User's Manual

RL78 ファミリ

EEPROM エミュレーション・ソフトウェア

RL78 Type01

ユーザーズマニュアル

ルネサスマイクロコンピュータ

RL78 / G2x

ルネサス エレクトロニクス
www.renesas.com

Rev.1.20 2023.09

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。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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。
13. 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。
14. 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.5.0-1 2020.10)

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。

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。

このマニュアルの使い方

対 象 者　このユーザーズマニュアルは、RL78/G2x マイクロコントローラの EEPROM エミュレーションの機能を理解し、それを用いたアプリケーション・システムを設計するユーザを対象としています。

目　　的　このユーザーズマニュアルは、RL78/G2x マイクロコントローラのデータ・フラッシュ・メモリに EEPROM エミュレーション・ソフトウェア(EES)でデータを格納する方法（アプリケーションによる定数データ書き込み）をユーザに理解していただくことを目的としています。

構　　成　このマニュアルは、大きく分けて次の内容で構成しています。

- 概要
- システム構成
- EEPROM エミュレーション
- EEPROM エミュレーションの使用方法
- ユーザインタフェース
- サンプル・プログラム
- サンプル・プロジェクトの作成

読 み 方　このマニュアルを読むにあたっては、電気、論理回路、マイクロコントローラ、C 言語とアセンブラの一般的な知識を持つことを想定しています。
RL78/G2x のハードウェア機能を理解するために、それぞれの RL78/G2x 製品のユーザーズマニュアルを参照してください。

凡　　例　データ表記の重み　：左が上位桁、右が下位桁
アクティブ・ロウの表記　：$\overline{\times\times\times}$（端子、信号名称に上線）
注　：本文中につけた注の説明
注意　：気をつけて読んでいただきたい内容
備考　：本文の補足説明
数の表記　：2 進数…××××または××××B
10 進数…xxxx
16 進数…XXXXH または 0xXXXX
2 の累乗を示す単位の表記（アドレス空間、メモリ容量）
：K (キロ) 2^{10} = 1024
M (メガ) 2^{20} = 1024^2

関連資料　関連資料は暫定版の場合がありますが、この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

No	Document Title	Document Number
1	RL78/G23 ユーザーズマニュアル ハードウェア編	R01UH0896JJ
2	RL78/G22 ユーザーズマニュアル ハードウェア編	R01UH0978JJ
3	RL78/G24 ユーザーズマニュアル ハードウェア編	R01UH0961JJ
4	RL78ファミリ Renesas Flash Driver RL78 Type01 ユーザーズマニュアル	R20UT4830JJ
5	E1/E20/E2エミュレータ, E2エミュレータLite ユーザーズマニュアル別冊 (RL78接続時の注意事項)	R20UT1994JJ
6	RL78用 Renesas Flash Driver, EEPROM Emulation Software 対象MCUリスト - General-Purpose	R20UT5228JJ

目　次

略語

用語	説明
EES	EEPROM Emulation Software（EEPROM エミュレーション・ソフトウェア）
RFD	Renesas Flash Driver（ルネサス・フラッシュ・ドライバ）
API	Application Program Interface（アプリケーション・プログラム・インタフェース）
BGO	Background operation データ・フラッシュ書き換え中に、プログラム・メモリ内の命令実行が可能。
RAM	Random Access Memory ランダムにアクセスできる揮発性メモリ。プログラム実行中に変更する値を保持するメモリです。
ROM	Read Only Memory 不揮発性メモリ。内容変更することができないメモリです。コード・フラッシュ・メモリを ROM と表現する場合があります。

専門用語

用語	説明
コード・フラッシュ・メモリ	アプリケーション・コードや定数データを格納するフラッシュ・メモリ ※本文中で、”CF”と略す場合があります。
データ・フラッシュ・メモリ	データを格納するフラッシュ・メモリ ※本文中で、”DF”と略す場合があります。
エクストラ領域	コンフィギュレーション設定領域、セキュリティ設定領域、ブロック保護設定領域、ブート・スワップ設定領域の総称
フラッシュ・メモリ・シーケンサ	RL78マイクロコントローラにはフラッシュ・メモリ制御用の専用回路が搭載されています。本書ではこの回路のことをフラッシュ・メモリ・シーケンサと呼びます。フラッシュ・メモリ・シーケンサに、コード・フラッシュ領域、またはデータ・フラッシュ領域を書き換える「コード/データ・フラッシュ領域シーケンサ」とエクストラ領域を書き換える「エクストラ領域シーケンサ」の総称です。
フラッシュ・メモリ制御モード	フラッシュ・メモリ・シーケンサの書き換え可否状態(モード)を示します。 - コード・フラッシュ・プログラミング・モード(Code flash programming mode) - データ・フラッシュ・プログラミング・モード(Data flash programming mode) - 非書き換えモード(Non-programmable mode)
コード・フラッシュ・プログラミング・モード	Code flash programming mode コード・フラッシュ・メモリ(および、エクストラ領域)を書き換え可能な状態(モード)を指します。
データ・フラッシュ・プログラミング・モード	Data flash programming mode データ・フラッシュ・メモリを書き換え可能な状態(モード)を指します。
非書き換えモード	Non-programmable mode フラッシュ・メモリ(および エクストラ領域)を書き換え不可の状態(モード)を指します。
セルフ・プログラミング	外部のフラッシュ・プログラミング・ツールを使用せず、ユーザ・プログラムを実行して、フラッシュ・メモリの書き換えを行うこと。
RFD 関数	RFD が提供する関数の総称です。
EES 関数	EES が提供する関数の総称です。
EES 用 RFD 制御関数	EES が提供する RFD を制御する関数の総称です。
EES ブロック	EEPROM エミュレーション・ソフトウェアがアクセスするブロックの略称です。なお、以降、本ユーザーズマニュアル内では EEPROM エミュレーション・ソフトウェア・ブロックを EES ブロックと呼称します。

1 概要

1.1 製品概要

EEPROM エミュレーションとは、マイコンに搭載されているデータ・フラッシュ・メモリへ EEPROM のようにデータを格納させるための機能です。EEPROM エミュレーションでは、EEPROM Emulation Software (EES) RL78 Type01 から Renesas Flash Driver (RFD) RL78 Type01 を操作して、データ・フラッシュ・メモリへの書き込みや読み出しを実行します。

EES RL78 Type01 は、ユーザ・プログラムにより RL78/G2x に搭載されているデータ・フラッシュ・メモリ内のデータを書き換えるためのソフトウェアです。

RL78/G2x 用の Renesas Flash Driver (RFD) RL78 Type01 については、RL78 ファミリ Renesas Flash Driver RL78 Type01 ユーザーズマニュアルを参照してください。

1.1.1 目的

このユーザーズマニュアルは、RL78/G2x マイクロコントローラのデータ・フラッシュ・メモリに EEPROM エミュレーション・ソフトウェア(EES)でデータを格納する方法（アプリケーションによる定数データ書き込み）をユーザに理解していただくことを目的としています。

1.2 製品内容

EES RL78 Type01 の API 関数をユーザ・プログラムから呼び出すことにより、データ・フラッシュ・メモリへ配置した EEPROM エミュレーション・ブロック（EES ブロック）の内容を書き換えることができます。

EES RL78 Type01 は、以下を含んでいます。

・本ユーザーズマニュアル
・RL78/G2x のデータ・フラッシュ・メモリを操作する EES のソース・コード・ファイル。
・EES を操作するためのサンプル・プログラム。

1.3 製品の特長

EES RL78 Type01 は、フラッシュ・メモリ・シーケンサを操作する RFD RL78 Type01 の API 関数を呼び出します。EES RL78 Type01 の API 関数は、1 つ、もしくは複数の関数で構成されており、各関数とユーザで行う処理を組み合わせて実現します。これは、ユーザ・アプリケーションに依存する処理、例えばタイムアウト処理のように、タイムアウト値がユーザ・アプリケーション・プログラムの実行条件によって異なるケースがあり、柔軟に対応できるよう、このような構成を採用しています。

ユーザ・アプリケーションが、EES RL78 Type01 の API 関数を使ってデータ・フラッシュ・メモリを操作するときのイメージを図 1-1 に示します。

EES RL78 Type01 では、複数の API 関数とユーザ・プログラムで行うべき処理を組み合わせた処理の例をサンプル・プログラムとして提供しています。EEPROM エミュレーションの処理をアプリケーションに組み込む際は、このサンプル・プログラムを参考にしてください。

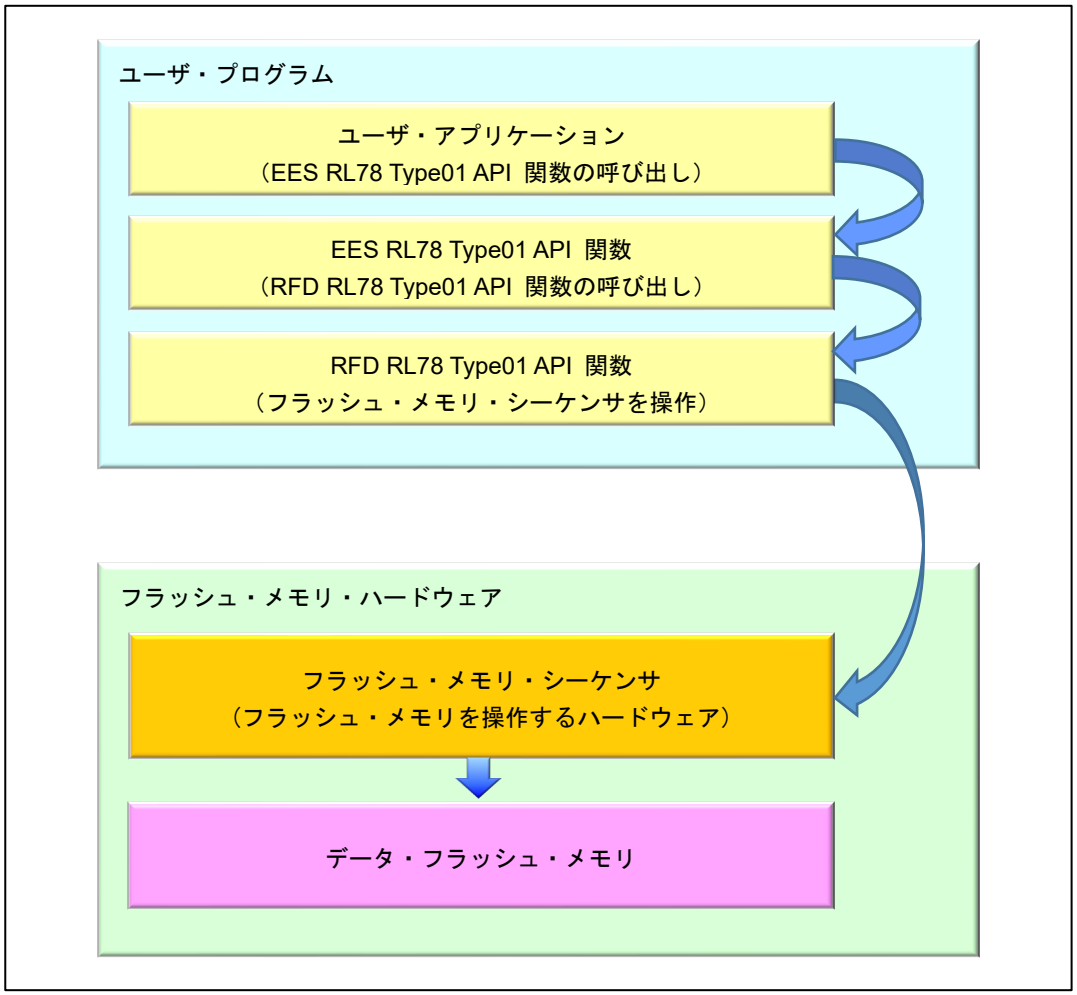


図 1-1 EES RL78 Type01 の API 関数を使ったデータ・フラッシュ・メモリの操作イメージ

1.4 動作環境

- ホスト・マシン

動作環境は、ホスト・マシンには依存しませんが、C コンパイラ・パッケージ、デバッガ、およびエミュレータが動作する環境が必要となります。（EES RL78 Type01 の開発は Windows10 Enterprise で実施）

- C コンパイラ・パッケージ

EES RL78 Type01 の対象の C コンパイラ・パッケージを表 1-1 に示します。

表 1-1 対象の C コンパイラ・パッケージ

パッケージ	統合開発環境	メーカー	バージョン
CC-RL コンパイラ	CS+, e2 studio	Renesas Electronics	V1.10 以降
IAR コンパイラ	IAR Embedded Workbench® for Renesas RL78	IAR システムズ®	V4.21 以降
LLVM コンパイラ	e2 studio	（オープンソース ソフトウェア）	V10.0.0.202306 以降

注）統合開発環境、およびコンパイラは、対象デバイスをサポートしている必要があります。

- エミュレータ

動作確認したエミュレータを表 1-2 に示します。

表 1-2 動作確認したエミュレータ

エミュレータ	メーカー
E2 エミュレータ	Renesas Electronics
E2 エミュレータ Lite	Renesas Electronics

- ターゲット MCU

RL78/G23, RL78/G22, RL78/G24

- EEPROM エミュレーション・ソフトウェア (EES)

本資料に対応している EEPROM エミュレーション・ソフトウェア(EES)を表 1-3 に示します。

表 1-3 EEPROM エミュレーション・ソフトウェア (EES)

パッケージ	メーカー	パッケージ・バージョン
EES RL78 Type01	Renesas Electronics	Ver 1.20

注）表 1-4 に記載されているバージョンの RFD RL78 Type01 をご使用ください。

- ルネサス・フラッシュ・ドライバ (RFD)

動作確認に使用したルネサス・フラッシュ・ドライバ(RFD)を表 1-4 に示します。

表 1-4 使用したルネサス・フラッシュ・ドライバ (RFD)

パッケージ	メーカー	パッケージ・バージョン
RFD RL78 Type01	Renesas Electronics	Ver 1.20

1.5 注意事項

EEPROM エミュレーションは、RL78/G2x に搭載しているデータ・フラッシュ・メモリを操作する機能を使用して実現しています。このため、次の点に注意する必要があります。

(1) 全ての EES のコードと定数は、同一の 64KB のフラッシュ・ブロック内に配置する必要があります。
（コンパイラに依存します。）

(2) R_EES_Init 関数による EES の初期化は、すべての EES 関数群を実行する前に行う必要があります。

(3) EES によるデータ・フラッシュ・メモリ操作中はデータ・フラッシュ・メモリを読み出せません。

(4) EES のコマンド実行中は RFD 関数を呼び出さないでください。

(5) EES 以外からは、直接 EES 用 RFD 制御関数を呼び出さないでください。

(6) EEPROM エミュレーション実行中に STOP 命令、および HALT 命令は実行しないでください。STOP 命令、および HALT 命令を実行する必要がある場合は必ず R_EES_Close 関数まで実行し、EEPROM エミュレーションを終了させてください。

(7) ウォッチドック・タイマは EES 実行中も停止しません。

(8) コマンド実行中はリクエスト・ストラクチャー(st_ees_request_t)を破壊しないでください。

(9) EEPROM エミュレーション・ソフトウェアで使用する引数(RAM)は一度初期化してください。初期化をしない場合、RAM パリティ・エラーが検出され、RL78/G2x にリセットが発生する可能性があります。 RAM パリティ・エラーについては、対象デバイスの「ユーザーズマニュアル：ハードウェア編」を参照してください。

(10) EES のコマンドを実行する前に、リクエスト・ストラクチャー(st_ees_request_t)のすべてのメンバは一度初期化する必要があります。リクエスト・ストラクチャー(st_ees_request_t)内に使用されないメンバがある場合には、そのメンバに任意の値を設定してください。初期化されない場合、RAM パリティ・エラーによって RL78/G2x にリセットが発生する可能性があります。詳細については対象デバイスの「ユーザーズマニュアル：ハードウェア編」を参照してください。

(11) EES は多重実行に対応していないため、EES 関数を割り込み処理内で実行しないで下さい。

(12) R_EES_Close 関数の実行後は、要求済みのコマンド、および実行中のコマンドは停止し、それを再開することはできません。R_EES_Close 関数を呼び出す前に、実行中のコマンドをすべて終了させてください。

(13) EEPROM エミュレーション実行中に、RFD RL78 Type01 でコード・フラッシュ・メモリの操作を実行しないでください。使用する場合は必ず R_EES_Close 関数まで実行し、EEPROM エミュレーションを終了状態にする必要があります。RFD RL78 Type01 でコード・フラッシュ・メモリの操作を実行後に EEPROM エミュレーションを使用する場合は、初期化関数(R_EES_Init 関数)から処理を行う必要があります。

(14) EEPROM エミュレーションを開始する前に高速オンチップ・オシレータを起動しておく必要があります。また、外部クロックを使用時も、高速オンチップ・オシレータは起動しておく必要があります。

(15) EES ではユーザ・データにチェックサムを追加しません。チェックサムが必要な場合、ユーザ・データにチェックサムを付加して判定する等ユーザ・プログラムで対応してください。

(16) EES の実行中に DFLCTL（データ・フラッシュ・コントロール・レジスタ）を操作しないでください。

(17) データ・フラッシュ・メモリを EEPROM エミュレーションで使用するためには初回起動時に R_EES_ENUM_CMD_FORMAT コマンドを実行し、データ・フラッシュ・メモリを EES ブロックとして使用できるように初期化をおこなう必要があります。

(18) EES を使用するためには、EES ブロック（仮想ブロック）に 3 ブロック以上を設定することを推奨します。

(19) EES 以外の RFD RL78 Type01 を使用したデータ・フラッシュを操作するユーザ・プログラム等で EES ブロックを破壊しないください。

(20) EES ディスクリプタが変更された場合、EEPROM エミュレーションの実行を継続することはできません。このような場合には、R_EES_CMD_FORMAT コマンドによる EES プールのフォーマットを行う必要があります。ただし、データの追加の場合は、実行を継続することができます。

(21) RL78/G2x の CPU の動作周波数と初期化関数（R_EES_Init 関数）で設定する CPU の動作周波数値について、以下の点に注意してください。

- RL78/G2x の CPU の動作周波数として 4MHz 未満の周波数を使用する場合は、1MHz、2MHz、3MHz のみを使用することができます（1.5 MHz のように整数値にならない周波数は使用できません）。
- RL78/G2x の CPU の動作周波数として 4 MHz 以上[注]の周波数を使用する場合は、RL78/G2x に任意の周波数を使用することができます。
- 高速オンチップ・オシレータの動作周波数ではありません。

注）最大周波数については、対象デバイスの「ユーザーズマニュアル：ハードウェア編」を参照してください。

1.6 C コンパイラ定義

EES RL78 Type01 のヘッダ・ファイル(r_ees_compiler.h)に記述する対象コンパイラの定義を示します。

コンパイラごとに異なる記述が必要なため、使用しているコンパイラを” r_ees_compiler.h”ファイルで判別し、対象のコンパイラ用の定義を使用します。

・C コンパイラの定義

- CC-RL コンパイラの定義 :

“__CCRL__”が定義されている場合

#define EES_COMPILER_CC (1)

- IAR コンパイラ(V2/V3/V4) :

“__IAR_SYSTEMS_ICC__”が定義されている場合

#define EES_COMPILER_IAR (2)

- LLVM コンパイラ:

“__llvm__”が定義されている場合

#define EES_COMPILER_LLVM (3)

<r_ees_compiler.h ファイル内の記述>

```
/* Compiler definition */
#define EES_COMPILER_CC      (1)
#define EES_COMPILER_IAR     (2)
#define EES_COMPILER_LLVM    (3)

#if defined (__llvm__)
    #define EES_COMPILER EES_COMPILER_LLVM
#elif defined (__CCRL__)
    #define EES_COMPILER EES_COMPILER_CC
#elif defined (__IAR_SYSTEMS_ICC__)
    #define EES_COMPILER EES_COMPILER_IAR
#else
    /* Unknown compiler error */
    #error   "Non-supported compiler."
#endif

/* Compiler dependent definition */
#if (EES_COMPILER_CC == EES_COMPILER)
    #define R_EES_FAR_FUNC                  __far
#elif (EES_COMPILER_IAR == EES_COMPILER)
    #define R_EES_FAR_FUNC                  __far_func
#elif (EES_COMPILER_LLVM == EES_COMPILER)
    #define R_EES_FAR_FUNC                  __far
#else
    /* Unknown compiler error */
    #error   "Non-supported compiler."
#endif
```

・C コンパイラ・オプション

以下に、動作確認した C コンパイラ・オプションを示します。

- [CC-RL(CS+)]

Major compile options:

-cpu=S3 -g -g_line -lang=c99

- [IAR(Embedded Workbench)]

Major compile options:

--core s3 –calling_convention v2 –code_model far –data_model near -e -Ol –no_cse –no_unroll –no_inline

--no_code_motion –no_tbaa –no_cross_call –no_scheduling –no_clustering –debug

- [LLVM(e^2 studio)]

Major compile options:

-Og -ffunction-sections -fdata-sections -fdiagnostics-parseable-fixits -Wunused -Wuninitialized -Wall

-Wmissing-declarations -Wconversion -Wpointer-arith -Wshadow -Waggregate-return -g -mcpu=s3

2 システム構成

2.1 システム構成

EESは、ユーザが定義するデータ・フラッシュ領域（EESプール）にアクセスするためのインタフェースです。

EES では、EES で提供している API 関数から EES 用 RFD 制御関数、および RFD を介して EES プールにアクセスします。「図 2-1 システム構成」の矢印が操作の流れです。

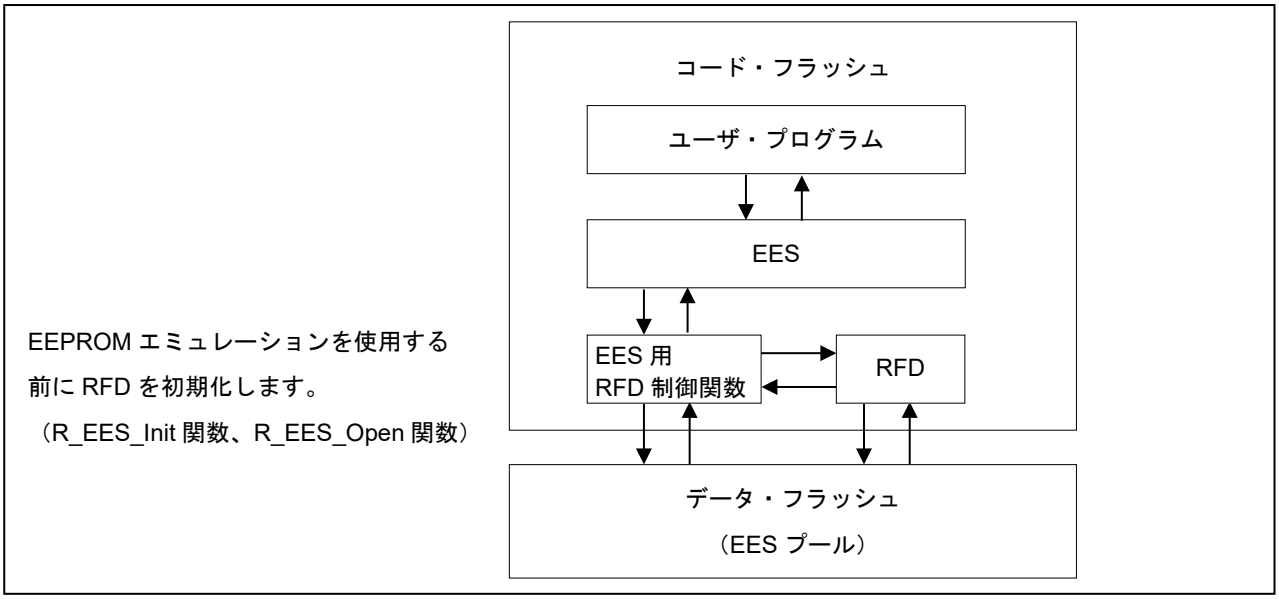


図 2-1 システム構成

2.2 EES アーキテクチャ

この章では、ユーザが EES を使用してデータ・フラッシュ・メモリ（EES プール）の書き換えを行う上で必要な EES のアーキテクチャについて説明します。

2.2.1 EES ブロック

EES では、複数のデータ・フラッシュ・メモリのブロックを 1 つの仮想ブロックとして使用します。この仮想ブロックを EES ブロックと呼びます。

RL78/G2x では、データ・フラッシュ・メモリの 1 ブロックのサイズが 256 バイトのため、EES ブロックサイズに 1K バイトを設定する場合は、データ・フラッシュ・メモリの 4 ブロックを 1K バイトの仮想ブロックとして扱います。また、EES ブロックサイズに 2K バイトを設定する場合は、データ・フラッシュ・メモリの 8 ブロックを 2K バイトの仮想ブロックとして扱います。必ず対象デバイスに搭載されているデータ・フラッシュ・メモリのサイズ、および総ブロック数の値を考慮し、EES ブロックのサイズを設定してください。なお、設定方法につきましては、「4.2 ユーザ設定初期値」をご参照ください。EES ブロックに 1K バイト、および 2K バイトを設定した時の EES ブロック 0 の概念図を「図 2-2 EES ブロック 0 の概念図」に示します。

なお、EES ブロックに設定できる最大ブロック数は、データ・フラッシュ・メモリを 8K バイト搭載している製品の場合、EES ブロックに 1K バイトを設定すると 8 ブロック。EES ブロックに 2K バイトを設定すると 4 ブロックです。

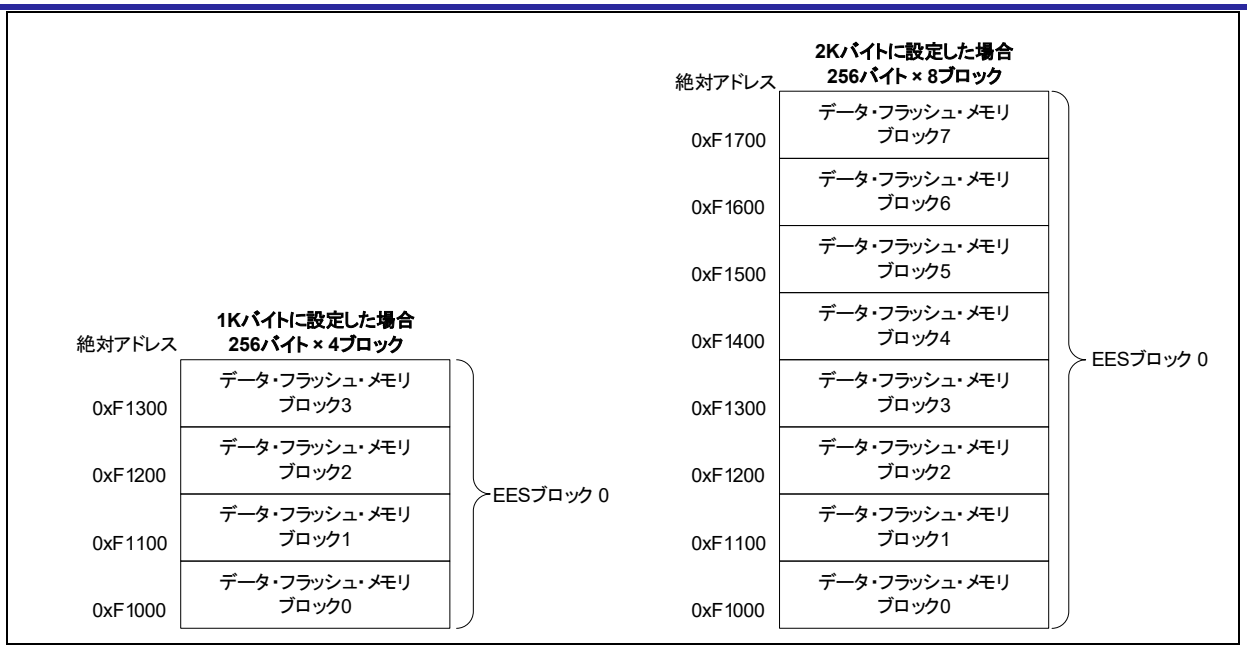


図 2-2 EES ブロック 0 の概念図

2.2.2 EES プール

EES プールはユーザによって定義される EES がアクセス可能なデータ・フラッシュ領域です。ユーザ・プログラムからのデータ・フラッシュへのアクセスは、EES 経由での EES 用 RFD 制御関数、および RFD から EES プールへのアクセスのみ許可されます。

必ず対象デバイスに搭載されているデータ・フラッシュサイズ内の値を EES プールに設定してください。なお、設定方法につきましては、「4.2 ユーザ設定初期値」をご参照ください。

8KB のデータ・フラッシュ・メモリを有するデバイスの EES プール構成例を「図 2-3 EES プール構成例（EES ブロックサイズに 1K バイトを設定）」に示します。

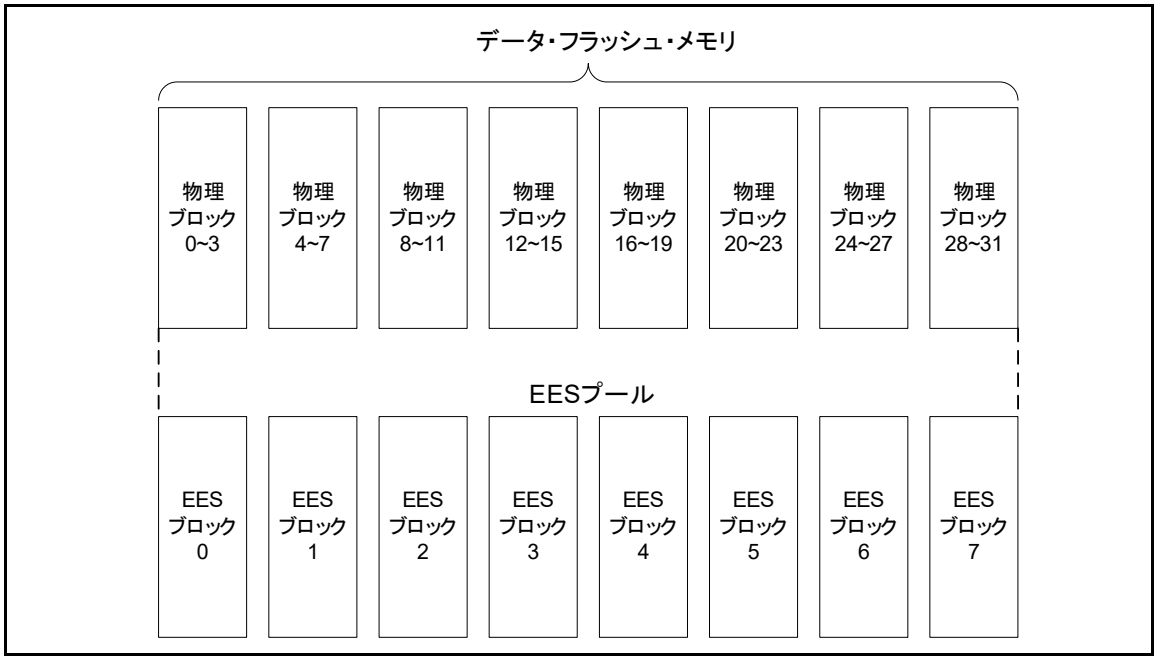


図 2-3 EES プール構成例（EES ブロックサイズに 1K バイトを設定）

2.3 ファイル構成

2.3.1 フォルダ構成

EES RL78 Type01 のフォルダ構成を図 2-4 に示します。

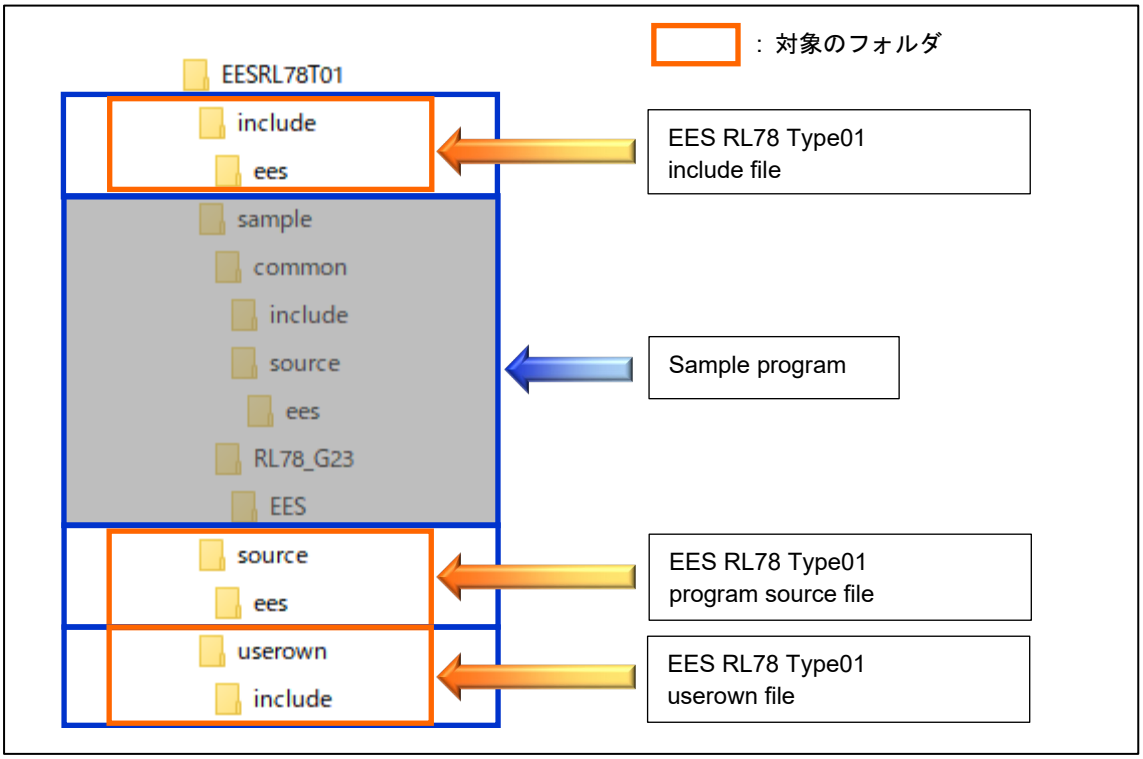


図 2-4 EES RL78 Type01 のフォルダ構成

注）図 2-4 では RL78/G23 を使用する場合の例を記載しています。実際にインストールした”sample”フォルダには、デバイスグループごとのフォルダが含まれます（例: RL78_G23, RL78_G24）。
デバイスグループごとのサンプル用フォルダについては、「6.1.1 フォルダ構成」をご確認ください。

2.3.2 ファイル・リスト

2.3.2.1 ソース・ファイル・リスト

“source\ees\”フォルダ内のプログラム・ソース・ファイルを表 2-1 に示します。

表 2-1 “source\ees\”フォルダ内プログラム・ソース・ファイル

No	ソース・ファイル名	概略(Summary)
1	r_ees_api.c	EES を制御する API 関数ファイル
2	r_ees_exrfd_api.c	EES 用 RFD を制御する API 関数ファイル
3	r_ees_sub_api.c	EES 制御関数の内部で使用される API 関数ファイル

“userown\”フォルダ内のプログラム・ソース・ファイルを表 2-2 に示します。

表 2-2 “userown\”フォルダ内プログラム・ソース・ファイル

No	ヘッダ・ファイル名	概略(Summary)
1	r_ees_descriptor.c	EES で使用するディスクリプタを実装するファイル

2.3.2.2 ヘッダ・ファイル・リスト

“include\”フォルダ内のプログラム・ヘッダ・ファイルを表 2-3 に示します。

表 2-3 “include\”フォルダ内プログラム・ヘッダ・ファイル

No	ヘッダ・ファイル名	概略(Summary)
1	r_ees_api.h	EES を制御する API 関数のプロトタイプ宣言を定義したファイル
2	r_ees_exrfd_api.h	EES 用 RFD 制御関数のプロトタイプ宣言を定義したファイル
3	r_ees_sub_api.h	EES 制御関数の内部で使用される関数のプロトタイプ宣言を定義したファイル

“userown\include”フォルダ内のプログラム・ヘッダ・ファイルを表 2-4 に示します。

表 2-4 “userown\include”フォルダ内プログラム・ヘッダ・ファイル

No	ヘッダ・ファイル名	概略(Summary)
1	r_ees_descriptor.h	EES で使用するディスクリプタを定義するファイル
2	r_ees_user_types.h	EES RL78 Type01 内で使用するユーザ・データの型を定義したファイル

“include\ees”フォルダ内のプログラム・ヘッダ・ファイルを表 2-5 に示します。

表 2-5 “include\ees\”フォルダ内プログラム・ヘッダ・ファイル

No	ヘッダ・ファイル名	概略(Summary)
1	r_ees.h	共通ヘッダ・ファイルを記述したファイル。
2	r_ees_compiler.h	EES RL78 Type01 で使用するコンパイラ依存のマクロを定義したファイル
3	r_ees_defines.h	EES RL78 Type01 で使用するマクロを定義したファイル
4	r_ees_device.h	EES RL78 Type01 で使用するハードウェア固有のマクロを定義したファイル
5	r_ees_memmap.h	EES RL78 Type01 で使用するセクションを記述するためのマクロを定義したファイル
6	r_ees_types.h	EES RL78 Type01 で使用する変数の型を定義したファイル
7	r_typedefs.h	EES RL78 Type01 で使用するデータの型を定義したファイル

2.4 RL78/G2x リソース

2.4.1 メモリ・マップ

RL78/G23, G22, G24 のコード・フラッシュ: CF(1 ブロック: 2Kbyte)、データ・フラッシュ: DF(1 ブロック: 256byte)、RAM のメモリ・マップを表 2-6 に示します。

