *const p_ctrl)
.resume	ssp_err_t(* sf_audio_playback_api_t::resume)(sf_audio_playback_ctrl_t *const p_ctrl)
.volumeSet	ssp_err_t(* sf_audio_playback_api_t::volumeSet)(sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const volume)
.versionGet	ssp_err_t(* sf_audio_playback_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.32 インタフェース： sf_audio_playback_hw_api_t

説明： オーディオ再生フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_audio_playback_hw_api_t::open)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)
.start	ssp_err_t(* sf_audio_playback_hw_api_t::start)(sf_audio_playback_hw_ctrl_t *const p_ctrl)

関数名	定義
.stop	ssp_err_t(* sf_audio_playback_hw_api_t::stop)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
.play	ssp_err_t(* sf_audio_playback_hw_api_t::play)(sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const *const p_buffer, uint32_t length)
.dataTypeGet	ssp_err_t(* sf_audio_playback_hw_api_t::dataTypeGet)(sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)
.close	ssp_err_t(* sf_audio_playback_hw_api_t::close)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_audio_playback_hw_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.33 インタフェース： sf_block_media_api_t

説明： ブロックメディアフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_block_media_api_t::open)(sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_block_media_api_t::read)(sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
.write	ssp_err_t(* sf_block_media_api_t::write)(sf_block_media_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count)
.ioctl	ssp_err_t(* sf_block_media_api_t::ioctl)(sf_block_media_ctrl_t *p_ctrl, ssp_command_t const command, void *p_data)

関数名	定義
.close	ssp_err_t(* sf_block_media_api_t::close)(sf_block_media_ctrl_t *p_ctrl)
.versionGet	ssp_err_t(* sf_block_media_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.34 インタフェース： sf_comms_api_t

説明： 通信フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_comms_api_t::open)(sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_comms_api_t::close)(sf_comms_ctrl_t *const p_ctrl)
.read	ssp_err_t(* sf_comms_api_t::read)(sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout)
.write	ssp_err_t(* sf_comms_api_t::write)(sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)
.lock	ssp_err_t(* sf_comms_api_t::lock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type, UINT timeout)
.unlock	ssp_err_t(* sf_comms_api_t::unlock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type)
.versionGet	ssp_err_t(* sf_comms_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.35 インタフェース： sf_console_api_t

説明： コンソールフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_console_api_t::open)(sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_console_api_t::close)(sf_console_ctrl_t *const p_ctrl)
.prompt	ssp_err_t(* sf_console_api_t::prompt)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_menu, UINT const timeout)
.parse	ssp_err_t(* sf_console_api_t::parse)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes)
.read	ssp_err_t(* sf_console_api_t::read)(sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout)
.write	ssp_err_t(* sf_console_api_t::write)(sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout)
.argumentFind	ssp_err_t(* sf_console_api_t::argumentFind)(uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data)
.versionGet	ssp_err_t(* sf_console_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.36 インタフェース： sf_el_gx_api_t

説明： GUIXTM インタフェース

関数名	定義
.open	ssp_err_t(* sf_el_gx_api_t::open)(sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_el_gx_api_t::close)(sf_el_gx_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_el_gx_api_t::versionGet)(ssp_version_t *p_version)

関数名	定義
.setup	UINT(* sf_el_gx_api_t::setup)(GX_DISPLAY *p_display)
.canvasInit	ssp_err_t(* sf_el_gx_api_t::canvasInit)(sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT *p_window_root)

11.2.1.37 インタフェース： sf_external_irq_api_t

説明： 外部 IRQ フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_external_irq_api_t::open)(sf_external_irq_ctrl_t *const p_ctrl, sf_external_irq_cfg_t const *const p_cfg)
.wait	ssp_err_t(* sf_external_irq_api_t::wait)(sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)
.versionGet	ssp_err_t(* sf_external_irq_api_t::versionGet)(ssp_version_t *const p_version)
.close	ssp_err_t(* sf_external_irq_api_t::close)(sf_external_irq_ctrl_t *const p_ctrl)

11.2.1.38 インタフェース： sf_i2c_api_t

説明： I2C フレームワーク

関数名	定義
.open	ssp_err_t(* sf_i2c_api_t::open)(sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_i2c_api_t::read)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout)
.write	ssp_err_t(* sf_i2c_api_t::write)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout)

