RAファミリ
RA MCUのためのIEC60730/60335セルフテスト・ライブラリ (RA0_CM23)

R01AN7509JJ0100
Rev.1.00
Jun.30.25

要旨

今日、自動電子制御システムが多くの多様なアプリケーションに拡大し続けているため、信頼性と安全性の要件は、システム設計においてますます増大する要素になりつつあります。

たとえば、家電製品向けの IEC60730 安全規格を導入するには、製造業者が製品の安全で信頼性の高い動作を保証する自動電子制御を設計する必要があります。

IEC60730 規格は製品設計のすべての側面をカバーしていますが、Annex H はマイクロコントローラベースの制御システムの設計にとって非常に重要です。これにより、自動電子制御用の 3 つのソフトウェア分類が提供されます。

1. クラス A： 装置の安全性に寄与することを意図したものではない制御機能
   例：部屋のサーモスタット、湿度制御、照明制御、タイマ、スイッチ
2. クラス B： 装置の安全でない操作を防止するための制御機能
   例：洗濯機のサーマルカットオフおよびドアロック
3. クラス C： 特別な危険を防止するための制御機能
   例：自動バーナー制御と閉動作のためのサーマルカットアウト

洗濯機、食器洗い機、乾燥機、冷蔵庫、冷凍庫、クッカー/ストーブなどの器具は、クラス B の分類に分類される傾向があります。

このアプリケーションノートでは、柔軟なサンプルソフトウェアルーチンを使用して、IEC60730 クラス B 安全規格への準拠を支援する方法のガイドラインを示します。これらのルーチンは VDE Test and Certification Institute GmbH によって認定されており、テスト証明書のコピーは、このアプリケーションノートのダウンロードパッケージで入手できます。

提供されるソフトウェアルーチンは、リセット後およびプログラムの実行中に使用されます。このドキュメントとそれに付随するサンプルコードは、これを行う方法の例を提供します。

動作確認デバイス

- デバイス：

  ルネサス RA ファミリ MCU（Arm® Cortex®-M23）※シリーズとグループは、次ページ参照

- 開発環境：

  GCC(GNU) Arm Embedded 13.2.1.arm-13-7 / e2 studio 2024-07 (24.7.0)

本書において「RA MCU」と表記している場合は、以下の製品のことを指します。

表 1. RA ファミリ MCU セルフテスト機能リスト

	CPU コア	Arm® Cortex®-M23
	シリーズ	RA0
	グループ	RA0E1
テスト機能	CPU	○(注)
	ROM	○
	RAM	○
	クロック	○
	独立ウォッチドッグタイマ (IWDT)	○
	ADC12	○
	GPIO	○

【注】　FPU 非搭載のため、FPU テストを除く

目次

1. テスト

1.1 CPU

この章では、CPU テストルーチンについて説明します。(参照：*IEC 60730-1:2013+A1:2015+A2:2020 Annex H - Table H.11.12.7 1.CPU*)

このソフトウェアは、次の CPU レジスタをテストします。

表 1-1 テストされるレジスター覧表

	CPU コア	Arm® Cortex®-M23
	グループ	RA0E1
テスト対象レジスタ	汎用レジスタ	R0-R12
	制御レジスタ (注)	MSP, PSP, LR, PSR(APSR, IPSR, EPSR), CONTROL
	プログラムカウンタ	PC

【注】 レジスタ名が同じでも、デバイスによってビットフィールドの構成が異なる場合があります。

ソースファイル cpu_test.c は、C 言語を使用した CPU テストの実装を提供し、アセンブリ言語関数（CPU_Test_Control）に依存してレジスタにアクセスします。汎用レジスタのカップリングテストを使用するには、ファイル cpu_test_coupling.c も必要です。結合テストはアセンブリ言語関数に依存します。

- TestGPRsCouplingStart_A
- TestGPRsCouplingR0_A
- TestGPRsCouplingR1_R3_A
- TestGPRsCouplingR4_R6_A
- TestGPRsCouplingR7_R9_A
- TestGPRsCouplingR10_R12_A
- TestGPRsCouplingStart_B
- TestGPRsCouplingR0_B
- TestGPRsCouplingR1_R3_B
- TestGPRsCouplingR4_R6_B
- TestGPRsCouplingR7_R9_B
- TestGPRsCouplingR10_R12_B
- TestGPRsCouplingEnd

あるいは、CPU_Test_General_Low、CPU_Test_General_High アセンブリ言語関数を使用して、汎用レジスタをテストします。

cpu_test.h ヘッダファイルは、CPU テストへのインタフェースを提供します。

本テストは各レジスタの基本的な読み書きをテストしています。API 関数には、テストの結果を示す戻り値はありません。代わりにこれらのテストのユーザは、次の宣言でエラー処理関数を作成する必要があります。

```
extern void CPU_Test_ErrorHandler(void);
```

エラーが検出された場合、CPU テストはこの関数にジャンプします。この関数は return してはいけません。

すべてのテスト関数は、C 関数呼び出し後のレジスタ保存の規則に従います。したがって、ユーザはこれらの関数を通常の C 関数のように呼び出すことができ、事前にレジスタ値を保存するいかなる責任もありません。

1.1.1　CPU テスト　ソフトウェア API

表 1-2 CPU テスト　ソフトウェア API ソースファイル

ファイル名
cpu_test.h
cpu_test.c cpu_test_coupling.c
TestGPRsCouplingStart_A.asm TestGPRsCouplingR0_A.asm TestGPRsCouplingR1_R3_A.asm TestGPRsCouplingR4_R6_A.asm TestGPRsCouplingR7_R9_A.asm TestGPRsCouplingR10_R12_A.asm TestGPRsCouplingStart_B.asm TestGPRsCouplingR0_B.asm TestGPRsCouplingR1_R3_B.asm TestGPRsCouplingR4_R6_B.asm TestGPRsCouplingR7_R9_B.asm TestGPRsCouplingR10_R12_B.asm TestGPRsCouplingEnd.asm CPU_Test_Control.asm CPU_Test_General_Low.asm CPU_Test_General_High.asm

■ cpu_test.c ファイル

Syntax
void CPU_Test_All(void)
Description
以下に詳述するすべてのテストを次の順序で実行します。 1. カップリング汎用レジスタテストを使用する場合（下記、注 1 を参照）: CPU_Test_GPRsCouplingPartA CPU_Test_GPRsCouplingPartB 2. カップリング汎用レジスタテストを使用しない場合: CPU_Test_General_Low CPU_Test_General_High 3. CPU_Test_Control 4. CPU_Test_PC プロセッサが特権モードであることを確認するのは、呼び出し元の関数の責任です。この関数が非特権モードで呼び出された場合、一部のレジスタビットは非特権モードでアクセスできないため、テストは失敗します。 さらに CPU_Test_Control 関数はスタックポインタレジスタ（MSP と PSP）をテストするため、このテストの間、割り込みが発生しないようにするのは呼び出し元関数の責任です。 エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。 詳細については、個々のテストを参照してください。 【注】 1. コード内の #define USE_TEST_GPRS_COUPLING は、汎用レジスタのテストに使用される関数を選択するために使用されます。

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void CPU_Test_PC(void)	
Description	
この関数は、プログラムカウンタ（PC）レジスタをテストします。 別関数を呼び出すことで PC が動作していることをテストします。 指定されたパラメータの反転値を返す別関数(TestPC_TestFunction)を呼び出して、戻り値をチェックします。 これにより、PC が確実に動作していることを確認します。 エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ CPU_Test_General_Low.asm ファイル

Syntax	
void CPU_Test_General_Low(void)	
Description	
汎用レジスタ R0、R1、R2、R3、R4、R5、R6 および R7 をテストします。レジスタはペアでテストされます。 レジスタの各ペアについて、 1. 両方のレジスタに h'55555555 を書き込みます。 2. 両方のレジスタを読み、一致していることを確認します。 3. 両方のレジスタに h'AAAAAAAA を書き込みます。 4. 両方のレジスタを読み、一致していることを確認します。 このテストの間、例外を無効にするのは呼び出し元関数の責任です。 エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ CPU_Test_General_High.asm ファイル

Syntax	
void CPU_Test_General_High(void)	
Description	
汎用レジスタ R8、R9、R10、R11 および R12 をテストします。レジスタはペアでテストされます。 レジスタの各ペアについて、 1. 両方のレジスタに h'55555555 を書き込みます。 2. 両方のレジスタを読み、一致していることを確認します。 3. 両方のレジスタに h'AAAAAAAA を書き込みます。 4. 両方のレジスタを読み、一致していることを確認します。 このテストの間、例外を無効にするのは呼び出し元関数の責任です。 エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ CPU_Test_Control.asm ファイル

Syntax
void CPU_Test_Control(void)

Description

この関数は、制御レジスタ（デバイスにより異なるため「表 1-1 テストされるレジスタ」を参照）をテストします。
このテストでは、レジスタ R1～R4 が機能していることを前提としています。
通常、各レジスタのテスト手順は次のとおりです。
各レジスタについて、

1. h'55555555 を書き込みます。
2. 読み返して、値が h'55555555 であることを確認します。
3. h'AAAAAAAA を書き込みます。
4. 読み返して、値が h'AAAAAAAA であることを確認します。

ただし、レジスタ内の特定のビットの制限により、この手順が許可されない場合があることに注意してください。
これらのケースでは他のテスト値が選択されています。
プロセッサが特権モードであることを確認するのは、呼び出し元の関数の責任です。非特権モードでこの関数が呼び出されると、非特権モードでは一部のレジスタビットにアクセスできないため、テストは失敗します。
このテストの間、例外を無効にするのは呼び出し元関数の責任です。

【注】 このテストでは例外を無効にする必要があるため、FAULTMASK および PRIMASK はテストされません。したがって、これらは FAULTMASK および PRIMASK を変更するテスト中にアクティブ化されません。

エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ cpu_test_coupling.c ファイル

Syntax	
void CPU_Test_GPRsCouplingPartA(void)	
Description	
汎用レジスタ R0 から R12 をテストします。レジスタ間の結合障害が検出されます。 汎用レジスタテストはパート A とパート B で構成され、この関数はパート A です。 このテストの間、割り込みが発生しないようにするのは呼び出し元関数の責任です。 エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void CPU_Test_GPRsCouplingPartB(void)	
Description	
汎用レジスタ R0 から R12 をテストします。レジスタ間の結合障害が検出されます。 汎用レジスタテストはパート A とパート B で構成され、この関数はパート B です。 このテストの間、割り込みが発生しないようにするのは呼び出し元関数の責任です。 エラーが検出された場合、外部関数 CPU_Test_ErrorHandler が呼び出されます。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

1.2　ROM

この章では、CRC ルーチンを使用した ROM /フラッシュメモリテストについて説明します。(参照：*IEC 60730-1:2013+A1:2015+A2:2020 Annex H– H2.19.4.2 CRC – Double Word*)

CRC は、メモリの内容に基づいて単一ワードまたはチェックサムを生成する不具合/エラー制御方法です。

CRC チェックサムは、メッセージビットストリームのビット繰り上がりなし（減算ではなく XOR を使用）n 次の多項式の係数を表す、長さ n+1 の定義済み（short）ビットストリームによるバイナリ除算の剰余です。除算の前に、n 個のゼロがメッセージストリームに追加されます。CRC は、バイナリハードウェアへの実装が簡単で数学的にも分析しやすいため、よく使用されます。

ROM テストは、ROM 内容の CRC 値を予め生成して保存することで実現できます。ROM セルフテストでは、同じ CRC アルゴリズムを用いて新たに CRC 値を生成し、保存しておいた CRC 値と比較します。この手法は、すべての 1 ビットエラーと高い割合のマルチビットエラーを認識します。