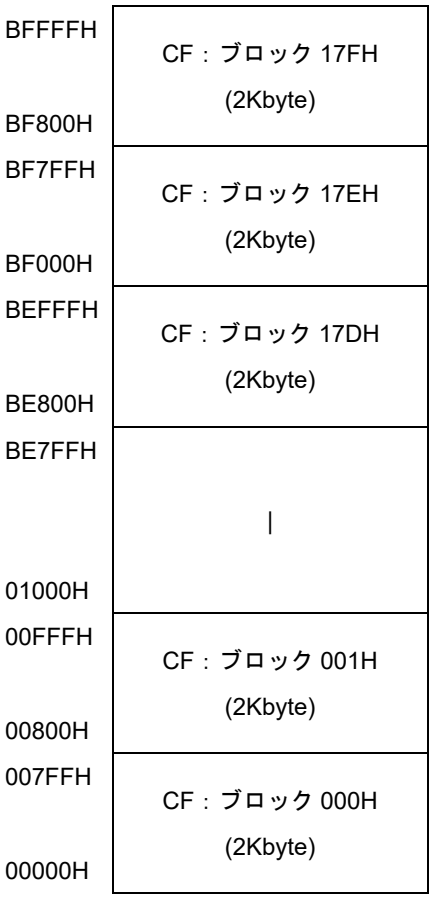
表 2-6 コード・フラッシュ、データ・フラッシュ、RAM のメモリ・マップ

RL78	デバイス型名	コード・フラッシュ: CF	RAM
G23	R7F100GxF (x=A,B,C,E,F,G,J,L)	96KB (00000H-17FFFH)	12KB (FCF00H-FFEFFH)
	R7F100GxG (x=A,B,C,E,F,G,J,L,M,P)	128KB (00000H-1FFFFH)	16KB (FBF00H-FFEFFH)
	R7F100GxH (x= A,B,C,E,F,G,J,L,M,P)	192KB (00000H-2FFFFH)	20KB (FAF00H-FFEFFH)
	R7F100GxJ (x=A,B,C,E,F,G,J,L,M,P,S)	256KB (00000H-3FFFFH)	24KB (F9F00H-FFEFFH)
	R7F100GxK (x=F,G,J,L,M,P,S)	384KB (00000H-5FFFFH)	32KB (F7F00H-FFEFFH)
	R7F100GxL (x=F,G,J,L,M,P,S)	512KB (00000H-7FFFFH)	48KB (F3F00H-FFEFFH)
	R7F100GxN (x=F,G,J,L,M,P,S)	768KB (00000H-BFFFFH)	48KB (F3F00H-FFEFFH)
	データ・フラッシュ: DF	8KB (F1000H-F2FFFH) RL78/G23 共通	
G22	R7F102GxC (x=4,6,7,8,A,B,C,E,F,G)	32KB (00000H-07FFFH)	4KB (FEF00H-FFEFFH)
	R7F102GxE (x=4,6,7,8,A,B,C,E,F,G)	64KB (00000H-0FFFFH)	4KB (FEF00H-FFEFFH)
	データ・フラッシュ: DF	2KB (F1000H-F17FFH) RL78/G22 共通	
G24	R7F101GxE (x=6,7,8,A,B,E,F,G,J,L)	64KB (00000H-0FFFFH)	12KB (FCF00H-FFEFFH)
	R7F101GxG (x=6,7,8,A,B,E,F,G,J,L)	128KB (00000H-1FFFFH)	12KB (FCF00H-FFEFFH)
	データ・フラッシュ: DF	4KB (F1000H-F1FFFH) RL78/G24 共通	

2.4.2 ブロックイメージ

RL78/G23 のコード・フラッシュ(CF)、データ・フラッシュ(DF)のブロックイメージの例を図 2-5、図 2-6 に示します。その他のデバイスのブロックイメージについては、対象デバイスのユーザーズマニュアルをご参照ください。

(1) R7F100GxN(コード・フラッシュ 768Kbyte)



(2) R7F100GxF(コード・フラッシュ 96Kbyte)

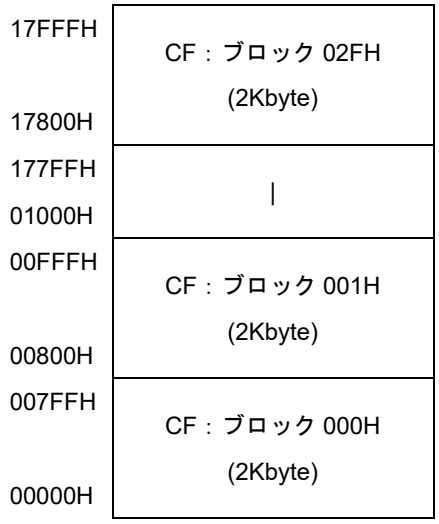


図 2-5 コード・フラッシュ のブロックイメージ

RL78/G23 共通(データ・フラッシュ 8Kbyte)

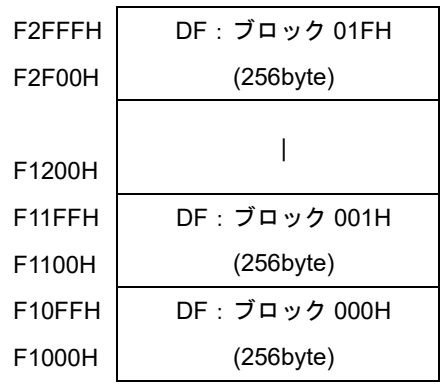


図 2-6 データ・フラッシュのブロックイメージ

2.4.3 フラッシュ動作モード

(1) RL78/G23 のフラッシュ動作モードごとの動作周波数範囲

RL78/G23 のフラッシュ動作モードごとの動作周波数範囲を表 2-7 に示します。

表 2-7 RL78/G23 のフラッシュ動作モードごとの動作周波数範囲

電源電圧（V_{DD}）	フラッシュ動作モード	動作周波数
1.8 V ≦ V_{DD} ≦ 5.5 V	HS（高速メイン）モード	1 MHz ～ 32 MHz
	LS（低速メイン）モード	1 MHz ～ 24 MHz
1.6 V ≦ V_{DD} ＜ 1.8 V	HS（高速メイン）モード	1, 2 MHz
	LS（低速メイン）モード	1, 2 MHz

注）LP（低消費メイン）モードでのフラッシュ書き換えはできません。

(2) RL78/G22 のフラッシュ動作モードごとの動作周波数範囲

RL78/G22 のフラッシュ動作モードごとの動作周波数範囲を表 2-8 に示します。

表 2-8 RL78/G22 のフラッシュ動作モードごとの動作周波数範囲

電源電圧（V_{DD}）	フラッシュ動作モード	動作周波数
1.8 V ≦ V_{DD} ≦ 5.5 V	HS（高速メイン）モード	1 MHz ～ 32 MHz
	LS（低速メイン）モード	1 MHz ～ 24 MHz

注）LP（低消費メイン）モードでのフラッシュ書き換えはできません。

(3) RL78/G24 のフラッシュ動作モードごとの動作周波数範囲

RL78/G24 のフラッシュ動作モードごとの動作周波数範囲を表 2-9 に示します。

表 2-9 RL78/G24 のフラッシュ動作モードごとの動作周波数範囲

電源電圧（V_{DD}）	フラッシュ動作モード	動作周波数
2.4 V ≦ V_{DD} ≦ 5.5 V	HS（高速メイン）モード （プリフェッチ ON）	48 MHz
1.8 V ≦ V_{DD} ≦ 5.5 V	HS（高速メイン）モード （プリフェッチ OFF）	1 MHz ～ 32 MHz
	LS（低速メイン）モード	1 MHz ～ 24 MHz

注 1）HS（高速メイン）モード（プリフェッチ ON）」では、RL78/G24 固有のプリフェッチバッファを有効にする必要があります。

2 LP（低消費メイン）モードでのフラッシュ書き換えはできません。

2.5 EES RL78 Type01 使用リソース

2.5.1 EES RL78 Type01 使用時のセクション

EES で使用するセクションと配置の一覧を表 2-10 に示します。

表 2-10 EES 使用時のセクション

セクション名	内容	配置
EES_CODE	EES 制御 API 関数のプログラム・セクション	ROM
EES_CNST	EES 定数のセクション	ROM
EES_VAR	EES 変数のデータ・セクション	RAM
SMP_EES	EES 制御サンプル関数のプログラム・セクション	ROM
SMP_VAR	EES 制御サンプル変数のデータ・セクション	RAM

2.5.2 ソフトウェア・リソース

表 2-11 に EES RL78 Type01 のソフトウェア・リソース（参考値）を示します。

表 2-11 EES RL78 Type01 のソフトウェア・リソース注 1,2（参考値）

項目	容量（バイト）		
	CC-RL	IAR	LLVM
スタックサイズ	44	48	44
コードサイズ注 3	4624	5177	5830

注 1 「1.6 C コンパイラ定義」のコンパイラ・オプション使用時の数値です。

注 2 サンプル・プログラムのスタックサイズ、コードサイズは含みません。

注 3 RFD RL78 Type01 のコードサイズは含みません。

3 EEPROM エミュレーション

3.1 EEPROM エミュレーションの仕様

EES RL78 Type01 では、ユーザ・プログラムから EES RL78 Type01 が提供する API 関数を呼び出す事により、データ・フラッシュ・メモリに関する操作を意識することなく使用することができます。

EES RL78 Type01 では、1 バイトの識別子（データ ID：1～254）をデータごとにユーザが割り振り、割り振った識別子ごとに読み出しや書き込みを 1～255 バイトの任意の単位で操作することができます（識別子は最大 254 個まで扱うことができます）。

また、データを格納するための EES ブロック（仮想ブロック）は、3 ブロック以上（推奨）[注]の領域を使用します。EEPROM エミュレーションによって書き込まれるデータは、参照用の参照データと、ユーザが指定したユーザ・データに分けられ、参照データはブロックの小さいアドレスから、ユーザ・データはブロックの大きいアドレスから EES ブロックに書き込みが行われます。

注） EEPROM エミュレーションを行う場合、最低限 2 ブロック以上の EES ブロックが必要です。EES ブロックを 2 ブロックと指定した場合、1 度でも書き込みエラーが発生した時に、以降はそれまで正常に書き込めたデータの読み出しのみ可能で、書き込みを継続して行うことができません。次に EES でデータの書き込みを行う場合は対象 2 ブロックをフォーマットする必要があり、それまで書き込んでいたデータは全て消去されます。また、システムにより電圧低下など、不慮の状態が発生する可能性もあるので、EES ブロックには、3 ブロック以上をご指定いただくことを推奨いたします。

3.2 機能概要

EES では以下のような特徴を含む基本的な書き込み、および読み出し機能等が提供されています。

- EES ブロックのサイズに、
  RL78/G23,G24：1024 または 2048 バイトのいずれかを設定可能
  RL78/G22：512 バイト
- ユーザ・データは 1～254 個まで設定可能
- ユーザ・データのサイズは 1～255 バイトまで設定可能
- BGO(Background Operation) に対応
- EES 管理用データ（ブロック・ヘッダ、セパレータ）のメモリ消費量、EES ブロックあたり 10 バイト
- 参照領域の参照データのメモリ消費量、EES ブロックに書き込まれる 1 データあたり 3 バイト
- R_EES_ENUM_CMD_WRITE、または R_EES_ENUM_CMD_REFRESH 実行中の CPU リセットによる中断に対して、R_EES_ENUM_CMD_REFRESH で復旧
- ブロック・ローテーション（データ・フラッシュ使用頻度の均衡化）

EES 機能使用時の設定範囲を「表 3-1」に示します。

表 3-1 EES 機能使用時の設定範囲

項目	範囲
EESブロックサイズ	512（バイト）RL78/G22 使用時 1024もしくは2048（バイト）RL78/G23, G24 使用時
ユーザ・データ長	1～255
格納ユーザ・データ数 注1	1～254
ユーザ・データ ID 番号	1～254 （登録順に1から最大254が割り当てられ、任意の設定は不可）
EEPROM エミュレーション・ブロック数注2	3～255
ユーザ・データの推奨サイズ 注1	EES ブロックを512（バイト）に設定時：502/2（バイト）以下 EES ブロックを1024（バイト）に設定時：1014/2（バイト）以下 EES ブロックを2048（バイト）に設定時：2038/2（バイト）以下

注1 ユーザ・データは、すべてのユーザ・データが EES ブロックへ書き込まれるときに必要な合計サイズを 1 ブロックの 1/2 以内に収められる状態にする必要があります。そのため、格納するユーザ・データのサイズによって格納ユーザ・データ数の使用範囲は変わります。また、合計サイズも管理用としてデータごとに付与される参照データ分のサイズも考慮に入れる必要があります。格納ユーザ・データ数や合計サイズの詳細については「4.1 格納ユーザ・データ数とユーザ・データの合計サイズ」を参照してください。

注2 搭載されているデータ・フラッシュ・メモリの最大容量以上には設定できません。

3.3 EES プール

この章では、ユーザが EES を使用してデータ・フラッシュ・メモリ（EES プール）の書き換えを行う上で必要な EES のアーキテクチャについて説明します。

3.3.1 EES プールの状態

EES プールの各ブロックには状態があり、表 3-2 にブロックの使用状態を示しています。

表 3-2 EES ブロックの状態一覧

状態	内容
有効	1 つの EES ブロックが有効となり、定義済みのデータを格納します。有効ブロックは EES プールに割り当てられた EES ブロック群を循環します。
無効	無効ブロックにはデータは格納されません。EES ブロックは EES によって無効とされるか、消去ブロックの場合は無効となります。
使用禁止	機能動作が失敗してデータ・フラッシュの故障の可能性が判明した場合は、EES では該当ブロックを使用禁止とし、その後はそのブロックを EEPROM エミュレーションには使用しません。

EES プールの状態例を「図 3-1 EES プール状態例」に示します。

有効ブロック（例では EES ブロック1）に書き込み可能領域がなくなり追加データの格納ができなくなった時（write コマンドの失敗）には、新規の有効ブロックが循環的に選定され、その時点で有効なデータ群が新規の有効ブロックにコピーされます。このプロセスは「リフレッシュ」と呼ばれます。R_EES_ENUM_CMD_REFRESH コマンド実行後に元の有効ブロックは無効となり、1つの有効ブロックのみが存在します。このプロセスにおいて使用禁止ブロック（例では EES ブロック7）は無視され、次の有効ブロック選定の候補にはなりません。

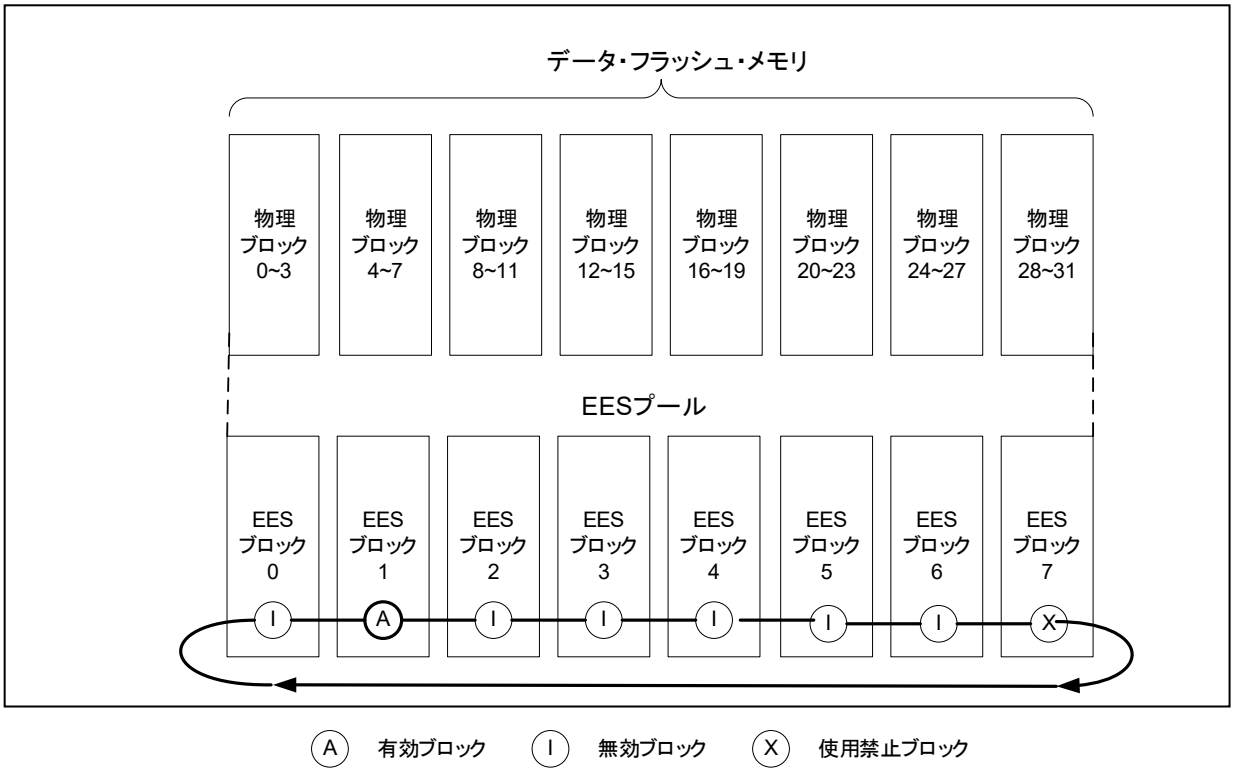


図 3-1 EES プール状態例（EES ブロックに1K バイト設定時）

EES プール内の EES ブロックのライフサイクルのイメージを「図 3-2 EES ブロックのライフサイクル」に示します。通常動作時では、EES ブロックは有効状態と無効状態の間を行き来します。EES ブロックへのアクセス中にエラーが発生した場合には、エラーが発生した EES ブロックは使用禁止ブロックとなります。このブロックは EES ブロックのライフサイクルに再投入されることはありません。ただし、ユーザが EES プール全体をフォーマットすることにより使用禁止ブロックの修復を試みることは可能ですが、この際に既存データの内容はすべて消去されます。

注意 EES ブロックは、仮想ブロックであるため EES ブロック内で使用されているデータ・フラッシュ・メモリの物理ブロックのうち 1 ブロックでも故障などで使用できない場合、そのブロックを含む EES ブロックは使用禁止ブロックと判断されます。

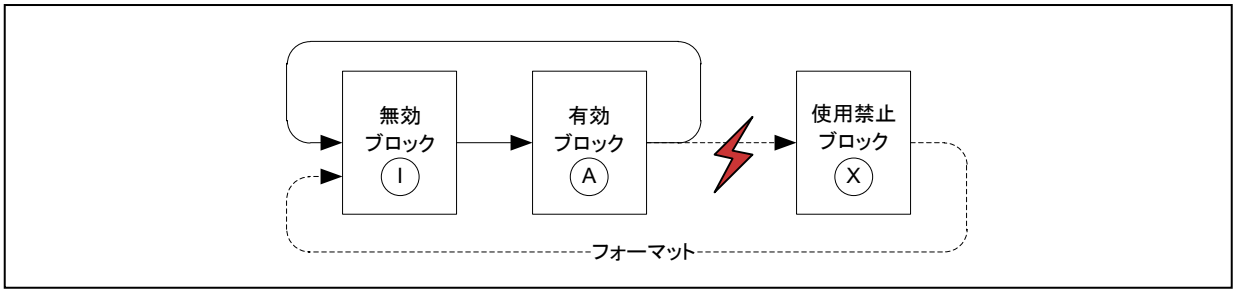


図 3-2 EES ブロックのライフサイクル

EESプールには、以下のような4種類の状態があります。

表 3-3 EESプールの状態一覧

状態	説明
プール動作可能	EES 動作時の通常状態です。すべてのコマンドが実行可能です。
プール・フル	使用中の有効ブロックは、書き込みを行うための空き容量が足りません。リフレッシュの実行が必要であることを示しています。
プール消耗	継続して使用できる EES ブロックがなくなりました（EES の動作には、最低でも使用禁止でないブロックが 2 つ必要です）。
プール不整合	プールの状態に不整合があり、EES ブロック内に存在するデータ構造が、ユーザが設定した構造と一致しません。EES ブロックが不定状態（有効ブロックがない等）です。

3.3.2 EES ブロック構造

EES が使用する EES ブロック構造を「図 3-3 EES ブロック（EES ブロックに 1K バイト設定時）の構成」に示します。EES ブロックは、ブロック・ヘッダ、参照領域、データ領域の 3 つの利用領域で構成されています。

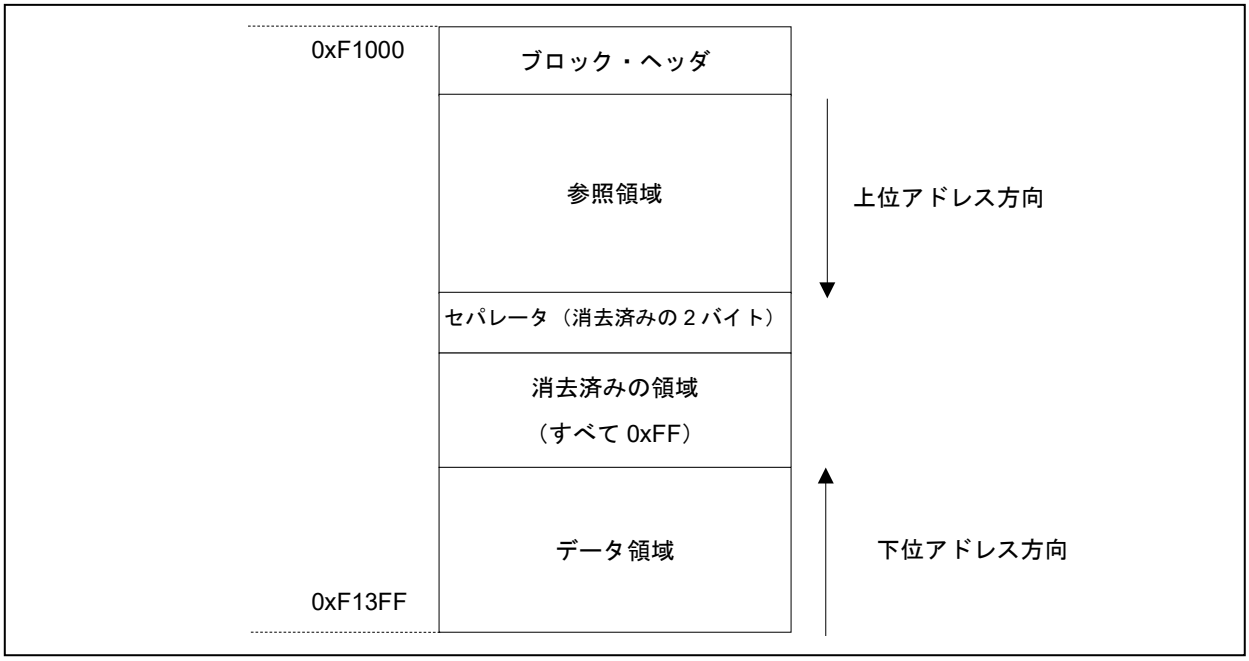


図 3-3 EES ブロック（EES ブロックに 1K バイト設定時）の構成

表 3-4 EESブロックの構成一覧

名称	説明
ブロック・ヘッダ	EES プール内のブロック管理に必要なブロック状態の情報が格納されています。8 バイトの固定サイズです。
参照領域	データの管理に必要な参照データが格納されています。データが書き込まれると、上位アドレス方向に拡大します。
データ領域	ユーザ・データが格納されています。データが書き込まれると、下位アドレス方向に拡大します。

参照領域とデータ領域の間には、消去済み領域があります。データが更新される（データの書き込みが行われる）たびに、この領域は減少します。しかし、参照領域とデータ領域の間には、領域の分離とブロック管理のために最低でも2バイトの未使用領域が常に残されています。これは「図 3-3 EESブロック（EESブロックに1Kバイト設定時）の構成」ではセパレータとして示されています。

EESブロック・ヘッダの詳細を「3.3.3 EESブロック・ヘッダ」に、参照領域とデータ領域に格納されるデータの構造を「3.3.4格納データの構造」で説明します。

3.3.3 EES ブロック・ヘッダ

EES ブロック・ヘッダの構成を「図 3-4 EES ブロック・ヘッダの構成」に示します。8 バイトで構成され、その中の 3 バイトはシステムで予約されています。

ブロック内の相対アドレス		
0x0000	A	N
0x0001	B	0xFF - N
0x0002	B’	0x00
0x0003	I	0x00
0x0004	X	0x00
0x0005	-	予約
0x0006	-	予約
0x0007	-	予約

図 3-4 EES ブロック・ヘッダの構成

ブロック状態フラグは、ブロックの先頭から A フラグ、B フラグ、B’フラグ、I フラグ、X フラグの各 1 バイトずつ計 5 バイトのデータとして配置されます。各フラグの組み合わせにより EES ブロックの状態を示します。

「図 3-4 EES ブロック・ヘッダの構成」に各フラグの配置状態を、「表 3-5 ブロック状態フラグの説明」に各フラグの組み合わせによる状態を示します。

表 3-5 ブロック状態フラグの説明

ブロック状態フラグ					状 態	概 要
A フラグ	B フラグ	B‘ フラグ	I フラグ	X フラグ		
0x01	0xFE	0x00	0xFF	0xFF	有効	現在使用中のブロック R_EES_ENUM_CMD_REFRESH コマンドを実行後、新しい有効ブロックの A フラグには0x02が設定されます。
0x02	0xFD	0x00	0xFF	0xFF		現在使用中のブロック R_EES_ENUM_CMD_REFRESH コマンドを実行後、新しい有効ブロックの A フラグには0x03が設定されます。
0x03	0xFC	0x00	0xFF	0xFF		現在使用中のブロック R_EES_ENUM_CMD_REFRESH コマンドを実行後、新しい有効ブロックの A フラグには0x01が設定されます。
0x01	0xFE	0x01~ 0xFE	0xFF	0xFF	有効	現在使用中のブロック。ただし、B‘フラグの書き込みが完了していないため、新しいデータを追加することはできません。読み出しは可能です。 R_EES_ENUM_CMD_REFRESH コマンドを実行後、新しい有効ブロックの A フラグは、0x01,0x02,0x03,0x01,…の順序で設定されます。
0x02	0xFD		0xFF	0xFF		
0x03	0xFC		0xFF	0xFF		
―		0xFF	0xFF	0xFF	無効	無効状態となったブロック
―			0xFF 以外	0xFF		
―			―	0xFF 以外	使用禁止	使用禁止となったブロック

3.3.4 格納データの構造

EES ブロックにユーザ・データを書き込むときの格納データの構造を示します。データは、レコード開始（SoR）フィールド、レコード終了（EoR, EoR’）フィールド、データフィールドの 3 つの部分から成ります。EES ディスクリプタ・テーブルを使って、EES 内部で使用するデータを設定できます。各データは識別番号（ID）によって参照され、1～255 バイトまでのサイズが設定可能です。（EES ディスクリプタ・テーブルのフォーマットの正確な仕様は「4.2 ユーザ設定初期値」に記載されています。）

データが書き込まれるたびに EES ブロック内に格納データが増加し、複数の格納データが存在しますが、参照されるのは最新の格納データのみです。

SoR と EoR, EoR’は、データの管理に必要な参照データを構成します。参照データとユーザ・データは有効ブロック内の別々のフィールドに格納されますが、これらのフィールドはそれぞれ参照領域、データ領域と呼ばれます。データの全体構造の使用例を「図 3-6 有効な EES ブロックの例」に示します。

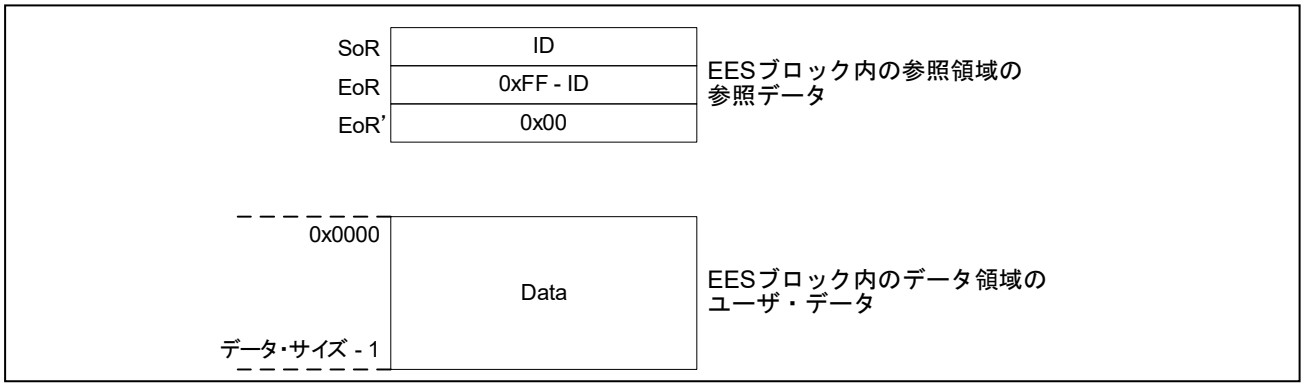


図 3-5 格納データの構造

表 3-6 データの各領域の説明

名称	説明
SoR フィールド (Start of Record)	1 バイトの SoR フィールドにはデータのデータ ID が格納されています。 このフィールドは、書き込み処理の開始を示します。消去済みセルのパターンを避けるため、データ ID には、0x00 と 0xFF は使用されません。
EoR フィールド (End of Record)	1 バイトの EoR フィールドには（0xFF – データ ID）の値が格納されています。 このフィールドは、データ書き込み処理の正常終了を示します。デバイスのリセット等により書き込みが不完全である場合には、対応する格納データは EES から無視されます。
EoR' フィールド (End of Record')	1 バイトの EoR'フィールドは、EoR フィールド書き込み処理の正常終了を示します。 このフィールドは、EoR フィールドの書き込み終了後に 0x00 が書き込まれます。 0x01~0xFE の場合は、格納データは有効ですが書き込みが最後まで完了していないと判断されるため、以降そのブロックは追記できないブロックとして扱われます。0xFF の場合、EoR フィールドの書き込みが正常終了していないと判定し、無効データとして扱われます。
データフィールド	データフィールドにはユーザ・データが格納されています。データ範囲は 1～255 バイトです。 サイズが 2 バイト以上のデータの場合、小さいアドレスのデータが、データフィールドの小さいアドレス側に格納されます（図 3-6 で確認してください）。

格納データは、SoR → データフィールド → EoR → EoR' の順番でEESブロックへの書き込みが行われます。また、EoRフィールドの値が正常に書き込まれていない場合、ひとつ前のデータが有効です。

注意1 各格納データによって消費される参照データの合計サイズは3バイトです。R_EES_GetSpace関数を用いてデータの書き込み前に空き容量を確認する際には、参照データの合計サイズも考慮する必要があります。

注意2 ユーザ・データにチェックサムは追加されません。チェックサムが必要な場合、ユーザ・データにチェックサムを付加し判定する等、ユーザ・プログラムで対応してください。

3.3.5 EES ブロックの概要

「図 3-6 有効なEESブロックの例」に、複数の格納データを含むEESブロックの例を示します。:

- データID 0x01 ：データ長 4バイト
- データID 0x02 ：データ長 1バイト
- データID 0x03 ：定義されていますが、書き込まれていません。
- データID 0x04 ：データ長 2バイト

データは、データID 0x01 → データID 0x04 → データID 0x02 の順に書かれています。

この例では、データID 0x03のデータはまだ書き込まれていません。

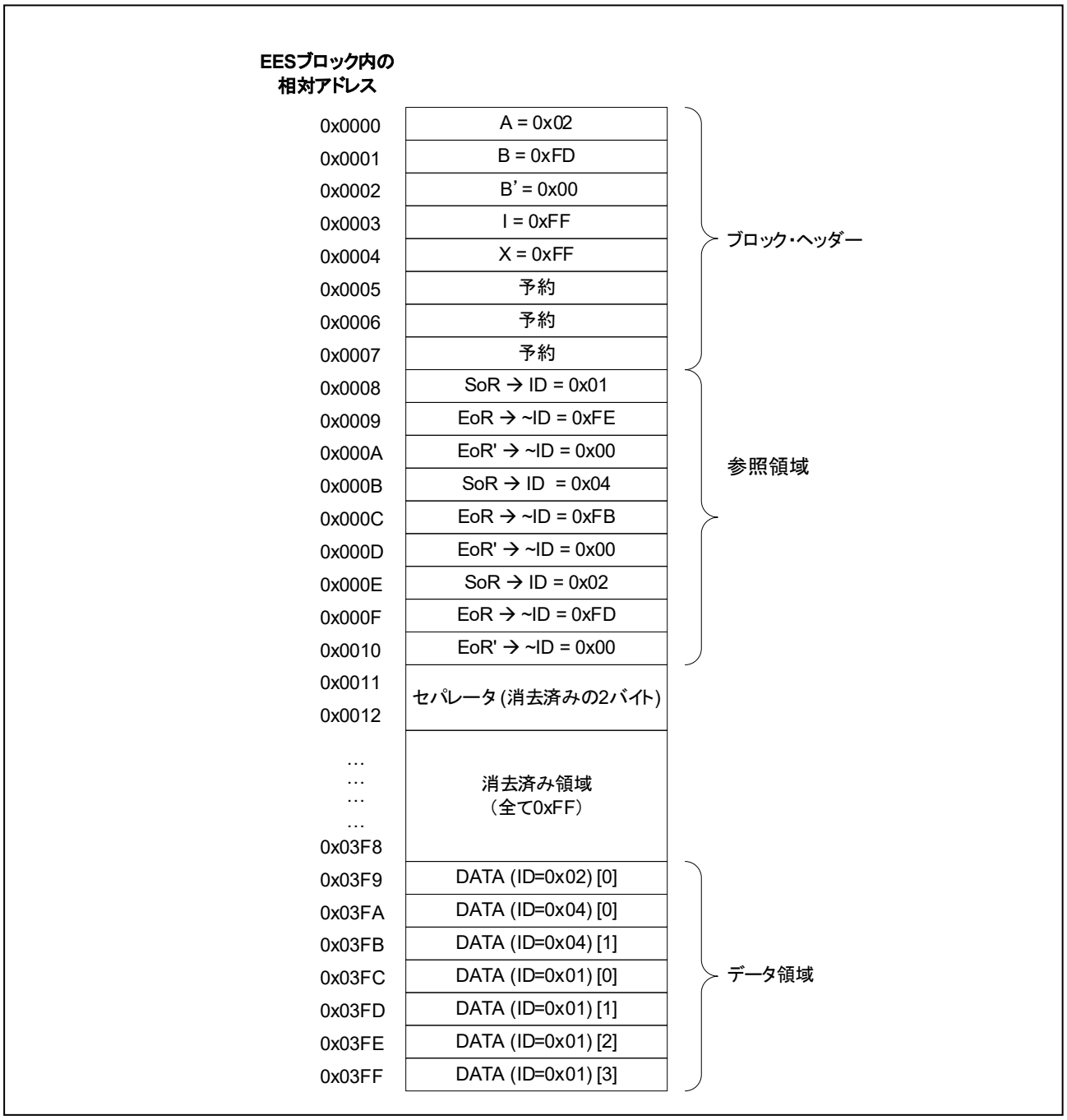


図 3-6 有効な EES ブロックの例

4 EEPROM エミュレーションの使用方法

EEPROM エミュレーションは、3 ブロック以上（推奨）の EES ブロックを使用することにより、1～255 バイトのデータを最大 254 個まで EEPROM エミュレーションによりデータ・フラッシュ・メモリに格納することができます。

EES をユーザ・プログラムに組み込み、そのプログラムを実行することにより、EEPROM エミュレーションを実現することができます。

4.1 格納ユーザ・データ数とユーザ・データの合計サイズ

EEPROM エミュレーションで使用できるユーザ・データの合計サイズには制限があり、すべてのユーザ・データが EES ブロックに書き込まれる場合に必要なサイズを 1 ブロックの 1/2 以内に収まる状態にする必要があります。また、使用できる格納データ数は、EES ブロックのサイズ設定や、実際に格納するユーザ・データのサイズによって異なります。以下に実際にユーザ・データの書き込みで使用できるサイズ、およびユーザ・データの合計サイズの計算方法を示します。

【ユーザ・データの書き込みに使用できる 1 ブロックの最大使用可能サイズ】

EEPROM エミュレーションでブロックの管理に必要なサイズ ：8 バイト

終端用の情報として必ず必要な空き容量（セパレータ） ：2 バイト

- EES ブロックサイズを 1024 バイトに設定した場合

  EES ブロックサイズ：256 * 4 = 1024（バイト）

  1 ブロックの最大使用可能サイズ= 1024 – (8 + 2) = 1014（バイト）

- EES ブロックサイズを 2048 バイトに設定した場合

  EES ブロックサイズ：256 * 8 = 2048（バイト）

  1 ブロックの最大使用可能サイズ = 2048 – (8 + 2) = 2038（バイト）

【ユーザ・データごとの書き込みサイズの計算方法】[注]

書き込まれる個々のユーザ・データのサイズ = データ・サイズ + 参照エントリ・サイズ（3 バイト）

注） 詳細については「3.3.4 格納データの構造」の項を参照してください。

【ユーザ・データの基本合計サイズの計算方法】

基本合計サイズ = (ユーザ・データ 1 + 3) + (ユーザ・データ 2 + 3)・・・+ (ユーザ・データ n + 3)

【最大サイズと推奨サイズ】

データはすべて 1 ブロック内に収める必要があります。そのため、最大サイズは 1 ブロックの最大使用可能サイズですが、以下の関係式を満たすことを推奨します。全データを 1 回は更新できるようにするため、1 ブロックの最大使用可能サイズの半分の容量内で使用することを推奨しています。

最大サイズ：全データを書き込み後、一番サイズが大きなデータを 1 回更新できることを想定。
推奨サイズ：全データを書き込み後、全データを 1 回更新できることを想定。

- EES ブロックサイズを 1024 バイトに設定した場合
  最大サイズ = ユーザ・データの基本合計サイズ + 最大のデータ・サイズ + 3 ≤ 1014
  推奨サイズ = 1014 / 2 = 507（バイト）以下

- EES ブロックサイズを 2048 バイトに設定した場合
  最大サイズ = ユーザ・データの基本合計サイズ + 最大のデータ・サイズ + 3 ≤ 2038
  推奨サイズ = 2038 / 2 = 1019（バイト）以下

4.2 ユーザ設定初期値

EESの設定初期値は、次に示す項目をユーザが必ず設定する必要があります。また、EESを実行する前に、高速オンチップ・オシレータを起動しておく必要があります。外部クロック使用時も、高速オンチップ・オシレータを起動しておく必要があります。

- EES 設定初期値

＜EEPROM エミュレーション・ソフトウェア ユーザ・インクルード・ファイル (r_ees_descriptor.h) ＞注 2、3

```
#define R_EES_EXRFD_VALUE_U16_PHYSICAL_BLOCK_SIZE     (256u)
                                                  : (1) データ・フラッシュ・メモリ 1 ブロックの
                                                    サイズ（物理ブロックサイズ）
#define R_EES_EXRFD_VALUE_U08_PHYSICAL_BLOCKS_PER_VIRTUAL_BLOCK       (4u)
                                                  : (2) EES ブロック（1 仮想ブロックあたり）に
                                                    設定するデータ・フラッシュ・メモリ・ブロッ
                                                    ク数（物理ブロック数）注 1
#define R_EES_EXRFD_VALUE_U08_POOL_VIRTUAL_BLOCKS                     (4u)
                                                  : (3) EES プールサイズ（仮想ブロック数）
#define R_EES_VALUE_U08_VAR_NO        (8u)        : (4) 格納データ数
```

注 1 EES ブロック（1 仮想ブロックあたり）に設定可能なデータ・フラッシュ・メモリ・ブロック数、
RL78/G22：2u
RL78/G23,G24：4u または 8u

＜EEPROMエミュレーション・ソフトウェア ユーザ・データ定義ファイル (r_ees_user_types.h) ＞注3

```
typedef   uint8_t   type_A[2];                    : (5) 識別子(データ ID)毎の
typedef   uint8_t   type_B[3];                      データ・サイズ定義
typedef   uint8_t   type_C[4];
typedef   uint8_t   type_D[5];
typedef   uint8_t   type_E[6];
typedef   uint8_t   type_F[10];
typedef   uint8_t   type_X[20];
typedef   uint8_t   type_Z[255];
```

＜EEPROMエミュレーション・ソフトウェア ユーザプログラムファイル (r_ees_descriptor.c) ＞注3

```
__far const uint8_t                                   : (6) 各データ識別子(データ ID)
g_ar_u08_ees_descriptor [R_EES_VALUE_U08_VAR_NO + 2u] =    のデータのサイズ
{
 (uint8_t)( R_EES_VALUE_U08_VAR_NO),   /* variable count */   \
 (uint8_t)(sizeof(type_A)),            /* id=1              */   \
 (uint8_t)(sizeof(type_B)),            /* id=2              */   \
 (uint8_t)(sizeof(type_C)),            /* id=3              */   \
 (uint8_t)(sizeof(type_D)),            /* id=4              */   \
 (uint8_t)(sizeof(type_E)),            /* id=5              */   \
 (uint8_t)(sizeof(type_F)),            /* id=6              */   \
 (uint8_t)(sizeof(type_X)),            /* id=7              */   \
 (uint8_t)(sizeof(type_Z)),            /* id=8              */   \
 (uint8_t)(0x00),                      /* zero terminator      */   \
};
```