関数名	定義
.reset	ssp_err_t(* sf_i2c_api_t::reset)(sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)
.close	ssp_err_t(* sf_i2c_api_t::close)(sf_i2c_ctrl_t *const p_ctrl)
.lock	ssp_err_t(* sf_i2c_api_t::lock)(sf_i2c_ctrl_t *const p_ctrl)
.unlock	ssp_err_t(* sf_i2c_api_t::unlock)(sf_i2c_ctrl_t *const p_ctrl)
.version	ssp_err_t(* sf_i2c_api_t::version)(ssp_version_t *const p_version)

11.2.1.39 インタフェース： sf_jpeg_decode_api_t

説明： JPEG デコードフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_jpeg_decode_api_t::open)(sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg)
.inputBufferSet	ssp_err_t(* sf_jpeg_decode_api_t::inputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const buffer_size)
.outputBufferSet	ssp_err_t(* sf_jpeg_decode_api_t::outputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
.linesDecodedGet	ssp_err_t(* sf_jpeg_decode_api_t::linesDecodedGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
.horizontalStrideSet	ssp_err_t(* sf_jpeg_decode_api_t::horizontalStrideSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
.imageSubsampleSet	ssp_err_t(* sf_jpeg_decode_api_t::imageSubsampleSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

関数名	定義
.wait	ssp_err_t(* sf_jpeg_decode_api_t::wait)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status, uint32_t timeout)
.statusGet	ssp_err_t(* sf_jpeg_decode_api_t::statusGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
.imageSizeGet	ssp_err_t(* sf_jpeg_decode_api_t::imageSizeGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
.pixelFormatGet	ssp_err_t(* sf_jpeg_decode_api_t::pixelFormatGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)
.close	ssp_err_t(* sf_jpeg_decode_api_t::close)(sf_jpeg_decode_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_jpeg_decode_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.40 インタフェース： sf_message_api_t

説明： メッセージングフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_message_api_t::open)(sf_message_ctrl_t *const p_ctrl, sf_message_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_message_api_t::close)(sf_message_ctrl_t *const p_ctrl)
.bufferAcquire	ssp_err_t(* sf_message_api_t::bufferAcquire)(sf_message_ctrl_t const *const p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const p_acquire_cfg, uint32_t const wait_option)

関数名	定義
.bufferRelease	ssp_err_t(* sf_message_api_t::bufferRelease)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t const option)
.post	ssp_err_t(* sf_message_api_t::post)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)
.pend	ssp_err_t(* sf_message_api_t::pend)(sf_message_ctrl_t const *const p_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option)
.versionGet	ssp_err_t(* sf_message_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.41 インタフェース： sf_power_profiles_api_t

説明： パワー プロファイルフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_power_profiles_api_t::open)(sf_power_profiles_ctrl_t *const p_ctrl, sf_power_profiles_cfg_t const *const p_cfg)
.sleep	ssp_err_t(* sf_power_profiles_api_t::sleep)(sf_power_profiles_ctrl_t *const p_ctrl)
.close	ssp_err_t(* sf_power_profiles_api_t::close)(sf_power_profiles_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_power_profiles_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.42 インタフェース： sf_spi_api_t

説明： SPI フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_spi_api_t::open)(sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_spi_api_t::read)(sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
.write	ssp_err_t(* sf_spi_api_t::write)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
.writeRead	ssp_err_t(* sf_spi_api_t::writeRead)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
.close	ssp_err_t(* sf_spi_api_t::close)(sf_spi_ctrl_t *const p_ctrl)
.lock	ssp_err_t(* sf_spi_api_t::lock)(sf_spi_ctrl_t *const p_ctrl)
.unlock	ssp_err_t(* sf_spi_api_t::unlock)(sf_spi_ctrl_t *const p_ctrl)
.version	ssp_err_t(* sf_spi_api_t::version)(ssp_version_t *const p_version)

11.2.1.43 インタフェース： sf_thread_monitor_api_t

説明： スレッド監視フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_thread_monitor_api_t::open)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_thread_monitor_api_t::close)(sf_thread_monitor_ctrl_t *const p_ctrl)