他の CRC ジェネレータによって事前に生成された CRC 値と比較する場合、基本的な CRC アルゴリズムが同じであっても、計算結果が同一にならない要因がいくつかあるため注意が必要です。たとえば、データをアルゴリズムに供給する順序、使用されるルックアップテーブルで想定されるビット順序、あるいは実際の CRC 値のビットに必要な順序の組み合わせ等です。システムがビッグエンディアンとリトルエンディアンの両方に対応する場合も問題になります。また、一部のデバッガは ROM 上でのソフトウェアブレイクを実現するものがあり、その場合はデバッグ中に ROM の内容が書き換えられてしまう可能性があります。

参照用 CRC 値の計算方法は、使用するツールチェーンで異なります。詳しい手順は、2. 使用例の 2.2 ROM を参照ください。

1.2.1　CRC32 アルゴリズム

RA MCU には、CRC32 アルゴリズムのサポートが可能な CRC（巡回冗長検査）演算器が内蔵されています。テストソフトウェアは、32 ビット CRC32 を生成するように CRC 演算器を設定します。

- 多項式 = 0x04C11DB7 ($x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$)
- 幅 = 32 bit
- 初期値 = 0xFFFFFFFF
- h'FFFFFFFF との XOR 演算結果が CRC に出力される

1.2.2　ROM テスト　ソフトウェア API

このセクションの関数は、CRC 値を計算し、ROM に格納されている値と比較してその正確性を検証するために使用されます。

すべてのソースは ANSI C で記述されます。renesas.h ヘッダファイルには、RA MCU のレジスタ定義が含まれます。

表 1-3 CRC ソフトウェア API ソースファイル

ファイル名
crc.h crc_verify.h
crc.c CRC_Verify.c

■ CRC_Verify.c ファイル

Syntax	
bool_t CRC_Verify(const uint32_t ui32_NewCRCValue, const uint32_t ui32_AddrRefCRC)	
Description	
この関数は、参照 CRC が格納されているアドレスを提供することにより、新しい CRC 値を参照 CRC と比較します。	
Input Parameters	
const uint32_t ui32_NewCRCValue	計算された新しい CRC 値
const uint32_t ui32_AddrRefCRC	32 ビット参照 CRC 値が格納されるアドレス
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テスト失敗

■ crc.c ファイル

Syntax	
void CRC_Init(void)	
Description	
CRC モジュールを初期化します。この関数は、他の CRC 関数を呼び出す前に呼び出す必要があります。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
uint32_t CRC_Calculate(const uint32_t* pui32_Data, uint32_t ui32_Length)	
Description	
この関数は、単一の指定されたメモリ領域の CRC を計算します。	
Input Parameters	
const uint32_t* pui32_Data	テストするメモリの開始を指すポインタ
uint32_t ui32_Length	ロングワード単位のデータの長さ
Output Parameters	
NONE	N/A
Return Values	
Uint32_t	計算された CRC32 値

以下の関数は、メモリ領域を単純に開始アドレスと長さで指定できない場合に使用されます。それらは範囲/セクションにメモリ領域を追加する方法を提供します。これは、関数 CRC_Calculate が 1 回の関数呼び出しで時間がかかり過ぎる場合にも使用できます。

■ crc.c ファイル

Syntax	
void CRC_Start(void)	
Description	
CRC モジュールを開始条件にリセットします。 関数 CRC_AddRange を使用する前にこれを 1 回呼び出します。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void CRC_AddRange(const uint32_t* pui32_Data, uint32_t ui32_Length)	
Description	
複数のアドレス範囲で構成されるデータの CRC を計算する場合は、CRC_Calculate ではなくこの関数を使用します。 最初に CRC_Start を呼び出し、次に必要なアドレス範囲ごとに CRC_AddRange を呼び出し、その後 CRC_Result を呼び出して CRC 値を取得します。	
Input Parameters	
const uint32_t* pui32_Data	テストするメモリ範囲の先頭を指すポインタ
uint32_t ui32_Length	ロングワード単位のデータの長さ
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
uint32_t CRC_Result(void)	
Description	
CRC データ出力レジスタ(CRCDOR)からの読み出し値をビット反転した値を戻り値として返します。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
uint32_t	計算された CRC32 の値

1.3 RAM

March テストは、RAM をテストする効果的な方法としてよく知られている一連のテストです。

March テストは、March エレメントの有限シーケンスで構成されます。March エレメントは、次のセルに進む前に、メモリアレイ内のすべてのセルに適用される有限の操作シーケンスです。

一般に、アルゴリズムを構成する March エレメントが多いほど、その障害カバー率は向上しますが、実行時間が長くなります。

アルゴリズム自体は破壊的ですが (現在の RAM 値は保持されない)、提供されているテスト関数は非破壊的なオプションを提供するため、メモリの内容を保持できます。これは、実際のアルゴリズムを実行する前にメモリを提供されたバッファにコピーし、テストの最後にバッファからメモリを復元することによって実現されます。API には RAM テスト領域だけでなく、バッファも自動的にテストするオプションが含まれています。

テスト中の RAM 領域は、テスト中は他の用途に使用できません。そのためスタックに使用される RAM のテストが特に困難になります。この問題を解決するために、API にはスタックのテストに使用できる関数が含まれています。

次のセクションでは、特定の March テストについて説明します。続いて、ソフトウェア API の仕様を示します。

1.3.1 RAM テストアルゴリズム

1.3.1.1 March C-

March C- アルゴリズム（van de Goor 1991）は、合計で 10 の演算がある 6 つのマーチエレメントで構成され、次の故障を検出します:

1. 縮退故障（SAF：Stuck-At Faults）
   - セルまたはラインの論理値は常に 0 または 1 です
2. 遷移故障（TF：Transition Faults）
   - 0→1 または 1→0 の遷移ができないセルまたはライン
3. カップリング故障（CF：Coupling Faults）
   - 1 つのセルへの書き込み操作により、2 番目のセルの内容が変更される
4. デコーダ故障（AF：Address decoder Faults）
   - アドレスデコーダに影響を与える障害
   - 特定のアドレスで、セルにアクセスできない
   - 特定のセルにアクセスできない
   - 特定のアドレスで、複数のセルに同時にアクセスされる
   - 特定のセルに、複数のアドレスからアクセスできる

6 つの March エレメントがあります。

1. アレイにすべて 0 を書き込む
2. 最下位アドレスから開始して、0 をリード、1 をライト、アレイをビットごとにインクリメント
3. 最下位アドレスから開始して、1 をリード、0 をライト、アレイをビットごとにインクリメント
4. 最上位アドレスから始めて、0 をリード、1 をライト、アレイをビットごとにデクリメント
5. 最上位アドレスから始めて、1 をリード、0 をライト、アレイをビットごとにデクリメント
6. アレイからすべての 0 をリード

1.3.1.2 March X

注：このアルゴリズムは RA MCU 用には実装されていないため、下記の March X WOM アルゴリズムの参考情報として提示します。

March X アルゴリズムは、合計で 6 つの演算がある 4 つの March エレメントで構成されます。次の故障を検出します。

1. 縮退故障（SAF）
2. 遷移故障（TF）
3. 反転カップリング故障（CFin：inversion Coupling Faults）
4. アドレスデコーダ故障（AF）

4 つの March エレメントがあります。

1. アレイにすべて 0 をライト
2. 最下位アドレスから開始して、0 をリード、1 をライト、アレイをビットごとにインクリメント
3. 最上位アドレスから開始して、1 をリード、0 をライト、アレイをビットごとにデクリメント
4. アレイからすべて 0 をリード

1.3.1.3 March X WOM (Word-Oriented Memory version)

March X Word-Oriented Memory（WOM）アルゴリズムは、2 段階で標準 March X アルゴリズムから作成されました。まず、標準の March X は、シングルビットデータパターンの使用から、メモリアクセス幅に等しいデータパターンの使用に変換されます。この段階でのテストは、主にアドレスデコーダ障害を含むワード間障害を検出しています。2 番目の段階は、2 つの March エレメントを追加することです。1 つ目は H/L ビットが交互になるデータパターンを使用し、2 つ目はその逆を使用します。これらのエレメントの追加は、ワード内カップリングフォールトを検出することです。

6 つの March エレメントがあります。

1. アレイにすべて 0 をライト
2. 最下位アドレスから開始して、0 をリード、1 をライト、アレイをワード単位でインクリメント
3. 最上位アドレスから開始して、1 をリード、0 をライト、ワードごとにデクリメント
4. 最下位アドレスから開始して、0 をリード、h'AA をライト、アレイをワード単位でインクリメント
5. 最上位のアドレスから始めて、h'AA をリード、h'55 をライト、ワードごとにデクリメント
6. アレイからすべての h'55 をリード

1.3.2　RAM テスト　ソフトウェア API

RAM テストでは 2 つのテストアルゴリズムに対応した API を準備しています。

各 API について説明します。

1.3.2.1　March C- アルゴリズム　ソフトウェア API

このテストは、8、16、または 32 ビットの RAM アクセスを使用するように構成できます。

これは、ヘッダファイルの #defining RAMTEST_MARCH_C_ACCESS_SIZE を次のいずれかにすることで実現されます。

1. RAMTEST_MARCH_C_ACCESS_SIZE_8BIT
2. RAMTEST_MARCH_C_ACCESS_SIZE_16BIT
3. RAMTEST_MARCH_C_ACCESS_SIZE_32BIT

単一の関数呼び出しでテストできる RAM の最大サイズを制限するとテストが高速化し、スタックとコードのサイズが小さくなる場合があります。これは、テスト領域に含まれる「word」の数を保持するために使用される変数のサイズを制限することによって行われます。「word」サイズは選択したアクセス幅です。

これは、ヘッダファイルの #defining RAMTEST_MARCH_C_MAX_WORDS を次のいずれかにすることで実現されます:

1. RAMTEST_MARCH_C_MAX_WORDS_8BIT　(Max words in test area is 0xFF)
2. RAMTEST_MARCH_C_MAX_WORDS_16BIT　(Max words in test area is 0xFFFF)
3. RAMTEST_MARCH_C_MAX_WORDS_32BIT　(Max words in test area is 0xFFFFFFFF)

表 1-4　March C- アルゴリズム　ソフトウェア API ソースファイル

ファイル名
ramtest_march_c.h
ramtest_march_c.c

ソースは ANSI C で記述され、renesas.h ファイルを使用してペリフェラルレジスタにアクセスします。

【注】　API は関数呼び出しで 1 つのワードだけをテストします。しかし、ワード間でテストする結合故障は、関数を使用して 1 ワードより大きいデータ範囲をテストすることが重要です。

■ ramtest_march_c.c ファイル

Syntax	
bool_t RamTest_March_C(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
March C- (Goor 1991) アルゴリズムを用いた RAM メモリテスト	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word アドレス。これは、選択したメモリアクセス幅に合わせる必要があります。
const uint32_t ui32_EndAddr	テストする RAM の最後の word アドレス。これは、選択したメモリアクセス幅に合わせて、ui32_StartAddr 以上の値にする必要があります。
void* const p_RAMSafe	破壊的なメモリテストの場合は、NULL に設定します。 非破壊メモリテストの場合は、テスト領域の内容をコピーするのに十分な大きさで、選択したメモリアクセス幅に合わせたバッファの先頭を設定します。
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