注2 使用しているマクロは、EES共通のパラメータとして使用していますので、数値以外は変更しないでください。

注3 EESブロック初期化後(R_EES_ENUM_CMD_FORMATコマンド実行後)は各値を変更しないでください。変更する場合は、EESブロックの再初期化(R_EES_ENUM_CMD_FORMATコマンド実行)を行ってください。

(1) データ・フラッシュ・メモリ1ブロックのサイズ（物理ブロックサイズ）

対象デバイスに搭載されている、データ・フラッシュ・メモリ1ブロックのサイズを設定します。

(2) EESブロックに使用するデータ・フラッシュ・メモリ・ブロック数

EESブロック1ブロックに使用するデータ・フラッシュ・メモリ・ブロック数を設定します。

(3) EESプールサイズ 注

EESプールに使用するEESブロック数（仮想ブロック数）を設定します。必ず対象デバイスに搭載されているデータ・フラッシュ・メモリのサイズを考慮しEESプールのブロック数に設定してください。

注）EES プールサイズには3ブロック以上の値を設定してください（推奨）

(4) 格納データ数

EEPROM エミュレーションで使用するデータ数を設定します。設定できる値は1～254の範囲です。

(5) 識別子(データID)毎のデータ・サイズ定義

ユーザが指定する各ユーザ・データのバイトサイズのデータ型名を定義します。EESディスクリプタ・テーブルに各ユーザ・データのバイトサイズが反映されます。

(6) 各データ識別子(データID)のデータのサイズ

各識別子のデータのサイズを規定するテーブルです。これをEESディスクリプタ・テーブルといいます。書き込みを行うデータはEESディスクリプタ・テーブルに事前に登録する必要があります。

__far const uint8_t g_ar_u08_ees_descriptor[格納データ数 ＋ 2]

R_EES_VALUE_U08_VAR_NO
データ ID1 のバイトサイズ
データ ID2 のバイトサイズ
データ ID3 のバイトサイズ
データ ID4 のバイトサイズ
データ ID5 のバイトサイズ
データ ID6 のバイトサイズ
データ ID7 のバイトサイズ
データ ID8 のバイトサイズ
0x00

図 4-1 EESディスクリプタ・テーブル（8件の異なったデータがある場合）

・R_EES_VALUE_U08_VAR_NO

ユーザが指定するEESで使用するデータの数です。

・データIDxのバイトサイズ

ユーザが指定する各ユーザ・データのバイトサイズです。

・終端領域(0x00)

終端情報として0を設定します。

5 ユーザインタフェース

5.1 リクエスト・ストラクチャー（st_ees_request_t）設定

データ・フラッシュへの書き込み、読み出し等の基本操作は一つの実行関数で実行されます。実行関数へリクエスト・ストラクチャー(st_ees_request_t)経由でコマンドやデータ ID を EES へ受け渡します。また、逆に EES の状態、エラー情報をリクエスト・ストラクチャー(st_ees_request_t)経由で取得します。

以降ユーザによるリクエスト・ストラクチャー(st_ees_request_t)への書き込みアクセスを「ユーザ・ライトアクセス」と呼び、読み出しアクセスを「ユーザ・リードアクセス」と呼びます。

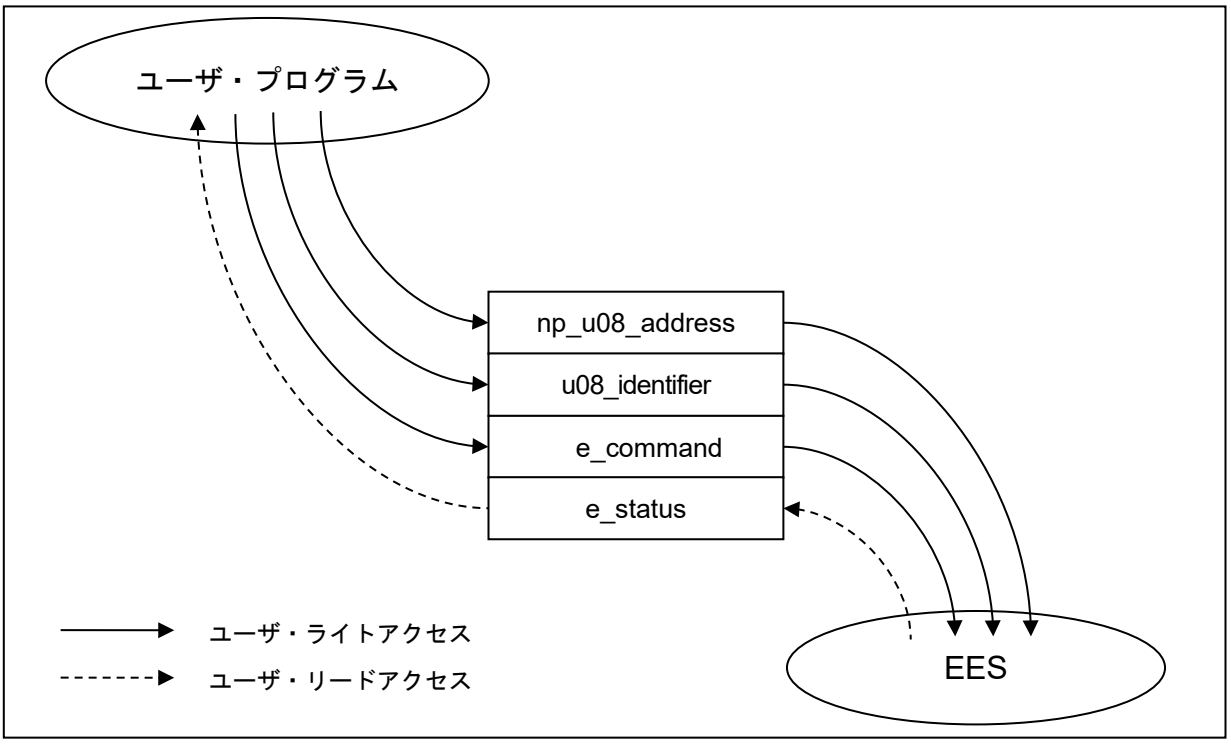


図 5-1 リクエスト・ストラクチャー(st_ees_request_t)

リクエスト・ストラクチャー(st_ees_request_t)はファイル r_ees_types.h に記述されています。ユーザによる変更は禁止です。

【リクエスト・ストラクチャー(st_ees_request_t)の定義】

```
typedef struct st_ees_request
{
  uint8_t __near *          np_u08_address;
  uint8_t                   u08_identifier;
  e_ees_command_t           e_command;
  e_ees_ret_status_t        e_status;
} st_ees_request_t;
```

uint8_t __near * np_u08_address	
uint8_t u08_identifier	e_ees_command_t e_command
e_ees_ret_status_t e_status	

bit0 bit15

図 5-2 リクエスト・ストラクチャー（st_ees_request_t）変数配置

5.1.1 ユーザ・ライトアクセス

(1) **np_u08_address**

R_EES_ENUM_CMD_WRITE コマンド、R_EES_ENUM_CMD_READ コマンド時に使用するデータ・バッファの先頭アドレスへのポインタを設定します。

対応コマンド名（マクロ名）	設定値
R_EES_ENUM_CMD_WRITE	データ・バッファ[注1] の先頭アドレスへのポインタ
R_EES_ENUM_CMD_READ	データ・バッファ[注2] の先頭アドレスへのポインタ

注1 ユーザの書き込むデータが配置されているバッファ
注2 データ・フラッシュから読み出したデータを配置するバッファ

(2) **u08_identifier**

各コマンドで使用するデータ ID を設定します。設定方法の詳細は「5.7 API 関数仕様 R_EES_Execute」のページをご参照ください。

対応コマンド名（マクロ名）	設定値
R_EES_ENUM_CMD_WRITE	書き込みデータの ID 指定
R_EES_ENUM_CMD_READ	読み出しデータの ID 指定

(3) **e_command**

共通実行関数へ設定するコマンド

コマンド名（マクロ名）	説明
R_EES_ENUM_CMD_UNDEFINED	コマンド未定義 （初期値：初期化以外で使用することはありません。）
R_EES_ENUM_CMD_STARTUP	スタートアップ処理
R_EES_ENUM_CMD_WRITE	書き込み処理
R_EES_ENUM_CMD_READ	読み出し処理
R_EES_ENUM_CMD_REFRESH	リフレッシュ処理
R_EES_ENUM_CMD_FORMAT	フォーマット処理
R_EES_ENUM_CMD_SHUTDOWN	シャットダウン処理

5.1.2 ユーザ・リードアクセス

– e_status

EES の状態、エラー情報。各関数において発生する可能性のある状態、エラーにつきましては「5.7 API 関数仕様」の各関数をご参照ください。

5.2 EES API 関数、および R_EES_Execute 関数のコマンド機能一覧

5.2.1 EES API 関数

EES RL78 Type01 の EES プールを制御する API 関数一覧を表 5-1 に示します。

表 5-1 EES RL78 Type01 API 関数一覧

	API 関数名	概要
1	R_EES_Init	すべての内部データ、変数の初期化、およびディスクリプタなどの構成のチェックを行います。
2	R_EES_Open	EEPROM エミュレーション準備処理 EEPROM エミュレーションを実行できる状態にします。
3	R_EES_Close	EEPROM エミュレーション終了処理 EEPROM エミュレーションを実行できない状態にします。
4	R_EES_Execute	EEPROM エミュレーション実行関数 EEPROM エミュレーションを操作するための各処理をコマンド形式で本関数の引数に設定し、処理を開始します。
5	R_EES_Handler	EEPROM エミュレーション継続実行処理 R_EES_Execute 関数で開始されたコマンドの処理を進行させ、終了を確認します。
6	R_EES_GetSpace	有効ブロックの空き容量を取得します。

5.2.2 R_EES_Execute 関数のコマンド

R_EES_Execute 関数において実行できる各コマンドの機能一覧を表 5-2 に示します。

表 5-2 R_EES_Execute 関数コマンドの機能一覧

	Command Name	Outline
1	R_EES_ENUM_CMD_STARTUP	【スタートアップ処理】 EES ブロックの状態を確認し、EEPROM エミュレーション（データ・アクセス）可能(Full Access)状態にします。有効ブロックが 2 個あった場合等は、不正な EES ブロックを無効ブロックに変更します。R_EES_ENUM_CMD_FORMAT コマンド以外のコマンドについては、必ず本コマンドを事前に実行し、正常終了させてください。
2	R_EES_ ENUM_CMD_WRITE 注 1	【書き込み処理】 EES ブロックへ指定データの書き込みを行います。 ※実行には以下の引数の設定が必要です。 ・np_u08_address：書き込みデータが保存されている RAM 領域の先頭アドレスへのポインタを指定 ・u08_identifier：書き込みデータのデータ ID 指定
3	R_EES_ ENUM_CMD_READ 注 1	【読み出し処理】 EES ブロックから指定データの読み出しを行います。 ※実行には以下の引数の設定が必要です。 ・np_u08_address：読み出したデータを保存する RAM 領域の先頭アドレスへのポインタを指定 ・u08_identifier：読み出すデータのデータ ID 指定
4	R_EES_ ENUM_CMD_REFRESH 注 1,2	【リフレッシュ処理】 有効ブロック（コピー元 EES ブロック）から EES プールの次のブロック（コピー先 EES ブロック）に対し、消去処理後に各データの最新の格納データをコピーします。これにより、コピー先 EES ブロックが新たな有効ブロックとなります。
5	R_EES_ ENUM_CMD_FORMAT	【フォーマット処理】 EES プール全体を記録されていたデータも含め、すべて初期化（消去）します。EEPROM エミュレーションを最初に使用する場合に必ず使用します。また、EES ブロックに異常が発生（有効ブロックがなくなる等）した場合や、ディスクリプタ・テーブル等の値（変更できない固定値）を修正する場合にも本コマンドを使用し、ブロック全体を初期化する必要があります。処理終了後は結果に関わらず必ず停止状態(opened)に遷移しますので、EEPROM エミュレーションを継続して使用する場合は、R_EES_ENUM_CMD_STARTUP コマンドを実行してください。
6	R_EES_ ENUM_CMD_SHUTDOWN 注 1	【シャットダウン処理】 EEPROM エミュレーションを停止状態(opened)にします。

注 1 R_EES_ENUM_CMD_STARTUP コマンドを正常に終了させてからコマンドを実行してください。

注 2 R_EES_ENUM_CMD_REFRESH コマンドを実行することで、消去処理が実行されます。

5.2.3 EES 用 RFD 制御 API 関数

EES 用 RFD 制御 API 関数一覧を表 5-3 に示します。

本関数は EES 内部で使用される関数で、ユーザが直接使用する必要はありません。

表 5-3 EES 用 RFD 制御 API 関数一覧

	API 関数名	概要
1	R_EES_EXRFD_Init	RFD RL78 Type01 の初期化を行います。
2	R_EES_EXRFD_Open	データ・フラッシュ・コントロール・レジスタ(DFLCTL)をデータ・フラッシュ・メモリへのアクセス許可状態(DFLEN = 1)に設定します。
3	R_EES_EXRFD_Close	データ・フラッシュ・コントロール・レジスタ（DFLCTL）をデータ・フラッシュ・メモリへのアクセス禁止状態（DFLEN = 0）に設定します。動作中のすべての EES 処理が停止します。
4	R_EES_EXRFD_Erase	EES ブロックの消去を開始します。
5	R_EES_EXRFD_Write	指定したデータ・フラッシュのアドレスに書き込みを開始します。
6	R_EES_EXRFD_BlankCheck	データ・フラッシュのブランク・チェック（対象データ・サイズ分）を開始します。
7	R_EES_EXRFD_Read	指定したアドレスからデータ（読み込みデータ・サイズ分）を読み出します。
8	R_EES_EXRFD_Handler	実行中の EES 用 RFD 制御関数の処理を進行し終了を確認します。

5.3 状態遷移

ユーザ・プログラムから EEPROM エミュレーションを使用するためには EES の初期化処理を行い、書き込みや読み出し等 EES ブロックの操作を行う関数を実行する必要があります。全体の状態遷移図を「図 5-3 状態遷移図」に、基本的な機能を使用するための操作フローを「図 **5-4** EES 基本フローチャート」に示します。EEPROM エミュレーションを使用する場合は、この流れに沿ってユーザ・プログラムに組み込んでください。

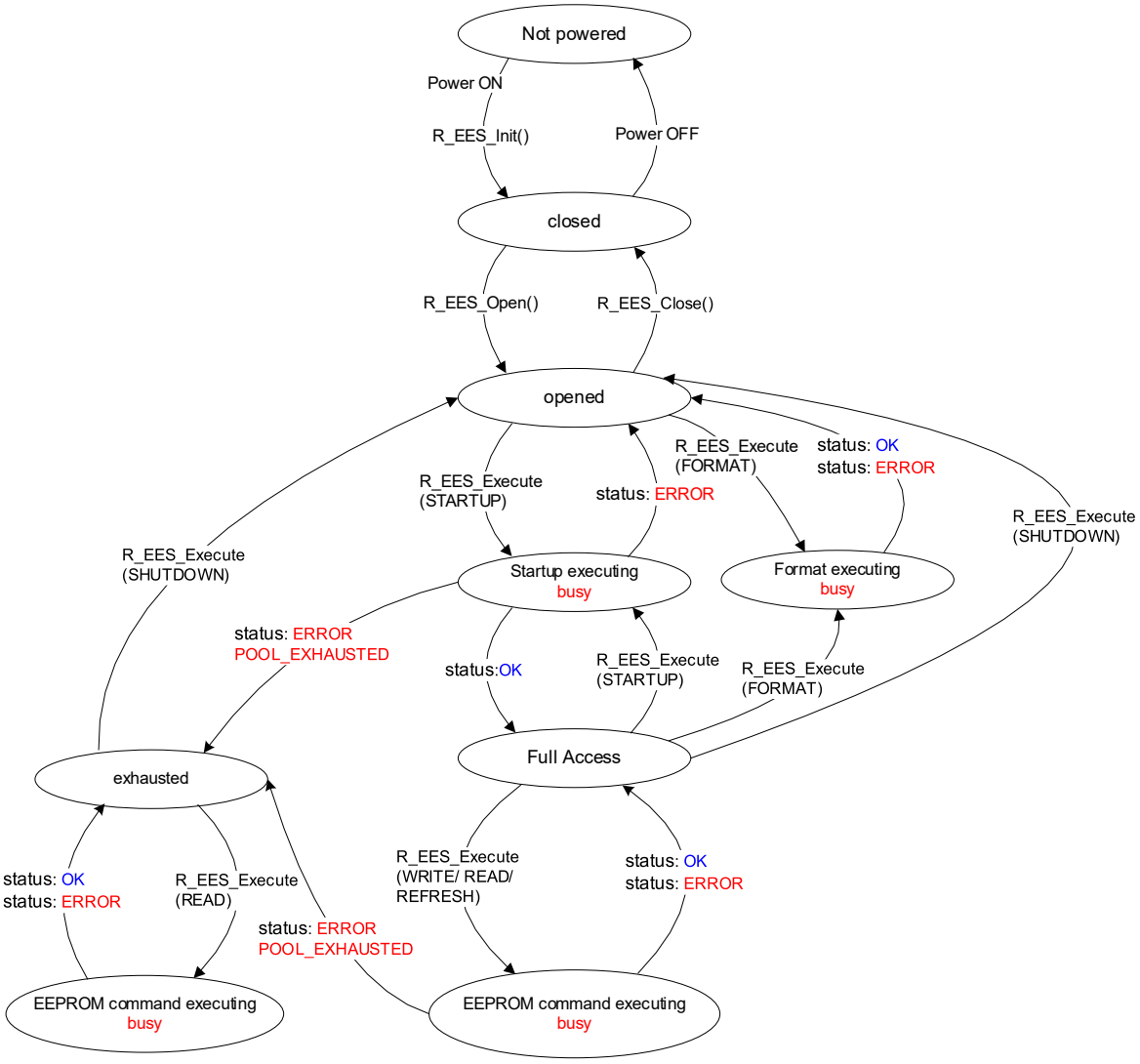


図 5-3 状態遷移図

注意1 R_EES_ENUM_CMD_FORMATコマンドを開始した場合、R_EES_Handler関数を実行して、必ず終了を確認してください。

【状態遷移図の概要】

EES を使用してデータ・フラッシュ・メモリを操作するためには、用意されている関数を順に実行し処理を進める必要があります。

(1) Not powered

Power OFF の状態です。

(2) closed

R_EES_Init 関数を実行し、EEPROM エミュレーションを実行するためのデータを初期化した状態（データ・フラッシュ・メモリへの操作は停止状態）です。EEPROM エミュレーションを動作させた後に RFD RL78 Type01 を使用したコード・フラッシュ・メモリの操作や、STOP モード、HALT モードを実行する場合は、opened 状態から R_EES_Close 関数を実行し、この状態に遷移させてください。

(3) opened

closed 状態から R_EES_Open 関数を実行し、データ・フラッシュ・メモリへの操作が可能になった状態です。R_EES_Close 関数を実行し、closed 状態に遷移するまでの間は RFD RL78 Type01 を使用したコード・フラッシュ・メモリの操作や STOP モード、HALT モードは実行できません。

(4) Full Access

opened 状態から R_EES_ENUM_CMD_STARTUP コマンドを実行し、EEPROM エミュレーションが実行できるようになった状態です。この状態から EEPROM エミュレーションを使用した書き込みや読み出しを行います。

(5) exhausted

opened 状態および Full Access 状態から、コマンド実行中に継続して使用できる EES ブロックがなくなった状態です。この状態では、R_EES_ENUM_CMD_READ コマンド、R_EES_ENUM_CMD_SHUTDOWN コマンドのみ実行できます。

(6) busy

指定された各コマンドを実行している状態です。実行コマンドと終了状況によっては遷移する状態が変わる場合もあります。

5.4 基本フローチャート

「図 **5-4** EES 基本フローチャート」に、EES を用いてデータ・フラッシュを操作（書き込み、読み出し等）する際の基本手順を示します。

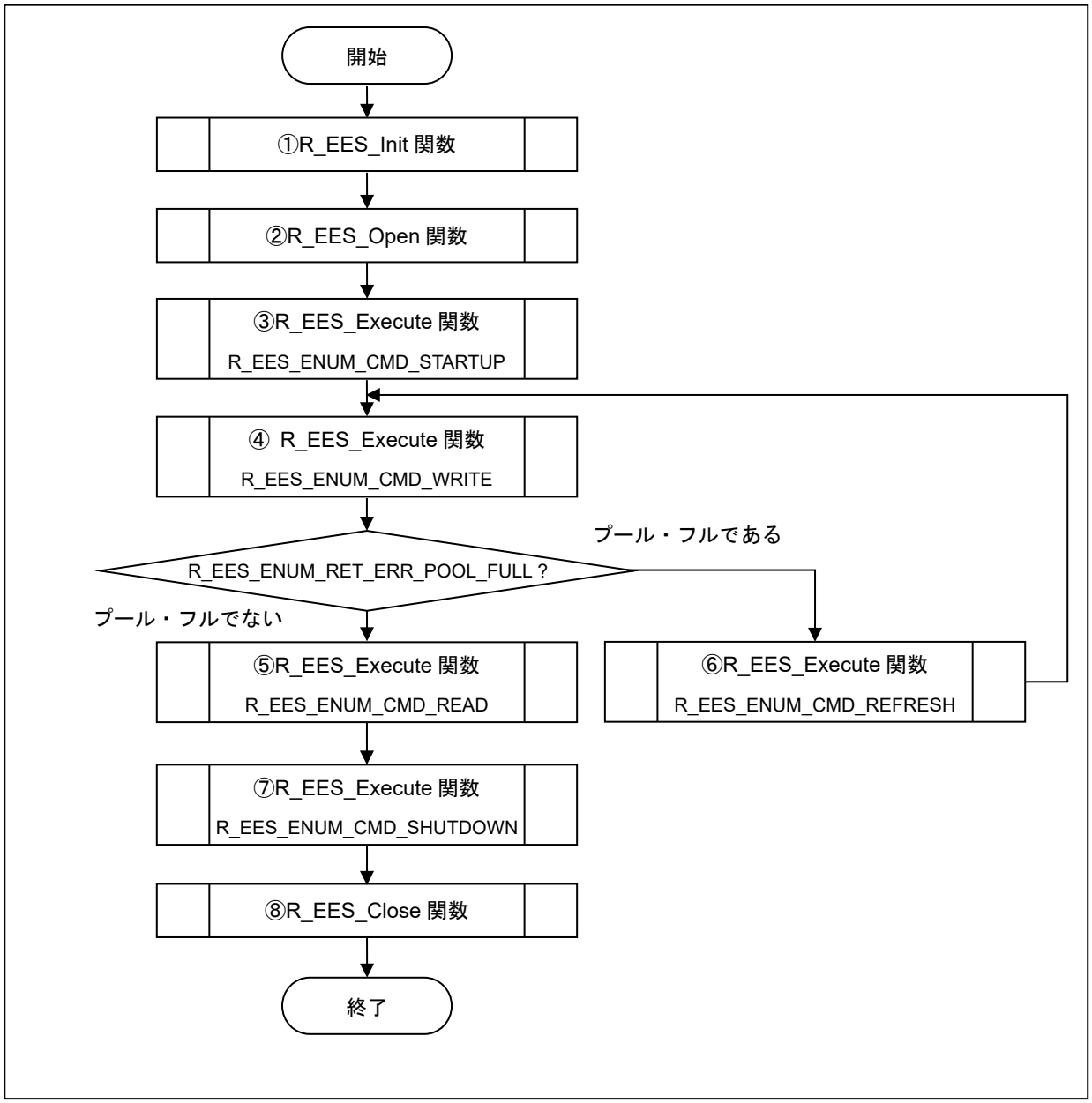


図 **5-4** EES 基本フローチャート

注意1　EEPROMエミュレーションを初めて使用する場合、必ずR_EES_ENUM_CMD_FORMATコマンドを実行してください。

注意2　上記フローは、コマンド実行後のR_EES_Handler処理とエラー処理を省略しています。

【基本操作フローの概要】

① EESの初期化処理 (R_EES_Init関数)
EESで使用する内部データ、変数の初期化、およびディスクリプタなどの構成のチェックを行います。

② EEPROMエミュレーション準備処理 (R_EES_Open関数)
EEPROMエミュレーションを実行するため、データ・フラッシュ・メモリを制御可能な状態(opened)にします。

③ EEPROMエミュレーション実行開始処理 (R_EES_Execute関数:R_EES_ENUM_CMD_STARTUPコマンド)
EEPROMエミュレーション (データ・アクセス)可能(Full Access)状態にします。

④ EEPROMエミュレーション・データ書き込み処理 (R_EES_Execute関数:R_EES_ENUM_CMD_WRITEコマンド)
指定されたIDのデータをEESブロックへ書き込みます。

⑤ EEPROMエミュレーション・データ読み出し処理 (R_EES_Execute関数:R_EES_ENUM_CMD_READコマンド)
指定されたIDのデータをEESブロックから読み出します。

⑥ EEPROMエミュレーション・リフレッシュ処理 (R_EES_Execute関数:R_EES_ENUM_CMD_REFRESHコマンド)
有効ブロック（コピー元ブロック）からEESプールの次のブロック（コピー先ブロック）に対し、消去処理後に各データの最新の格納データをコピーします。これにより、コピー先ブロックが新たな有効ブロックとなります。

⑦ EEPROMエミュレーション実行停止処理 (R_EES_Execute関数:R_EES_ENUM_CMD_SHUTDOWNコマンド)
EEPROMエミュレーションの動作を停止状態(opened)にします。

⑧ EEPROMエミュレーション終了処理 (R_EES_Close関数)
EEPROMエミュレーションを終了するために、データ・フラッシュ・メモリを制御出来ない状態(closed)にします。

5.5 コマンド操作フローチャート

「図 5-5 コマンド操作フローチャート」に、EES を用いてデータ・フラッシュを操作（書き込み、読み出し等）する際の基本手順を示します。

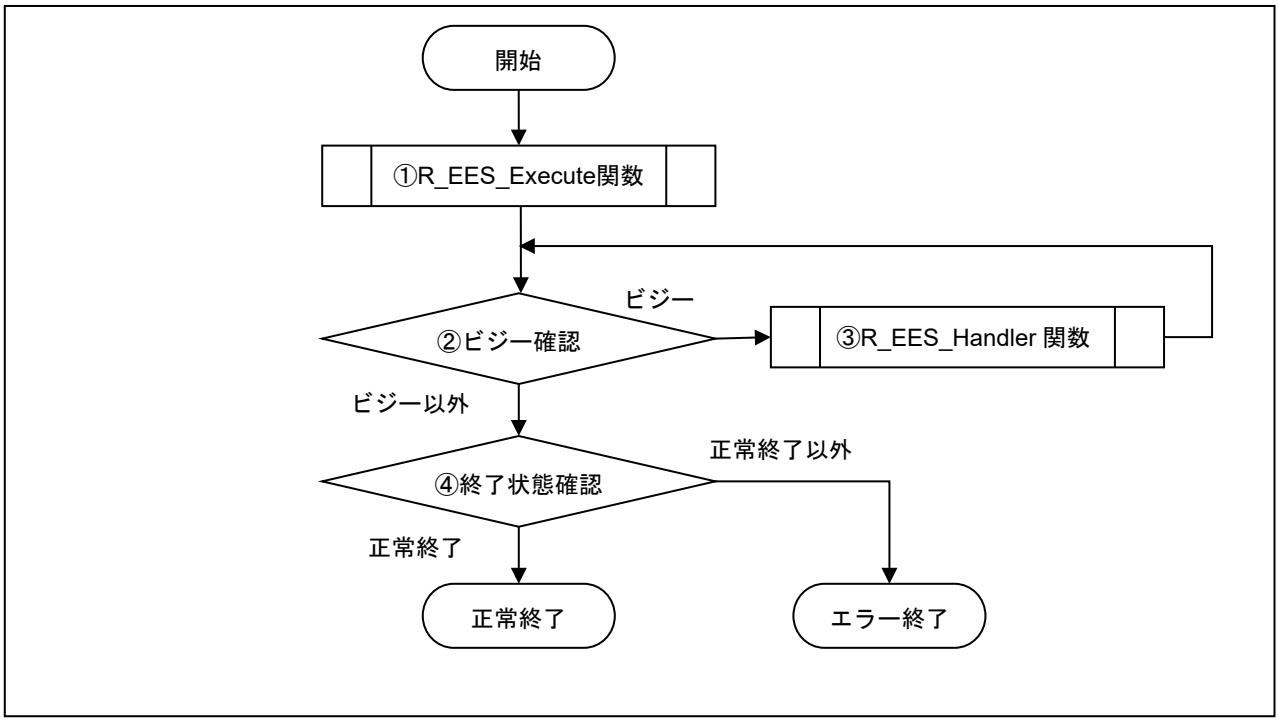


図 5-5 コマンド操作フローチャート

① R_EES_Execute 関数

データ・フラッシュ操作を実行します。

② ビジー確認

リクエスト・ストラクチャー（st_ees_request_t）の e_status を確認します。

R_EES_ENUM_RET_STS_BUSY であれば、引き続きデータ・フラッシュ操作を実行します。

R_EES_ENUM_RET_STS_BUSY 以外であれば終了状態確認をしてください。

③ R_EES_Handler 関数

実行中の EES を制御します。R_EES_Handler 関数を繰り返し実行することによってデータ・フラッシュ操作を進行させます。

④ 終了状態確認

R_EES_ENUM_RET_STS_OK であれば、正常終了です。R_EES_ENUM_RET_STS_OK 以外であればエラー終了してください。

5.6 データ型定義

5.6.1 データ型

EES RL78 Type01 のデータ型定義一覧を表 5-4 に示します。

表 5-4 EES RL78 Type01 データ型定義一覧

Macro value	Type	Description
int8_t	signed char	1byte signed integer
uint8_t	unsigned char	1byte unsigned integer
int16_t	signed short	2byte signed integer
uint16_t	unsigned short	2byte unsigned integer
int32_t	signed long	4byte signed integer
uint32_t	unsigned long	4byte unsigned integer
bool	unsigned char	Boolean (false:0 / true:1)

補足：これらのデータ型は C 言語規格 C99 以降では標準整数型として stdint.h と stdbool.h に定義されています。

5.6.2 グローバル変数

EES RL78 Type01 で使用するグローバル変数を以下に示します。

(1) g_ar_u08_ees_descriptor[R_EES_VALUE_U08_VAR_NO + 2u]

型 / 名称	uint8_t g_ar_u08_ees_descriptor[]
初期値	(uint8_t)(R_EES_VALUE_U08_VAR_NO), /* variable count */ (uint8_t)(sizeof(type_A)), /* id=1 */ (uint8_t)(sizeof(type_B)), /* id=2 */ (uint8_t)(sizeof(type_C)), /* id=3 */ (uint8_t)(sizeof(type_D)), /* id=4 */ (uint8_t)(sizeof(type_E)), /* id=5 */ (uint8_t)(sizeof(type_F)), /* id=6 */ (uint8_t)(sizeof(type_X)), /* id=7 */ (uint8_t)(sizeof(type_Z)), /* id=8 */ (uint8_t)(0x00u) /* zero terminator */
説明	各データ識別子(データID)のデータのサイズを格納
定義ファイル	r_ees_descriptor.c

(2) g_st_ees_exrfd_descriptor

型 / 名称	st_ees_exrfd_descriptor_t g_st_ees_exrfd_descriptor
初期値	(uint16_t) R_EES_EXRFD_VALUE_U16_PHYSICAL_BLOCK_SIZE (uint8_t) R_EES_EXRFD_VALUE_U08_PHYSICAL_BLOCKS_PER_VIRTUAL_BLOCK (uint8_t) R_EES_EXRFD_VALUE_U08_POOL_VIRTUAL_BLOCKS
説明	EESプールを構成している設定値を格納 - uint16_t u16_ees_physical_block_size; データ・フラッシュ・メモリ1ブロックのサイズ（物理ブロックサイズ） 例）RL78/G2xでは(256u)固定 - uint8_t u08_ees_physical_blocks_per_virtual_block; EESブロックに設定するデータ・フラッシュ・メモリ・ブロック数（物理ブロック数） 例）EESブロック512byteの場合は、物理ブロック数に(2u)を使用 例）EESブロック1024byteの場合は、物理ブロック数に(4u)を使用 - uint8_t u08_ees_pool_virtual_blocks; EESプールサイズ（仮想ブロック数） 例）EESブロック総数(4u)個
定義ファイル	r_ees_descriptor.c

(3) g_ar_u16_ram_ref_table[R_EES_VALUE_U08_VAR_NO]

型 / 名称	uint16_t g_ar_u16_ram_ref_table[]
初期値	-
説明	各データ識別子(データID)の参照情報を格納
定義ファイル	r_ees_descriptor.c

5.6.3 列挙型

- e_ees_command (列挙変数名 : e_ees_command_t)

EES 実行コマンド

Symbol Name	Value	Description
R_EES_ENUM_CMD_UNDEFINED	0x00	コマンド未定義（初期値）
R_EES_ENUM_CMD_STARTUP	0x01	スタートアップ処理
R_EES_ENUM_CMD_WRITE	0x02	書き込み処理
R_EES_ENUM_CMD_READ	0x03	読み出し処理
R_EES_ENUM_CMD_REFRESH	0x04	リフレッシュ処理
R_EES_ENUM_CMD_FORMAT	0x06	フォーマット処理
R_EES_ENUM_CMD_SHUTDOWN	0x07	シャットダウン処理

- e_ees_ret_status (列挙変数名 : e_ees_ret_status_t)

EES 関数戻り値

Symbol Name	Value	Description
R_EES_ENUM_RET_STS_OK	0x00	正常終了
R_EES_ENUM_RET_STS_BUSY	0x01	コマンド実行中
R_EES_ENUM_RET_ERR_CONFIGURATION	0x82	EES 構成エラー
R_EES_ENUM_RET_ERR_INITIALIZATION	0x83	EES 初期化エラー
R_EES_ENUM_RET_ERR_ACCESS_LOCKED	0x84	EEPROM エミュレーション・ロック・エラー
R_EES_ENUM_RET_ERR_PARAMETER	0x85	パラメータ・エラー
R_EES_ENUM_RET_ERR_WEAK	0x86	書き込み不十分エラー
R_EES_ENUM_RET_ERR_REJECTED	0x87	リジェクト・エラー
R_EES_ENUM_RET_ERR_NO_INSTANCE	0x88	データ未書き込みエラー
R_EES_ENUM_RET_ERR_POOL_FULL	0x89	プール・フル・エラー
R_EES_ENUM_RET_ERR_POOL_INCONSISTENT	0x8A	EES ブロック不整合エラー
R_EES_ENUM_RET_ERR_POOL_EXHAUSTED	0x8B	EES ブロック消耗エラー
R_EES_ENUM_RET_ERR_INTERNAL	0x8C	内部エラー
R_EES_ENUM_RET_ERR_FLASH_SEQ	0x8D	フラッシュ・シーケンサー・エラー

- e_ees_exrfd_ret_status (列挙変数名 : e_ees_exrfd_ret_status_t)

本列挙型は EES 内部で使用される列挙型で、ユーザが直接使用する必要はありません。

EES 用 RFD 制御関数戻り値

Symbol Name	Value	Description
R_EES_EXRFD_ENUM_RET_STS_OK	0x00	正常終了
R_EES_EXRFD_ENUM_RET_STS_BUSY	0x01	コマンド実行中
R_EES_EXRFD_ENUM_RET_ERR_CONFIGURATION	0x10	構成エラー
R_EES_EXRFD_ENUM_RET_ERR_INITIALIZATION	0x11	初期化エラー
R_EES_EXRFD_ENUM_RET_ERR_REJECTED	0x12	リジェクト・エラー
R_EES_EXRFD_ENUM_RET_ERR_PARAMETER	0x13	パラメータ・エラー
R_EES_EXRFD_ENUM_RET_ERR_INTERNAL	0x14	内部エラー
R_EES_EXRFD_ENUM_RET_ERR_MODE_MISMATCHED	0x20	モード不一致エラー
R_EES_EXRFD_ENUM_RET_ERR_CFDF_SEQUENCER	0x21	シーケンサ・エラー
R_EES_EXRFD_ENUM_RET_ERR_ERASE	0x22	消去処理エラー
R_EES_EXRFD_ENUM_RET_ERR_BLANKCHECK	0x23	ブランク・チェック処理エラー
R_EES_EXRFD_ENUM_RET_ERR_WRITE	0x24	書き込み処理エラー

5.7 API 関数仕様

この章では、EEPROM Emulation Software (EES) RL78 Type01 の API 関数の詳細仕様について説明します。EES RL78 Type01 の API 関数を使用して、フラッシュ・メモリの書き換えを実施する上での前提条件があります。この前提条件と異なる条件で EES RL78 Type01 の API 関数を使用した場合、各関数の動作が不定となる可能性がありますので、ご注意ください。

≪前提条件≫

・R_EES_Init 関数は、全ての EES 関数を使用する前に、1 回実行してください。
・セルフ・プログラミング実行中は、高速オンチップ・オシレータを起動しておく必要があります。EES RL78 Type01 の全ての API 関数は、高速オンチップ・オシレータが起動している状態で実行してください。
・EEPROM エミュレーションを操作する場合、データ・フラッシュへのアクセスを許可した状態で EES RL78 Type01 の API 関数を実行してください。データ・フラッシュへのアクセス許可方法については、対象デバイスの「ユーザーズマニュアル：ハードウェア編」を参照してください。

以下に API 関数仕様の記述例を示します。

≪API 関数仕様の記述例≫

Information

Syntax	この関数を C 言語で記述されたプログラムから呼び出す際の書式を示します。	
Reentrancy	再帰可否 ：Reentrant（再帰可能）、または Non-reentrant（再起不可）。	
Parameters (IN)	この関数の引数（入力）	引数［値、範囲、引数の意味等］
Parameters (IN/OUT)	この関数の引数（入出力）	引数［値、範囲、引数の意味等］
Parameters (OUT)	この関数の引数（出力）	引数［値、範囲、引数の意味等］
Return Value	この関数からの戻り値の型 （列挙型、ポインタ等）	戻り値の列挙子（定数）：値 ［定数の意味：詳細説明］
		戻り値の列挙子（定数）：値 ［定数の意味：詳細説明］
Description	機能概要	
Preconditions	事前条件の概要	
Remarks	特記事項	

動作概要：

この関数の機能概要を示します。

備考：

この関数の使用条件や制限事項を示します。

5.7.1 EES RL78 Type01 EEPROM エミュレーション制御関数仕様

EEPROM エミュレーションを制御する API 関数を示します。

5.7.1.1 R_EES_Init

Information

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_Init(uint8_t i_u08_cpu_frequency);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_cpu_frequency	CPU 動作周波数 [1–32(MHz)]（対象：全デバイス） [48(MHz)]（対象：RL78/G24）
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK : 0x00 [正常終了] R_EES_ENUM_RET_ERR_CONFIGURATION : 0x82 [EES 構成エラー]
Description	すべての内部データ、変数の初期化、およびディスクリプタなどの構成のチェックを行います。	
Preconditions	高速オンチップ・オシレータを起動している状態で実行してください。	
Remarks	EES 関数を使用する前に、1 回実行してください。	