関数名	定義
.threadRegister	ssp_err_t(* sf_thread_monitor_api_t::threadRegister)(sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max)
.threadUnregister	ssp_err_t(* sf_thread_monitor_api_t::threadUnregister)(sf_thread_monitor_ctrl_t *const p_ctrl)
.countIncrement	ssp_err_t(* sf_thread_monitor_api_t::countIncrement)(sf_thread_monitor_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_thread_monitor_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.44 インタフェース： sf_touch_ctsu_api_t

説明： CTSU フレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_touch_ctsu_api_t::open)(sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_touch_ctsu_api_t::read)(sf_touch_ctsu_ctrl_t *const p_ctrl, void *p_dest, ctsu_read_t opts, const ctsu_channel_pair_t *channels, const uint16_t count)
.close	ssp_err_t(* sf_touch_ctsu_api_t::close)(sf_touch_ctsu_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_ctsu_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.45 インタフェース： sf_touch_ctsu_button_api_t

説明： CTSU ボタンフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_touch_ctsu_button_api_t::open)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_cfg_t const *const p_cfg)
.enable	ssp_err_t(* sf_touch_ctsu_button_api_t::enable)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_id const button_id)
.disable	ssp_err_t(* sf_touch_ctsu_button_api_t::disable)(sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_id const button_id)
.close	ssp_err_t(* sf_touch_ctsu_button_api_t::close)(sf_touch_ctsu_button_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_ctsu_button_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.46 インタフェース： sf_touch_ctsu_slider_api_t

説明： CTSU スライダフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_touch_ctsu_slider_api_t::open)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_cfg_t const *const p_cfg)
.enable	ssp_err_t(* sf_touch_ctsu_slider_api_t::enable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
.disable	ssp_err_t(* sf_touch_ctsu_slider_api_t::disable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)

関数名	定義
.close	ssp_err_t(* sf_touch_ctsu_slider_api_t::close)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_ctsu_slider_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.47 インタフェース： sf_touch_panel_api_t

説明： タッチパネルフレームワークインタフェース

関数名	定義
.open	ssp_err_t(* sf_touch_panel_api_t::open)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_cfg_t const *const p_cfg)
.calibrate	ssp_err_t(* sf_touch_panel_api_t::calibrate)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_calibrate_t const *const p_expected, sf_touch_panel_payload_t const *const p_actual, ULONG timeout)
.start	ssp_err_t(* sf_touch_panel_api_t::start)(sf_touch_panel_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_touch_panel_api_t::stop)(sf_touch_panel_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* sf_touch_panel_api_t::reset)(sf_touch_panel_ctrl_t *const p_ctrl)
.close	ssp_err_t(* sf_touch_panel_api_t::close)(sf_touch_panel_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_panel_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.48 インタフェース： slcdc_api_t

説明： SLCDC インタフェース

関数名	定義
.open	ssp_err_t(* slcdc_api_t::open)(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)
.write	ssp_err_t(* slcdc_api_t::write)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
.modify	ssp_err_t(* slcdc_api_t::modify)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment, slcdc_size_t const data_mask, slcdc_size_t const data)
.start	ssp_err_t(* slcdc_api_t::start)(slcdc_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* slcdc_api_t::stop)(slcdc_ctrl_t *const p_ctrl)
.contrastIncrease	ssp_err_t(* slcdc_api_t::contrastIncrease)(slcdc_ctrl_t *const p_ctrl)
.contrastDecrease	ssp_err_t(* slcdc_api_t::contrastDecrease)(slcdc_ctrl_t *const p_ctrl)
.setdisplayArea	ssp_err_t(* slcdc_api_t::setdisplayArea)(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)
.close	ssp_err_t(* slcdc_api_t::close)(slcdc_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* slcdc_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.49 インタフェース： spi_api_t

説明： SPI インタフェース

関数名	定義
.open	ssp_err_t(* spi_api_t::open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
.read	ssp_err_t(* spi_api_t::read)(spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

関数名	定義
.write	ssp_err_t(* spi_api_t::write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
.writeRead	ssp_err_t(* spi_api_t::writeRead)(spi_ctrl_t *const p_ctrl, void const *p_src, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
.close	ssp_err_t(* spi_api_t::close)(spi_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* spi_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.50 インタフェース： tdes_api_t