Syntax	
bool_t RamTest_March_C_Extra(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
March C-（Goor 1991）アルゴリズムを用いた非破壊 RAM メモリテスト。 この関数は、RamTest_March_C 関数とは異なり、使用前に「RAMSafe」バッファをテストします。 「RAMSafe」バッファのテストが失敗すると、テストは中止され、関数は False を返します。	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word アドレス。これは、選択したメモリアクセス幅に合わせる必要があります。
const uint32_t ui32_EndAddr	テストする RAM の最後の word アドレス。これは、選択したメモリアクセス幅に合わせて、ui32_StartAddr 以上の値にする必要があります。
void* const p_RAMSafe	テスト領域の内容をコピーするのに十分な大きさで、選択したメモリアクセス幅に揃えられたバッファの先頭に設定します。
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

1.3.2.2　March X WOM アルゴリズム　ソフトウェア API

このテストは、8、16、または 32 ビットの RAM アクセス用に構成できます。

これは、ヘッダファイル内の #defining RAMTEST_MARCH_X_WOM_ACCESS_SIZE に次のいずれかを選ぶことで実現されます:

1. RAMTEST_MARCH_X_WOM_ACCESS_SIZE_8BIT
2. RAMTEST_MARCH_X_WOM_ACCESS_SIZE_16BIT
3. RAMTEST_MARCH_X_WOM_ACCESS_SIZE_32BIT

テストの実行時間を短縮するために、1 回の関数呼び出しでテストできる RAM の最大サイズを制限することを選択できます。これは、テスト領域に含まれる「word」の数を保持するために使用される変数のサイズを制限することによって行われます。「word」サイズは、選択したアクセス幅と同じです。

これは、ヘッダファイルの #defining RAMTEST_MARCH_X_WOM_MAX_WORDS を次のいずれかにすることによって実現されます:

1. RAMTEST_MARCH_X_WOM_MAX_WORDS_8BIT　(Max words in test area is 0xFF)
2. RAMTEST_MARCH_X_WOM_MAX_WORDS_16BIT　(Max words in test area is 0xFFFF)
3. RAMTEST_MARCH_X_WOM_MAX_WORDS_32BIT　(Max words in test area is 0xFFFFFFFF)

表 1-5　March X WOM アルゴリズム　ソフトウェア API ソースファイル

ファイル名
ramtest_march_x_wom.h
ramtest_march_x_wom.c

ソースは ANSI C で記述され、renesas.h ファイルを使用してペリフェラルレジスタにアクセスします。

【注】　API は関数呼び出しで 1 つのワードだけをテストします。しかし、ワード間でテストする結合故障は、関数を使用して 1 ワードより大きいデータ範囲をテストすることが重要です。

■ ramtest_march_x_wom.c ファイル

Syntax	
bool_t RamTest_March_X_WOM(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
WOM 用に変換された March X アルゴリズムに基づく RAM メモリテスト	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word アドレス。これは、選択したメモリアクセス幅に合わせる必要があります。
const uint32_t ui32_EndAddr	テストする RAM の最後の word アドレス。これは、選択したメモリアクセス幅に合わせ、ui32_StartAddr 以上の値にする必要があります。
void* const p_RAMSafe	破壊的なメモリテストの場合は、NULL に設定します。 非破壊メモリテストの場合は、テスト領域の内容をコピーするのに十分な大きさで、選択したメモリアクセス幅に合わせたバッファの先頭を設定します。
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

Syntax	
bool_t RamTest_March_X_WOM_Extra(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
WOM 用に変換された March X アルゴリズムに基づく非破壊 RAM メモリテスト。 この関数は、RamTest_March_X_WOM 関数とは異なり、使用前に「RAMSafe」バッファをテストします。「RAMSafe」バッファのテストが失敗すると、テストは中止され、関数は False を返します。	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word アドレス。これは、選択したメモリアクセス幅に合わせる必要があります。
const uint32_t ui32_EndAddr	テストする RAM の最後の word アドレス。これは、選択したメモリアクセス幅に合わせ、ui32_StartAddr 以上の値にする必要があります。
void* const p_RAMSafe	テスト領域の内容をコピーするのに十分な大きさで、選択したメモリアクセス幅に揃えられたバッファの先頭に設定します。
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

1.3.2.3　RAM テスト(スタック)　ソフトウェア API

この API を使用すると、スタックを含む RAM 領域で RAM テストを実行できます。RAM テストを実行する関数にはスタックが必要なので、これらの関数はスタックを提供された新しい RAM 領域に再配置して、元のスタック領域をテストできるようにします。テスト領域にあるスタック（メインまたはプロセス）に応じて、または両方のスタックに応じて、呼び出すことができる 3 つの関数が提供されます。

プロセッサが特権モードであることを確認するのは、呼び出し元関数の責任です。この関数が非特権モードで呼び出されると、一部のレジスタビットが非特権モードでアクセスできないため、テストは失敗します。

【注】　スタックテスト関数は、関数ポインタ渡しで前述の March RAM テストの 1 つを利用します。使用前に初期化を必要とするテストの場合、これらの関数のいずれかを呼び出してテストする前に、初期化が完了していることを確認するのは、ユーザの責任です。

表 1-6 RAM テスト(スタック)　ソフトウェア API ソースファイル

ファイル名
ramtest_stack.h
ramtest_stack.c
StartBothTestAssembly.asm StartMainTestAssembly.asm StartProcTestAssembly.asm

■ ramtest_stack.c ファイル

Syntax	
bool_t RamTest_Stack_Main(	const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe, const uint32_t ui32_NewMSP, const TEST_FUNC fpTest_Func)
Description	
メインスタックを含む（プロセススタックは含まない）領域の RAM テスト	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word のアドレス。これは fpTest_Func の要件に適応しなければなりません。
const uint32_t ui32_EndAddr	テストする RAM の最後の word のアドレス。これは fpTest_Func の要件に適応しなければなりません。
void* const p_RAMSafe	テスト RAM 領域と同じサイズのバッファの先頭に設定します。これは、fpTest_Func の要件に適応しなければなりません。
const uint32_t ui32_NewUSP	再配置先のメインスタックの新しいスタックポインタ値
const TEST_FUNC fpTest_Func	使用する実際のメモリテストへの TEST_FUNC タイプの関数ポインタ。 Typedef bool_t(*TEST_FUNC)(uint32_t, uint32_t, void*); 例：RamTest_March_X_WOM
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

Syntax	
bool_t RamTest_Stack_Proc(	const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe, const uint32_t ui32_NewPSP, const TEST_FUNC fpTest_Func)
Description	
プロセススタックを含む（メインスタックは含まない）領域の RAM テスト	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word のアドレス。これは fpTest_Func の要件に適応しなければなりません。
const uint32_t ui32_EndAddr	テストする RAM の最後の word のアドレス。これは fpTest_Func の要件に適応しなければなりません。
void* const p_RAMSafe	テスト RAM 領域と同じサイズのバッファの先頭に設定します。これは、fpTest_Func の要件に適応しなければなりません。
const uint32_t ui32_NewPSP	再配置先のプロセススタックの新しいスタックポインタ値
const fpTest_Func	使用する実際のメモリテストへの TEST_FUNC タイプの関数ポインタ。 Typedef bool_t(*TEST_FUNC)(uint32_t, uint32_t, void*); 例：RamTest_March_X_WOM
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

Syntax	
bool_t RamTest_Stacks(	const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe, const uint32_t ui32_NewPSP, const uint32_t ui32_NewMSP, const TEST_FUNC fpTest_Func)
Description	
メインスタックとプロセススタックの両方のスタックを含む領域の RAM テスト	
Input Parameters	
const uint32_t ui32_StartAddr	テストする RAM の最初の word のアドレス。これは fpTest_Func の要件に適応しなければなりません。
const uint32_t ui32_EndAddr	テストする RAM の最後の word のアドレス。これは fpTest_Func の要件に適応しなければなりません。
void* const p_RAMSafe	テスト RAM 領域と同じサイズのバッファの先頭に設定します。これは、fpTest_Func の要件に適応しなければなりません。
const uint32_t ui32_NewPSP	再配置先のプロセススタックの新しいスタックポインタ値
const uint32_t ui32_NewMSP	再配置先のメインスタックの新しいスタックポインタ値
const TEST_FUNC fpTest_Func	使用する実際のメモリテストへの TEST_FUNC タイプの関数ポインタ。 Typedef bool_t(*TEST_FUNC)(const uint32_t, const uint32_t, void* const); 例：RamTest_March_X_WOM
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストまたはパラメータチェックのフェイル

1.4　クロック

本テストは、MCU内蔵のタイマアレイユニット(TAU)　チャネル5による入力パルスのインターバル測定機能を使用してシステムクロックが許容範囲内の周波数であることを確認します。範囲外の場合、周波数異常を検出してエラー処理を実施します。

1.4.1　TAU0 ch5(入力パルスのインターバル測定機能)によるシステムクロック周波数の監視

タイマ入力選択レジスタ0(TIS0.TIS[2:0])で選択された入力パルス信号の有効エッジでカウントが開始します。タイマのカウント値は次のパルスの有効エッジでキャプチャされます。この方法で入力パルスのインターバルは測定されます。

タイマ入力選択レジスタ 0(TIS0.TIS[2:0])で選択されるタイマ入力に「**LOCO**」を選択し、関連レジスタ(*1)でチャネル 5 の動作クロックに「**PCLKB(分周なし)**」を選択します。

(*1)　TMR05.CCS=0, TMR05.CKS[1:0]=b'00 , TPS0.PRS0[3:0]=0x0 を設定

RA0E1 では PCLKB(周辺モジュールクロック)はシステムクロック(ICLK)と同じクロック源です。

以下、今回使用するタイマアレイユニット 0 のチャネル 5 の内部ブロック図を示します。

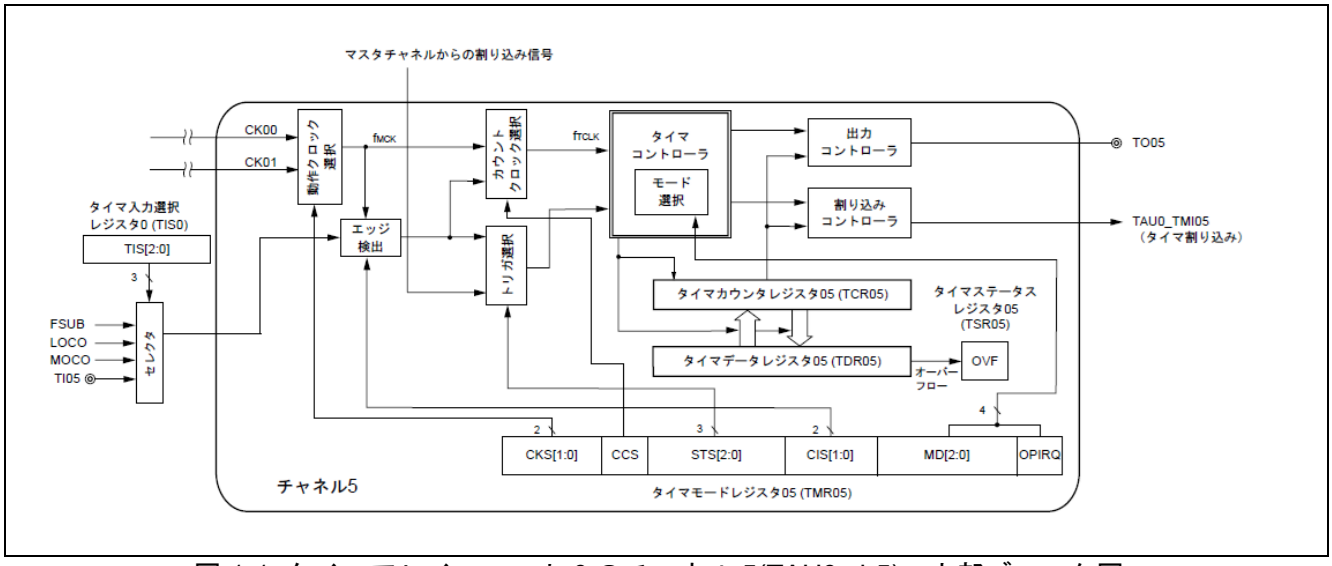

図 1-1 タイマアレイユニット 0 のチャネル 5(TAU0 ch5)　内部ブロック図

本サンプルソフトでは、LOCO 入力の有効エッジを検出すると、PCLKB を動作クロックとしてカウントを開始し、次の有効エッジを検出すると TCR05 レジスタの値がタイマデータレジスタ 05 (TDR05) にキャプチャされ、TAU0_TMI05 が発生します。(TCR05 レジスタは再び 0x0000 からカウント開始します。)