動作概要：

・引数(CPU 動作周波数)を R_EES_EXRFD_Init 関数に設定して実行します 。

備考：

・EES プールや、EES ブロックサイズなどの EEPROM エミュレーションを実行するための構成が異常な場合、リターン値に EES 構成エラー(R_EES_ENUM_RET_ERR_CONFIGURATION)が返ります。

・EEPROM エミュレーション実行中は、高速オンチップ・オシレータを起動しておく必要があります。高速オンチップ・オシレータが起動している状態で、本関数を実行してください。
※EES RL78 Type01 では、高速オンチップ・オシレータの起動やチェックは行っていません。

・引数(i_u08_cpu_frequency)には、実際に CPU が動作する周波数の値の小数点以下を切り上げた整数値を設定します。（例：CPU が動作する周波数が 4.5MHz の場合は、初期化関数で 5 を設定してください）
CPU の動作周波数を 4 MHz 未満で使用する場合は、1 MHz, 2 MHz, 3 MHz を使用することができます。その際、整数値でない周波数(1.5MHz など)は使用できません。
引数(i_u08_cpu_frequency)に設定する周波数は、フラッシュ書き換え時、実際に CPU が動作する周波数であり、必ずしも高速オンチップ・オシレータの周波数を設定するということではありません。
- CPU 動作周波数と異なる値を指定した場合、その後の動作は不定となります。その際、フラッシュの書き換えが完了した場合でも、データの値、及びその後の保持期間を満たすことができない可能性があります。
※CPU 動作周波数の範囲については、対象デバイスの「ユーザーズマニュアル：ハードウェア編」を参照してください。

5.7.1.2 R_EES_Open

Information

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_Open(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
arameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK：0x00 [正常終了] R_EES_ENUM_RET_ERR_REJECTED：0x87 [リジェクト・エラー]
Description	EEPROM エミュレーション準備処理 EEPROM エミュレーションを実行できる状態にします。	
Preconditions	R_EES_Init 関数を正常終了させていること。	
Remarks		

動作概要：

・R_EES_EXRFD_Open 関数を実行し、データ・フラッシュ・メモリにアクセスできる状態にします。

備考：

・R_EES_Init 関数を実行せず、内部変数が初期化されていなかった場合、リターン値にはリジェクト・エラー(R_EES_ENUM_RET_ERR_REJECTED)が返ります。

5.7.1.3 R_EES_Close

Information

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_Close(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK [正常終了]
Description	EEPROM エミュレーション終了処理 EEPROM エミュレーションを実行できない状態にします。	
Preconditions	-	
Remarks	-	

動作概要：

・R_EES_EXRFD_Close 関数を実行し、EEPROM エミュレーションを終了します。

備考：

・EEPROM エミュレーションを実行していた場合は、R_EES_ENUM_CMD_SHUTDOWN コマンドで EEPROM エミュレーションを停止状態（opened 状態）にしてから実行します。

5.7.1.4 R_EES_Execute

Information

Syntax	R_EES_FAR_FUNC void R_EES_Execute(st_ees_request_t __near * ionp_st_ees_request);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	st_ees_request_t __near * ionp_st_ees_request	リクエスト・ストラクチャー(st_ees_request_t)のポインタ
Parameters (OUT)	N/A	
Return Value	N/A	
Description	EEPROM エミュレーション実行関数 EEPROM エミュレーションを操作するための各処理をコマンド形式で本関数の引数に設定し、処理を開始します。	
Preconditions	R_EES_Init, R_EES_Open 関数を正常終了させていること。	
Remarks	-	

動作概要：

・リクエスタに設定されたコマンドを設定し処理を開始します。

備考：

・R_EES_Execute 関数は、コマンドの処理を開始させたのち、制御を直ちにユーザ・プログラムに戻します。コマンドの処理の継続は R_EES_Handler の実行によって行われます。そのため、コマンドの処理が完了するまで、R_EES_Handler 関数を継続的に実行しなければいけません。

・リクエスト・ストラクチャー(st_ees_request_t)の e_status が R_EES_ENUM_RET_STS_BUSY の間は、繰り返し R_EES_Handler 関数を実行してください。

・割り込み処理内で R_EES_Execute 関数の呼び出しは許可していません。

R_EES_Execute / R_EES_Handler のコマンド実行状態(e_status)一覧(1/2)

コマンド実行状態	カテゴリ	説　明	対応コマンド
R_EES_ENUM_RET_STS_OK	説明	正常終了	すべてのコマンド
	原因	なし	
	対処方法	なし	
R_EES_ENUM_RET_STS_BUSY	説明	コマンド実行中	R_EES_ENUM_CMD_SHUTDOWN 以外のコマンド
	原因	なし	
	対処方法	状態が変化するまでR_EES_Handler関数を呼び出してください。	
R_EES_ENUM_RET_ERR_INITIALIZATION	説明	EES初期化エラー	すべてのコマンド
	原因	R_EES_Init関数、R_EES_Open関数が正常に完了していません。	
	対処方法	R_EES_Init関数、R_EES_Open関数を正常に完了させてください。	
R_EES_ENUM_RET_ERR_ACCESS_LOCKED	説明	EEPROMエミュレーション・ロック・エラー	R_EES_ENUM_CMD_STARTUP、R_EES_ENUM_CMD_FORMAT以外のコマンド
	原因	EEPROMエミュレーションが実行できない状態です。	
	対処方法	R_EES_ENUM_CMD_STARTUPコマンドを正常に終了させてください。	
R_EES_ENUM_RET_ERR_PARAMETER	説明	パラメータ・エラー	すべてのコマンド
	原因	コマンドの設定パラメータに誤りがあります。	
	対処方法	設定したパラメータを見直してください。	
R_EES_ENUM_RET_ERR_WEAK	説明	ブロック・ヘッダもしくは最後に書き込まれた格納データ書き込みが正常に完了していません。	R_EES_ENUM_CMD_STARTUP
	原因	有効ブロック・ヘッダ、もしくは書き込まれた格納データの書き込み処理が中断された可能性があります。	
	対処方法	R_EES_ENUM_CMD_REFRESHコマンドを実行してください。	
R_EES_ENUM_RET_ERR_REJECTED	説明	リジェクト・エラー	すべてのコマンド
	原因	別コマンドが実行中です。	
	対処方法	R_EES_Handler関数を呼び出して実行中のコマンドを終了させてください。	

R_EES_Execute / R_EES_Handler のコマンド実行状態(e_status)一覧(2/2)

コマンド実行状態	カテゴリ	説　明	対応コマンド
R_EES_ENUM_RET_ERR_NO_INSTANCE	説明	データ未書き込みエラー	R_EES_ENUM_CMD_READ
	原因	指定された識別子のデータが書き込まれていません。	
	対処方法	R_EES_ENUM_CMD_WRITEコマンドで指定された識別子にデータを書いてください。	
R_EES_ENUM_RET_ERR_POOL_FULL	説明	プール・フル・エラー	R_EES_ENUM_CMD_WRITE
	原因	データを書き込める領域が存在しません。	
	対処方法	R_EES_ENUM_CMD_REFRESHを実行し、書き込みを再実行してください。	
R_EES_ENUM_RET_ERR_POOL_INCONSISTENT	説明	EESブロック不整合エラー	R_EES_ENUM_CMD_STARTUP
	原因	EESブロックが不定状態(有効ブロックがない等)です。	
	対処方法	R_EES_ENUM_CMD_FORMATコマンドを実行し、EESブロックを初期化してください。	
R_EES_ENUM_RET_ERR_POOL_EXHAUSTED	説明	EES ブロック消耗エラー	R_EES_ENUM_CMD_STARTUP R_EES_ENUM_CMD_FORMAT R_EES_ENUM_CMD_REFRESH R_EES_ENUM_CMD_WRITE
	原因	継続して使用できる EES ブロックがなくなりました。	
	対処方法	EEPROMエミュレーションを終了してください。 R_EES_ENUM_CMD_FORMATコマンドを実行し修復(既存のデータはすべて消去されます)を試行、もしくは既存データの読み出しのみ実行可能。	
R_EES_ENUM_RET_ERR_INTERNAL	説明	内部エラー	R_EES_ENUM_CMD_SHUTDOWN 以外のコマンド
	原因	予期しないエラーが発生しました。	
	対処方法	EESを終了してください。 デバイス状態を確認してください。	
R_EES_ENUM_RET_ERR_FLASH_SEQ	説明	フラッシュ・シーケンサー・エラー	R_EES_ENUM_CMD_SHUTDOWN 以外のコマンド
	原因	フラッシュ・メモリ・モードの変更、またはフラッシュ・シーケンサーの起動に失敗しました。	
	対処方法	EESを終了してください。 EEPROMエミュレーションの操作以外で、RFD RL78 Type01を使用したフラッシュ・メモリの操作を実行していないか確認してください。	

5.7.1.5 R_EES_Handler

Information

Syntax	R_EES_FAR_FUNC void R_EES_Handler(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	EEPROM エミュレーション継続実行処理 R_EES_Execute 関数で開始されたコマンドの処理を進行させ、終了を確認します。	
Preconditions	R_EES_Init, R_EES_Open 関数を正常終了させていること。	
Remarks	-	

動作概要：

・R_EES_Execute 関数で開始された EEPROM エミュレーションの処理を進行させます。

備考：

・リクエスト・ストラクチャー(st_ees_request_t)の e_status が R_EES_ENUM_RET_STS_BUSY の間は、繰り返し R_EES_Handler 関数を実行してください。

・割り込み処理内で R_EES_Handler 関数の呼び出しは許可していません。

・R_EES_Handler 関数のコマンド実行状態は R_EES_Execute 関数の引数で使用されたリクエスト・ストラクチャー(st_ees_request_t)に設定されます。そのため、R_EES_Handler 関数を使用する場合、リクエスト・ストラクチャー(st_ees_request_t)を解放しないでください。

5.7.1.6 R_EES_GetSpace

Information

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_GetSpace(uint16_t __near * onp_u16_space);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	uint16_t __near * onp_u16_space	現在の有効ブロックの空き容量の情報が入力される変数へのポインタ
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK：0x00 [正常終了] R_EES_ENUM_RET_ERR_INITIALIZATION：0x83 [EES 初期化エラー] R_EES_ENUM_RET_ERR_ACCESS_LOCKED：0x84 [EEPROM エミュレーション・ロック・エラー] R_EES_ENUM_RET_ERR_REJECTED：0x87 [リジェクト・エラー]
Description	有効ブロックの空き容量を取得します。	
Preconditions	R_EES_Init, R_EES_Open 関数を正常終了させていること。 R_EES_Execute 関数で R_EES_ENUM_CMD_STARTUP コマンドを正常に終了させていること。	
Remarks		

動作概要：

・有効ブロックの空き容量を計算します。

備考：

・R_EES_Init 関数を実行せず、内部変数が初期化されていなかった場合、リターン値には EES 初期化エラー(R_EES_ENUM_RET_ERR_INITIALIZATION)が返ります。

・R_EES_Execute 関数で R_EES_ENUM_CMD_STARTUP コマンドが正常に終了していなかった場合、リターン値には EEPROM エミュレーション・ロック・エラー(R_EES_ENUM_RET_ERR_ACCESS_LOCKED)が返ります。

・R_EES_Execute 関数で EES のコマンド処理が実行中の場合、リターン値にはリジェクト・エラー(R_EES_ENUM_RET_ERR_REJECTED)が返ります。

・EES プールがプール消耗状態の場合、空き容量には常に 0x0000 が戻ります。

・有効ブロック・ヘッダ、もしくは書き込まれた格納データの書き込み処理が中断された可能性がある場合、空き容量には 0x0000 が戻ります。

・エラー値が戻る場合、空き容量の情報は取得されません。

5.7.2 EES 用 RFD 制御関数

RFD を制御する API 関数を示します。これらの関数は、EEPROM エミュレーション制御関数から呼び出されます。ユーザ・プログラムからは直接呼び出さないでください。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Init(uint8_t i_u08_cpu_frequency);
Description	RFD RL78 Type01 の初期化を行います。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Open(void);
Description	データ・フラッシュ・コントロール・レジスタ(DFLCTL)をデータ・フラッシュ・メモリへのアクセス許可状態(DFLEN = 1)に設定します。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Close(void);
Description	データ・フラッシュ・コントロール・レジスタ（DFLCTL）をデータ・フラッシュ・メモリへのアクセス禁止状態（DFLEN = 0）に設定します。動作中のすべての EES 処理が停止します。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Erase(uint8_t i_u08_virtual_block_number);
Description	EES ブロックの消去を開始します。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Write(uint16_t i_u16_offset_addr, uint8_t __near * inp_u08_write_data, uint16_t i_u16_size);
Description	指定したデータ・フラッシュのアドレスに書き込みを開始します。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_BlankCheck(uint16_t i_u16_offset_addr, uint16_t i_u16_size);
Description	データ・フラッシュのブランク・チェック（対象データ・サイズ分）を開始します。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Read(uint16_t i_u16_offset_addr, uint8_t __near * onp_u08_read_data, uint16_t i_u16_size);
Description	指定したアドレスからデータ（読み込みデータ・サイズ分）を読み出します。

Information

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Handler(void);
Description	実行中の EES 用 RFD 制御関数の処理を進行し終了を確認します。

Information

Syntax	static R_EES_FAR_FUNC e_ees_exrfd_ret_status_t r_ees_exrfd_get_seq_error_status(void);
Description	データ・フラッシュ・メモリ・シーケンサから処理結果を取得します。

Information

Syntax	static R_EES_FAR_FUNC e_ees_exrfd_ret_status_t r_ees_exrfd_finish_state(void);
Description	EES 用 RFD 制御関数を終了状態にします。

Information

Syntax	static R_EES_FAR_FUNC e_ees_exrfd_ret_status_t r_ees_exrfd_check_cmd_executable(void);
Description	EES 用 RFD 制御関数の実行状態とフラグを確認します。

Information

Syntax	static R_EES_FAR_FUNC bool r_ees_exrfd_is_valid_byte_parameter(uint16_t i_u16_offset_addr, uint16_t i_u16_size);
Description	EES 用 RFD 制御関数で使用するパラメータを確認します。

Information

Syntax	static R_EES_FAR_FUNC void r_ees_exrfd_clear_cmd_workarea(void);
Description	EES 用 RFD 制御関数で使用するデータ領域をクリアします。

Information

Syntax	static R_EES_FAR_FUNC void r_ees_exrfd_blankcheck_byte_req(uint32_t i_u32_start_addr, uint16_t i_u16_size);
Description	データ・フラッシュのブランク・チェック（指定サイズ分）を開始します。

5.7.3 EEPROM エミュレーションを制御する API 関数用内部関数

EEPROM エミュレーションを制御する関数内で使用されている内部関数を示します。ユーザ・プログラムからは直接呼び出さないでください。

Information

Syntax	R_EES_FAR_FUNC bool r_ees_is_valid_configuration(void);
Description	EES の構成を確認し、使用する内部データを初期化します。

Information

Syntax	R_EES_FAR_FUNC bool r_ees_is_valid_requester(st_ees_request_t __near * ionp_st_ees_request);
Description	リクエスト・ストラクチャーと EES の状態を確認し、内部ステータスを更新します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_startup_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_startup_state_09(void);
Description	スタートアップ処理用の内部ステータスを更新します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_write_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_write_state_04(void);
Description	書き込み処理用の内部ステータスを更新します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_read_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_read_state_01(void);
Description	読み込み処理用の内部ステータスを更新します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_refresh_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_refresh_state_17(void);
Description	リフレッシュ処理用の内部ステータスを更新します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_format_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_format_state_11(void);
Description	フォーマット処理用の内部ステータスを更新します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_shutdown_state_00(void);
Description	EEPROM エミュレーションのシャットダウン処理をします。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_erase_state_00(void);
Description	消去処理を開始します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_bw_state_00(void);
Description	ブランク・チェック処理と書き込み処理を開始します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_inner_blankcheck_state_00(void);
Description	ブランク・チェックの内部処理を開始します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_write_state_00(void);
Description	書き込み処理を開始します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_inner_write_state_00(void);
Description	書き込みの内部処理を開始します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_read_state_00(void);
Description	読み込み処理を開始します。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_state_01(void);
Description	開始した EES 用 RFD 制御関数の内部処理を進めます。

Information

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exit_state(void);
Description	ダミー処理

Information

Syntax	static R_EES_FAR_FUNC uint8_t r_ees_calculate_next_a_flag(uint8_t i_u08_a_flag_value);
Description	A フラグの値を計算します。

Information

Syntax	static R_EES_FAR_FUNC void r_ees_fsm_finish_command(void);
Description	実行コマンドの終了処理を行います。

Information

Syntax	static R_EES_FAR_FUNC void r_ees_fsm_swap_acvive_block_info(void);
Description	有効ブロックの情報を入れ替えます。

Information

Syntax	static R_EES_FAR_FUNC bool r_ees_fsm_exrfd_cmd_detect_fatal_error(e_ees_exrfd_ret_status_t i_e_ees_exrfd_ret_value);
Description	EES 用 RFD 制御処理結果に EES が継続実行不可となるエラーがないか確認します。

Information

Syntax	static R_EES_FAR_FUNC e_ees_block_status_t r_ees_fsm_get_ees_block_status(void);
Description	EES ブロックの状態を取得します。

6 サンプル・プログラム

EES RL78 Type01 に添付しているサンプル・プログラムについて説明します。

この章では、RL78/G23 用のサンプル・プログラムを例に説明しています。RL78/G23 以外を使用する場合は、“G23”を対象のデバイスに読みかえてください。

6.1 ファイル構成

6.1.1 フォルダ構成

図 6-1 は、RL78/G23 を使用する場合の例です。実際にインストールした“sample”フォルダには、デバイスグループごとのサンプル用フォルダが含まれます（例: RL78_G23, RL78_G24）。RL78/G23 以外を使用する場合は、“G23”を対象のデバイスに読みかえてください。

RL78/G24 を使用する場合のフォルダ名：“RL78_G24”

サンプル・プログラムを使用する場合は、対象デバイス用のフォルダのみインクルードしてください。

サンプル・プログラムのフォルダ構成を図 6-1 に示します。

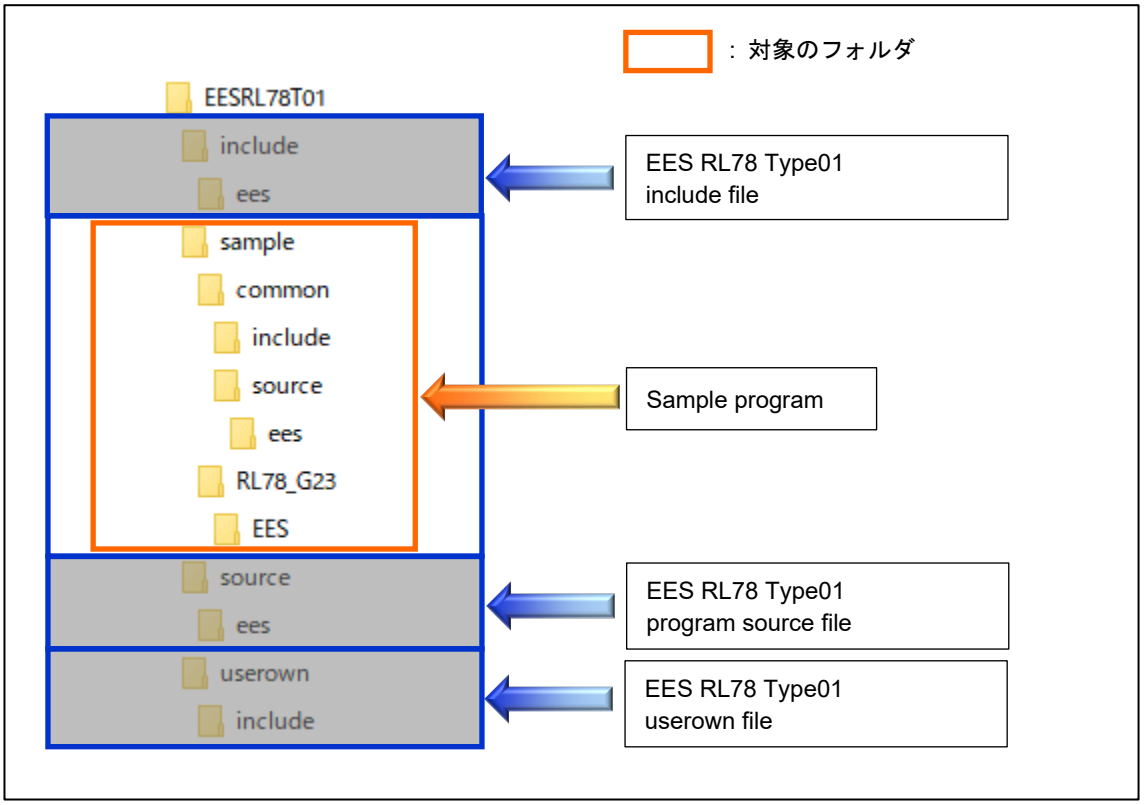


図 6-1 サンプル・プログラムのフォルダ構成

6.1.2 ファイル・リスト

6.1.2.1 ソース・ファイル・リスト

“sample\common\source\ees\”フォルダ内のプログラム・ソース・ファイルを表 6-1 に示します。

表 6-1 “sample\common\source\ees\”フォルダ内プログラム・ソース・ファイル

No	ソース・ファイル名	概略(Summary)
1	sample_control_ees.c	EEPROM エミュレーション制御用関数サンプル・ファイル

“sample\RL78_G23”フォルダ内のメイン処理のプログラム・ソース・ファイルを表 6-2 に示します。

“sample\RL78_G23\EES\[コンパイラ名]\source\”フォルダ

表 6-2 メイン処理のプログラム・ソース・ファイル

No	ソース・ファイル名	概略(Summary)
1	main.c	メイン処理関数サンプル・ファイル

6.1.2.2 ヘッダ・ファイル・リスト

“sample\common\include\”フォルダ内のプログラム・ヘッダ・ファイルを表 6-3 に示します。

表 6-3 “sample\common\include\”フォルダ内プログラム・ヘッダ・ファイル

No	ヘッダ・ファイル名	概略(Summary)
1	sample_control_ees.h	EES を制御する関数サンプルのプロトタイプ宣言を定義したファイル
2	sample_ees_defines.h	EES を制御する関数サンプルのマクロを定義したファイル
3	sample_ees_memmap.h	EES を制御する関数サンプルで使用するセクションを記述するためのマクロを定義したファイル

6.2 データ型定義

6.2.1 マクロ定義

- 周波数設定マクロ

サンプルで使用している CPU の動作に使用している周波数。

Symbol Name	Value	Description
SAMPLE_VALUE_U08_CPU_FREQUENCY	32u	CPU の動作周波数（RL78_G23 フォルダ）
	48u	CPU の動作周波数（RL78_G24 フォルダ）

6.3 サンプル・プログラム関数

サンプル・プログラム関数一覧を表 6-4 に示します。

表 6-4 サンプル・プログラム関数一覧

	API Name	Outline
1	main	EES 制御サンプル・プログラムのメイン関数
2	Sample_EES_Control	EES の基本的な使用手順に従い、EES ブロックの書き込み、読み出しを実行します。

6.3.1 EEPROM エミュレーション制御サンプル・プログラム

EES RL78 Type01 の書き換え制御サンプルでは、EES を使用するための基本的な操作手順に従い EES ブロックの書き換え、読み出し処理を実行します。

注）EES のコマンド処理を実行中は、データ・フラッシュ上のデータを参照できないため、参照するデータは、事前に RAM へコピーして、RAM 上で参照する必要があります。

動作条件(RL78/G23 用サンプル・プログラムの例) :

・CPU 動作周波数: 32MHz

（メイン・システム・クロックに高速オンチップ・オシレータ・クロック(HOCO)を使用）

EES RL78 Type01 のサンプルのメイン処理実行フローを図 6-2 に示します。

6.3.1.1 main 関数

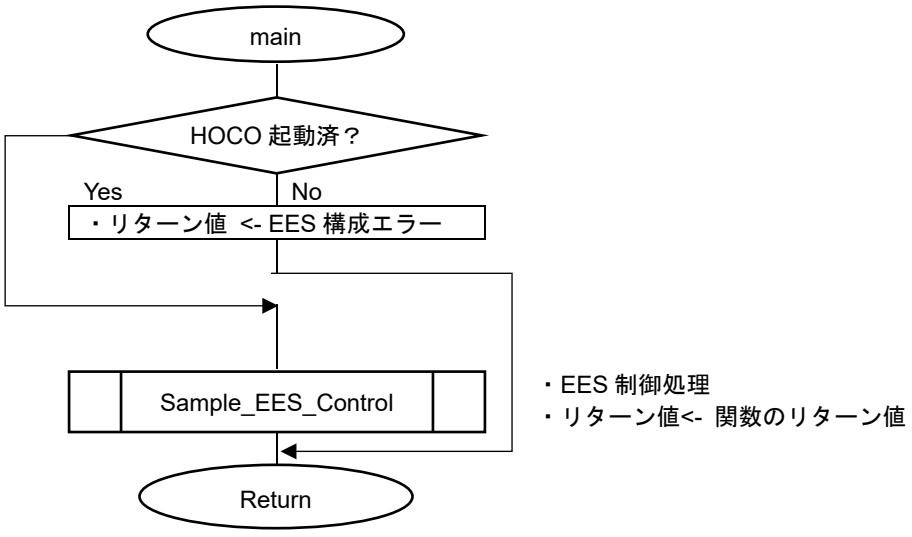
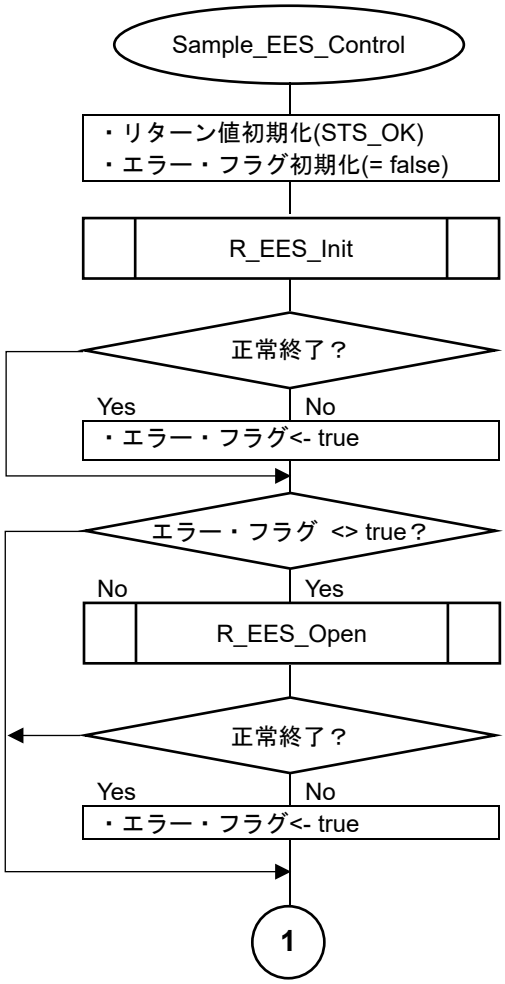


図 6-2 EEPROM エミュレーション制御サンプルのメイン処理実行フロー

6.3.1.2 Sample_EES_Control 関数

EES を使用する際に必要な事前処理と、基本的な書き込み、読み出しの処理フローを図 6-3 に示します。

・EES の初期化を行います。



・EES を初期化します。

・EES を Open 状態にします。

図 6-3 EEPROM エミュレーション制御サンプルの処理実行フロー(1/5)

・EES のスタートアップを実行し、ブロック不整合の場合、フォーマット実施後にスタートアップを再実行します。

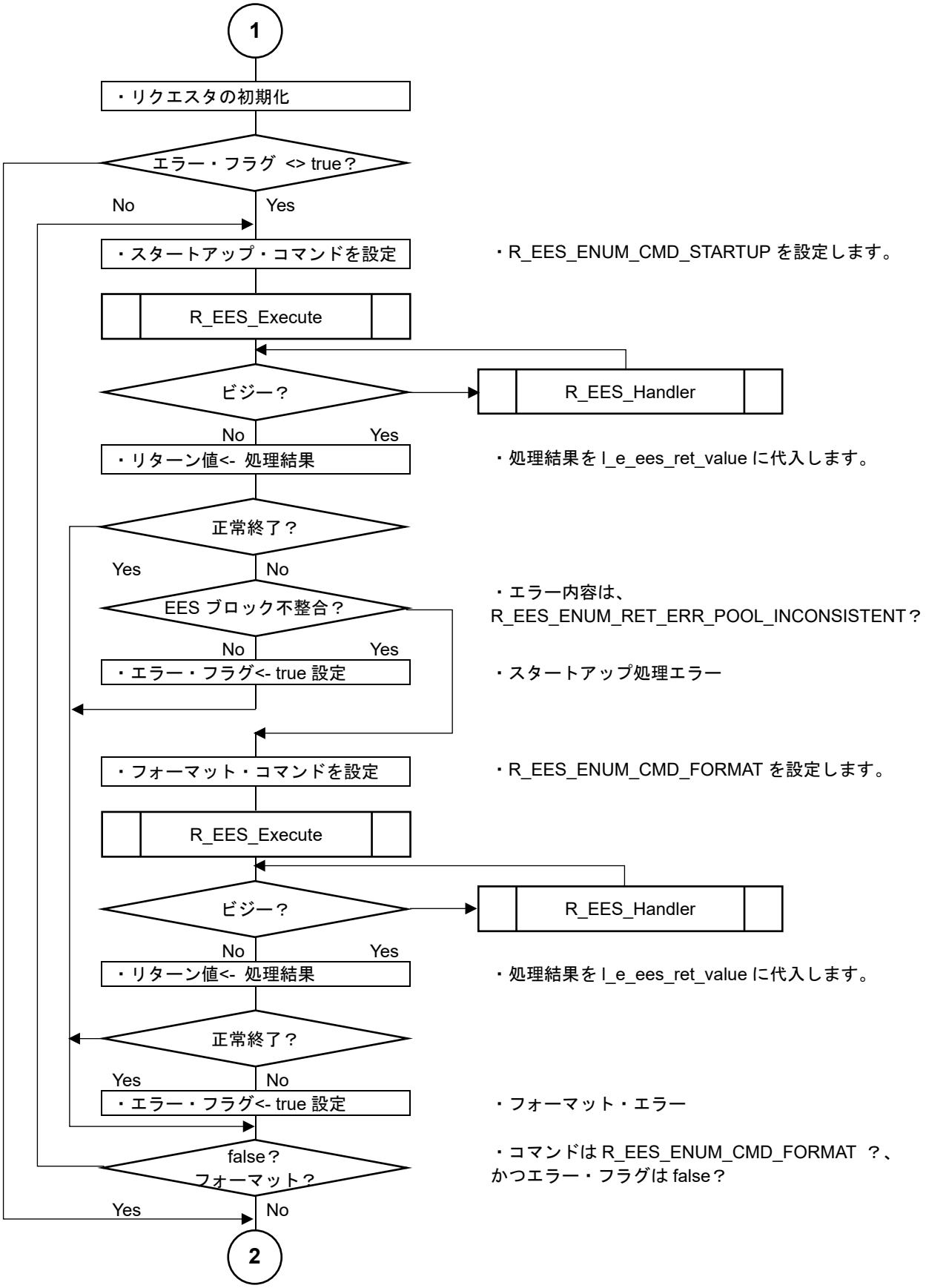


図 6-4 EEPROM エミュレーション制御サンプルの処理実行フロー(2/5)

・書き込み処理を実行しプール・フルの場合は、リフレッシュ実施後に書き込み処理を再実行します。

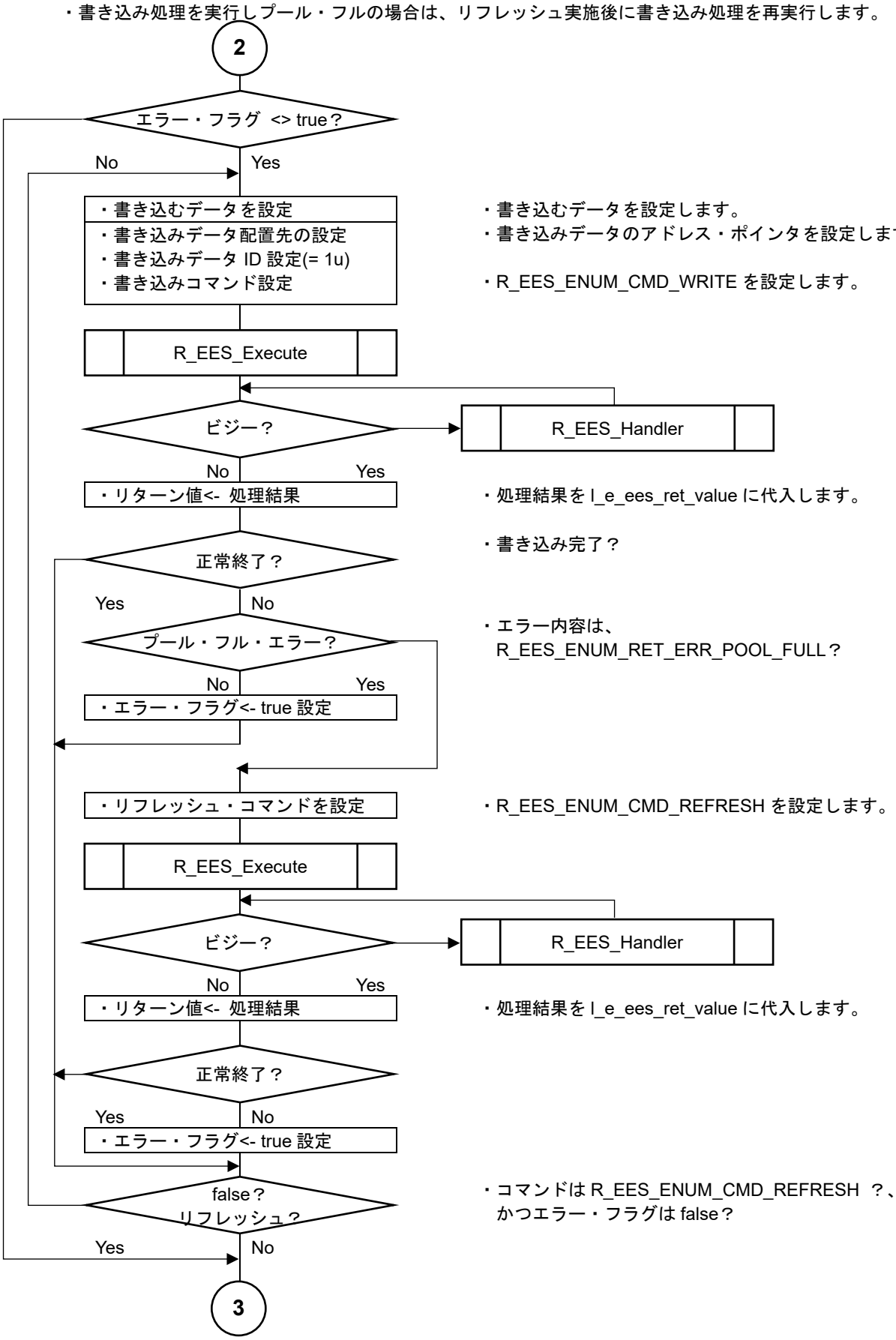
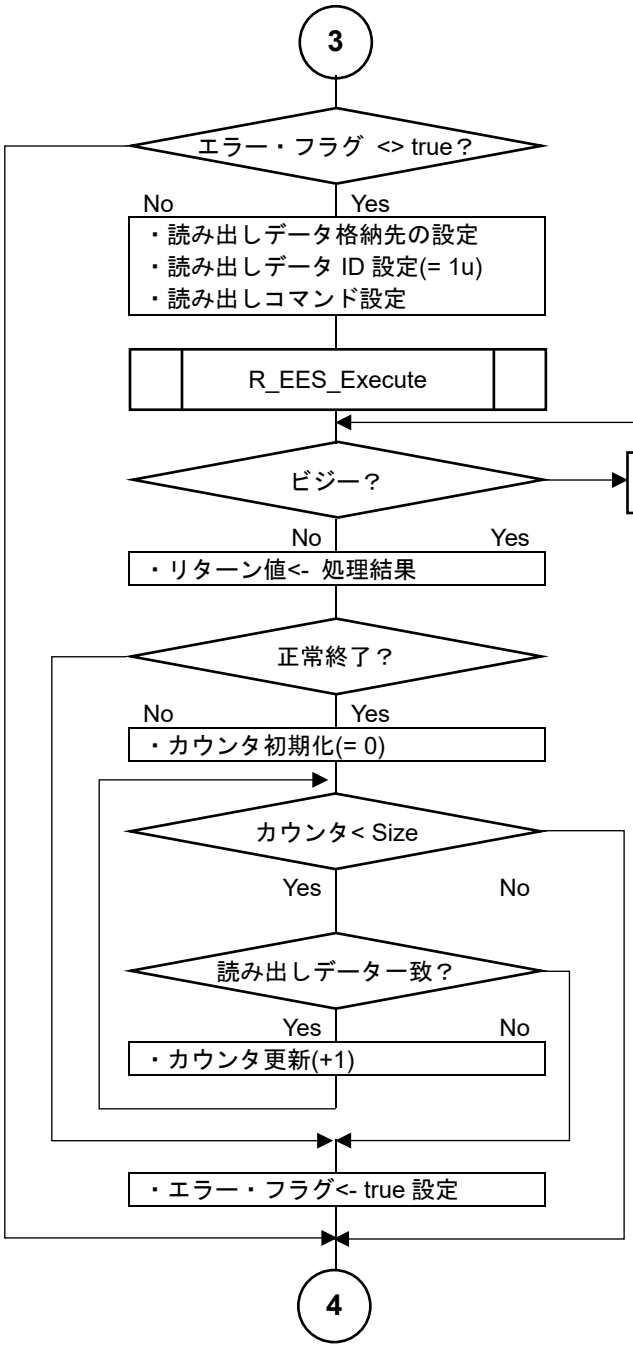


図 6-5 EEPROM エミュレーション制御サンプルの処理実行フロー(3/5)