説明： TDES インタフェース

関数名	定義
.open	uint32_t(* tdes_api_t::open)(tdes_ctrl_t *const p_ctrl, tdes_cfg_t const *const p_cfg)
.encrypt	uint32_t(* tdes_api_t::encrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.decrypt	uint32_t(* tdes_api_t::decrypt)(tdes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.close	uint32_t(* tdes_api_t::close)(tdes_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* tdes_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.51 インタフェース： timer_api_t

説明： タイマインタフェース

関数名	定義
.open	ssp_err_t(* timer_api_t::open)(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

関数名	定義
.stop	ssp_err_t(* timer_api_t::stop)(timer_ctrl_t *const p_ctrl)
.start	ssp_err_t(* timer_api_t::start)(timer_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* timer_api_t::reset)(timer_ctrl_t *const p_ctrl)
.counterGet	ssp_err_t(* timer_api_t::counterGet)(timer_ctrl_t *const p_ctrl, timer_size_t *const p_value)
.periodSet	ssp_err_t(* timer_api_t::periodSet)(timer_ctrl_t *const p_ctrl, timer_size_t const period, timer_unit_t const unit)
.dutyCycleSet	ssp_err_t(* timer_api_t::dutyCycleSet)(timer_ctrl_t *const p_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)
.infoGet	ssp_err_t(* timer_api_t::infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
.close	ssp_err_t(* timer_api_t::close)(timer_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* timer_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.52 インタフェース： transfer_api_t

説明： 転送インタフェース

関数名	定義
.open	ssp_err_t(* transfer_api_t::open)(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
.reset	ssp_err_t(* transfer_api_t::reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)
.enable	ssp_err_t(* transfer_api_t::enable)(transfer_ctrl_t *const p_ctrl)
.disable	ssp_err_t(* transfer_api_t::disable)(transfer_ctrl_t *const p_ctrl)
.start	ssp_err_t(* transfer_api_t::start)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)

関数名	定義
.stop	ssp_err_t(* transfer_api_t::stop)(transfer_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* transfer_api_t::infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_info)
.close	ssp_err_t(* transfer_api_t::close)(transfer_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* transfer_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.53 インタフェース： trng_api_t

説明： 乱数生成

関数名	定義
.open	uint32_t(* trng_api_t::open)(trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
.read	uint32_t(* trng_api_t::read)(trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)
.close	uint32_t(* trng_api_t::close)(trng_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* trng_api_t::versionGet)(ssp_version_t *const p_version)

11.2.1.54 インタフェース： uart_api_t

説明： UART インタフェース

関数名	定義
.open	ssp_err_t(* uart_api_t::open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
.read	ssp_err_t(* uart_api_t::read)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest, uint32_t const bytes)
.write	ssp_err_t(* uart_api_t::write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

関数名	定義
.baudSet	ssp_err_t(* uart_api_t::baudSet)(uart_ctrl_t const *const p_ctrl, uint32_t const baudrate)
.infoGet	ssp_err_t(* uart_api_t::infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
.close	ssp_err_t(* uart_api_t::close)(uart_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* uart_api_t::versionGet)(ssp_version_t *p_version)

11.2.1.55 インタフェース： wdt_api_t

説明： WDT インタフェース

関数名	定義
.cfgGet	ssp_err_t(* wdt_api_t::cfgGet)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)
.open	ssp_err_t(* wdt_api_t::open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
.refresh	ssp_err_t(* wdt_api_t::refresh)(wdt_ctrl_t *const p_ctrl)
.statusGet	ssp_err_t(* wdt_api_t::statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
.statusClear	ssp_err_t(* wdt_api_t::statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
.counterGet	ssp_err_t(* wdt_api_t::counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
.timeoutGet	ssp_err_t(* wdt_api_t::timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
.versionGet	ssp_err_t(* wdt_api_t::versionGet)(ssp_version_t *const p_data)

Renesas Synergy™ ソフトウェアパッケージ（SSP）
v1.1.0ユーザーズマニュアル（参考資料）

発行年月日　2016年10月13日　Rev.0.94

発行　　ルネサス エレクトロニクス株式会社
〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

Renesas Synergy™ Software Package（SSP）v1.1.0ユーザーズマニュアル（参考資料）

ルネサス エレクトロニクス株式会社

R01US0171JU0094