※詳細は、ユーザーズマニュアル ハードウェア編「タイマアレイユニット(TAU)」章を参照ください。

また、TAU0 ch5 割込み内でシステムクロックの周波数が設定範囲内かを判定し、範囲外の場合はエラー処理を実行します。

このテストモジュールの使用例については、2.4 章を参照してください。

1.4.2　クロックテスト　ソフトウエア API

表 1-7 Clock テスト　ソフトウェア API ソースファイル

ファイル名
clock_monitor.h
clock_monitor.c

テストモジュールは、renesas.h ヘッダファイルを使用してペリフェラルレジスタにアクセスします。

■ clock_monitor.c ファイル

Syntax	
void ClockMonitor_Init(CLOCK_MONITOR_ERROR_CALL_BACK CallBack)	
Description	
1. エラー判定時の処理関数を登録 2. TAU0(ch5)モジュールの初期設定を行い、PCLKB(システムクロックがクロック源)の監視を開始します。 3. LOCO が停止していた場合、発振を開始させる。	
Input Parameters	
CLOCK_MONITOR_ERROR_CALL_BACK CallBack	PCLKB が許容範囲外の場合に呼び出される関数
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void tau0_tmi05_int (void)	
Description	
TAU0 ch5 キャプチャ終了割り込みハンドラ。 入力パルスのキャプチャ結果が有効範囲内かを判定し、有効範囲外の場合は ClockMonitor_Init 関数で登録されたコールバック関数を呼び出します。 ※有効範囲は、clock_monitor.h で”hwMAXTIME”, “hwMINTIME”を定義します。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

1.5　独立ウォッチドッグタイマ (IWDT)

ウォッチドッグタイマは、異常なプログラムの実行を検出するために使用されます。プログラムが期待どおりに実行されていない場合、ソフトウェアによるウォッチドッグタイマ更新が必要なタイミングで行われないため、エラーを検出します。

これには、RA MCU の独立ウォッチドッグタイマ (IWDT) モジュールが使用されます。ウィンドウ機能が含まれているため、指定した時間の直前ではなく、指定したウィンドウ内で更新を行う必要があります。エラーが検出された場合、内部リセットまたはノンマスカブル割り込み (NMI) を生成するように構成できます。

IWDT のすべての構成は、「オプション設定メモリ」内のオプション機能選択レジスタ 0 (OFS0) で行います（構成の例については、2.5 章を参照）。オプション設定メモリとは、リセット後のマイコンの状態を選択するために利用可能な一連のレジスタのことで、コードフラッシュの領域に配置されます。

テストモジュールは、renesas.h ヘッダファイルを使用してペリフェラルレジスタにアクセスします。

1.5.1　IWDT　ソフトウェア API

表 1-8 独立ウォッチドッグタイマソースファイル

ファイル名
iwdt.h
iwdt.c

Syntax	
void IWDT_Init (void)	
Description	
独立ウォッチドッグタイマを初期化します。この関数を呼び出した後は、ウォッチドッグタイマエラーを防ぐために、IWDT_kick 関数を正しい時間に呼び出す必要があります。 【注】　割り込みを生成するように構成されている場合、これはノンマスカブル割り込み (NMI) になります。これは NMISR.IWDTST フラグをチェックするユーザコードで処理する必要があります。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void IWDT_Kick(void)	
Description	
ウォッチドッグタイマのカウントをリフレッシュします。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
bool_t IWDT_DidReset(void)	
Description	
IWDT がタイムアウトしたか、正しく更新されなかった場合は True を返します。 また、アンダーフローフラグおよびリフレッシュエラーフラグはクリアします。 ※Class-B ではデフォルト設定は「ウィンドウ制御なし(100%)」です。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool_t	IWDT がタイムアウトした場合又は、正しく更新されなかった場合は True、それ以外の場合は False

1.6　ADC

本 ADC(ADC12)には、＋側基準電圧、－側基準電圧、内部基準電圧をテスト用 A/D 変換に選択する機能があります。

この機能を使用して A/D コンバータが正常に動作しているかを確認することができます。本サンプルソフトでは、12bit 分解能を前提としております。

本 ADC のテストモード設定については、ユーザーズマニュアル　ハードウエア編「12 ビット A/D コンバータ (ADC12)」章を参照ください。

表 1-9 MCU 内蔵の ADC

グループ	RA0E1
内蔵 ADC	ADC12
ユニット数	1
診断基準電圧	「-」側の基準電圧／内部基準電圧／「+」側の基準電圧 (注)

【注】　各測定モードと診断基準電圧の関係は以下の通り　※本サンプルソフト

ゼロスケール測定時に “「-」側の基準電圧” を変換対象に選択

フルスケール測定時に”「+」側の基準電圧”を変換対象に選択

テストモジュールは、renesas.h ヘッダファイルを使用してペリフェラルレジスタにアクセスします。

1.6.1　ADC テスト　ソフトウェア API

表 1-10 ADC テスト　ソフトウェア API ソースファイル

ファイル名
test_adc.h
test_adc.c

Syntax	
void Test_ADC_Init(void)	
Description	
ADC モジュール（ユニット 0）を初期化します。 本関数は他の ADC 関数を使用する前に呼び出す必要があります。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
bool_t Test_ADC_Wait(void)	
Description	
この関数は、ADC モジュール（ユニット 0）の基本設定を実施し、テスト用 A/D 変換対象の切替を行い、A/D 変換が行われている間は待機します。 A/D 変換結果が許容範囲かを判断し、範囲内の場合は TRUE(1)を、範囲外の場合はエラーと判断し、FALSE(0)を返します。なお、本関数内でエラー処理関数へは移行しません。 ※ADM0、ADM1、ADM2、ADUL、ADLL、ADTES の各レジスタを設定します。	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストフェイル

Syntax	
void Test_ADC_Start(const ADC_ERROR_CALL_BACK Callback)	
Description	
・ADC モジュール（ユニット 0）のテスト用 A/D 変換対象の切替えを行い、A/D 変換を実施します。 ※開始タイミングによって 1 回目の A/D 変換データを無視する必要があるため、本関数内で 2 度の A/D 変換動作を行います。 ・A/D 変換完了後に Test_ADC_CheckResult 関数を呼び出して、A/D 変換結果を確認する必要があります。 ※ADM0、ADUL、ADLL、ADTES の各レジスタを設定します。	
Input Parameters	
const ADC_ERROR_CALL_BACK Callback	エラーが検出された場合に呼び出す関数。 【注】　この関数はパラメータ bCallErrorHandler が True に設定されて Test_ADC_CheckResult が呼び出された場合にのみ、呼び出されます。 本関数では g_ADC_ErrorCallBack へ Callback 内容を格納するのみ
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
bool_t Test_ADC_CheckResult(const bool_t bCallErrorHandler)	
Description	
・ADC モジュール（ユニット 0）のテスト用 A/D 変換結果が各対象の許容範囲内(注)かを確認します。 ・Test_ADC_Start 関数の後に呼び出し、テスト用 A/D 変換が完了するたびに呼び出す必要があります。 ・A/D 変換結果が範囲内の場合は TRUE(1)を返します。 ・A/D 変換結果が範囲外の場合はエラーと判断し、FALSE(0)を返します。 なお、引数:“bCallErrorHandler”が”True”の場合、エラーと判断し、エラー処理関数へ移行します。 【注】　許容範囲内については、test_adc.h を参照。	
Input Parameters	
const bool_t bCallErrorHandler	Test_ADC_Start 関数に提供されるエラーコールバック関数を呼び出すには True を設定し、そうでない場合は False を設定します。
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = テストパス、False = テストフェイル

1.7　GPIO

GPIO リードバックレベル検出機能は、ポートが出力モード時に、端子のデジタル出力レベルを読み出す機能です。端子部の不良診断を行うことができます。

GPIO リードバックレベル検出機能では、以下のような手順で端子をチェックします。

1. 前処理として、該当端子の PMC ビット(端子モード制御)を”0”(デジタル入出力)に設定する ※RA0E1 は必要
2. Pmn 方向レジスタ(PDRm : m=0-9)の PDRn(n=00-15) ビットを 1 にして出力ポートに設定する
3. Pmn 出力設定レジスタ(POSRm : m=0-9)の POSRn(n=00-15) ビットに 1 をセットすることで High を出力する
4. Pmn 状態レジスタ(PIDRm : m=0～9)の PIDRn(n=00～15) で端子の状態を読み出し、High レベルを確認する。
5. Pmn 出力リセットレジスタ(PORRm : m=0-9)の PORRn(n=00-15)ビットに 1 をセットすることで Low を出力
6. Pmn 状態レジスタ(PIDRm : m=0～9)の PIDRn(n=00～15) で端子の状態を読み出し、Low レベルを確認する。

テスト対象のポートは、gpio_config.h ヘッダファイルで指定します。

1.7.1　GPIO テスト　ソフトウェア API

表 1-11 GPIO テスト　ソフトウェア API ソースファイル

ファイル名
gpio.h gpio_config.h
gpio.c

Syntax	
void GPIO_Start(const GPIO_ERROR_CALL_BACK Callback)	
Description	
この関数は GPIO リードバックレベル検出機能を利用して、ポートが出力モード時に端子のデジタル出力レベルを切り替えて読み出し、端子部の不良診断を行います。 テスト対象のポートは、gpio_config.h ヘッダファイルで指定します。	
Input Parameters	
const GPIO_ERROR_CALL_BACK Callback	端子部の不良を検出した（ポートの読み出し値が期待値と異なる）場合に呼び出される関数
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

2.　使用例

このセクションでは、アプリケーションソフトウェアにセルフテストライブラリを適用する方法に関する、いくつかの有用な提案をユーザに提供します。

セルフテストは次の 2 つのパターンに分けられます。

(a)　電源投入時のテスト

リセット後に一度実行されるテストです。これらはできるだけ早く実行する必要がありますが、特に起動時間が重要な場合は、すべてのテストを実行する前に初期化コードを実行して、たとえばより高速なメインクロックを選択できるようにすることもできます。

(b)　定期的なテスト

通常のプログラム操作を通じて定期的に実行されるテストです。このドキュメントでは、特定のテストを実行する頻度を判断することはできません。定期的なテストのスケジューリング方法は、アプリケーションの構造に応じてユーザが決定します。

以下のセクションでは、各テストタイプの使用例を示します。

2.1　CPU

いずれかの CPU テストで障害が検出されると、CPU_Test_ErrorHandler と呼ばれるユーザ指定の関数が呼び出されます。CPU のエラーは非常に深刻なので、この機能の目的は、ソフトウェアの実行に依存しない安全な状態にできるだけ早く到達することです。

2.1.1　電源投入時

すべての CPU テストは、リセット後できるだけ早く実行する必要があります。

【注】　この関数は、デバイスを非特権モードにする前に呼び出す必要があります。
関数 CPU_Test_All を使用して、すべての CPU テストを自動的に実行できます。