・読み出しを実行



・読み出しデータ格納先へのアドレス・ポインタを設定します。
・R_EES_ENUM_CMD_READ を設定します。

・指定された ID の読み出しを開始します。

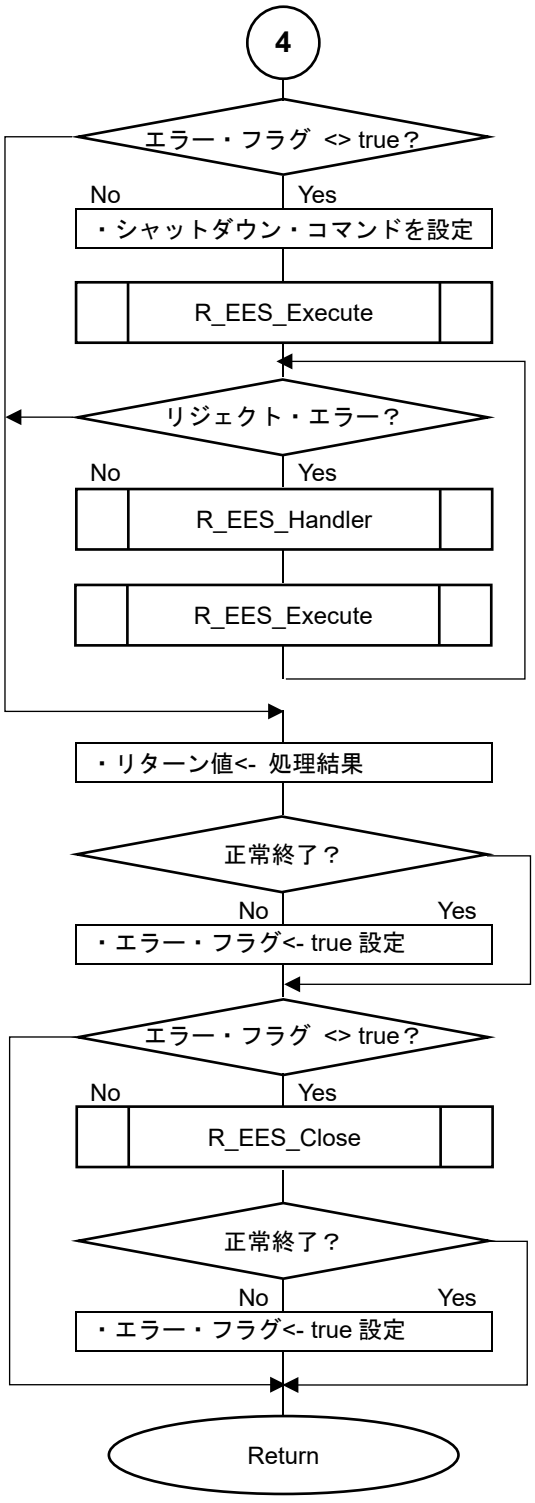
・処理結果を l_e_ees_ret_value に代入します。

・読み出したデータのベリファイ・チェック

・書き込みデータ[カウンタ位置]と読み出しデータ[カウンタ位置]を比較します。

図 6-6 EEPROM エミュレーション制御サンプルの処理実行フロー(4/5)

・EES の終了処理を実行。



・R_EES_ENUM_CMD_SHUTDOWN を設定します。

・他の未完了なコマンド処理を進行させます。

・シャットダウン・コマンドを再実行します。

・処理結果を l_e_ees_ret_value に代入します。

・EES を Close 状態にします。

図 6-7 EEPROM エミュレーション制御サンプルの処理実行フロー(5/5)

注）エラー処理、および正常終了時のユーザ処理は記載していません。

6.4 サンプル・プログラム関数仕様

この章では、EES RL78 Type01 のサンプル・プログラム関数仕様について説明します。EES RL78 Type01 のサンプル・プログラムは、EEPROM エミュレーションの基本的な実行手順の例を示しています。EEPROM エミュレーションを使用するアプリケーションを開発する上で参考にしていただくことができます。

開発されたアプリケーション・プログラムについては、お客様自身で必ず十分な動作確認を行ってください。

6.4.1 EEPROM エミュレーションを使用したサンプル・プログラム関数仕様

6.4.1.1 main

Information

Syntax	int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	int (e_ees_ret_status_t)	R_EES_ENUM_RET_STS_OK : 0x00 [正常終了] R_EES_ENUM_RET_STS_BUSY : 0x01 [コマンド実行中] R_EES_ENUM_RET_ERR_CONFIGURATION : 0x82 [EES 構成エラー] R_EES_ENUM_RET_ERR_INITIALIZATION : 0x83 [EES 初期化エラー] R_EES_ENUM_RET_ERR_ACCESS_LOCKED : 0x84 [EEPROM エミュレーション・ロック・エラー] R_EES_ENUM_RET_ERR_PARAMETER : 0x85 [パラメータ・エラー] R_EES_ENUM_RET_ERR_WEAK : 0x86 [書き込み不十分エラー] R_EES_ENUM_RET_ERR_REJECTED : 0x87 [リジェクト・エラー] R_EES_ENUM_RET_ERR_NO_INSTANCE : 0x88 [データ未書き込みエラー] R_EES_ENUM_RET_ERR_POOL_FULL : 0x89 [プール・フル・エラー] R_EES_ENUM_RET_ERR_POOL_INCONSISTENT : 0x8A [EES ブロック不整合エラー] R_EES_ENUM_RET_ERR_POOL_EXHAUSTED : 0x8B [EES ブロック消耗エラー] R_EES_ENUM_RET_ERR_INTERNAL : 0x8C [内部エラー] R_EES_ENUM_RET_ERR_FLASH_SEQ : 0x8D [フラッシュ・シーケンサー・エラー]
Description	EES 制御サンプル・プログラムのメイン関数	
Preconditions	-	
Remarks	-	

6.4.1.2 Sample_EES_Control

Information

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t Sample_EES_Control();	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK : 0x00 [正常終了] R_EES_ENUM_RET_STS_BUSY : 0x01 [コマンド実行中] R_EES_ENUM_RET_ERR_CONFIGURATION : 0x82 [EES 構成エラー] R_EES_ENUM_RET_ERR_INITIALIZATION : 0x83 [EES 初期化エラー] R_EES_ENUM_RET_ERR_ACCESS_LOCKED : 0x84 [EEPROM エミュレーション・ロック・エラー] R_EES_ENUM_RET_ERR_PARAMETER : 0x85 [パラメータ・エラー] R_EES_ENUM_RET_ERR_WEAK : 0x86 [書き込み不十分エラー] R_EES_ENUM_RET_ERR_REJECTED : 0x87 [リジェクト・エラー] R_EES_ENUM_RET_ERR_NO_INSTANCE : 0x88 [データ未書き込みエラー] R_EES_ENUM_RET_ERR_POOL_FULL : 0x89 [プール・フル・エラー] R_EES_ENUM_RET_ERR_POOL_INCONSISTENT : 0x8A [EES ブロック不整合エラー] R_EES_ENUM_RET_ERR_POOL_EXHAUSTED : 0x8B [EES ブロック消耗エラー] R_EES_ENUM_RET_ERR_INTERNAL : 0x8C [内部エラー] R_EES_ENUM_RET_ERR_FLASH_SEQ : 0x8D [フラッシュ・シーケンサー・エラー]
Description	EES の基本的な使用手順に従い、EES ブロックの書き込み、読み出しを実行します。	
Preconditions	-	
Remarks	読み出したデータのベリファイ・チェックでエラーとなった場合、Return Value には反映されません。	

6.5 サンプル・プログラム使用時の注意事項

・RL78/G24 を使用する場合の注意事項

オプション・バイト(000C2H/040C2H)の設定値を 0xF0 に設定して CPU の動作周波数を 24MHz で使用する場合に限り、対策が必要です。以下に示す RL78/G24 用のサンプル・プログラムに含まれているソース・コードの赤文字部分のようにコメントに変更するか、削除するなどにより、コンパイルされないように変更してください。

赤文字部分がコンパイルされると、プリフェッチバッファが有効となると共に、48MHz で動作するように設定されます。

対象フォルダ

\EESRL78T01\sample\RL78_G24\EES\[コンパイラ名]\source\

対象ファイル

CC-RL, LLVM: hdwinit.c

IAR: low_level_init.c

以下、コメントに変更した例を示します。

```
/* Start HOCO. It must be started before flash control. */
HIOSTOP = 0u;

/* Check CPU frequency in the user option byte (0x000C2). */
/* 0xF0 : HS mode 48 MHz */
// if (0xF0u == (*(volatile unsigned char __far *)0x000C2u))
// {
//       /* Set CPU frequency 48 MHz (Enables the prefetch buffer). */
//       HOCODIV = 0x00u;
//       PFBE      = 1u;
//       FIHSEL   = 1u;
//
//       /* Confirm the switching status flag. */
//       while (1u == FIHST)
//       {
//             /* No operation */
//       }
// }
// else
// {
//       /* No operation */
// }
```

7 サンプル・プロジェクトの作成

EES RL78 Type01 には、EEPROM エミュレーションを操作するサンプル・プログラムが含まれます。EES RL78 Type01 で使用できるコンパイラは、CC-RL コンパイラと IAR コンパイラと LLVM コンパイラです。それぞれのコンパイラに対応する統合開発環境を使用してサンプル・プロジェクトを作成することができます。

サンプル・プログラムは、サポートしているデバイス毎に分かれています（例：RL78_G23, RL78_G24）。この項では、RL78/G23(R7F100GLG)用のサンプル・プログラムを例に説明しています。RL78/G23 以外を使用する場合は、"G23"を対象のデバイスに読みかえてください。セクション設定のアドレスなどは、対象デバイスのユーザーズマニュアルを参照いただき変更する必要があります。

また、フラッシュ・メモリ制御方式は、対象デバイスにより異なるため、分類用のマクロを統合開発環境(IDE)で設定する必要があります。設定方法については、“7.1.3.2 ユーザ定義マクロの設定”(CC-RL)、“7.2.3.2 ユーザ定義マクロの設定”(IAR)、“7.3.3.2 ユーザ定義マクロの設定”(LLVM)に記述されています。

RL78/G22 を使用する場合は、RL78/G23 のサンプル・プログラムを使用できます。

注意 1. 対象の統合開発環境、およびコンパイラは、RL78/G2x を対象としたバージョンをご使用いただくことを前提としています。RL78/G2x が対象製品であることをご確認の上、ご使用ください。

2. EES RL78 Type01 では、データ・フラッシュを操作するために RFD RL78 Type01 を使用しますが、EES RL78 Type01 のインストーラには含まれていません。別途 RFD RL78 Type01 をインストールしてプロジェクトに登録する必要があります。また、この章では EEPROM エミュレーションを使用するために必要な RFD RL78 Type01 のファイルや使用するセクションについて記載していますが、RFD RL78 Type01 の詳細については RFD RL78 Type01 のユーザーズマニュアルを参照してください。

7.1 CC-RL コンパイラを使用する場合のプロジェクトの作成

RENESAS 製 CC-RL コンパイラは、統合開発環境として CS+、および e^2 studio を使用して作成したプロジェクトへ EES RL78 Type01 と RFD RL78 Type01 を登録し、ビルドすることができます。各統合開発環境を使用した場合のサンプル・プロジェクトの作成例を示します。CC-RL コンパイラ、および各統合開発環境を理解するため、それぞれのツール製品のユーザーズマニュアルを参照してください。

7.1.1 サンプル・プロジェクト作成例

(1) 統合開発環境 CS+を使用したサンプル・プロジェクト作成例

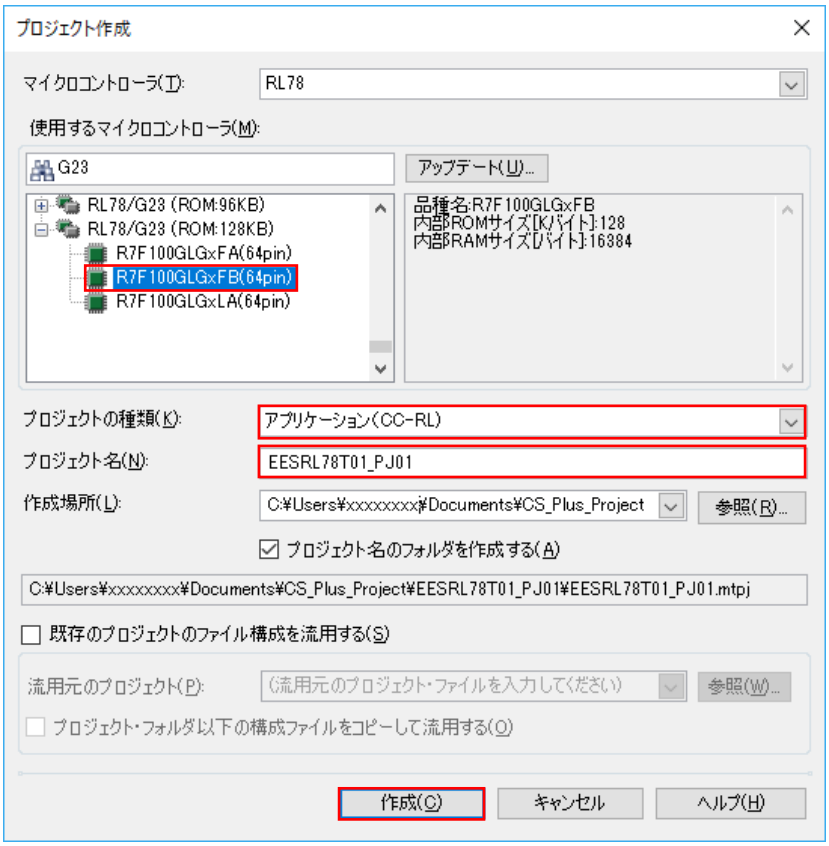
CS+を起動し、[プロジェクト]メニューの[新しいプロジェクトを作成]を選択し、以下に示す”プロジェクト作成”ウインドウを起動します。

・[使用するマイクロコントローラ]は、”RL78/G23 (ROM: 128KB)” – “R7F100GLGxFB(64pin)”を選択します。

・[プロジェクトの種類]は、”アプリケーション(CC-RL)”を選択します。

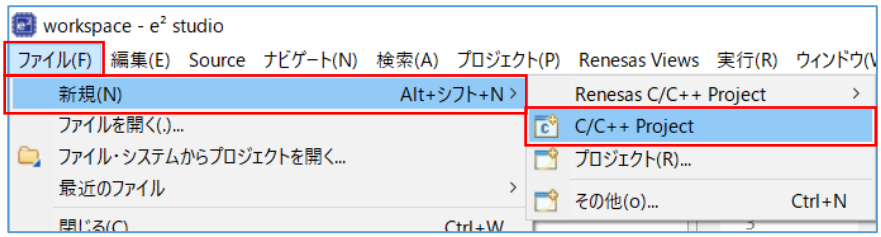
・ここでの[プロジェクト名]は、仮に”EESRL78T01_PJ01”とします。

・[作成]ボタンを押すと、新しいプロジェクトが作成されます。

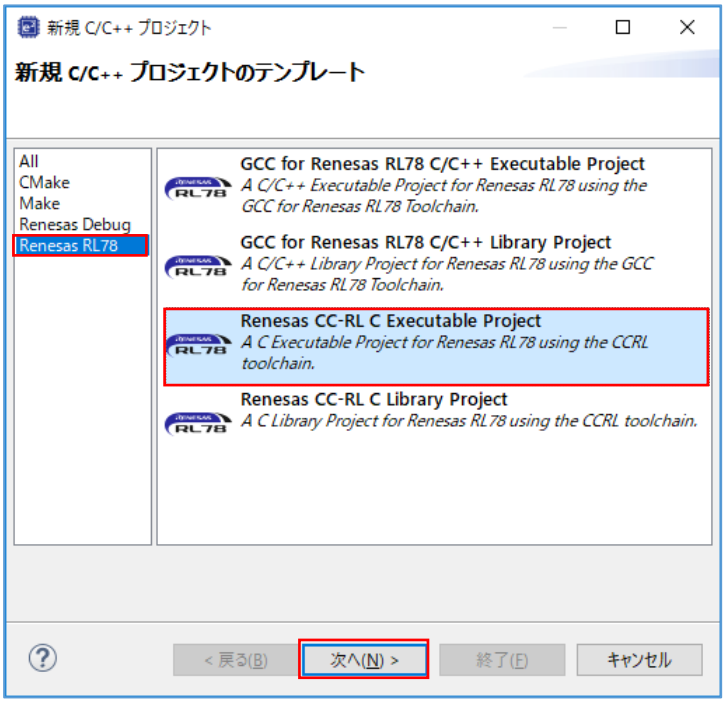


(2) 統合開発環境 e² studio を使用したサンプル・プロジェクト作成例

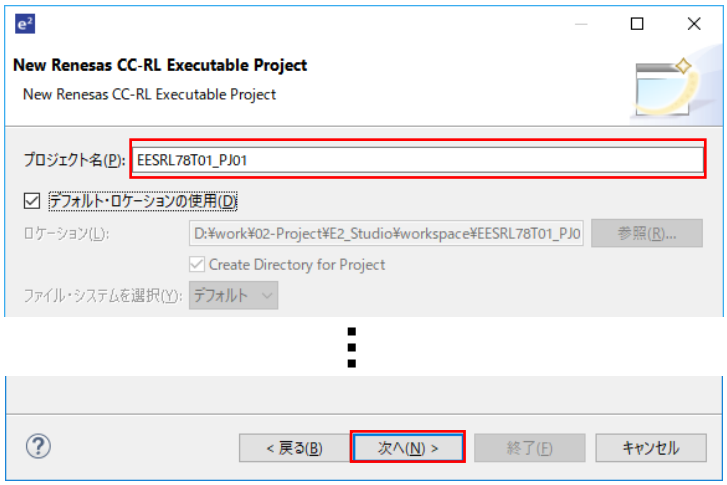
e² studio を起動し、[ファイル]メニューの[新規]から[プロジェクト]を選択し、"新規 C/C++プロジェクトのテンプレート"ウインドウを起動します。



・[Renesas RL78]を選択して表示した[Renesas CC-RL C Executable Project]を選択、”次へ”ボタンを押します。



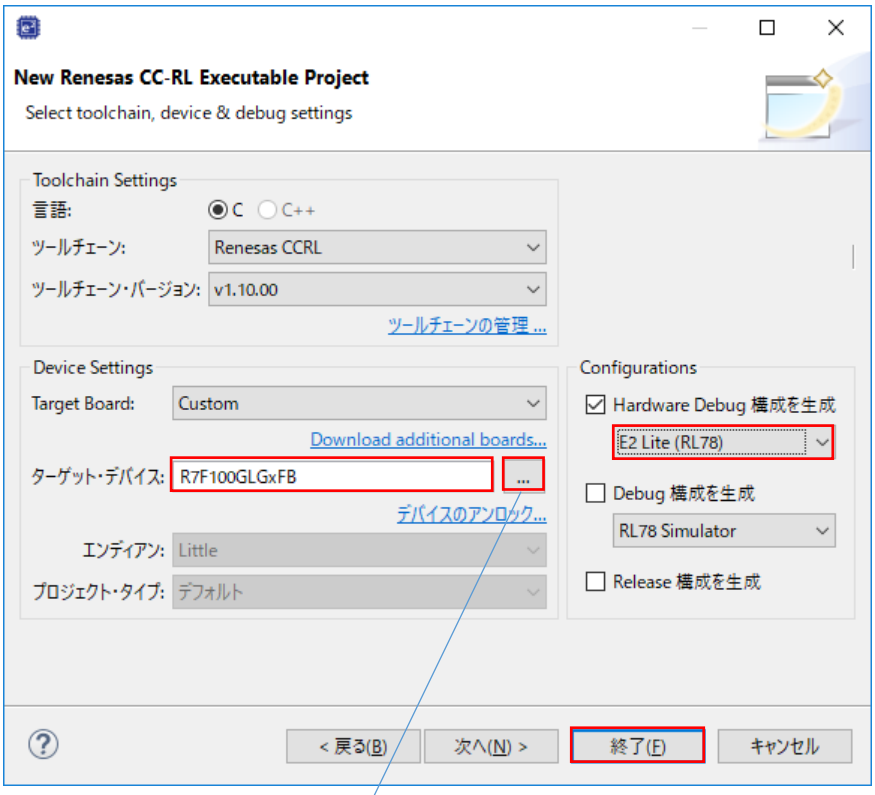
・”New Renesas CC-RL Executable Project”ウインドウで、プロジェクト名を入力して”次へ”ボタンを押します。
（ここでは、仮に”EESRL78T01_PJ01”とします。）

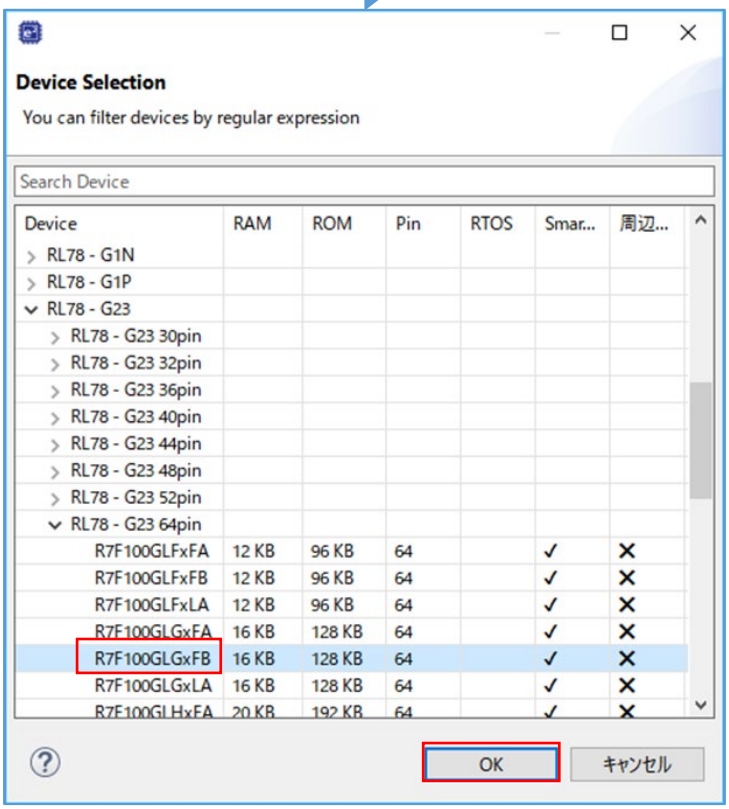


・[Device Settings]の[ターゲット・デバイスで、”RL78 – G23” – “RL78 – G23 64pin” – “R7F100GLGxFB”を選択し[OK]ボタンを押します。

・デバッグ・ツールに E2 Lite を選択し、オンチップ・デバッグを実施することを前提としています。
[Configurations]で”Hardware Debug 構成を生成”にチェックが入った状態で、E2 Lite (RL78)を選択します。

・[終了]ボタンを押します。

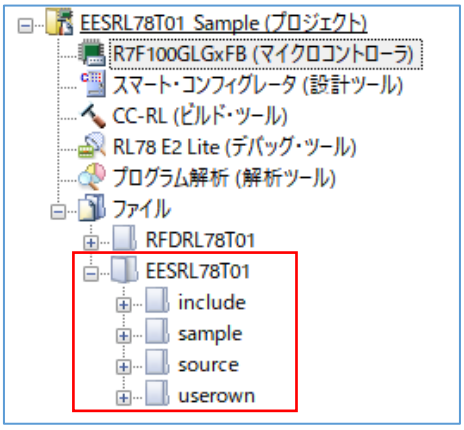




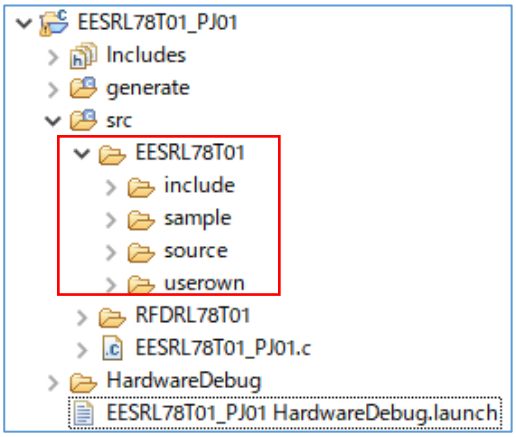
7.1.2 対象フォルダと対象ファイルの登録例

EES RL78 Type01 を使用して、EEPROM エミュレーションを実行するために必要なファイルの登録例を記述します。まず、EESRL78T01 ソースプログラムの”EESRL78T01”フォルダを登録します。このフォルダには、”include”, ”source”, ”userown”, ”sample”フォルダが含まれます。

その他の手順として、”include”, ”source”, ”userown”, ”sample”の全てのフォルダを登録し、不要なファイルとフォルダを、[プロジェクトから外す] 機能(CS+)、[リソース構成] – [ビルドから除外…]機能(e^2 studio)により、対象から外すこともできます。



CS+の EES RL78 Type01 登録時のツリー画面



e^2 studio の EES RL78 Type01 登録時のツリー画面

・統合開発環境で対象製品用に出力された最新の I/O ヘッダ・ファイルの登録

”iodefine.h”は、CS+、および e^2 studio が対象製品用に出力する I/O ヘッダ・ファイルです。EES RL78 Type01 に含まれている”iodefine.h”の代わりに置き換えてご使用頂くことを推奨いたします。統合開発環境が出力した”iodefine.h”を EES RL78 Type01 内の”iodefine.h”と入れ替える、もしくは上書きしてください。

統合開発環境が I/O ヘッダ・ファイル”iodefine.h”を出力するフォルダ：

- CS+：[プロジェクト名]フォルダ
- e^2 studio：[プロジェクト名]/generate フォルダ

”iodefine.h”ファイルを入れ替え、もしくは上書きするフォルダ：

- CS+：”\[プロジェクト名]\EESRL78T01\sample\RL78_G23\EES\CCRL\include”
- e^2 studio：”\[プロジェクト名]\src\EESRL78T01\sample\RL78_G23\EES\CCRL\include”

・統合開発環境の機能により自動的に追加されたファイルの除外

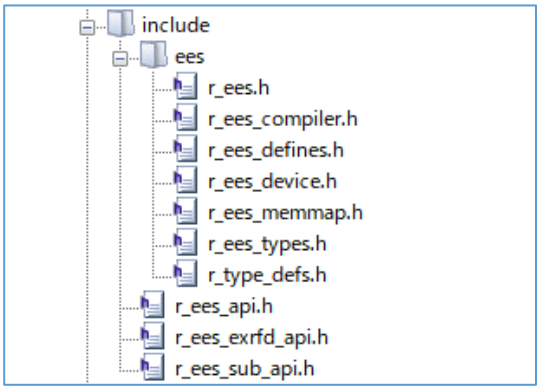
作成されたプロジェクトには、自動的に追加されるファイルがあります。これらと同様のファイルは、EES RL78 Type01 の”sample”フォルダ内にも存在するため、プロジェクト・ツリーから各ファイルを選択し、各統合開発環境の機能を使用して、プロジェクトから外します。

- CS+ではツリーでファイルをマウス右クリック、”プロジェクトから外す”機能で対象ファイルを除外します。[プロジェクト名]フォルダ内の cstart.asm, hdwinit.asm, stkinit.asm, main.c, iodefine.h が対象。
- e^2 studio ではツリーでファイルをマウス右クリック、”プロパティ”で表示された[設定]画面で、“ビルドからリソースを除外”にチェックを入れ、対象ファイル(対象フォルダ)を除外します。(フォルダから削除も可能)
  [プロジェクト名]/generate フォルダ内の cstart.asm, hdwinit.asm, iodefine.h, stkinit.asm、および[プロジェクト名]/src フォルダ内の[プロジェクト名].c(ここでは”EESRL78T01_PJ01.c”)が対象。

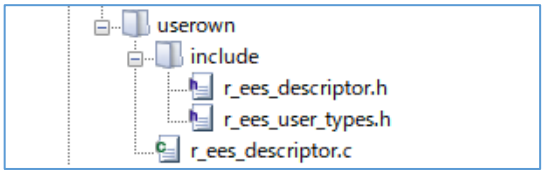
(1) EES RL78 Type01 の対象フォルダと対象ファイルの登録

EES RL78 Type01 ソースプログラムファイルの各フォルダ(”include”, ”source”, ”userown”, “sample”)と登録ファイルを以下に示します。

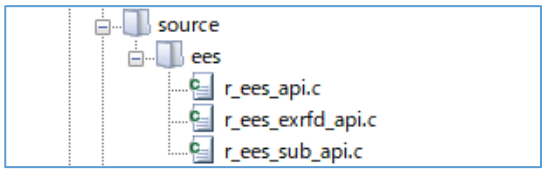
include フォルダ内



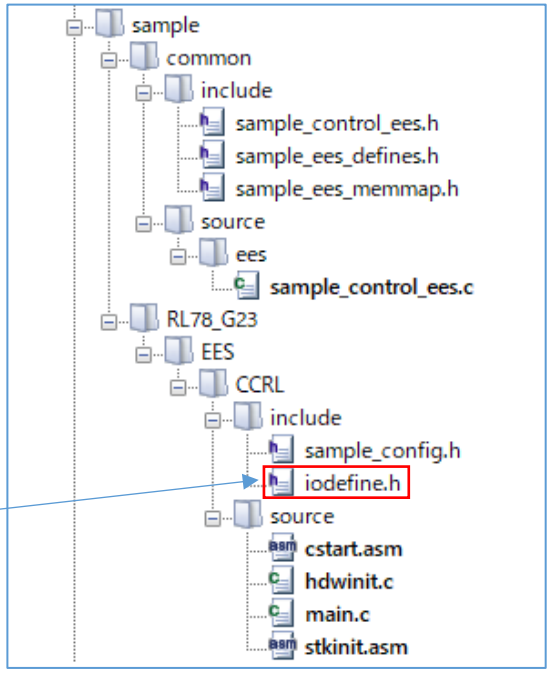
userown フォルダ内



source フォルダ内



sample フォルダ内

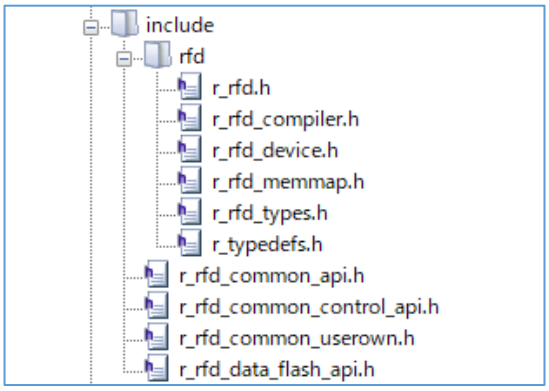


CS+、または e^2 studio で出力した“iodefine.h” に置き換えてください。

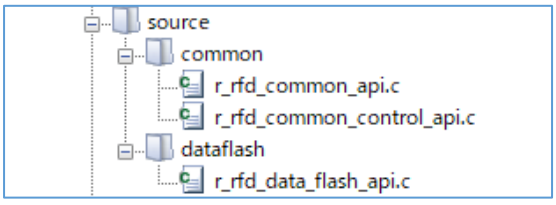
(2) RFD RL78 Type01 の対象フォルダと対象ファイルの登録

RFD RL78 Type01 ソースプログラムファイルの各フォルダ(“include”, “source”, “userown”)と登録ファイルを以下に示します。

include フォルダ内



source フォルダ内



userown フォルダ内

7.1.3 ビルド・ツールの設定

CC-RL コンパイラで EES RL78 Type01 をビルドして実行するための各統合開発環境の設定を行います。

CS+ではツリーで”CC-RL(ビルド・ツール)“のマウス右クリックで”プロパティ”を選択、e^2 studio ではツリーでプロジェクト(ここでは”EESRL78T01_PJ01“)のマウス右クリックで”プロパティ”を選択することにより、表示された画面内のビルド・ツールの各設定を行います。

7.1.3.1 インクルード・パスの設定

・CS+でのインクルード・パスの設定は、 “共通オプション”タブで設定

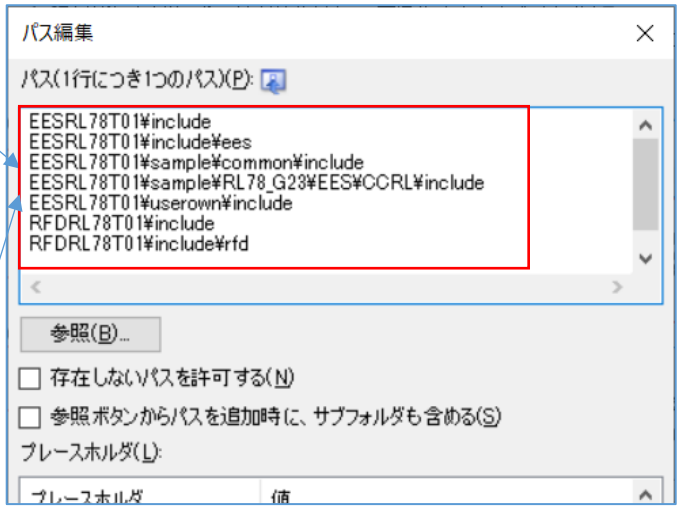
- [よく使うオプション(コンパイル)] – [追加のインクルード・パス]で”パス編集”ウインドウを表示して、インクルード・ファイルのパスを設定します。

(1) EES RL78 Type01 の include パス

```
EESRL78T01\include
EESRL78T01\include\ees
EESRL78T01\sample\common\include
EESRL78T01\sample\RL78_G23\EES\CCRL\include
EESRL78T01\userown\include
```

(2) RFD RL78 Type01 の include パス

```
RFDRL78T01\include
RFDRL78T01\include\rfd
```



・e^2 studio でのインクルード・パスの設定は、”プロパティ”ウインドウで設定

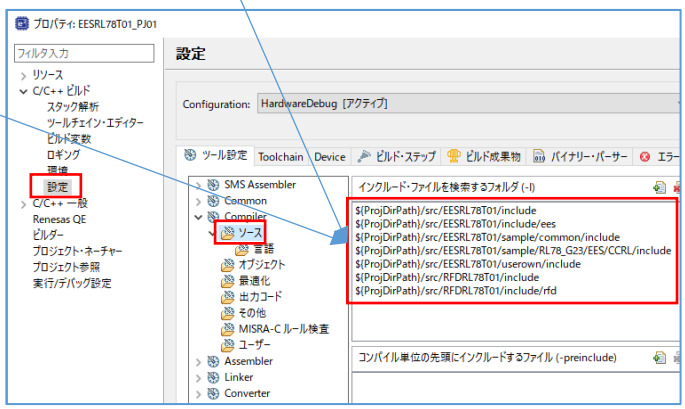
- ”C/C++ビルド” [設定] – “Compiler” [ソース] で表示した画面でインクルード・ファイルのパスを設定します。

(1) EES RL78 Type01 の include パス

```
${ProjDirPath}\src\EESRL78T01\include
${ProjDirPath}\src\EESRL78T01\include\ees
${ProjDirPath}\src\EESRL78T01\sample\common\include
${ProjDirPath}\src\EESRL78T01\sample\RL78_G23\EES\CCRL\include
${ProjDirPath}\src\EESRL78T01\userown\include
```

(2) RFD RL78 Type01 の include パス

```
${ProjDirPath}\src\RFDRL78T01\include
${ProjDirPath}\src\RFDRL78T01\include\rfd
```



7.1.3.2 ユーザ定義マクロの設定

・CS+でのフラッシュ・メモリ制御方式分類用マクロを“共通オプション”タブで定義します。

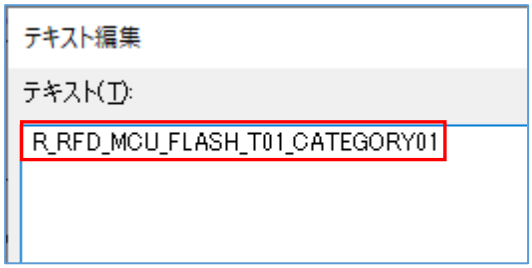
- [よく使うオプション(コンパイル)] – [定義マクロ]で “テキスト編集”ウインドウを表示して、以下のマクロを定義してください。使用するデバイスによって定義するマクロが異なります。

RL78/G23, RL78/G22 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY01

RL78/G24 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY02



・e² studio でのフラッシュ・メモリ制御方式分類用マクロを“プロパティ”ウインドウで定義します。

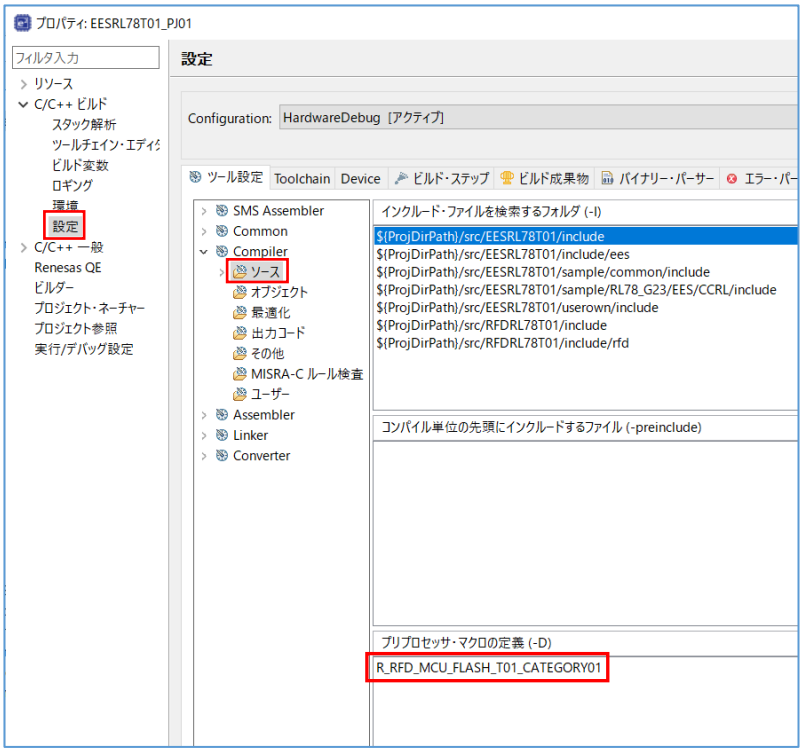
- “C/C++ビルド” [設定] – “Compiler” [ソース] で“プリプロセッサ・マクロの定義(-D)”の欄に以下のマクロを定義してください。使用するデバイスによって定義するマクロが異なります。

RL78/G23, G22 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY01

RL78/G24 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY02



注）マクロを定義していない場合、コンパイル・エラーが出力されます。

7.1.3.3 デバイス項目の設定

・CS+でのデバイス項目の設定は、“リンク・オプション”タブで設定

- [デバイス] 項目を設定します。

[オンチップ・デバッグの許可／禁止をリンク・オプション設定する]を”はい(-OCDBG)”に設定します。

注）オンチップ・デバッグを実施することを前提とした設定例です。

[オンチップ・デバッグ・オプション・バイト制御値]を”85”に設定します。(オンチップ・デバッグ動作許可の例)

注）対象デバイスのユーザーズマニュアルで「オプション・バイト」の章の「オンチップ・デバッグ・オプション・バイト」の内容をご確認いただき、使用する設定値を書き込んでください。

[デバッグ・モニタ領域を設定する]を”はい(範囲指定)(-OCDBG_MONITOR=<アドレス範囲>)”に設定します。

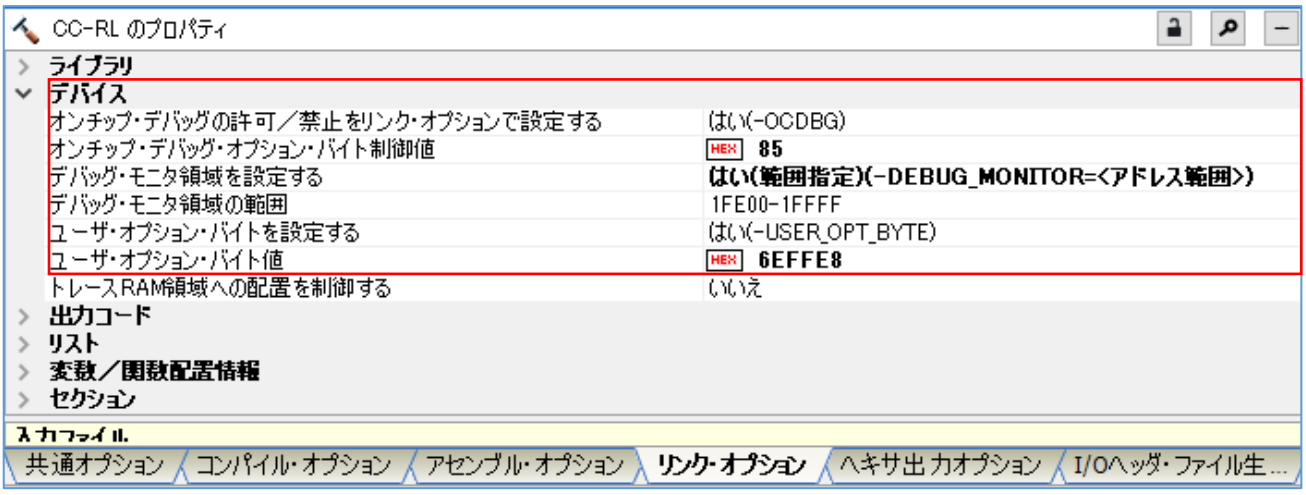
[デバッグ・モニタ領域の範囲]を”1FE00-1FFFF”に設定します。[RL78/G23 の例]

注）対象デバイスのユーザーズマニュアルで「オンチップ・デバッグ機能」の章の「ユーザ資源の確保」で「デバッグ用モニタ・プログラムが配置されるメモリ空間」をご確認いただき、使用する領域の範囲を書き込んでください。

[ユーザ・オプション・バイトを設定する]を”はい(-USER_OPT_BYTE)”に設定します。

[ユーザ・オプション・バイト値]を”6EFFE8”に設定します。(WDT 停止, LVD(reset モード),HS モード/32MHz [RL78/G23 の例])

注）対象デバイスのユーザーズマニュアルで「オプション・バイト」の章の「ユーザ・オプション・バイト」の内容をご確認いただき、使用する設定値を書き込んでください。



・e^2 studio でのデバイス項目の設定は、”プロパティ”ウインドウで設定

- ”C/C++ビルド” [設定] – “Linker” [デバイス] で表示した画面でデバイス項目を設定します。

[OCD モニタのメモリ領域を確保する(-debug_monitor)]をチェックします。

注）オンチップ・デバッグを実施することを前提とした設定例です。

[メモリ領域(-debug_monitor=<start address>-<end address>)]を”1FE00-1FFFF”に設定します。[RL78/G23 の例]

注）対象デバイスのユーザーズマニュアルで「オンチップ・デバッグ機能」の章の「デバッグ用モニタ・プログラムが配置されるメモリ空間」をご確認いただき、使用する領域の範囲を書き込んでください。

[オプション・バイト領域のユーザ・オプション・バイト値を設定する(-user_opt_byte)] をチェックします。

[ユーザ・オプション・バイト値(-user_opt_byte=<value>)]を”6EFFE8”に設定します。

(WDT 停止, LVD(reset モード),HS モード/32MHz [RL78/G23 の例])

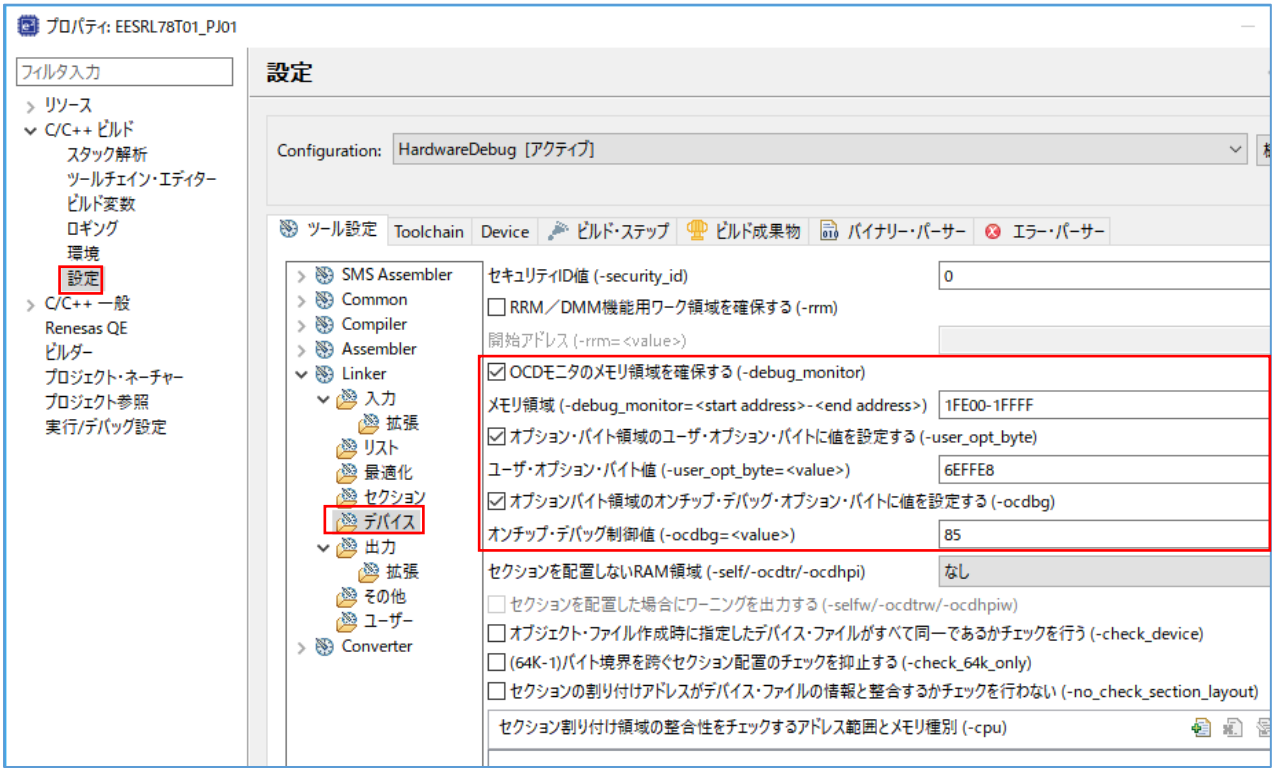
注）対象デバイスのユーザーズマニュアルで「オプション・バイト」の章の「ユーザ・オプション・バイト」の内容をご確認いただき、使用する設定値を書き込んでください。

[オプション・バイト領域のオンチップ・デバッグ・オプション・バイトに値を設定する(-ocdbg)]をチェックします。

注）オンチップ・デバッグを実施することを前提とした設定例です。

[オンチップ・デバッグ制御値(-ocdbg=<value>)]を”85”に設定します。(オンチップ・デバッグ動作許可の例)

注）対象デバイスのユーザーズマニュアルで「オプション・バイト」の章の「オンチップ・デバッグ・オプション・バイト」の内容をご確認いただき、使用する設定値を書き込んでください。

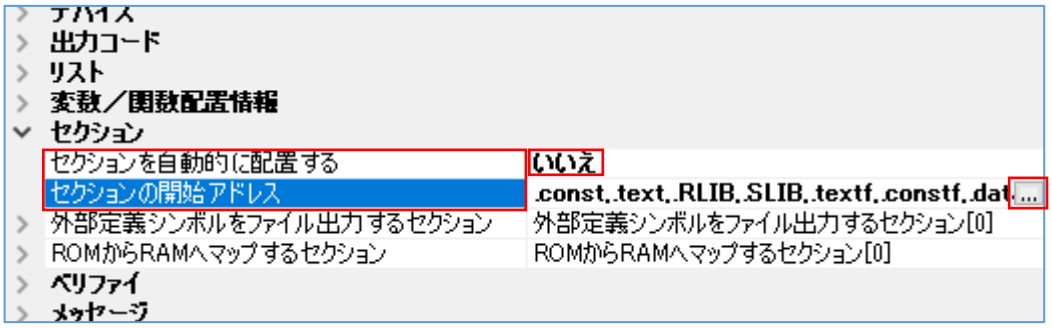


7.1.3.4 セクション項目の設定

・CS+でのセクション項目の設定は、“リンク・オプション”タブで設定

- [セクション] 項目を設定します。

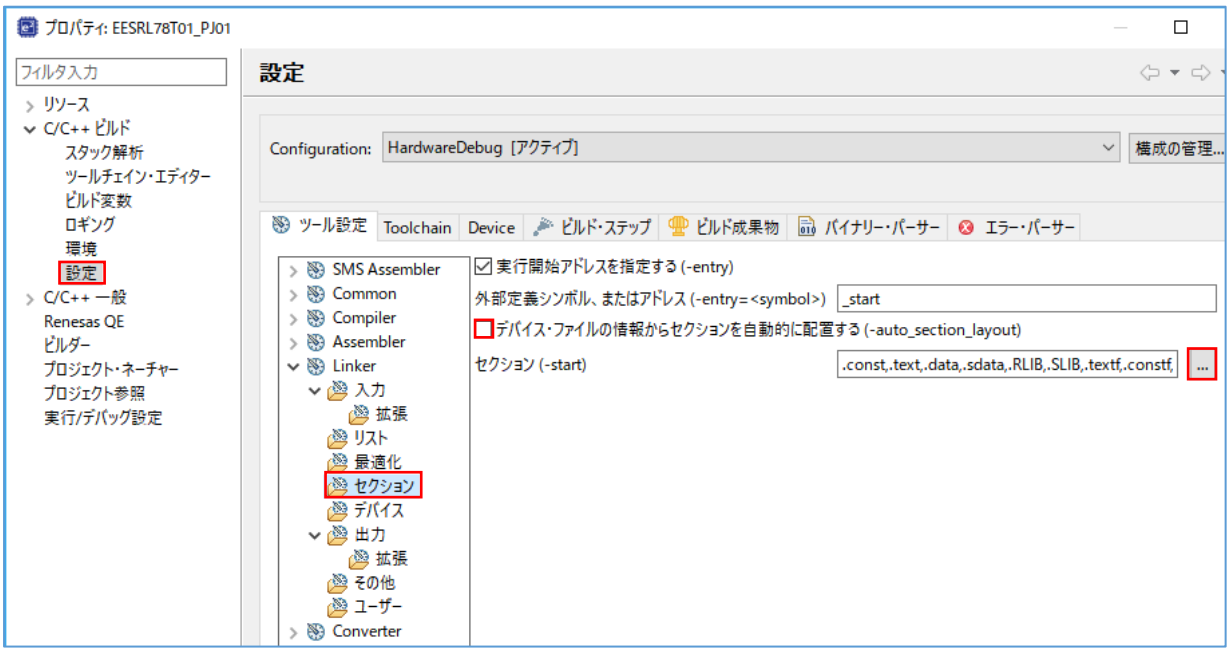
[セクションを自動的に配置する]を一度”いいえ”に設定すると[セクションの開始アドレス]にセクションが表示されるようになり、表示された最も右の”...”ボタンで、”セクション設定”画面を表示します。



・e² studio でのセクション項目の設定は、”プロパティ”ウインドウで設定

- ”C/C++ビルド” [設定] – “Linker” [セクション] で表示した画面でセクション項目を設定します。

[デバイス・ファイルの情報からセクションを自動的に配置する(-auto_section_layout)]のチェックを一度外します。ここで、[セクション(-start)]の最も右の”...”ボタンで、”セクション設定”画面を表示します。



・CS+、e2 studio のセクション設定操作

先頭アドレスに”0x03000”を設定します。

プログラム領域（コード・フラッシュ・メモリ）と RAM 領域に、EES RL78 Type01 内の”#pragma section”で定義されたセクションを追加します。各セクションの詳細は「表 2-10 EES 使用時のセクション」を参照してください。

※本説明では、[コンパイル・オプション]の[メモリ・モデル]がミディアム・モデル（R7F100GLG での”自動選択”と同じ）であることを前提としています。ROM から RAM へマップするセクションのコピー処理は、cstart.asm ファイル内で実施しています。また、”スモール・モデル”を選択した場合の各プログラムのセクション名については CC-RL のユーザーズマニュアルを参照してください。

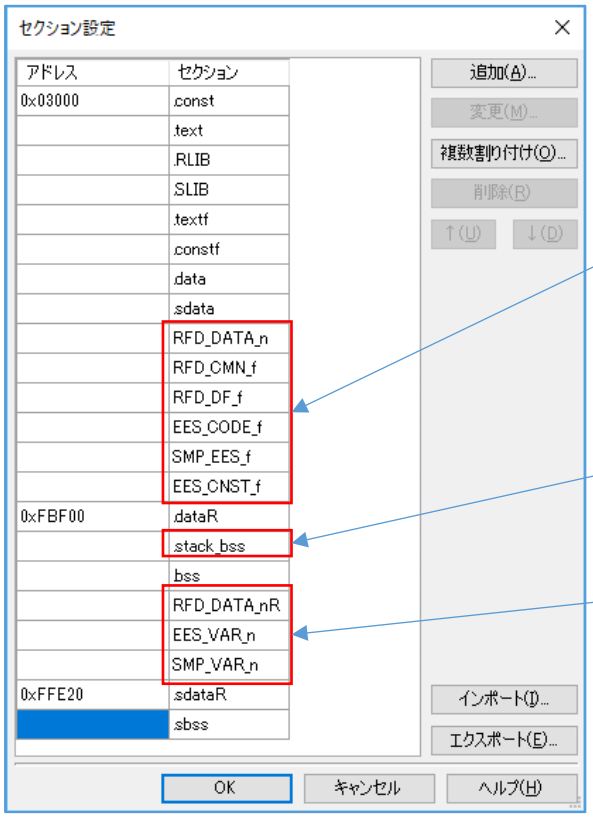
(1) EES 使用時に必要なセクションの追加

・CS+での EEPROM エミュレーションの実行に必要なセクション追加

EEPROM エミュレーションの操作に必要なセクションを“セクション設定”画面で追加します。RFD RL78 Type01 のセクションも含んでいます。

プログラム領域へ追加：RFD_DATA_n, RFD_CMN_f, RFD_DF_f, EES_CODE_f, SMP_EES_f, EES_CNST_f

RAM へ追加：.stack_bss, RFD_DATA_nR, EES_VAR_n, SMP_VAR_n

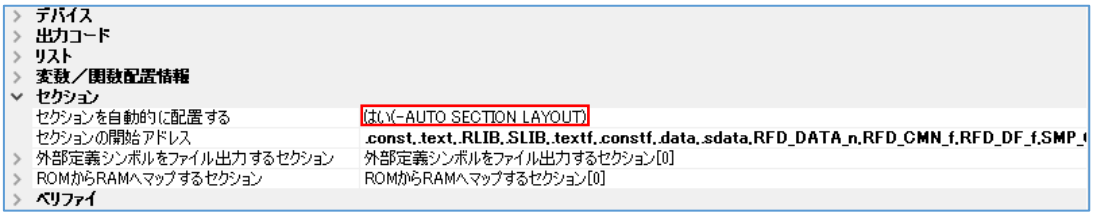


追加セクション
RFD_DATA_n
RFD_CMN_f
RFD_DF_f
EES_CODE_f
SMP_EES_f
EES_CNST_f

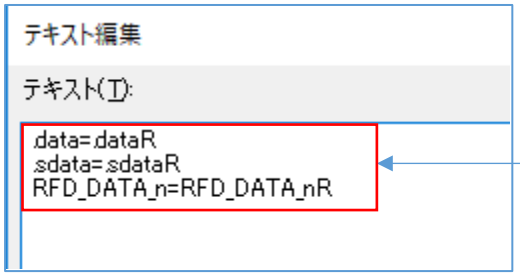
.stack_bss

RFD_DATA_nR
EES_VAR_n
SMP_VAR_n

“OK”ボタン押下後、[セクションを自動的に配置する]を”はい”に戻して下さい。



[ROM から RAM へマップするセクション]の最も右の” ... ”ボタンで、”テキスト編集”画面を表示して、ROM から RAM へマップするセクションを追加します。



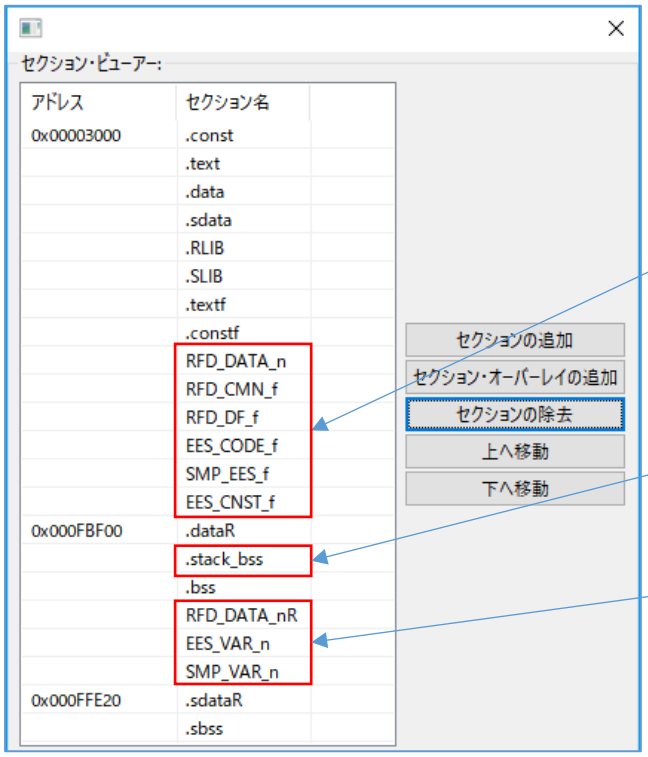
ROM から RAM へ マップするセクション
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

・e^2 studio での EEPROM エミュレーションの実行に必要なセクション追加

EEPROM エミュレーションの操作に必要なセクションを“セクション・ビューアー”で追加します。RFD RL78 Type01 のセクションも含んでいます。

プログラム領域へ追加：RFD_DATA_n, RFD_CMN_f, RFD_DF_f, EES_CODE_f, SMP_EES_f, EES_CNST_f

RAM へ追加：.stack_bss, RFD_DATA_nR, EES_VAR_n, SMP_VAR_n

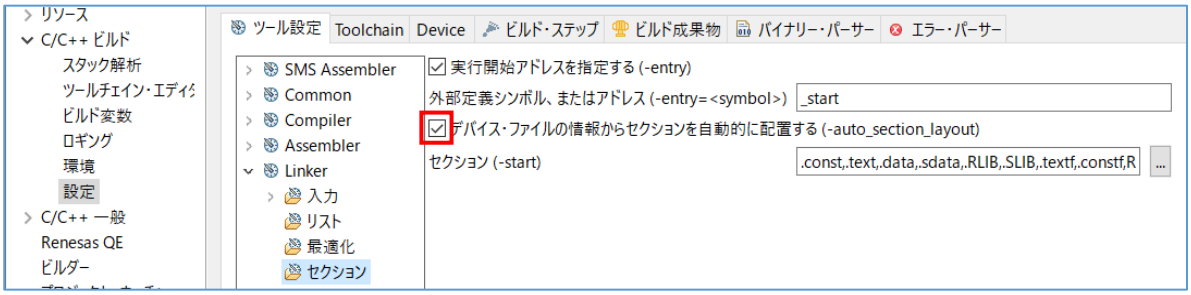


追加セクション
RFD_DATA_n
RFD_CMN_f
RFD_DF_f
EES_CODE_f
SMP_EES_f
EES_CNST_f

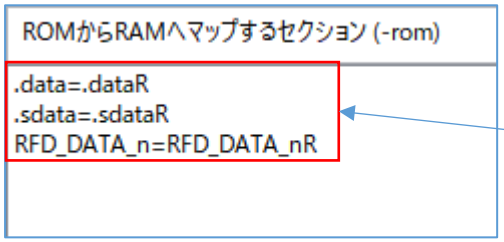
.stack_bss

RFD_DATA_nR
EES_VAR_n
SMP_VAR_n

“OK” ボタン押下後、[デバイス・ファイルの情報からセクションを自動的に配置する(-auto_section_layout)]をチェックして下さい。



”C/C++ビルド” [設定] – “Linker” [出力] で表示した画面で[ROM から RAM へマップするセクション(-rom)]を設定します。



ROM から RAM へ マップするセクション
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

7.1.4 デバッグ・ツールの設定

ここでは、デバッグ・ツールに E2 Lite を選択してオンチップ・デバッグを行う場合のターゲット・ボードとの接続の設定について説明します。その他のデバッグ・ツール設定の詳細については、各統合開発環境のユーザーズマニュアルを参照してご確認ください。

CS+では、ツリーで”RL78 シミュレータ(ビルドツール)“[初期設定]でマウス右クリックし、表示された”使用するデバッグ・ツール”で”RL78 E2 Lite”を選択します。この時、”RL78 E2 Lite のプロパティ”画面が表示されます。ここで各タブを選択して、デバッグ・ツール設定を行います。

e^2 studio では、ツリーで対象プロジェクトをマウス右クリックし、[デバッグ] – [デバッグの構成]を選択して表示された”デバッグ構成”画面のツリーで、[Renesas GDB Hardware Debugging]の対象プロジェクト(ここでは、”EESRL78T01_PJ01 HardwareDebug”)を選択し、表示された”Debugger”タブで、デバッグ・ツール設定を行います。

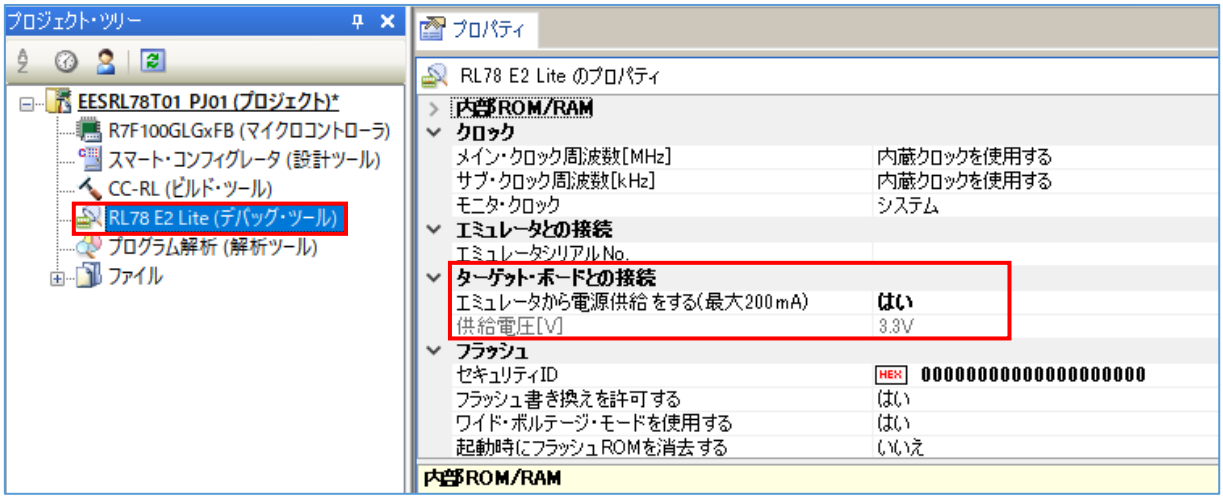
注）ターゲット・ボードに他の電源が供給されている場合や電源供給容量が不足するなど、E2 Lite を含むエミュレータからターゲット・ボードへの電源供給ができない場合があります。必ず、対象デバイス用のエミュレータのユーザーズマニュアル、およびユーザーズマニュアル別冊(RL78 接続時の注意事項)をご参照の上、ご使用ください。

7.1.4.1 ターゲット・ボードとの接続の設定

・CS+でのターゲット・ボードとの接続(E2 Lite 経由)は、 “接続用設定”タブで設定

- [ターゲット・ボードとの接続] 項目

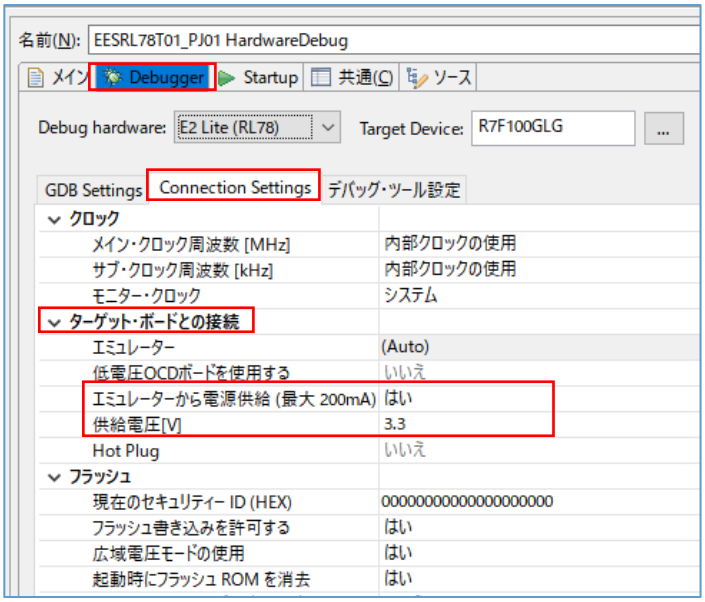
[エミュレータから電源供給をする(最大 200mA)]を”はい”に設定することで、E2 Lite からターゲット・ボードに電源供給(供給電圧:3.3V)することが可能です。



・e^2 studio でのターゲット・ボードとの接続(E2 Lite 経由)の設定は、 “Connection Settings”タブで設定

- [ターゲット・ボードとの接続] 項目

[エミュレータから電源供給(最大 200mA)]を”はい”に設定することで、E2 Lite からターゲット・ボードに電源供給(供給電圧:3.3V)することが可能です。



7.2 IAR コンパイラを使用する場合のプロジェクトの作成

IAR コンパイラは、統合開発環境として IAR Embedded Workbench を使用して作成したプロジェクトへ EES RL78 Type01 を登録し、ビルドすることができます。IAR Embedded Workbench を使用した場合のサンプル・プロジェクトの作成例を示します。IAR コンパイラ、および統合開発環境を理解するため、IAR Systems のツール製品のユーザーズマニュアルを参照してください。

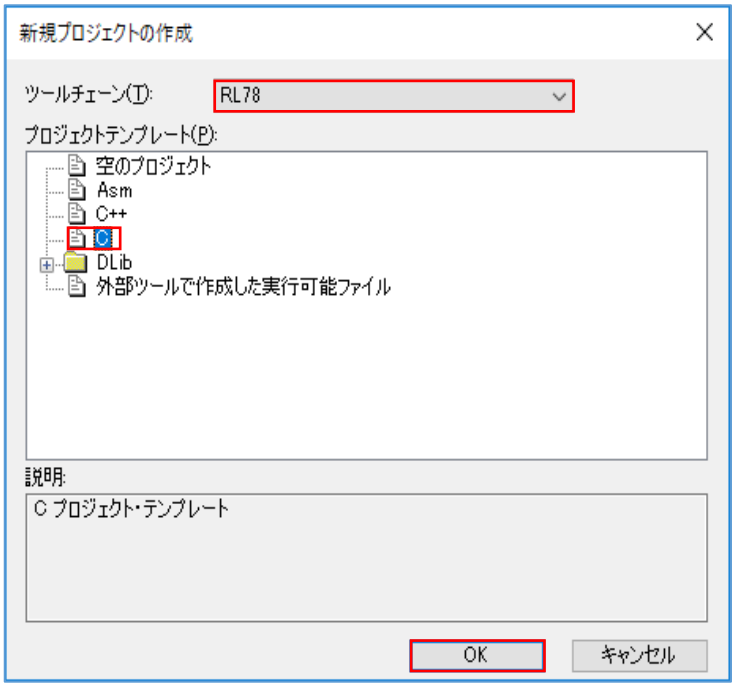
7.2.1 サンプル・プロジェクト作成例

(1) 統合開発環境 IAR Embedded Workbench を使用したサンプル・プロジェクト作成例

IAR Embedded Workbench を起動し、[プロジェクト]メニューの[新規プロジェクトの作成]を選択し、以下に示す ”新規プロジェクトの作成”ウインドウを起動します。

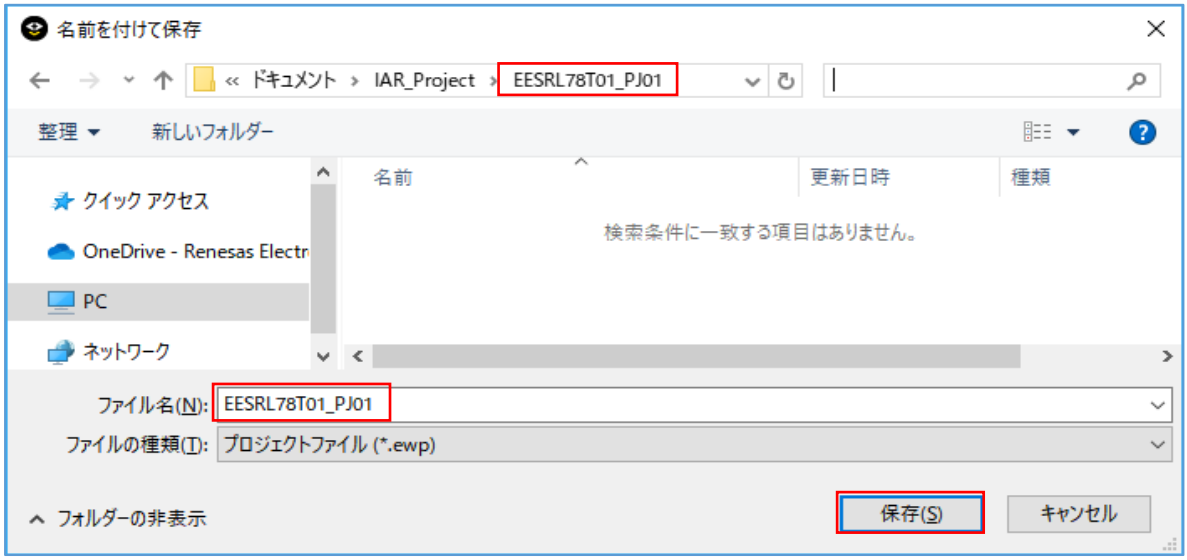
・[プロジェクトテンプレート]で、”C”を選択します。

・[OK]ボタンを押すと、[名前を付けて保存]ウインドウが表示されます。



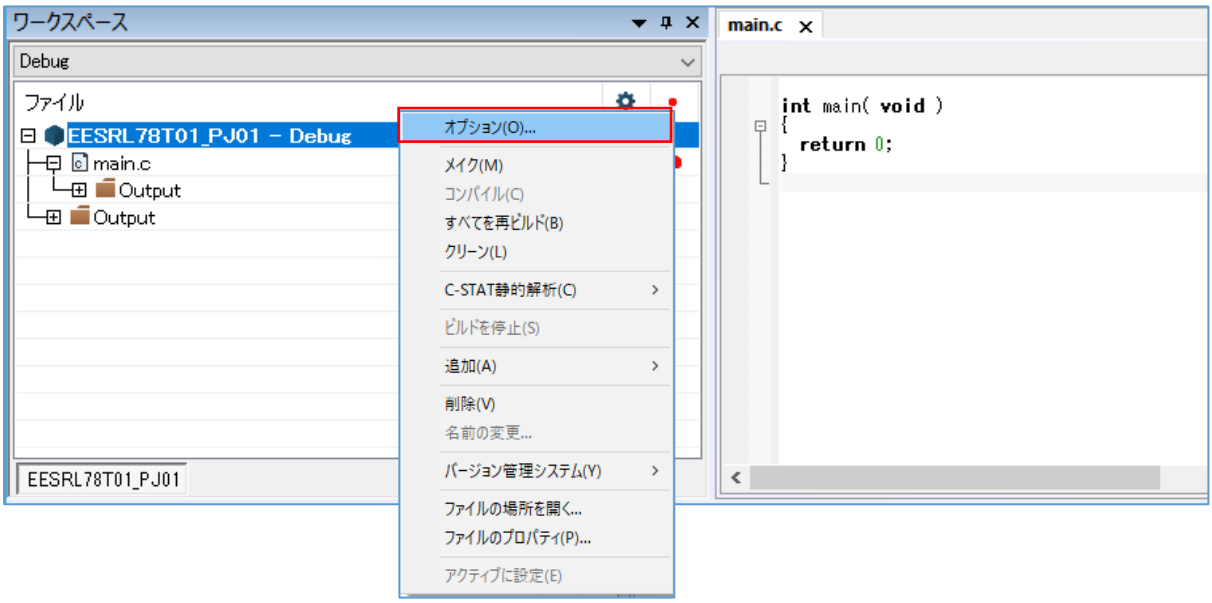
・ここでは、仮に”EESRL78T01_PJ01”フォルダを作成し、フォルダ内へ移動します。

・ここでの[プロジェクト名]は、仮に”EESRL78T01_PJ01”として保存します。



(2) 対象デバイスの選択

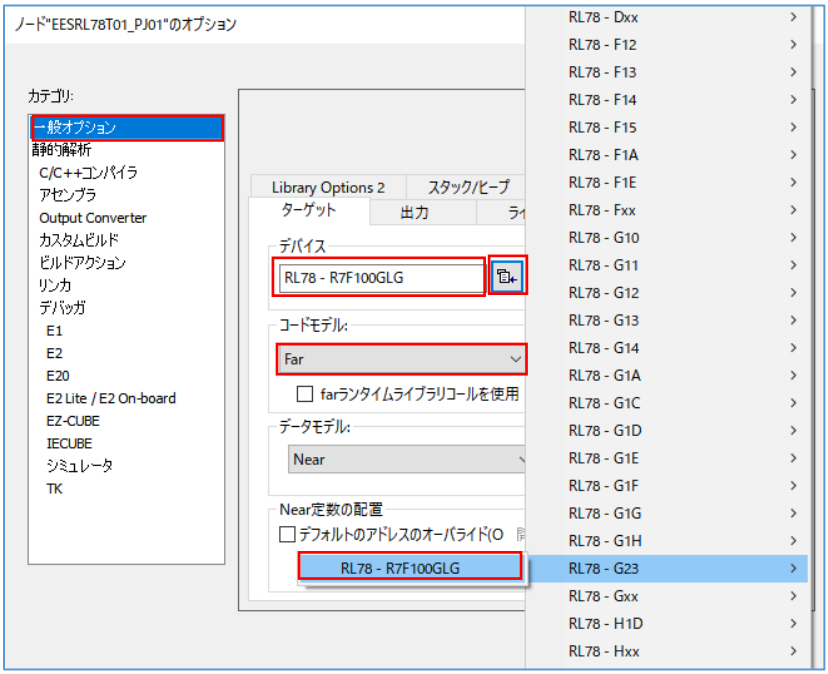
IAR Embedded Workbench ではツリーでプロジェクト(ここでは”EESRL78T01_PJ01 – Debug“)のマウス右クリックで”オプション”を選択することにより、”オプション”の画面を表示します。



・”オプション”の画面内の[一般オプション] – [ターゲット]タブの各設定を行います。

[デバイス]の” ”ボタンで”RL78 – G23” – “RL78 – R7F100GLG”を選択します。

ここでは、[コードモデル]に”Far”を選択し、[データモデル]に”Near”を選択します。

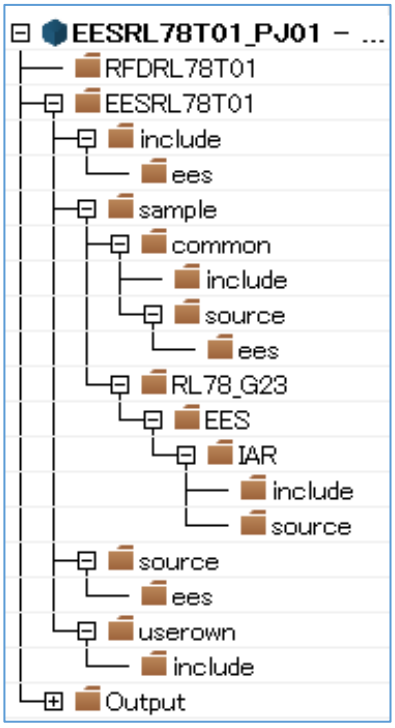


7.2.2 対象フォルダと対象ファイルの登録例

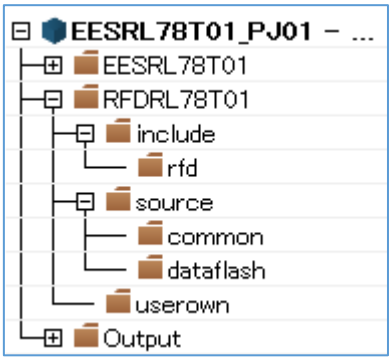
EEPROM エミュレーションを実行するために必要なファイルの登録例を記述します。

ここでは、IAR Embedded Workbench でフォルダを登録する代わりに、[プロジェクト]メニューの[グループの追加]を選択し、EES RL78 Type01 と RFD RL78 T01 のフォルダ構成と同様のグループを追加してファイルを登録する例を示します。（グループを作らずに登録することも可能です。）

(1) EES RL78 Type01、(2) RFD RL78 Type01 のグループを追加した例を示します。



(1) EES RL78 Type01



(2) RFD RL78 Type01

・統合開発環境の機能により自動的に追加されたファイルの除外

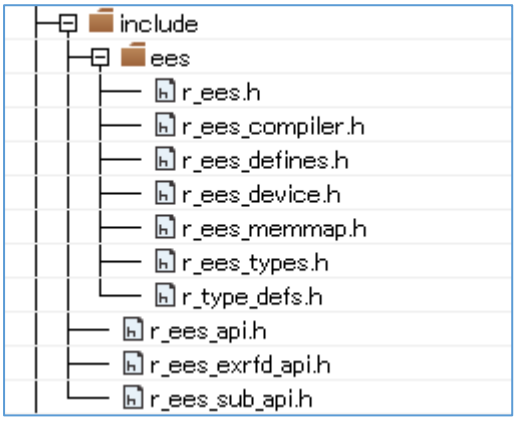
作成されたプロジェクトには、自動的に追加されるファイルがあります。これらと同様のファイルは、EES RL78 Type01 の”sample”フォルダ内にも存在するため、ツリーで各ファイルを選択し、各統合開発環境の機能を使用して、プロジェクトから外します。

- IAR Embedded Workbench では、ツリーでファイルをマウス右クリック、”削除”機能で対象の”main.c”ファイルを除外します。

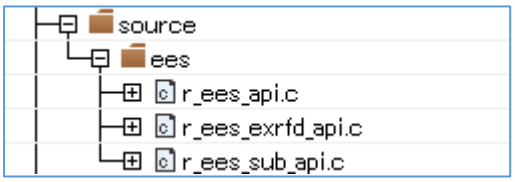
(1) EES RL78 Type01 の対象ファイルの登録

EES RL78 Type01 ソースプログラムファイルの各グループ(“include”, ”source”, ”userown”, ”sample”)に登録するファイルを以下に示します。