2.1.2　定期的

CPU を定期的にテストするには、電源投入テストと同様に、CPU_Test_All 関数を使用して、すべての CPU テストを自動的に実行できます。または 1 回の関数呼び出しで実行されるテストの量を減らすため、ユーザは CPU 定期テストがスケジュールされるたびに、個々の CPU テスト関数を順番に呼び出すことも選択できます。

2.2　ROM

ROM は、その内容の CRC 値を計算し（CRC32）、予め保存しておいた参照 CRC 値（CRC 計算に含まれていない ROM 内の特定の場所に追加する必要がある）と比較することでテストします。参照 CRC 値の計算方法は、使用する開発環境によって異なります。

RA MCU 内蔵の CRC モジュールは、CRC_Init 関数を呼び出して、使用する前に初期化する必要があります。また、結果が一致するためには、テスト対象となる ROM セクションが、事前の CRC 値計算と ROM テストの両方に同じように含まれていることを確認します。

2.2.1　事前の参照用 CRC 計算

GNU ツールには CRC の計算機能が付属しないため、以下に紹介する SRecord ツール（注）を使用して、参照 CRC 値を計算します。ユーザは、このツールを利用して、予め参照用の CRC 値を ROM に書き込んでおき、セルフテストではこの値とテストで計算した値を比較します。

【注】 SRecord は、SourceForge のオープンソースプロジェクトです。詳細は下記を参照ください。

- SRecord Web Site (SRecord v1.65)
  http://srecord.sourceforge.net/

- CRC Checksum Generation with “SRecord” Tools for GNU and Eclips
  https://llvm-gcc-renesas.com/wiki/index.php?title=CRC_Checksum_Generation_with_%E2%80%98SRecord%E2%80%99_Tools_for_GNU_and_Eclipse

ダウンロードしたファイル(srecord-1.65.0-win64.zip)を解凍すると、\srecord-1.65.0-win64\bin フォルダに以下のプログラムが展開されます。

※リファレンスマニュアルは、\srecord-1.65.0-win64\share\doc\srecord フォルダ内に同梱されております。

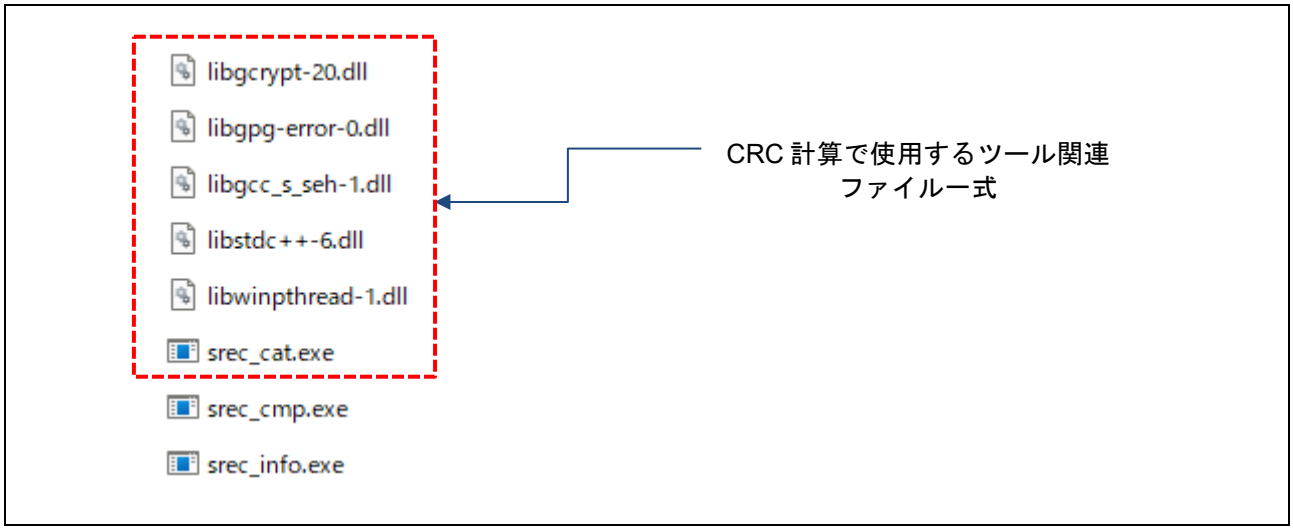

図 2-1　SRecord ツールの内容

プロジェクト及び SRecord ツールのフォルダ構成例を以下に示します。

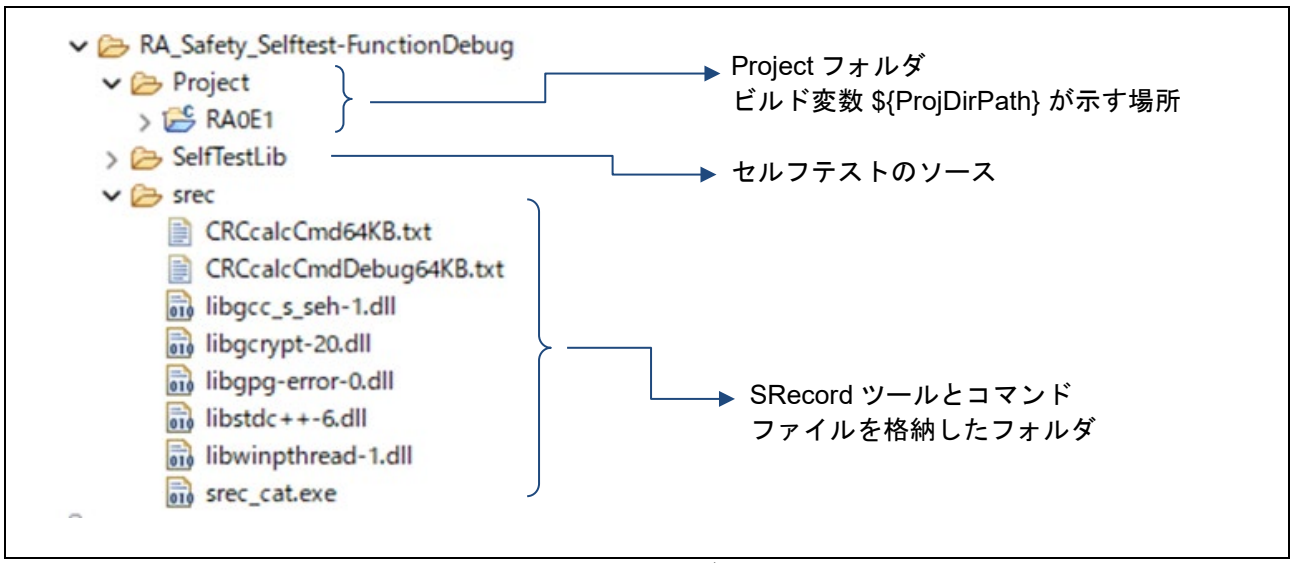

図 2-2　フォルダ構成例

e² studio の「プロジェクト(Project)」⇒「プロパティ(Properties)」を開き、ビルド後のステップで、objcopy コマンドを使って、.elf ファイルから S レコードファイルを生成します。ここでは、変換後のファイル名を Original.srec とします。このファイルが、SRecord ツールの入力になります。

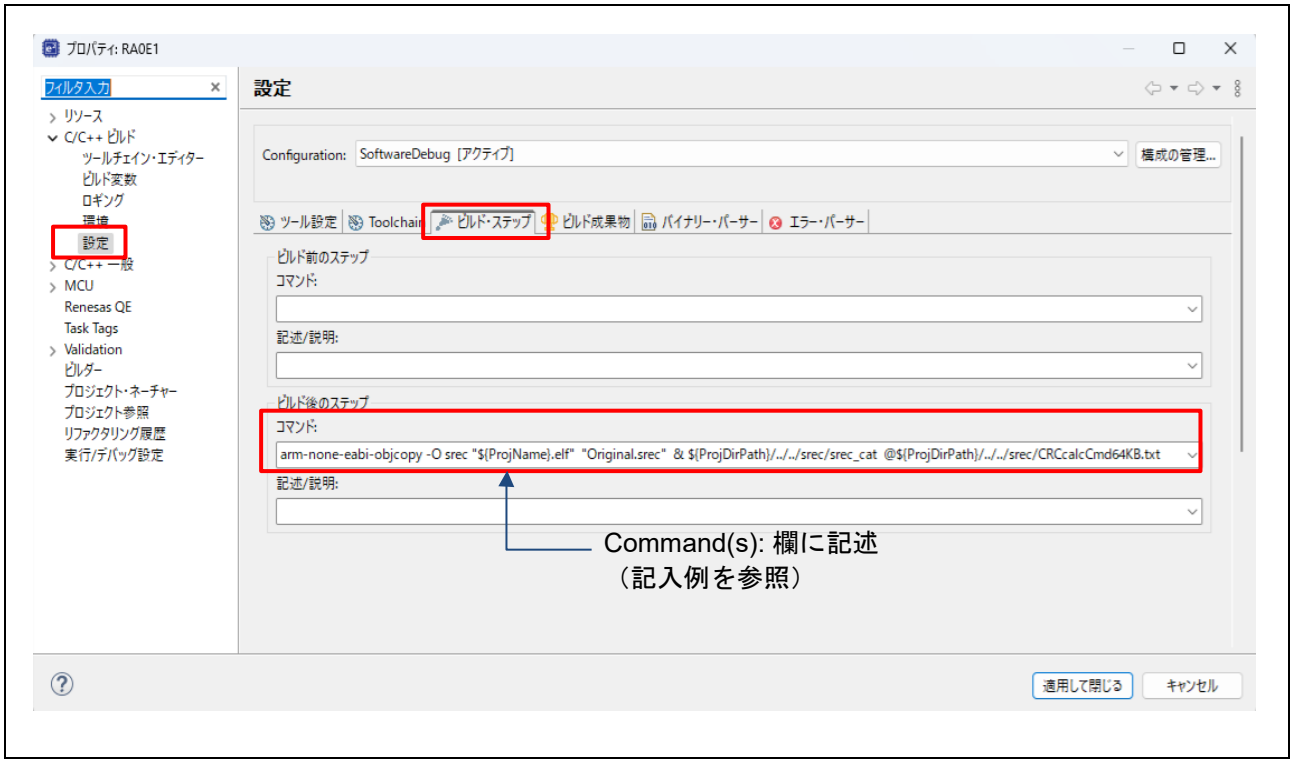

図 2-3　S レコードファイルの出力と SRecord ツールの起動

上図における”設定(Setting)”画面の「ビルド・ステップ(Build Steps)」タブの「ビルド後のステップ(Post-build steps)」で、次のように記述します。

■ Command(s):欄の記入例（改行せず 1 行に書きます）

arm-none-eabi-objcopy -O srec "${ProjName}.elf" "**Original.srec**" & ${ProjDirPath}/../../srec/**srec_cat @**${ProjDirPath}/../../srec/**CRCcalcCmd64KB.txt**

1 行目の"&"の前までが S レコードファイルの生成、2 行目の書式「**srec_cat @**コマンドファイル」が、srec_cat ツールの起動 になります。コマンドファイルとして「**CRCcalcCmd64KB.txt**」の記述例を以下に示します。

なお、RA0E1 搭載の CRC 演算器では「LSB ファースト通信用 CRC 生成」のみの対応となるため、以下のコマンドファイル内のオプション指定(赤色の点線枠部分)が異なることに注意してください。

■ CRCcalcCmd64KB.txt ファイルの内容（例）

```
# CRC calculate
Original.srec                          # Read srec file
-fill 0xFF 0x00000 0x010000            # 64KB ROM fill by 0xFF
-crop 0x00000 0x00FFFC                 # CRC calculate area
# -STM32-le 0x00FFFC                   # Calculate and output CRC value
-CRC32_Little_Endian 0x00FFFC -CCITT   # CRC32 LSB (with initial value :
0xFFFFFFFF)
-crop 0x0FFFC 0x10000
Original.srec
-fill 0xFF 0x000000 0x0FFFC            # -fill 0xFF 0x0000 0x0FFFC
-Output_Block_Size 16                  # Set Block size
#-Output addcrc.srec
# Motorola  S-Record fomat and the number of bytes which form each address is
"4".
-Output addcrc.srec -Motorola -address-length=4
```

デバイスにより ROM の容量が異なる場合は、アドレスの設定はデバイスに合わせて変更してください。また、デバッグを行う場合、デバッガによってはソフトウェアブレークのために ROM の内容を書き換えるものがあるので、その場合は演算の対象領域をデバッグ領域以外に設定する必要があります。

以上の操作で、プロジェクトフォルダ下のビルド構成フォルダ内に **addcrc.srec**（プログラムコードの後ろに CRC 演算結果を付加した S レコードファイル）ができるので、これをターゲットボードにダウンロードします。

e² studio の「実行(Run)」⇒「デバッグの構成(Debug Configrations)」でデバッグ構成ダイアログを開き、

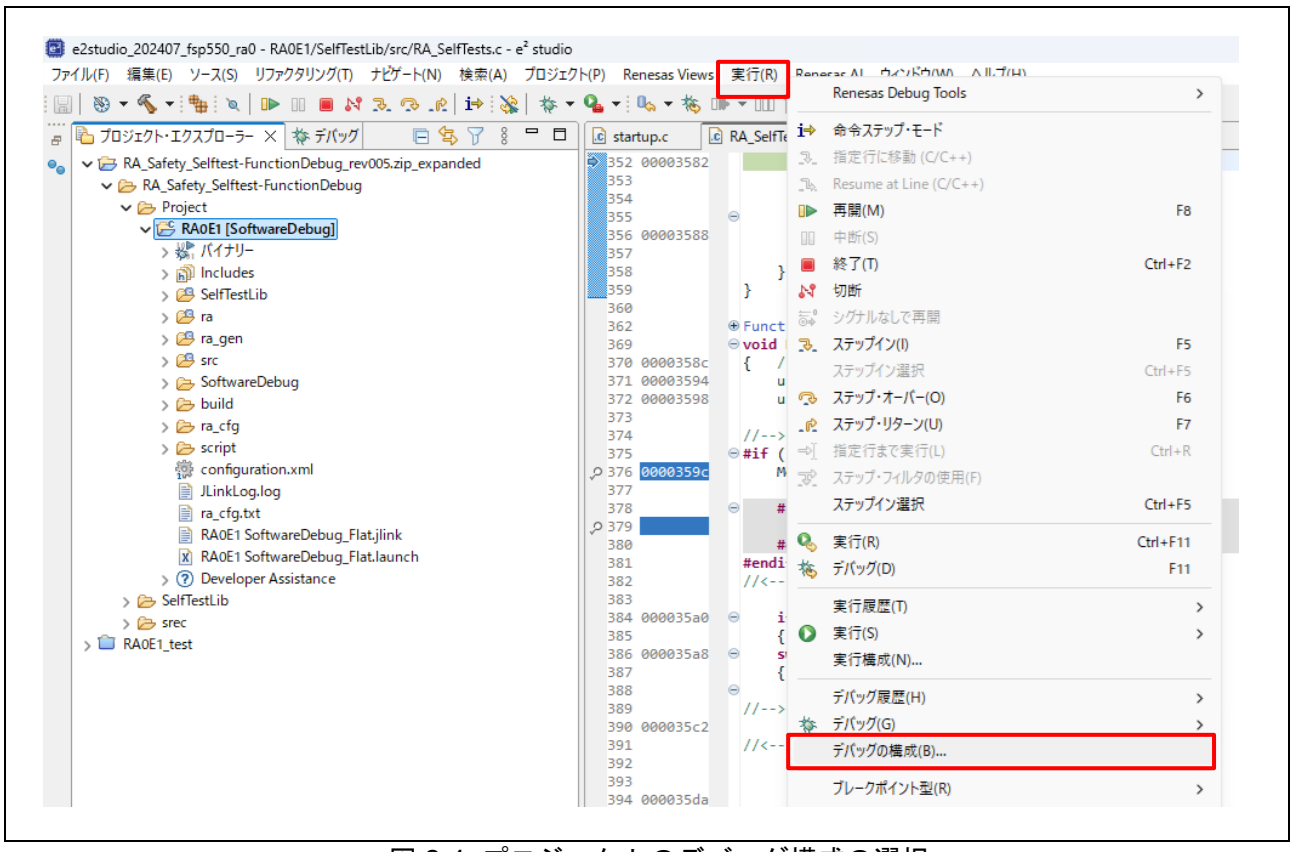

図 2-4　プロジェクトのデバッグ構成の選択

デバッグ構成のダイアログが表示されたら、Startup のタブを選び、使用するビルド構成を選択します。ELF ファイルからはシンボル情報だけを読み出し、addcrc.srec からは CRC 計算値を含むプログラムイメージを読み込むように設定します。

「デバッグ(Debug)」ボタンを押下すると CRC 演算値がターゲットにダウンロードされます。

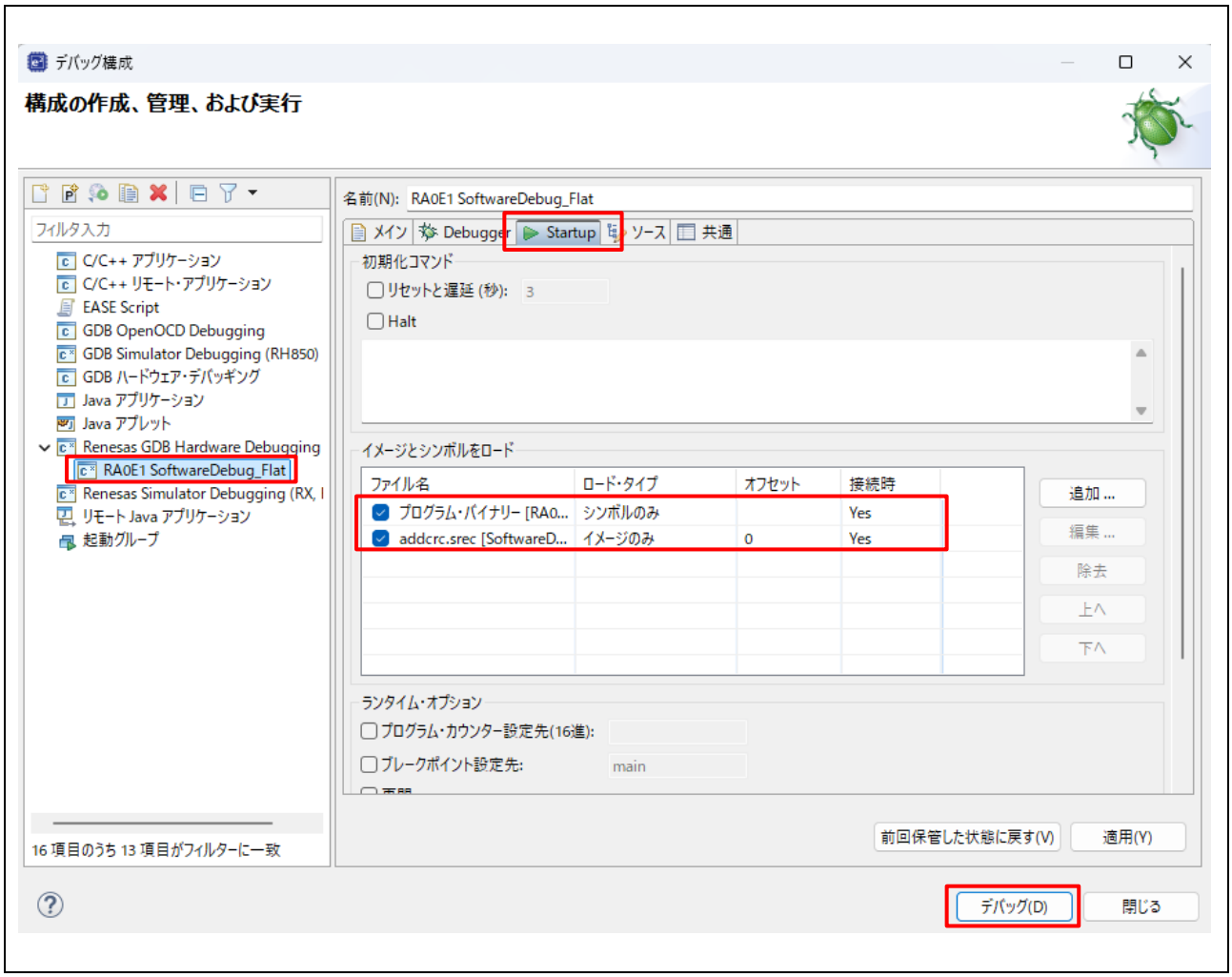

図 2-5　ロードイメージとシンボルの設定例

2.2.2　電源投入時

使用するすべての ROM メモリは、電源投入時にテストする必要があります。

この領域が 1 つの連続したブロックである場合、関数 CRC_Calculate を使用して、計算された CRC 値を計算して返すことができます。

使用する ROM が 1 つの連続したブロックにない場合は、次の手順を使用する必要があります。

1. CRC_Start を呼び出します。
2. CRC 計算に含めるメモリの各領域に対して CRC_AddRange を呼び出します。
3. CRC_Result を呼び出して、計算された CRC 値を取得します。

計算された CRC 値は、関数 CRC_Verify を使用して、ROM に格納されている参照 CRC 値と比較できます。

プロジェクトで使用されるすべての ROM 領域が CRC 計算に含まれるようにするのはユーザの責任です。

2.2.3　定期的

ROM が連続していても、CRC_AddRange を使用して ROM の定期的なテストを行うことをお勧めします。これにより、CRC 値をセクション単位で計算できるため、単一の関数呼び出しに時間がかかりすぎることはありません。電源投入テストで指定された手順に従い、各アドレス範囲が十分に小さいことを確認して、CRC_AddRange の呼び出しに時間がかかりすぎないようにします。

2.3　RAM

テストが必要な RAM の領域は、プロジェクトのメモリマップに応じて大きく変わる可能性があることを認識することが非常に重要です。

RAM をテストするときは、次の点に注意してください。

1. テスト中の RAM は、現在のスタックを含め、他の処理には使用しないでください。
2. 非破壊テストでは、テスト対象のメモリ領域を完全にコピーおよび復元できる RAM バッファが必要です。
3. スタックにはメインとプロセスの 2 つがあります。スタックテストでは、2 つのスタックのうちのいずれか、または両方をテストできます。スタックのテストには、対象のスタック領域を再配置できる RAM バッファが必要です。

※サンプルプロジェクトではプロセススタック領域はテスト未実施のため、非サポートです。

2.3.1　電源投入時

電源投入時に、スタック以外の RAM で完全な破壊テストを実行できます。スタックは非破壊テストでテストする必要があります。ただし、起動時間が非常に重要な場合は、これを微調整して、電源投入 RAM テストの前に使用されたスタックの領域のみが低速の非破壊テストを使用して実行され、残りのスタックが破壊テストされます。