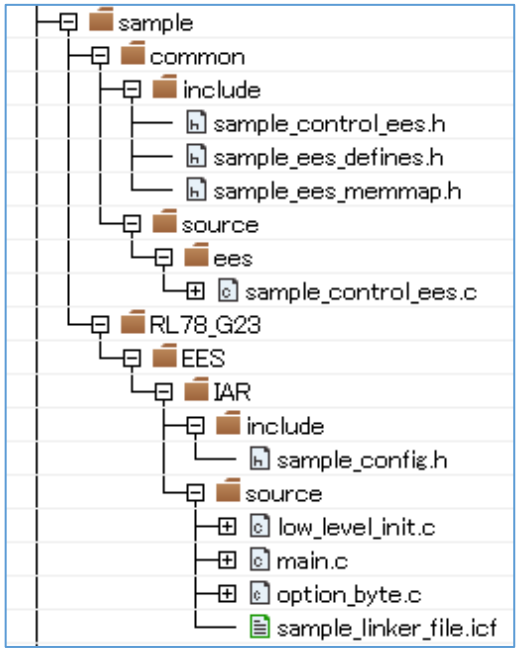
“include” グループ内



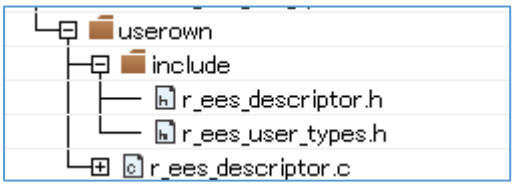
“source” グループ内



“sample” グループ内



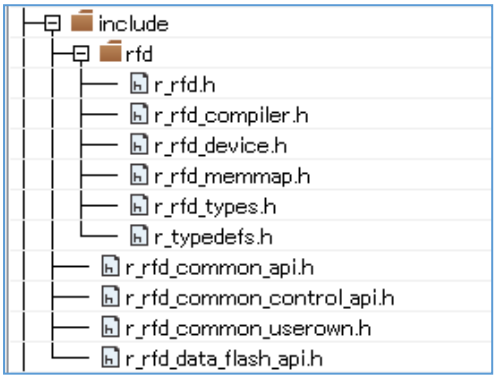
“userown” グループ内



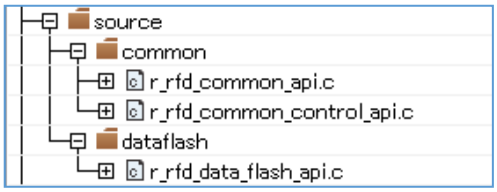
(2) RFD RL78 Type01 の対象ファイルの登録

RFD RL78 Type01 ソースプログラムファイルの各グループ(“include”, ”source”, ”userown”)に登録するファイルを以下に示します。

“include” グループ内



“source” グループ内



“userown” グループ内

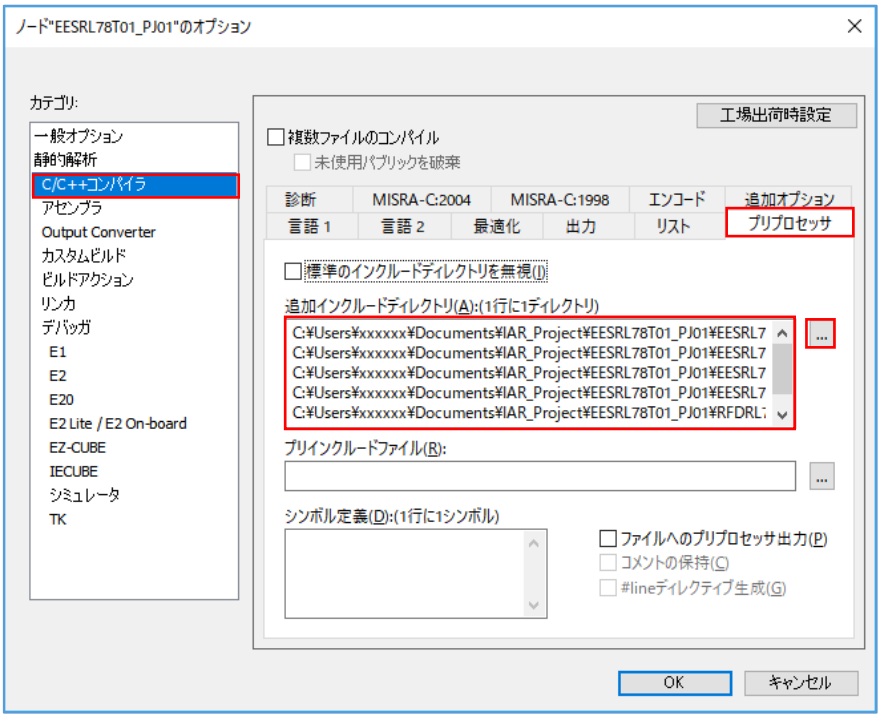
7.2.3 統合開発環境の設定

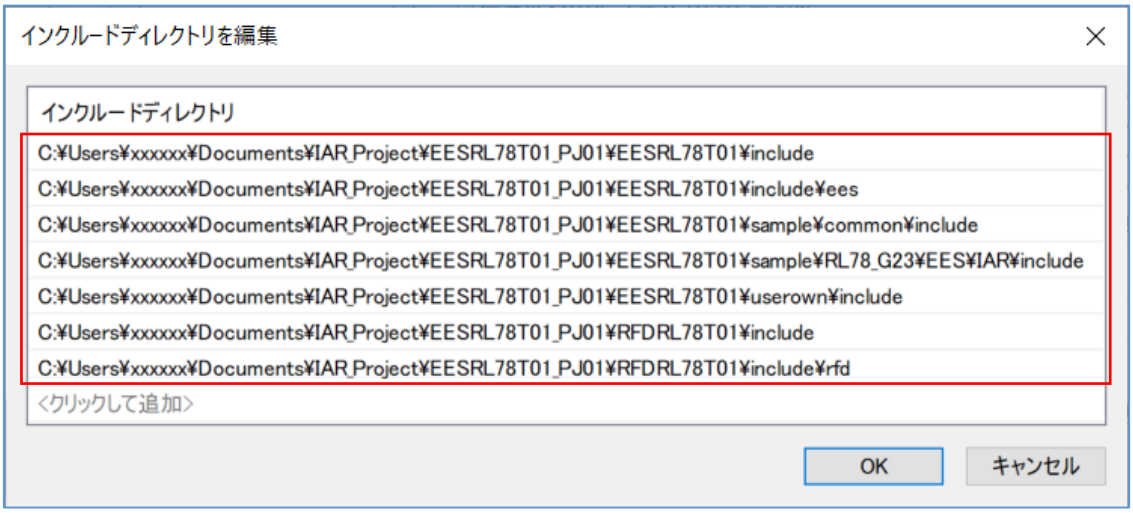
IAR コンパイラで EEPROM エミュレーションをビルドして実行するための統合開発環境の設定を行います。IAR Embedded Workbench ではツリーで[プロジェクト]をマウス右クリックして”オプション”を選択、表示された画面内の”カテゴリ”を選択して各設定を行います。

7.2.3.1 インクルード・パスの設定

IAR Embedded Workbench でのインクルード・パスの設定は、カテゴリの”C/C++コンパイラ”を選択し、“プリプロセッサ”タブで設定します。

- [追加インクルードディレクトリ(A)：(1 行に 1 ディレクトリ)]で"インクルードディレクトリを編集"ウインドウを表示して、インクルード・ディレクトリのパスを設定します。





- 設定するディレクトリパスの例

“C:\Users\xxxxxx\Documents\IAR_Project\”に、プロジェクトフォルダを置いた場合の例です。

(1) EES RL78 Type01 のインクルード・ディレクトリ

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\EESRL78T01\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\EESRL78T01\include\ees

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\EESRL78T01\sample\common\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\EESRL78T01\sample\RL78_G23\EES\IAR\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\EESRL78T01\userown\include

(2) RFD RL78 Type01 のインクルード・ディレクトリ

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\RFDRL78T01\include

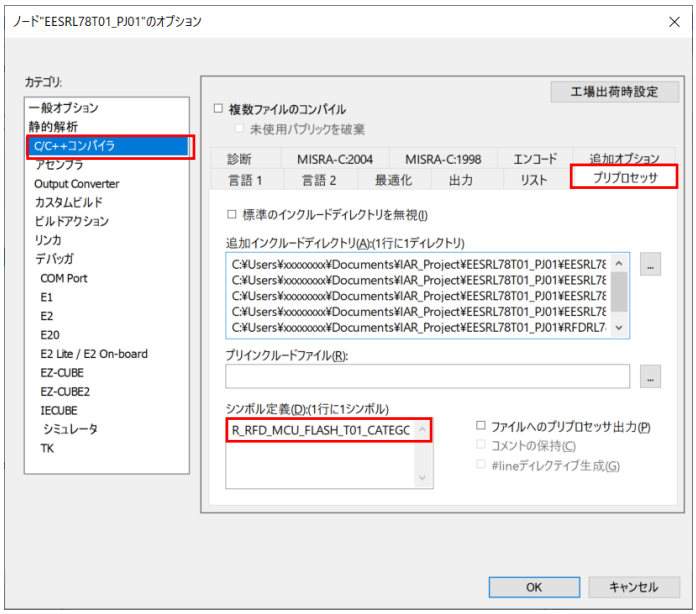
C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T01_PJ01\RFDRL78T01\include\rfd

注）インクルード・ディレクトリのパス設定については、絶対パスで指定しているとプロジェクトをコピーした時に再設定が必要になります。プロジェクトをコピーしても使用できるよう相対パス($PROJ_DIR$)を指定することも可能です。指定方法については、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをいただき、必要に応じて設定してください。

7.2.3.2 ユーザ定義マクロの設定

・IAR Embedded Workbench でのフラッシュ・メモリ制御方式分類用マクロを、"カテゴリ" – "C/C++コンパイラ"の"プリプロセッサ"タブで定義します。

– [シンボル定義(D) : (1 行に 1 シンボル)] 欄に以下のマクロを定義してください。使用するデバイスによって定義するマクロが異なります。



RL78/G23, RL78/G22 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY01

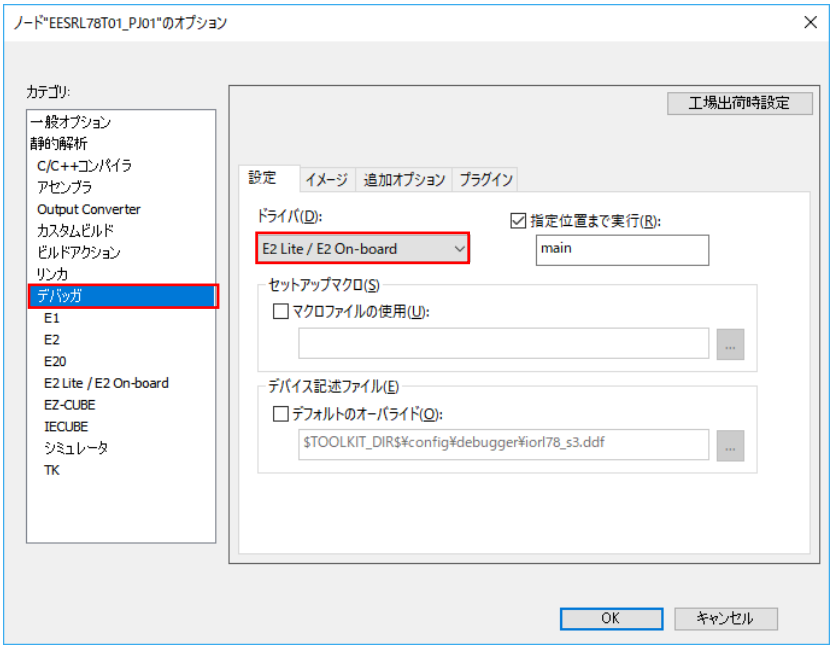
RL78/G24 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY02

注）マクロを定義していない場合、コンパイル・エラーが出力されます。

7.2.3.3 デバッガの設定

・オンチップ・デバッグを実施することを前提として、[デバッガ] – [設定]タブの[ドライバ]で”E2 Lite / E2 On-Board”を選択します。

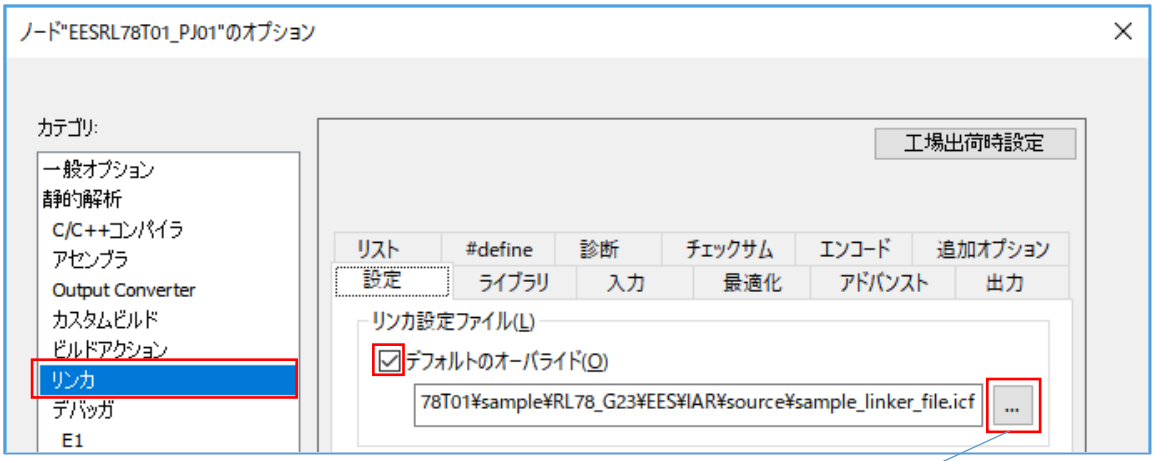


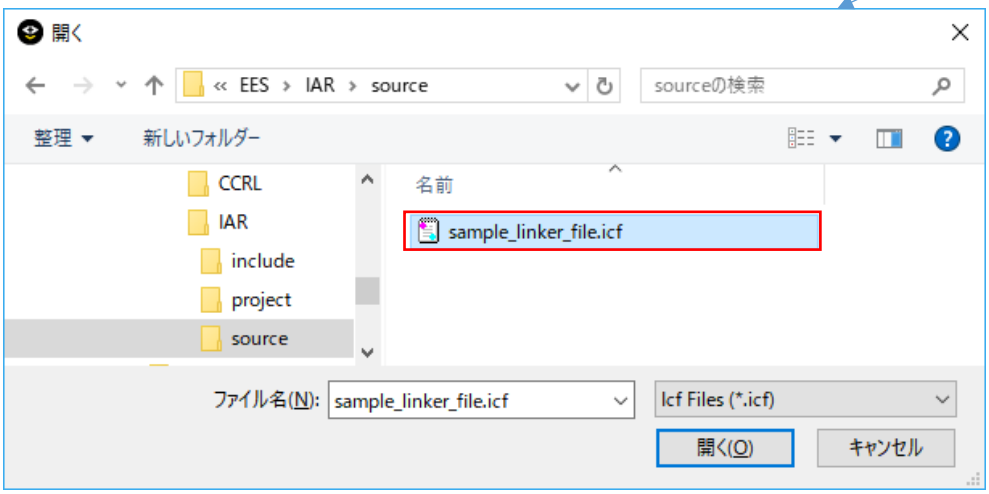
注）その他の設定項目については、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをご参照いただき、必要に応じて設定してください。

7.2.4 リンカ設定ファイル(.icf)の設定

IAR Embedded Workbench では、ビルドで実行するリンク設定をリンカ設定ファイル(*.icf)に記述します。ツリーで[プロジェクト]のマウス右クリックで”オプション”を選択、表示された画面内の[リンカ]で、[設定] – [デフォルトのオーバライド(O)]にチェックを入れ、”...”ボタンの”開く”画面でリンカ設定ファイル(*.icf)を選択します。ここでは、EES RL78 Type01 用に準備されている”sample_linker_file.icf”ファイルを選択します。

- sample_linker_file.icf (\sample\RL78_G23\EES\IAR\source\)





注）リンカ設定ファイルの記述内容、及び記述方法の詳細については、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをご参照ください。

7.2.4.1 セクション項目の設定

EES RL78 Type01 で準備されているリンカ設定ファイル(*.icf)で追加しているセクションの概要を記述します。

注）リンカ設定ファイルのセクション項目の設定、及び機能の詳細は、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをご参照ください。

(1) EES RL78 Type01 のセクション追加

EES_CODE, SMP_EES, EES_CNST の各セクションの初期値を ROM 領域(ROM_far)へ追加します。

- ROM_far 領域の追加セクション :

EES_CODE, SMP_EES, EES_CNST

- RAM_near 領域の追加セクション :

EES_VAR, SMP_VAR

(2) RFD RL78 Type01 のセクション追加

RFD_DATA の初期値と RFD_CMN, RFD_DF の各セクションを ROM 領域(ROM_far)へ追加し、RFD_DATA は RAM 領域(RAM_near)のセクションへコピーする必要があります。

- ROM_far 領域の追加セクション(プログラムと RAM 領域へコピーするためのデータ) :

RFD_DATA_init, RFD_CMN, RFD_DF

- RAM_near 領域の追加セクション(ROM 領域からコピーされるデータ) :

RFD_DATA

7.2.4.2 オプション・バイトの設定

RL78 のオプション・バイト定義は、IAR Embedded Workbench 付属のリンカ設定ファイル(*.icf)、及び EES RL78 Type01 用に準備されている” sample_linker_file.icf”ファイルに記述されています。EES RL78 Type01 でのオプション・バイト値は、”option_byte.c”ファイルに記述されています。

注）リンカ設定ファイルのオプション・バイトの設定については、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをご参照ください。

EES RL78 Type01 用リンカ設定ファイル(*.icf)のオプション・バイトの定義例

```
define block OPT_BYTE with size = 4   { R_OPT_BYTE,
                                        ro section .option_byte,
                                        ro section OPTBYTE };
                                  |
place at address mem:0x000C0            { block OPT_BYTE };
```

”option_byte.c”ファイル内のオプション・バイト値の記述例

```
#pragma location = "OPTBYTE"
__root const unsigned char option_bytes[4] = {
    0x6E,   /* 01101110 */
            /*       || */
            /*       |+-- Watchdog timer     */
            /*       |    operation stopped  */
            /*       |    in HALT/STOP mode  */
            /*       +--- Watchdog timer     */
            /*            overflow time is   */
            /*            2^17 / fIL =       */
            /*            3478.26 ms         */
            /*      +----- Watchdog timer     */
            /*             operation disabled */
            /*     ++------ 100% window open  */
            /*     |        period            */
            /*     +------- Interval interrupt */
            /*              is not used       */
    0xFF,   /* 11111111 */
            /*        | */
            /*        +-- LVD reset mode */
    0xE8,   /* HS mode 32 MHz */
    0x85    /* OCD: enables on-chip debugging function */
};
```

- ユーザ・オプション・バイト値の説明：

”option_byte.c”ファイル内のユーザ・オプション・バイト(000C0H-000C2H)の値は”6EFFE8”です。
(WDT 停止,LVD (reset モード),HS モード/32MHz [RL78/G23 の例])

”option_byte.c”ファイル内のオンチップ・デバッグ・オプション・バイト(000C3H/040C3H [RL78/G23 の例])の値は”85”です。(オンチップ・デバッグ動作許可の例)

注）対象デバイスのユーザーズマニュアルで「オプション・バイト」の章の「ユーザ・オプション・バイト」，「オンチップ・デバッグ・オプション・バイト」の内容をご確認いただき、ユーザ・アプリケーションで使用する設定値を書き込んでください。

7.2.5 オンチップ・デバッグの設定

プロジェクトのビルド実行後、E2 Lite を接続した状態で、 [プロジェクト]メニューから[ダウンロードしてデバッグ]を選択して、デバッグを開始します。

7.2.5.1 接続エラーに関する対処の例

ここでは、オンチップ・デバッグを実行時の接続エラーに関する対処(よくある例)として、”ID コード”の不一致や”電源”が正しく設定されていない場合について説明します。

注）その他の原因によりターゲットに接続できない場合は、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをご確認ください。

[ダウンロードしてデバッグ]を選択して、デバッグを開始するときに、”E2 Lite ハードウェア設定”画面が表示される場合があります。原因として、”ID コード”の不一致や”電源”が正しく設定されていない場合が考えられます。

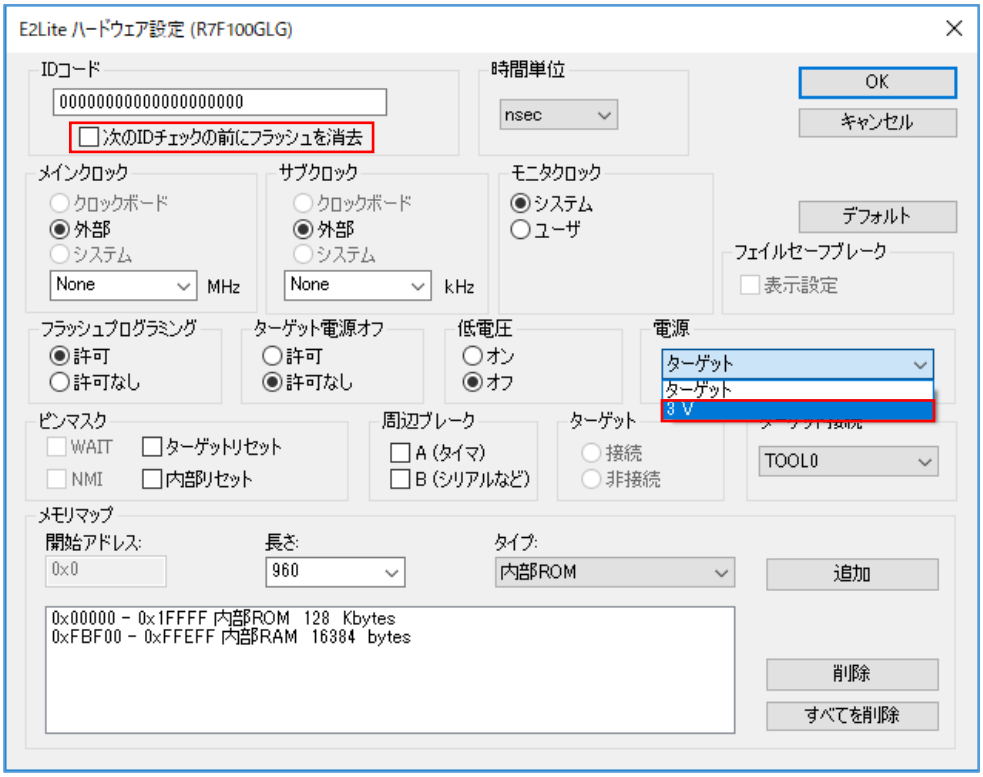
- ID コードが不一致の場合：

“ID コードをベリファイできない。”等のメッセージが表示されることがあります。この場合は、”E2 Lite ハードウェア設定”画面]の[ID コード]で、”次の ID チェックの前にフラッシュを消去”をチェックし、一度フラッシュ・メモリを消去することで、接続できる場合があります。

- 電源が設定されていない場合：

“電源”の初期状態は、”ターゲット”ですが E2 Lite から電源を供給する場合は、プルダウン・メニューで”3V”を選択します。

注）ターゲットに電源が供給されている場合、絶対に”3V”(E2 Lite から電源を供給)に設定しないでください。



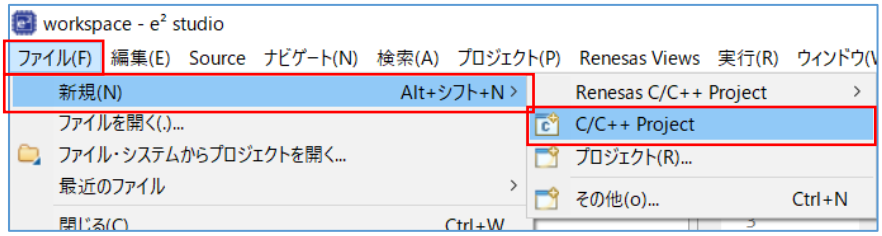
7.3 LLVM コンパイラを使用する場合のプロジェクトの作成

LLVM コンパイラは、統合開発環境として e² studio を使用して作成したプロジェクトへ EES RL78 Type01 と RFD RL78 Type01 を登録し、ビルドすることができます。e² studio を使用した場合のサンプル・プロジェクトの作成例を示します。LLVM コンパイラ、および e² studio を理解するため、それぞれのツール製品のユーザーズマニュアルを参照してください。

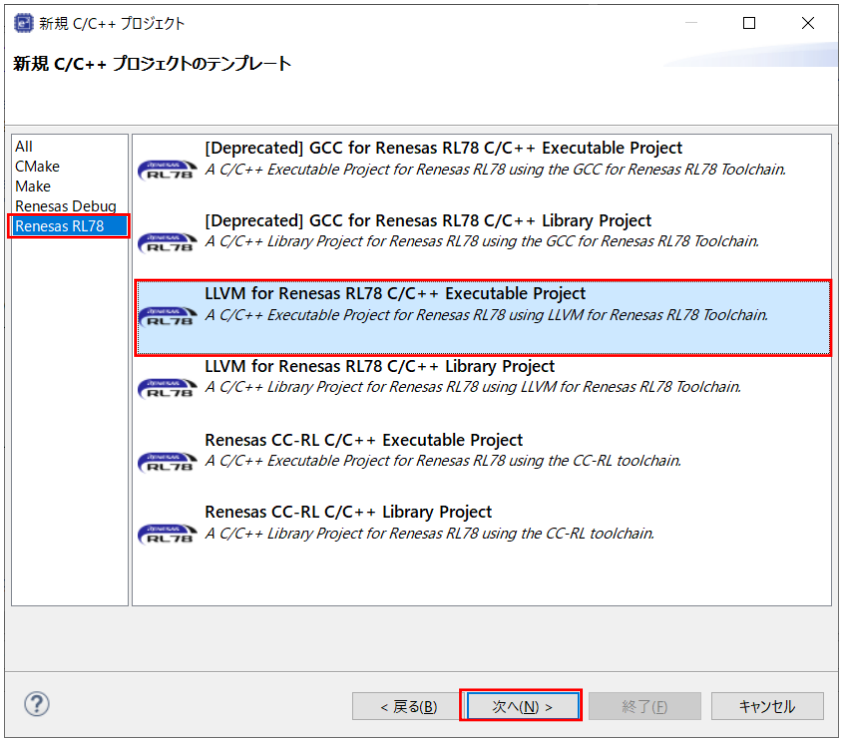
7.3.1 サンプル・プロジェクト作成例

統合開発環境(e² studio)を使用したサンプル・プロジェクト作成例

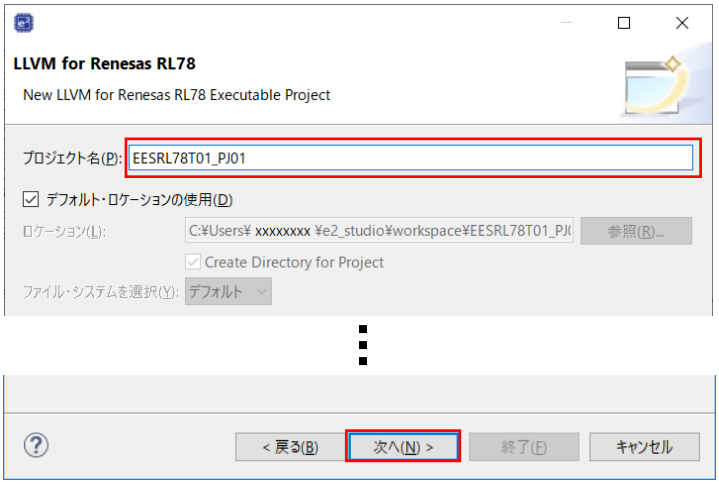
e² studio を起動し、[ファイル]メニューの[新規]から[C/C++ Project]を選択し、"新規 C/C++プロジェクトのテンプレート"ウインドウを起動します。



・[Renesas RL78]を選択して表示された[LLVM for Renesas RL78 C/C++ Executable Project]を選択、"次へ"ボタンを押します。



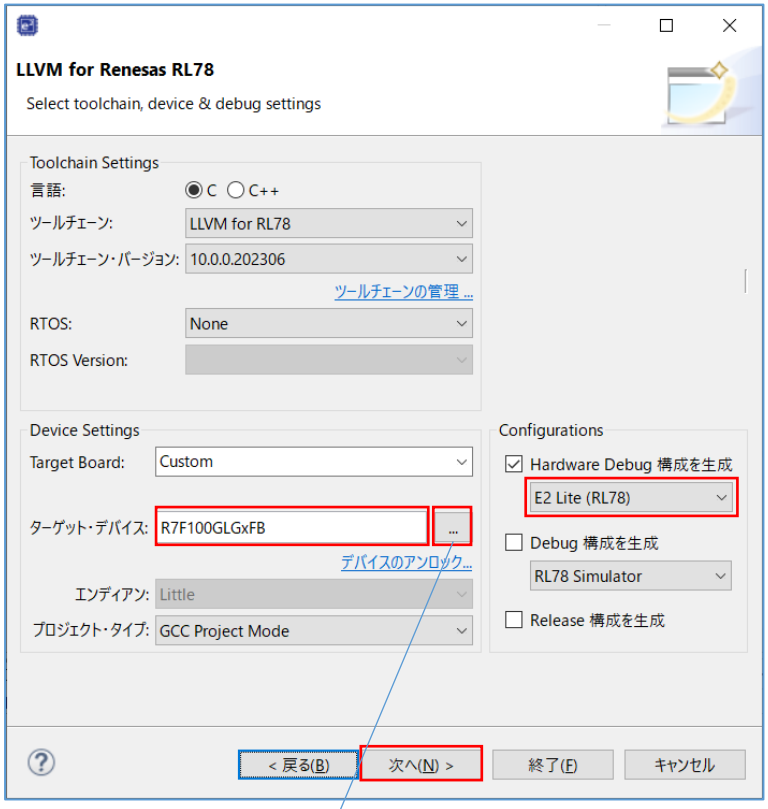
・"New LLVM for Renesas RL78 Executable Project"ウインドウで、プロジェクト名を入力して"次へ"ボタンを押します。（ここでは、仮に"EESRL78T01_PJ01"とします。）

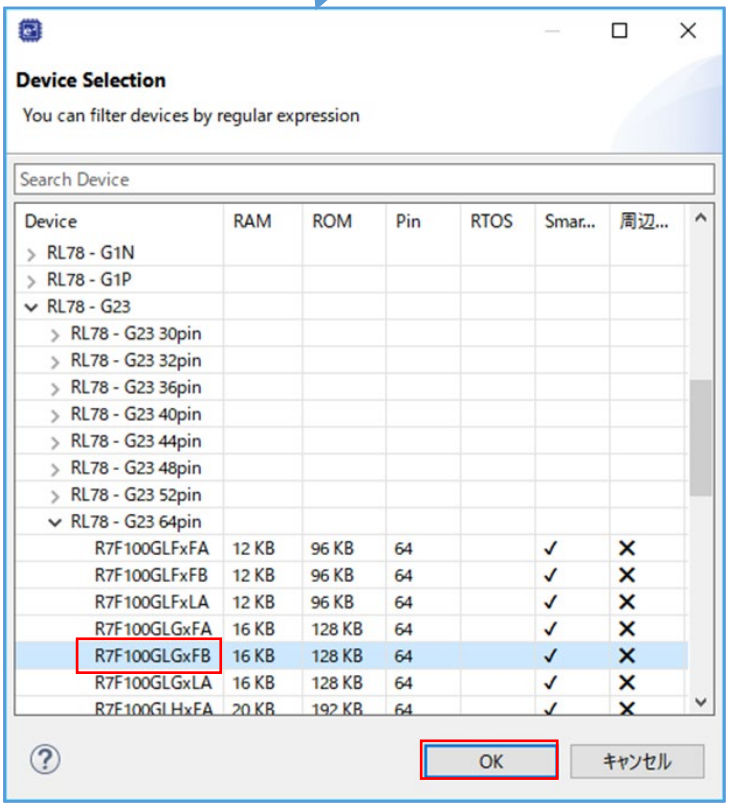


・[Device Settings]の[ターゲット・デバイスで、"RL78 - G23" - "RL78 - G23 64pin" - "R7F100GLGxFB"を選択し[OK]ボタンを押します。

・デバッグ・ツールに E2 Lite を選択し、オンチップ・デバッグを実施することを前提としています。
[Configurations]で"Hardware Debug 構成を生成"にチェックが入った状態で、E2 Lite (RL78)を選択します。

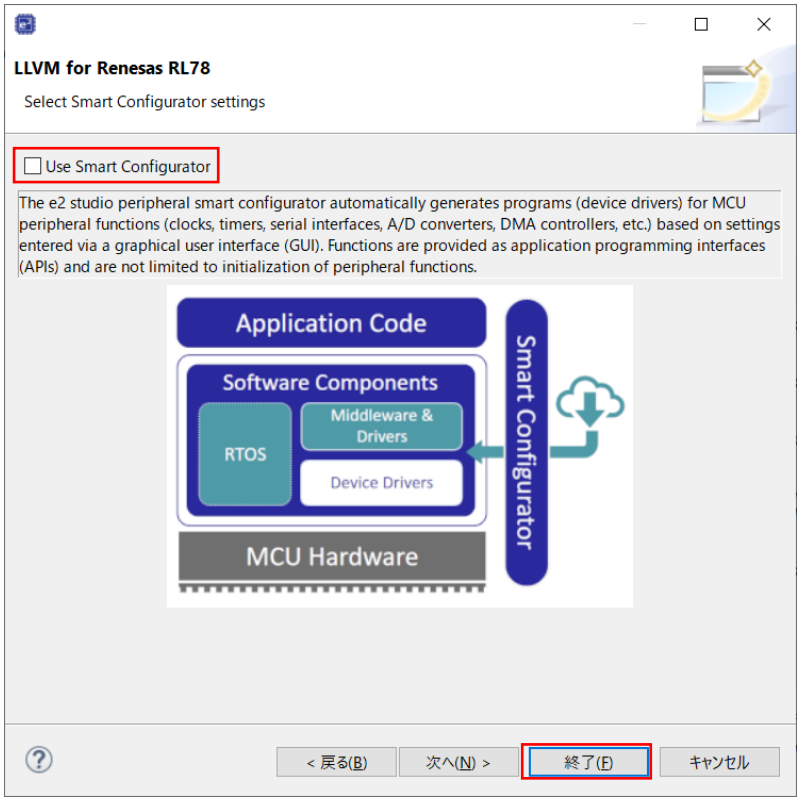
・[次へ]ボタンを押します。





・[Use Smart Configurator]のチェックを外します。

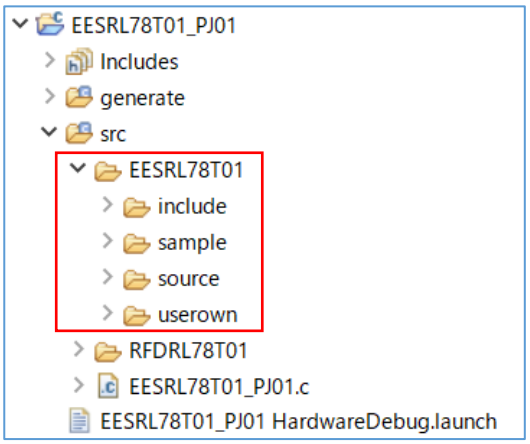
・[終了]ボタンを押します。



7.3.2 対象フォルダと対象ファイルの登録

EES RL78 Type01 を使用して、EEPROM エミュレーションを実行するために必要なファイルの登録例を記述します。まず、EESRL78T01 ソースプログラムの"EESRL78T01"フォルダを登録します。このフォルダには、"include", "sourse", "userown", "sample"フォルダが含まれます。

その他の手順として、”include”, "source", "userown", "sample"の全てのフォルダを登録し、不要なファイルとフォルダを、 [リソース構成] – [ビルドから除外…]機能により、対象から外すこともできます。



e² studio の EES RL78 Type01 登録時のツリー画面

注）e² studio が出力する"generate"フォルダは、必要に応じて登録してください。

・e² studio から対象製品用に出力された最新の I/O ヘッダ・ファイルの登録

"iodefine.h"と"iodefine_ext.h"は、e² studio が対象製品用に出力する I/O ヘッダ・ファイルです。 EES RL78 Type01 に含まれている"iodefine.h", "iodefine_ext.h"の代わりに置き換えてご使用頂くことを推奨いたします。

e² studio から出力された"iodefine.h", "iodefine_ext.h"を EES RL78 Type01 内の"iodefine.h", "iodefine_ext.h"と入れ替える、もしくは上書きしてください。

・e² studio から対象製品用に出力された最新のベクタ・テーブルファイルの登録

"interrupt_handlers.h"と"inthandler.c"と"vects.c"は、e² studio が対象製品用に出力するベクタテーブルが記載されているファイルです。製品によって異なるため、EES RL78 Type01 に含まれている"interrupt_handlers.h", "inthandler.c", "vects.c"の代わりに置き換えてご使用ください。置き換えた場合、"vects.c"のオプション・バイト値を変更してください。オプション・バイト値の設定については、"7.3.4 オプション・バイトの設定"をご参照ください。

e² studio が"iodefine.h", "iodefine_ext.h", "interrupt_handlers.h", "inthandler.c", "vects.c を出力するフォルダ：

"[プロジェクト名]/generate"

"iodefine.h", "iodefine_ext.h","interrupt_handlers.h"ファイルを入れ替え、もしくは上書きするフォルダ：

"\[プロジェクト名]\src\EESRL78T01\sample\RL78_G23\EES\LLVM\include"

"inthandler.c", "vects.c"ファイルを入れ替え、もしくは上書きするフォルダ：

"\[プロジェクト名]\src\EESRL78T01\sample\RL78_G23\EES\LLVM\source"

・e^2 studio の機能により自動的に追加されたファイルの除外

作成されたプロジェクトには、自動的に追加されるファイルがあります。これらと同様のファイルは、EES RL78 Type01 の"sample"フォルダ内にも存在するため、プロジェクト・ツリーから各ファイルを選択し、統合開発環境の機能を使用して、プロジェクトから外します。

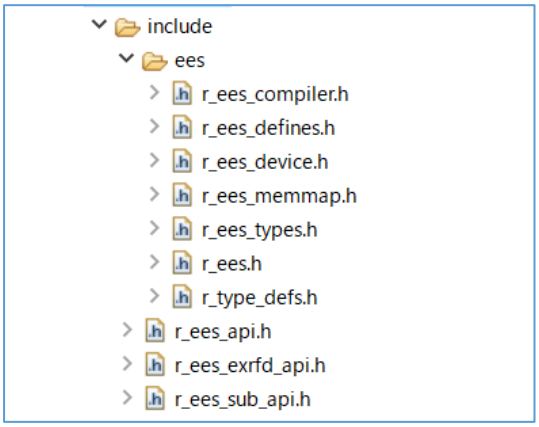
- e^2 studio ではツリーでファイルをマウス右クリックしメニューから"削除"、もしくは"プロパティ"で表示された［設定］画面で、"ビルドからリソースを除外"にチェックを入れ、対象ファイル（対象フォルダ）を除外します。

［プロジェクト名］/generate フォルダ内の"hwinit.c", "linker_script.ld", "start.S", "typedefine.h"、および[プロジェクト名]/src フォルダ内の[プロジェクト名].c(ここでは"EESRL78T01_PJ01.c")については、EES RL78 Type01 では未使用なのでプロジェクトから除外します。