2.3.2　定期的

すべての定期的なテストは非破壊的でなければなりません。定期的なテストは割り込みハンドラから呼び出されるため、デバイスは特権モードであること前提としてテストします。

2.4　クロック

システムクロックの監視のため、FSP の Configration 設定画面で TAU0 チャネル 5 に関連する設定を行っております。

下記” **Stacks Configuration** ”画面では、TAU0 チャネル 5 のキャプチャ機能に関する設定を確認できます。

操作手順としては、

- プロジェクト・エクスプローラーで赤枠部分の **configration.xlm** ファイルをダブルクリックする
- FSP Configuration 画面が表示 → 赤枠部分の”Stacks”タブを選択すると” **Stacks Configuration** ”画面が表示
- Thread で”g_timer05 Timer, Independent Channel, … ”を選択すると TAU0 ch5 が選択される。
- ”プロパティ”タブを選択する

下図のように” **Stacks Configuration** ”画面の設定内容(”Settings”)が表示されます。

続いて、“Commonl”, “Module g_timer05 Timer …”の”General”, “Input”を選択すると詳細な設定内容が表示されます。

赤い点線枠が主な TAU0 ch5 の設定項目となります。

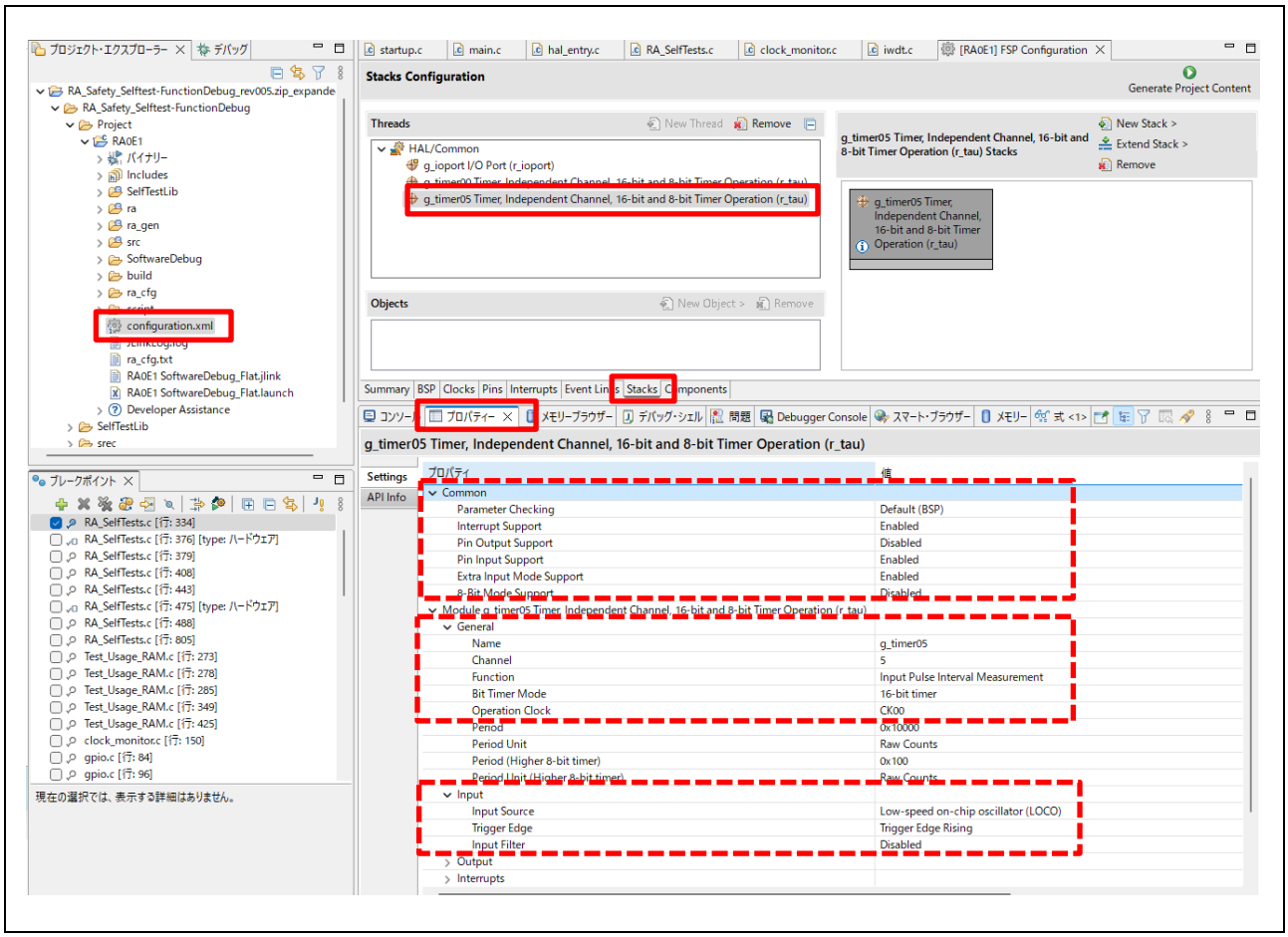

図 2-6　FSP　Configuration 設定画面（Stacks Configuration タブ：TAU0 ch5）

なお、TAU0 ch5 の詳細な仕様については、ユーザーズマニュアル　ハードウエア編の「タイマアレイユニット(TAU)」章を参照ください。

次に下記” Interrupts Configuration ”タブでは、TAU0 チャネル 5 のキャプチャ割込みに割り当てられた関数のユーザ名やイベント番号を確認できます。

なお、割り込みの詳細な仕様については、ユーザーズマニュアル　ハードウエア編の「割り込みコントローラユニット(ICU)」章を参照ください。

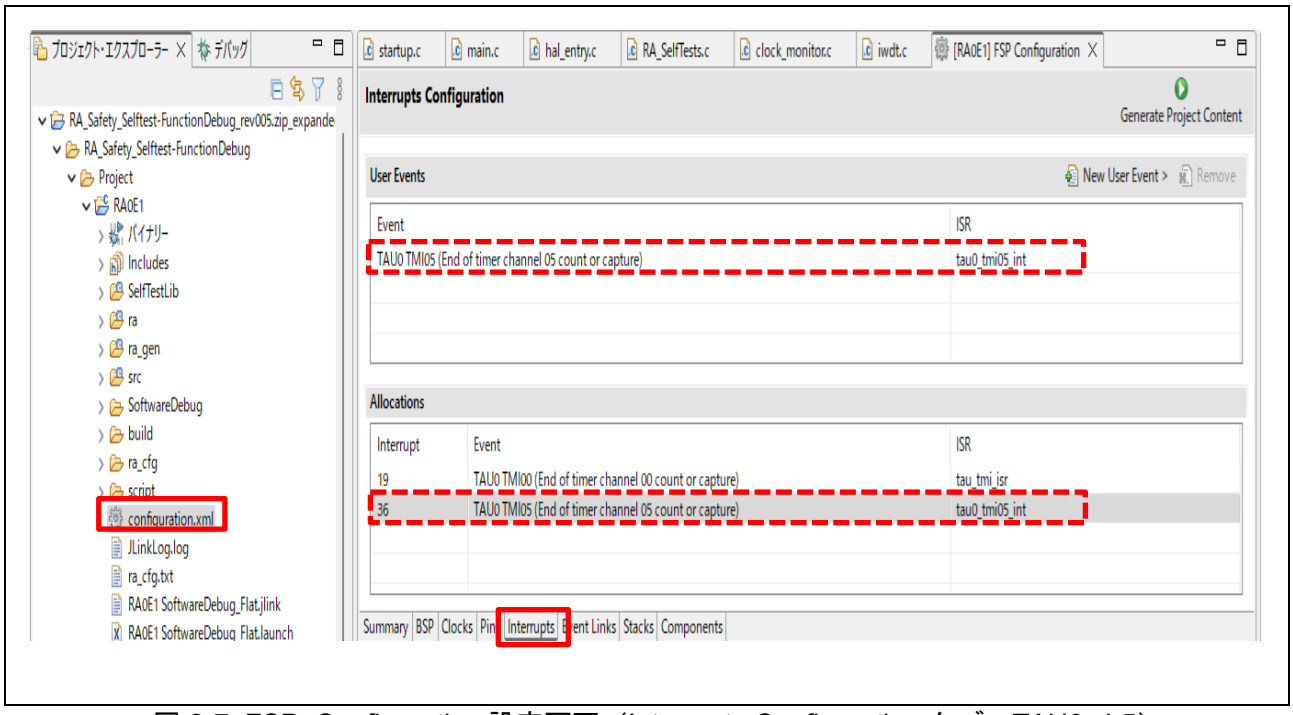

図 2-7　FSP　Configuration 設定画面（Interrupts Configuration タブ：TAU0 ch5）

上記の設定内容が “ClockMonitor_Init”関数の呼び出しで関連レジスタへ反映されます。

[記述例]

```
    // TAU0 ch5 initial setting
    // In RA0, reflect the Stack setting in FSP Configuration to each TAU0 ch5 related register.
    // At the same time, set the error functions.
    ClockMonitor_Init ( Clock_Test_Failure );
```

本サンプルプロジェクトでは、この関数内で TAU0 ch5 関連レジスタへ FSP Configuration 設定内容が反映されます。

また、クロック周波数の許容範囲は clock_monitor.h の以下で定義しております。

本サンプルソフトでは TAU0 ch5 のタイマ入力に「LOCO」、カウンタ動作クロックに「PCLKB」(ICLK=HOCO 選択)を設定しております。
なお、上記の対象クロックを変更した場合は、以下の定義を選択クロックに合わせて変更してください。

※以下の青色の点線枠部分が本サンプルソフトでの許容範囲の上限、下限値、緑文字部分は補足コメント。

["clock_monitor.h"内での定義箇所]

```
/* Hardware clock test reference values */
/*;hwMAXTIME= HOCO_FREQUENCY_MAX / LOCO_FREQUENCY_MIN */
/*;hwMINTIME= HOCO_FREQUENCY_MIN / LOCO_FREQUENCY_MAX
/*
;HOCO_FREQUENCY_MIN: Value considering -1% error of HOCO frequency (31.68MHz)
;LOCO_FREQUENCY_MAX: Value considering +15% error of low-speed on-chip
oscillator frequency (37.68KHz)
;HOCO_FREQUENCY_MAX: Value considering +1% error of HOCO frequency (32.32MHz)
;LOCO_FREQUENCY_MIN: Value considering -15% error of low-speed on-chip
oscillator frequency (27.85KHz)
;Change the lower and upper limits of the allowable range for pulse interval
measurement as appropriate depending on the system.
*/
// egde trig clock ; LOCO , count clock : PCLKB

#define hwMAXTIME      (0x0489)          // 1161
#define hwMINTIME      (0x0349)          //  841
```

2.5　独立ウォッチドッグタイマ（IWDT）

独立ウォッチドッグタイマを構成するには、オプション設定メモリの OFS0 レジスタを設定する必要があります。例えば、オプション設定メモリを以下のように設定するとします。

表 2-1 OFS0 レジスタの設定例（IWDT 関連）

項目	OFS0 レジスタの設定値（例）
IWDT スタートモード（IWDTSTRT）	**0: リセット後、IWDT は自動的に起動（オートスタートモード）**
IWDT タイムアウト期間選択（IWDTTOPS[1:0]）	11b：2048 サイクル
IWDT 専用クロック分周比（IWDTCKS[3:0]）	1111b：128 分周
IWDT ウィンドウ終了位置（IWDTRPES[1:0]）	11b：0%（ウィンドウの終了位置設定なし）
IWDT ウィンドウ開始位置（IWDTRPSS[1:0]）	11b：100%（ウィンドウの開始位置設定なし）
IWDT リセット割り込み要求（IWDTRSTIRQS）	0：ノンマスカブル割り込み要求、または割り込み要求を許可
IWDT 停止制御（IWDTSTPCTL）	1：スリープモード、スヌーズモード、またはソフトウェアスタンバイモードの状態にあるとき、カウント停止

e² studio で FSP (Flexible Software Package) を利用する場合、FSP の「BSP」タブのプロパティで、オプション設定メモリの設定ができます。

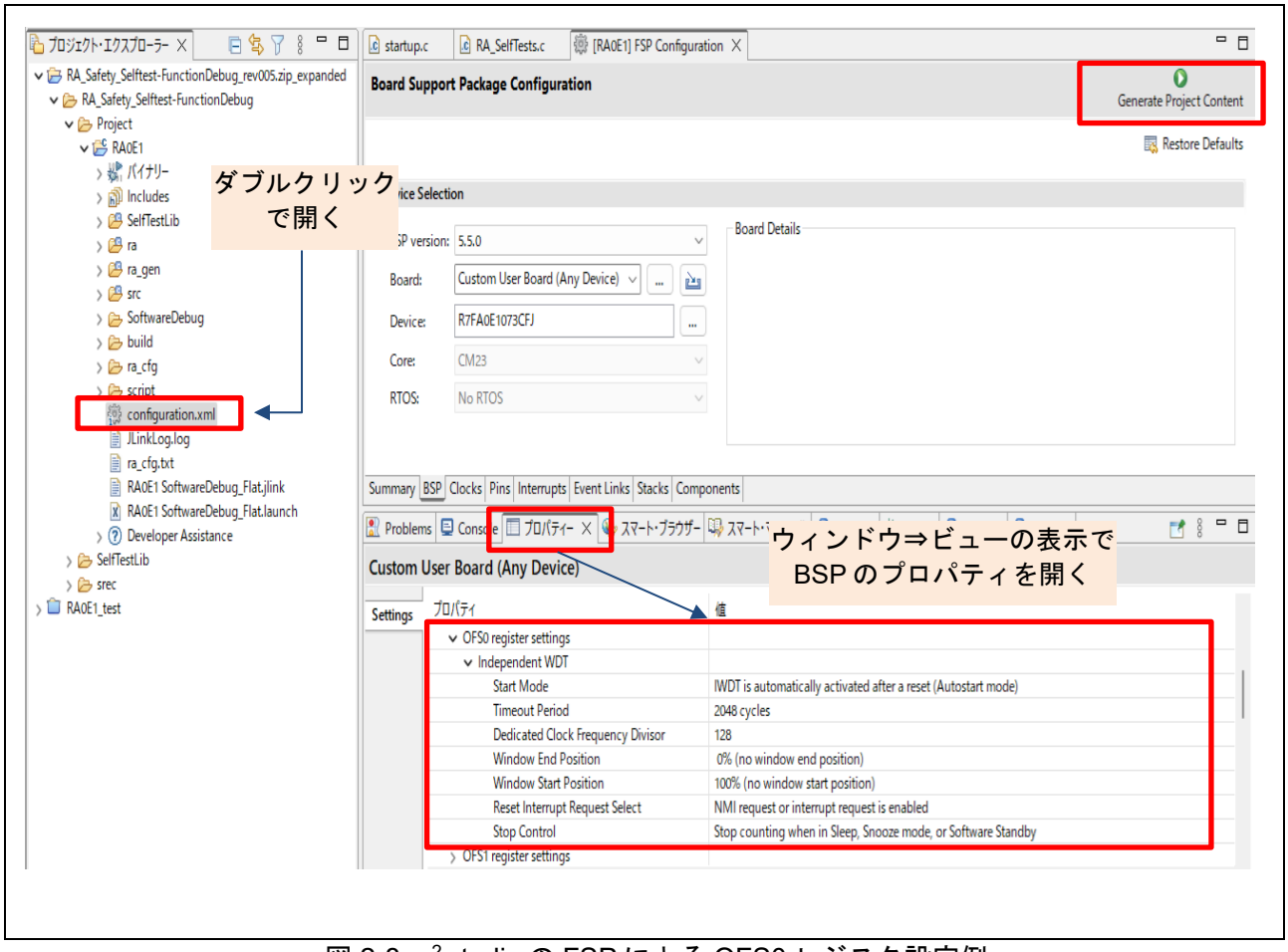

図 2-8　e² studio の FSP による OFS0 レジスタ設定例

「Generate Project Content」ボタンを押すと、プロパティでの設定内容が、下記ファイルの該当シンボルの定義に反映されます。

- 該当ファイル

  ..*project-name*\ra_cfg\fsp_cfg\bsp**bsp_mcu_family_cfg.h**

- 該当シンボル部分（抜粋）

```
#define OFS_SEQ1 0xA001A001 | (0 << 1) | (3 << 2)
#define OFS_SEQ2 (15 << 4) | (3 << 8) | (3 << 10)
#define OFS_SEQ3 (0 << 12) | (1 << 14) | (1 << 17)
  :    :
```

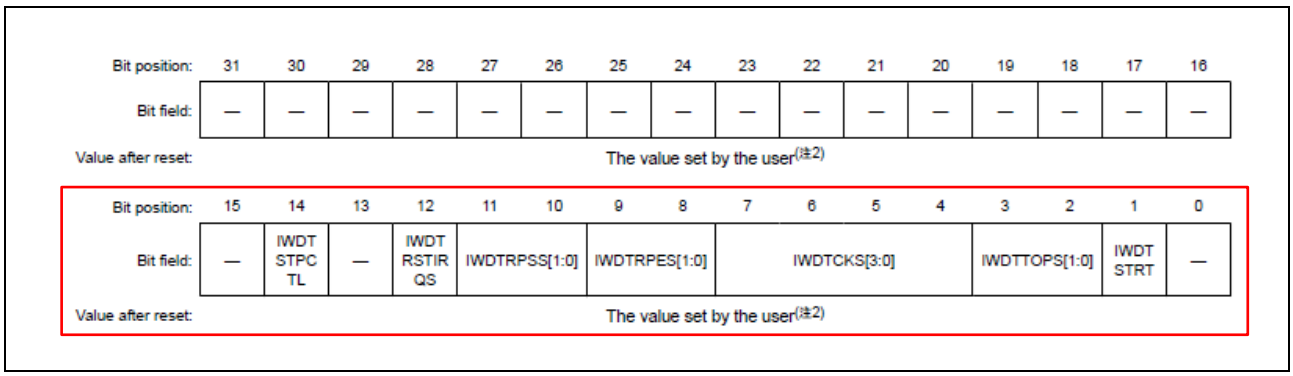

図 2-9 オプション機能選択レジスタ 0（OFS0）

IWDT の詳細につきましては、RA MCU のハードウェアユーザーズマニュアル「**26. 独立ウォッチドッグタイマ (IWDT)**」を参照ください。

独立ウォッチドッグタイマは、IWDT_Init を呼び出して、リセット後できるだけ早く初期化する必要があります。

```
/* Setup the Independent WDT. */
IWDT_Init();
```

この後、ウォッチドッグタイマは、ウォッチドッグタイマがタイムアウトしてリセットが実行されるのを防ぐために、定期的にリフレッシュする必要があります。ウィンドウ処理を使用する場合、リフレッシュは単に定期的であるだけでなく、指定されたウィンドウに一致するように時間を調整する必要があります。ウォッチドッグタイマの更新は以下で行います。

```
/* Regularly kick the watchdog to prevent it performing a reset. */
IWDT_Kick();
```

ウォッチドッグタイマがエラー検出時に NMI を生成するように設定されている場合、ユーザはその結果の割り込みを処理する必要があります。

2.6　ADC

本 ADC(ADC12)には、＋側基準電圧、－側基準電圧、内部基準電圧をテスト用 A/D 変換に選択する機能があります

以下の各定義シンボルを使用して、定義された許容範囲(青色背景色)内に収まるか ADC モジュールをテストします。

※システムに合わせてユーザが各定義を変更し使用してください。

校正済みのシステムでは、この許容誤差を厳しくすることができます。

[“test_adc.h”内での定義箇所]

```
// See "31.6.1 "A/D conversion characteristics" on UMH of RA0E1
#define OVERALL_ERROR_LSB_UNIT (9)                              /* AD Converter Overall error */
#define VSS_RANGE_MAX          (0 + OVERALL_ERROR_LSB_UNIT)     /* Vss Range Max */
#define VSS_RANGE_MIN          (0x0000)                         /* Vss Range Min */
#define AD_RESOLUTION_HEX      (0x0FFF)                         /* AD Converter Resolution */
#define VDD_RANGE_MIN          (AD_RESOLUTION_HEX - OVERALL_ERROR_LSB_UNIT)  /* Vdd Range Min */
#define VDD_RANGE_MAX          (AD_RESOLUTION_HEX)              /* Vdd Range Max */

/* Please change it according to the VDD voltage. */
#define VDD           (3.3)       /* Vdd */
//#define VDD           (5.0)       /* Vdd */
#define VBGR_MIN      (1.4)       /* Vbgr Min */
#define VBGR_MAX      (1.56)      /* Vbgr Max */
#define VBGR_RANGE_MIN ((((VBGR_MIN * AD_RESOLUTION_HEX)/VDD)) - OVERALL_ERROR_LSB_UNIT) /* Vdd
Range Min */
#define VBGR_RANGE_MAX ((((VBGR_MAX * AD_RESOLUTION_HEX)/VDD)) + OVERALL_ERROR_LSB_UNIT) /* Vdd
Range Max */
```

なお、上記の誤差や内部基準電圧の詳細な値については、ユーザーズマニュアル　ハードウエア編　「アナログ特性」を参照ください。

なお、本 ADC テストを実施する際は、Test_ADC_Init を呼び出して初期化する必要があります。

2.6.1　電源投入時

電源投入時に、Test_ADC_Wait 関数を使用して ADC モジュールをテストできます。

この関数では、－側基準電圧、＋側基準電圧のいずれかの AD 変換が実行され、AD 変換が実行されるまでの間はこの関数内で待機します。

この関数の戻り値で結果を確認する必要があります。

2.6.2　定期的

定期的なテストは、Test_ADC_Start の呼び出しで AD 変換を実施します。

この関数が呼び出されると ADC モジュールは、テスト用 A/D 変換対象を切替えます。

※本 ADC モジュールは、自動ローテーションは非搭載です。

従って、Test_ADC_Start の呼び出した後に Test_ADC_CheckResult 呼び出して AD 変換結果をチェックする必要があります。

2.7　GPIO

　各デバイスでテスト対象とするポートの指定は、gpio_config.h ヘッダファイル内の g_GPIO_Test_Port 構造体で（ポート m，ビット n のように）定義します。

ポートの指定例：

```
static const struct {
uint16_t m;
uint16_t n;
} g_GPIO_Test_Port[GPIO_TEST_NUM] =
{
{ 0 , 12 } ,    // PORT012
{ 1 ,  0 } ,    // PORT100
{ 2 ,  1 }      // PORT201
};
```

ウェブサイトとサポート

RA MCU に関する情報や、ツール、ドキュメントのダウンロード、技術サポートなどは、下記の各ウェブサイトを通じて利用できます。

- RA 製品情報：　www.renesas.com/ra
- RA FSP (Flexible Software Package)：　www.renesas.com/FSP
- RA サポートフォーラム：　www.renesas.com/ra/forum
- Renesas サポート：　www.renesas.com/support

改訂履歴

Rev.	発行日	説明	
		ページ	ポイント
1.00	2025.6.30	ー	初版

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

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生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。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. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
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。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。