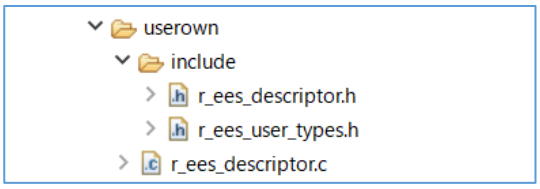
(1) EES RL78 Type01 の対象フォルダと対象ファイルの登録

EES RL78 Type01 ソースプログラムファイルの各フォルダ("include", "source", "userown", "sample")と登録ファイルを以下に示します。

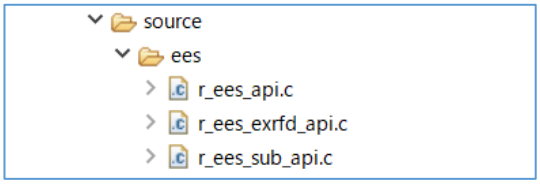
include フォルダ内



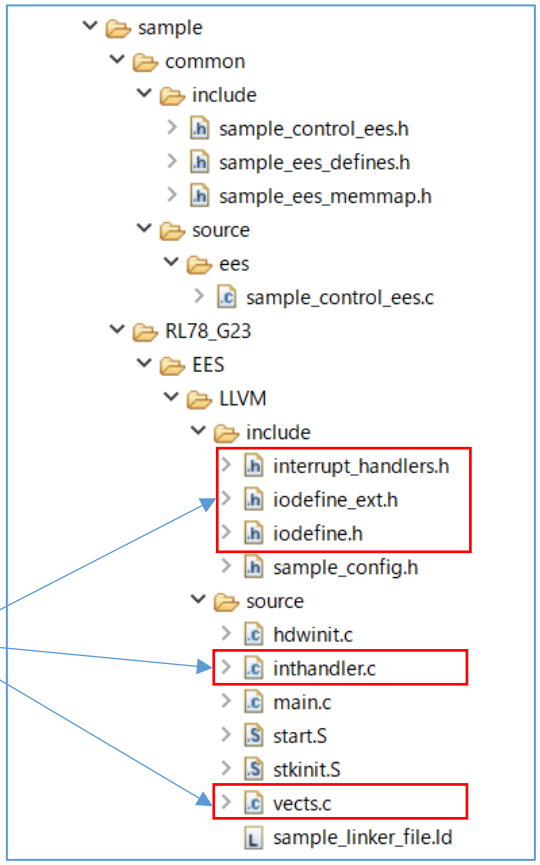
userown フォルダ内



source フォルダ内



sample フォルダ内



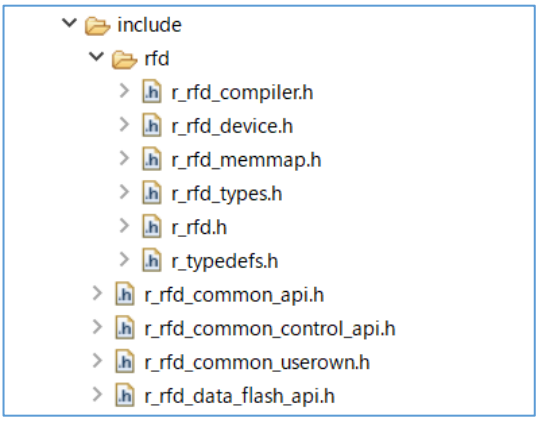
e² studio で出力した"iodefine.h", "iodefine_ext.h", "interrupt_handlers.h", "inthandler.c", "vects.c"に置き換えてください。

※**"vects.c"**は、オプション・バイト値を変更してください。

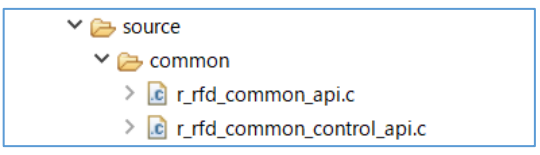
(2) RFD RL78 Type01 の対象フォルダと対象ファイルの登録

RFD RL78 Type01 ソースプログラムファイルの各フォルダ("include", "source", "userown")と登録ファイルを以下に示します。

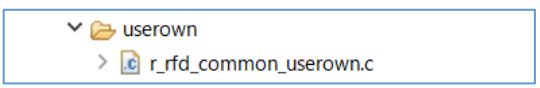
include フォルダ内



source フォルダ内



userown フォルダ内



7.3.3 ビルド・ツールの設定

LLVM コンパイラで EES RL78 Type01 をビルドして実行するための e2 studio の設定を行います。

e2 studio ではツリーでプロジェクト(ここでは"EESRL78T01_PJ01")のマウス右クリックで"プロパティ"を選択することにより、表示された画面内のビルド・ツールの各設定を行います。

7.3.3.1 インクルード・パスの設定

e2 studio でのインクルード・パスの設定は、"プロパティ"ウインドウで設定します。

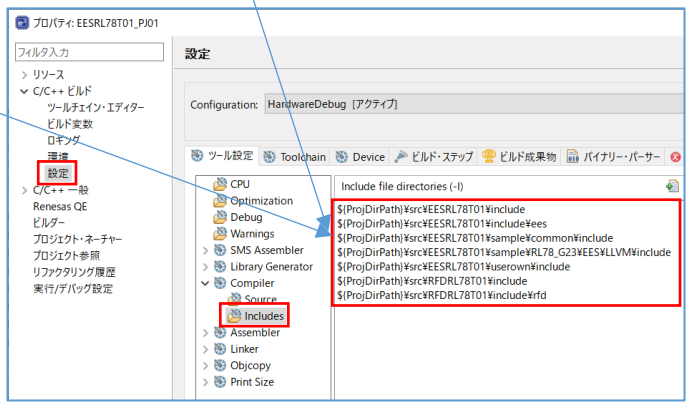
- "C/C++ビルド" [設定] – "Compiler" [includes]で表示された画面でインクルード・ファイルのパスを設定します。

(1) EES RL78 Type01 の include パス

```
${ProjDirPath}\src\EESRL78T01\include
${ProjDirPath}\src\EESRL78T01\include\ees
${ProjDirPath}\src\EESRL78T01\sample\common\include
${ProjDirPath}\src\EESRL78T01\sample\RL78_G23\EES\LLVM\include
${ProjDirPath}\src\EESRL78T01\userown\include
```

(2) RFD RL78 Type01 の include パス

```
${ProjDirPath}\src\RFDRL78T01\include
${ProjDirPath}\src\RFDRL78T01\include\rfd
```



7.3.3.2 ユーザ定義マクロの設定

e2 studio でのフラッシュ・メモリ制御方式分類用マクロを"プロパティ"ウインドウで定義します。

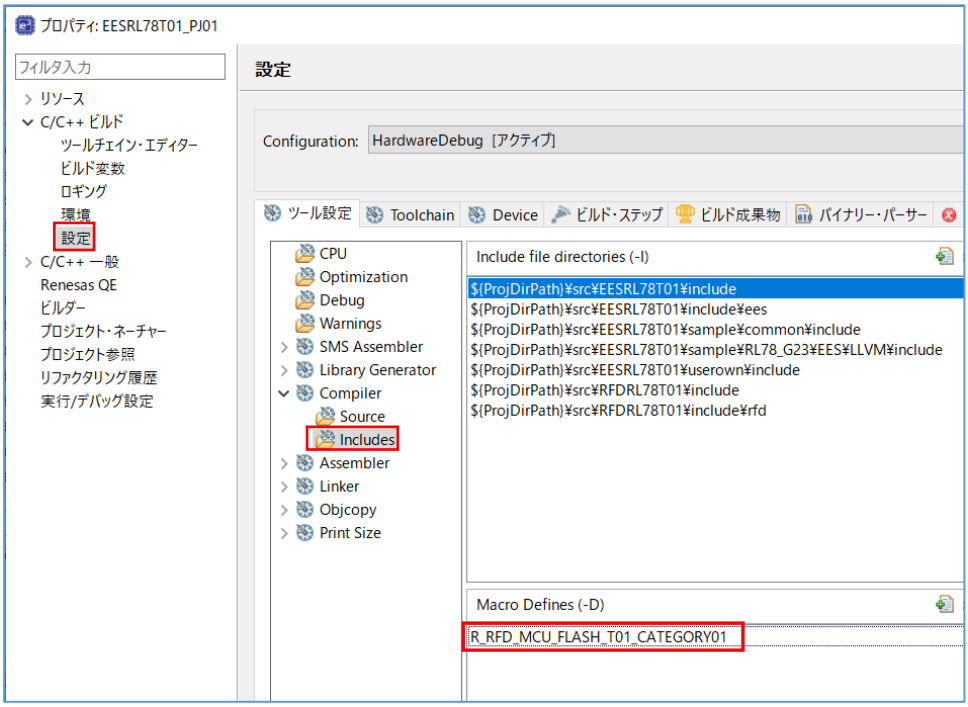
- "C/C++ビルド" [設定] – "Compiler" [Includes]で表示された"Macro Defines (-D)"の欄に以下のマクロを定義してください。使用するデバイスによって定義するマクロが異なります。

RL78/G23, G22 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY01

RL78/G24 を使用する場合に定義するマクロ：

R_RFD_MCU_FLASH_T01_CATEGORY02

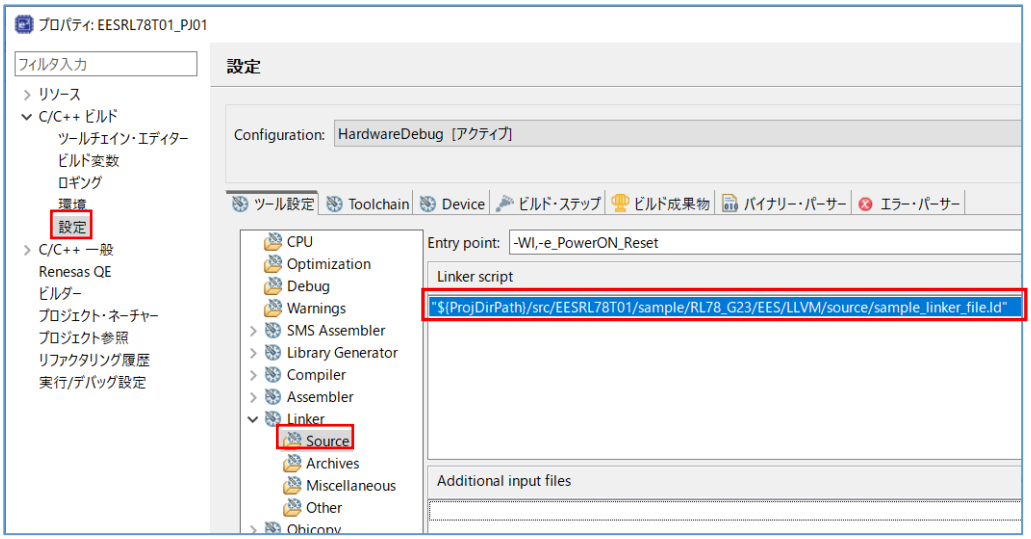


注）マクロを定義していない場合、コンパイル・エラーが出力されます。

7.3.3.3 リンカ・スクリプトファイル(.ld)の設定

LLVM では、ビルドで実行するリンク設定をリンカ・スクリプトファイル(*.ld)に記述します。ツリーの[プロジェクト]上でマウスの右クリックで"プロパティ"を選択、"C/C++ビルド" [設定] – "Linker" [Source]で表示された画面の"Linker script"欄に、リンカ・スクリプトファイルのパスを設定します。ここでは、EES RL78 Type01 用に準備されている"sample_linker_file.ld"ファイルを選択します。EES RL78 Type01 用のリンカ・スクリプトファイル(*.ld)は以下の通りです。

- sample_linker_file.ld (\sample\RL78_G23\EES\LLVM\source\)



注）リンカ・スクリプトファイルの記述内容、及び記述方法の詳細については、LLVM のリファレンスマニュアルをご参照ください。

7.3.3.4 セクション項目の設定

EES RL78 Type01 で準備されているリンカ・スクリプトファイル (*.ld)で追加しているセクションの概要を記述します。

(1) EES RL78 Type01 のセクション

- ROM 領域に配置されるコードのセクション：

  EES_CODE, SMP_EES

  (EES_ROM_CODE)

- ROM 領域に配置される Const データのセクション：

  EES_CNST

- RAM 領域に配置されるデータのセクション：

  EES_VAR, SMP_VAR

(2) RFD RL78 Type01 のセクション

- ROM 領域に配置されるコードのセクション：

  RFD_CMN, RFD_DF

  (RFD_ROM_CODE)

- ROM 領域から RAM 領域にコピーされるデータのセクション：

  RFD_DATA

注）LLVM コンパイラ使用時は、同一セクション内での共通処理が検出された場合に、コンパイラが自動的に別名のサブセクションを追加するような場合があるため、sample_linker_file.ld ファイル内の記述に "EES_XXXX.*", "SMP_XXXX.*" ("XXXX" = "CODE" or "EES" or "VAR" or "CNST")を追加しています。

追加される可能性があるサブセクションの例：EES_CODE.outlined-functions 等

その他、リンカ・スクリプトファイル(*.ld)の記述内容、及び記述方法の詳細については、LLVM のリファレンスマニュアルをご参照ください。

7.3.4 オプション・バイトの設定

LLVM コンパイラ使用時のオプション・バイトの設定は、"sample"フォルダに含まれる"vects.c"ファイルで設定します。

"\[プロジェクト名]\src\EESRL78T01\sample\RL78_G23\EES\LLVM\source\vects.c "

サンプル・プログラムで提供されている"vects.c"ファイルでは、オプション・バイト値とユーザ・オプション・バイト値を"Option_Bytes"に次のように設定しています。

"0x6e, 0xff, 0xe8, 0x85" (WDT 停止, LVD(reset モード), HS モード/32MHz, オンチップ・デバッグ動作許可 [RL78/G23 の例])

```
#include "interrupt_handlers.h"

extern void PowerON_Reset (void);

const unsigned char Option_Bytes[]   __attribute__ ((section (".option_bytes"))) = {
        0x6e, 0xff, 0xe8, 0x85
};
```

注）オンチップ・デバッグを実施することを前提とした設定例です。
対象デバイスのユーザーズマニュアルの「オプション・バイト」章の「ユーザ・オプション・バイト」、「オンチップ・デバッグ・オプション・バイト」の内容をご確認いただき、使用する設定値を書き込んでください。

7.3.5 デバッグ・ツールの設定

ここでは、デバッグ・ツールに E2 Lite を選択してオンチップ・デバッグを行う場合のターゲット・ボードとの接続の設定について説明します。デバッグ・ツール設定の詳細については、統合開発環境のユーザーズマニュアルを参照してご確認ください。

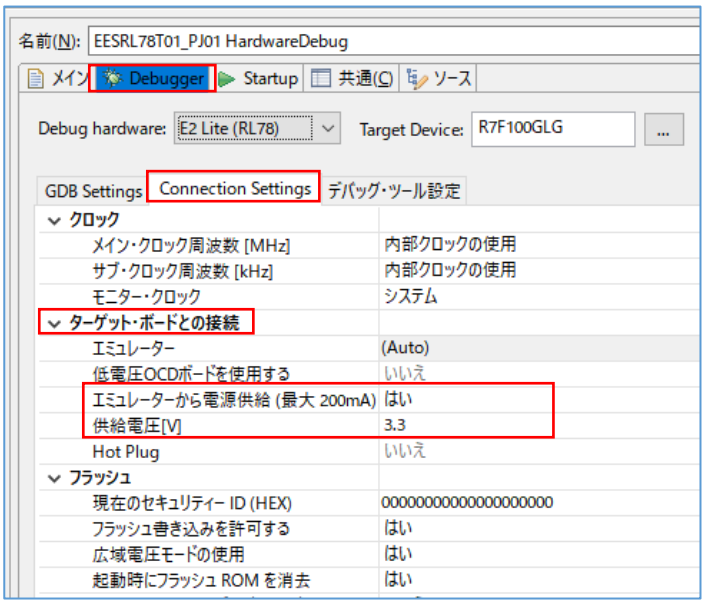
e2 studio では、ツリーで対象プロジェクトをマウス右クリックし、[デバッグ] – [デバッグの構成]を選択して表示された"デバッグ構成"画面のツリーで、[Renesas GDB Hardware Debugging]の対象プロジェクト（ここでは、"EESRL78T01_PJ01 HardwareDebug"）を選択し、表示された"Debugger"タブで、デバッグ・ツール設定を行います。

注）ターゲット・ボードに他の電源が供給されている場合や電源供給容量が不足するなど、E2 Lite を含むエミュレータからターゲット・ボードへの電源供給ができない場合があります。必ず、対象デバイス用のエミュレータのユーザーズマニュアル、およびユーザーズマニュアル別冊(RL78 接続時の注意事項)をご参照の上、ご使用ください。

・e2 studio でのターゲット・ボードとの接続（E2 Lite 経由）の設定は、"Connection Settings"タブで設定

- [ターゲット・ボードとの接続] 項目

[エミュレータから電源供給（最大 200mA）]を"はい"に設定することで、E2 Lite からターゲット・ボードに電源供給（供給電圧:3.3V）することが可能です。



7.4 デバイス変更に伴う設定

EES RL78 Type01 のサンプル・プログラムが対象としているデバイス以外を使用する場合、ROM や RAM、データ・フラッシュ・メモリのサイズが異なるため、セクションのアドレス設定やサンプル・プログラムの一部を変更する必要があります。この項では、RL78_G23 フォルダが対象としているデバイス以外を使用する場合の変更手順、および変更箇所について説明します。

“sample”フォルダの対象デバイス：

- RL78_G23 フォルダ[CATEGORY01]

  用意されているファイル群の対象デバイス：RL78/G23(R7F100GLG ROM:128KB, RAM:16KB,DF:8KB)

- RL78_G24 フォルダ[CATEGORY02]

  用意されているファイル群の対象デバイス：RL78/G24(R7F101GLG ROM:128KB, RAM:12KB,DF:4KB)

設定値の変更には"RL78 用 Renesas Flash Driver, EEPROM Emulation Software 対象 MCU リスト – General-Purpose"（以降、対象 MCU リスト）を参照し、使用しているデバイスにあわせて設定値を変更します。

“sample”フォルダ内に対象デバイスグループのフォルダ名が存在する場合は、そのフォルダを利用します。対象デバイスグループのフォルダ名が存在しない場合は、対象 MCU リストに記述されている"CATEGORY"番号が同じデバイスのフォルダを利用します。RL78/G22 を使用する場合、"RL78_G22"サンプルフォルダが存在しないため、同じ"CATEGORY01"の RL78/G23 用の RL78_G23 フォルダを利用します。

・対象 MCU リストの抜粋

対象MCU

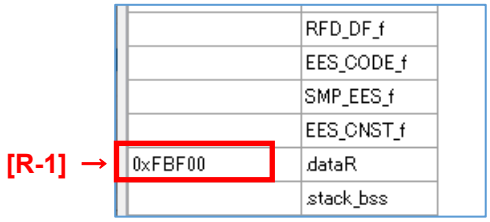
MCU Group	Code Flash memory		User RAM		Data F		[R-7]	[R-8]	Target MCU name
	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	Size (bytes)	S	END_BLOCK	CATEGORY	
RL78/G22	32K	0x00000 - 0x07FFF	4K	0xFEF00 - 0xFFEFF	2K	0x	16	01	R7F102GxC(x = 4, 6, 7, 8, A, B, C, E, F, G)
	64K	0x00000 - 0x0FFFF	4K	0xFEF00 - 0xFFEFF	2K	0x	32	01	R7F102GxE(x = 4, 6, 7, 8, A, B, C, E, F, G)
RL78/G23	96K	0x00000 - 0x17FFF	12K	0xFCF00 - 0xFFEFF	8K	0x	48	01	R7F100GxF(x = A, B, C, E, F, G, J, L)
	128K	0x00000 - 0x1FFFF	16K	0xFBF00 - 0xFFEFF	8K	0x	64	01	R7F100GxG(x = A, B, C, E, F, G, J, L, M, P)
	192K	0x00000 - 0x2FFFF	20K	0xFAF00 - 0xFFEFF	8K	0x	96	01	R7F100GxH(x = A, B, C, E, F, G, J, L, M, P)
	256K	0x00000 - 0x3FFFF	24K	0xF9F00 - 0xFFEFF	8K	0x	128	01	R7F100GxJ(x = A, B, C, E, F, G, J, L, M, P, S)
	384K	0x00000 - 0x5FFFF	32K	0xF7F00 - 0xFFEFF	8K	0x	192	01	R7F100GxK(x = F, G, J, L, M, P, S)
	512K	0x00000 - 0x7FFFF	48K	0xF3F00 - 0xFFEFF	8K	0x	256	01	R7F100GxL(x = F, G, J, L, M, P, S)
	768K	0x00000 - 0xBFFFF	48K	0xF3F00 - 0xFFEFF	8K	0x	384	01	R7F100GxN(x = F, G, J, L, M, P, S)
RL78/G24	64K	0x00000 - 0x0FFFF	12K	0xFCF00 - 0xFFEFF	4K	0x	32	02	R7F101GxE(x = 6, 7, 8, A, B, E, F, G, J, L)
	128K	0x00000 - 0x1FFFF	12K	0xFCF00 - 0xFFEFF	4K	0x	64	02	R7F101GxG(x = 6, 7, 8, A, B, E, F, G, J, L)

以降、対象 MCU リストの参照例と変更箇所の記載例を示します。

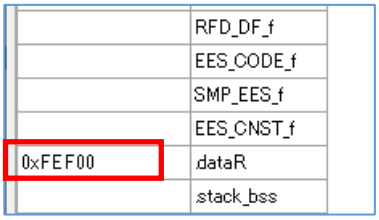
- 対象 MCU リストの参照例

例えば、次の図のように[R-1] が指している箇所の設定値（RAM の先頭アドレス）を変更するとします。ここでは、対象 MCU リストに記載されている RAM の先頭アドレス [R-1] (RAM Start Address)の設定値を参照して、RL78/G22(R7F102GxE)の値を設定します。

例）RAM の先頭アドレス変更箇所：RL78/G23(R7F100GxG RAM: 16KB)



例）RL78/G22(R7F102GxE RAM: 4KB)を使用する場合の RAM の先頭アドレス値を設定



[R-1] に設定する値は、対象 MCU リストを参照して対象デバイスの RAM の先頭アドレスを設定します。

対象 MCU リストの"Target MCU name"の列から、R7F102GxE の行を検索します。次に、[R-1] の列から R7F102GxE の行と交わるセルを検索します。

・対象 MCU リストの表示例

MCU Group	Code Flash memory		User RAM		Data Flash memory		[R-1]	[R-2]	[R-3]	[R-4]	[R-5]	[R-6]	[R-7]	[R-8]	Target MCU name
	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	RAM Start Address	ROM End Address 1	ROM End Address 2	Data Flash End Address	OCD_ROM	Trace_RAM	END_BLOCK	CATEGORY	
RL78/G22	32K	0x00000 - 0x07FFF	4K	0xFEF00 - 0xFFEFF	2K	0xF1000 - 0xF17FF	0xFEF00	0x07FFF	-	0xF17FF	0x7E00	0xFF300	16	01	R7F102GxC(x = 4, 6, 7, 8, A, B, C, E, F, G)
	64K	0x00000 - 0x0FFFF	4K	0xFEF00 - 0xFFEFF	2K	0xF1000 - 0xF17FF	0xFEF00	0x0FFFF	-	0xF17FF	0xFE00	0xFF300	32	01	R7F102GxE(x = 4, 6, 7, 8, A, B, C, E, F, G)

"0xFEF00"が該当するので、RL78/G22(R7F102GxE)における[R-1] の設定値に"0xFEF00"を設定します

[R-1] RAM Start Address	[R-2] ROM End Address 1	[R-3] ROM End Address 2	[R-4] Data Flash End Address	[R-5] OCD_ROM	[R-6] Trace_RAM	[R-7] END_BLOCK	[R-8] CATEGORY	Target MCU name
0xFEF00	0x07FFF	-	0xF17FF	0x7E00	0xFF300	16	01	R7F102GxC(x = 4, 6, 7, 8, A, B, C, E, F, G)
0xFEF00	0x0FFFF	-	0xF17FF	0xFE00	0xFF300	32	01	R7F102GxE(x = 4, 6, 7, 8, A, B, C, E, F, G)

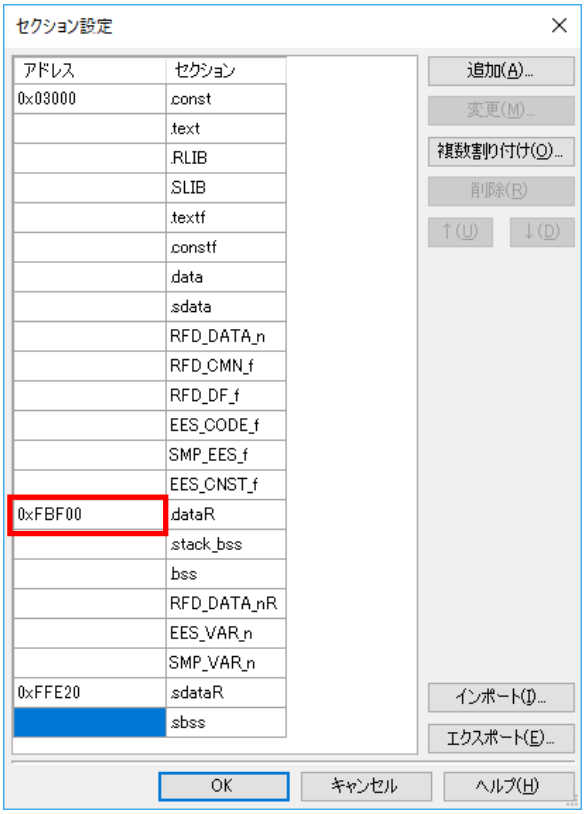
- 変更箇所の記載例

「7.3.1 CC-RL コンパイラ環境の設定」以降に、RL78/G23(R7F100GxG)の設定値から変更が必要な箇所を記載しています。その変更が必要な箇所には、"[R- x] →" のように示しているので、対象 MCU リストから使用しているデバイスに該当する[R- x] の設定値を検索し、[R- x] に設定値を入力します。(x = 1, 2, 3…)

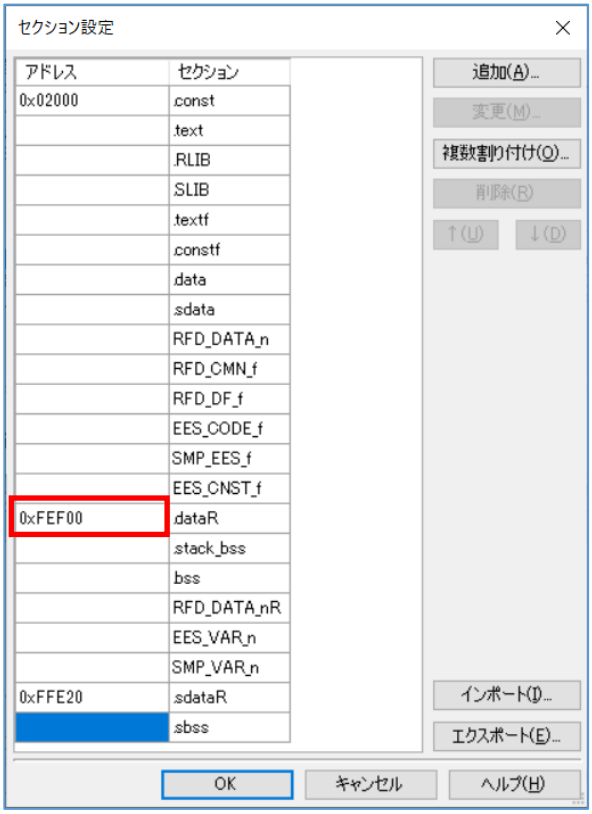
・セクション設定（RAM の先頭アドレス）の変更箇所の例：

CS+(CC-RL コンパイラ)

例） RL78/G23(R7F100GLG)用設定 RAM: 16KB



例） RL78/G22(R7F102GGE)用設定 RAM: 4KB



7.4.1 CC-RL コンパイラ環境の設定

CC-RL コンパイラ環境(CS+、e^2 studio)を使用する場合の変更箇所と変更例を記載します。

7.4.1.1 セクション設定

セクション設定で使用する製品の RAM 領域の先頭アドレスを設定します。

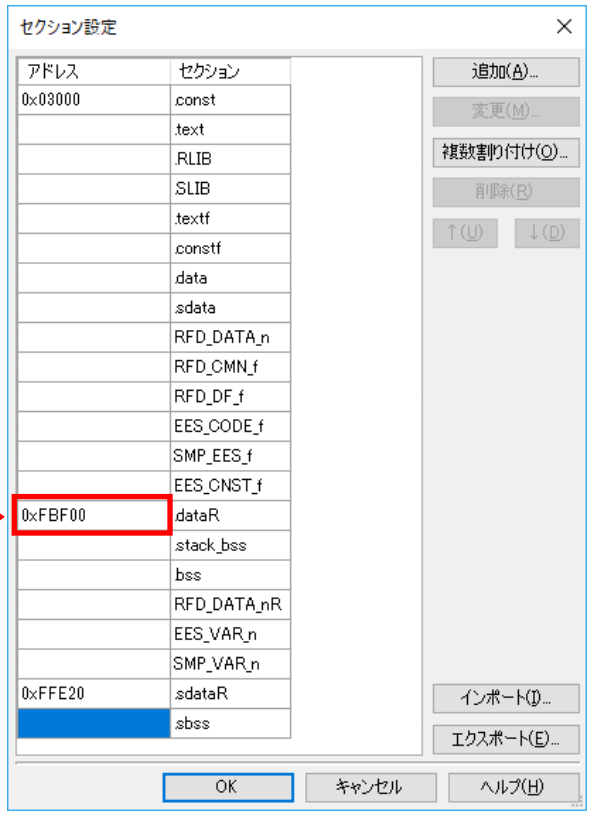
RL78/G23(R7F100GLG)から RL78/G22(R7F102GGE)へ変更する場合を例として示します。

RAM のサイズが 16KB から 4KB へ変更されるため、RAM の先頭アドレスを"0xFBF00"から"0xFEF00"へ変更します。

各製品の RAM の先頭アドレスについては、対象 MCU リストの[R-1] 列をご確認ください。

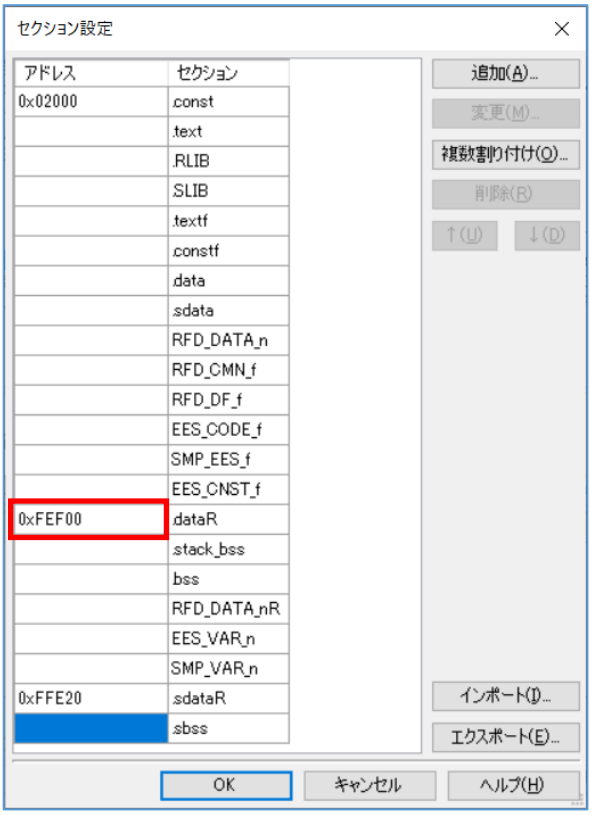
・CS+でのセクション設定（RAM の先頭アドレス）の変更箇所の例：

例） RL78/G23(R7F100GLG)用設定 RAM: 16KB



[R-1] →

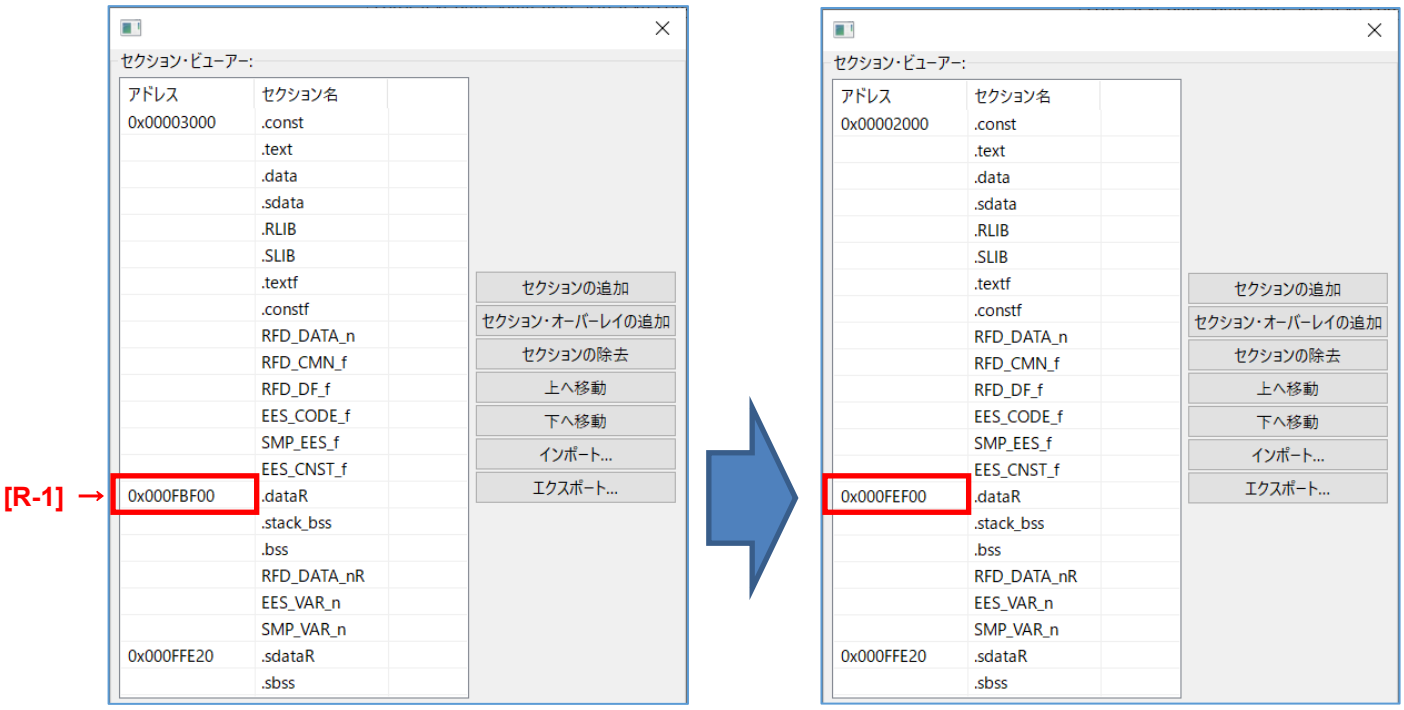
例） RL78/G22(R7F102GGE)用設定 RAM: 4KB



・e^2 studio でセクション設定（RAM の先頭アドレス）の変更箇所の例：

例）RL78/G23(R7F100GLG)用設定 RAM: 16KB

例）RL78/G22(R7F102GGE)用設定 RAM: 4KB



7.4.1.2 デバッグ設定

サンプル・プログラムが対象としているデバイス以外を使用する場合、デバッガ使用時のデバッグ・モニタ領域の範囲が異なります。

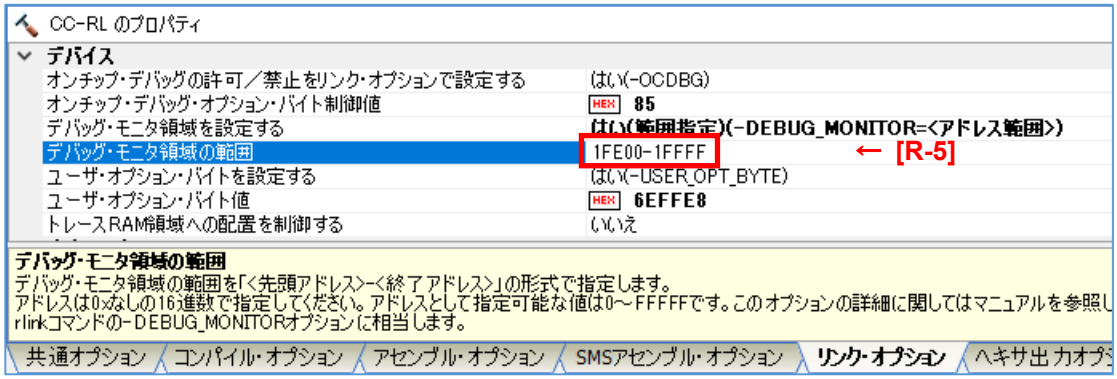
- デバッグ・モニタ領域の先頭アドレスは、ROM 領域の終了アドレスから"511byte(0x1FF)"を減算したアドレスを設定します。終了アドレスが"0x1FFFF"なら、"0x1FE00"を設定します。

RL78/G23(R7F100GLG)から RL78/G22(R7F102GGE)へ変更する場合を例として示します。
- RL78/G22 用にデバッグ・モニタ領域の範囲を[0x0FE00 - 0x0FFFF]に設定します。

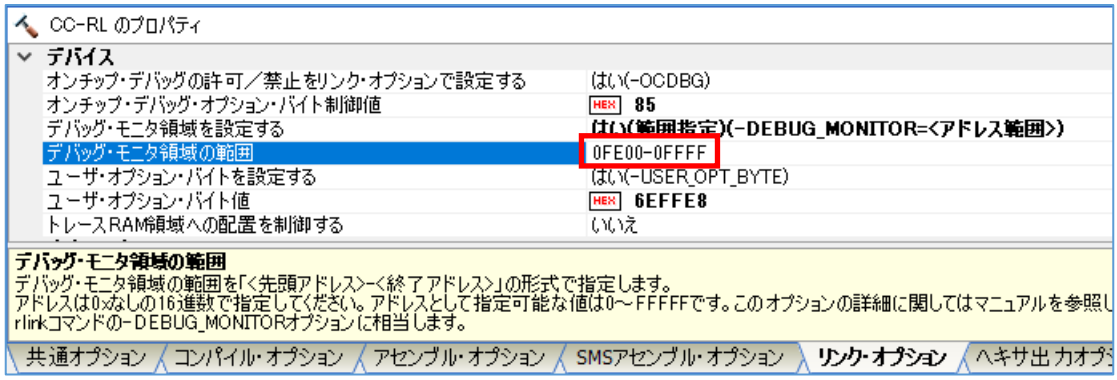
各製品のデバッグ・モニタ領域の先頭アドレスについては、対象 MCU リストの[R-5] 列をご確認ください。

・CS+でのデバッグ・モニタ領域の設定は、“リンク・オプション”タブで［デバイス］項目を選択します。

RL78/G23 用設定(ROM:128KB)　R7F100GLG の例

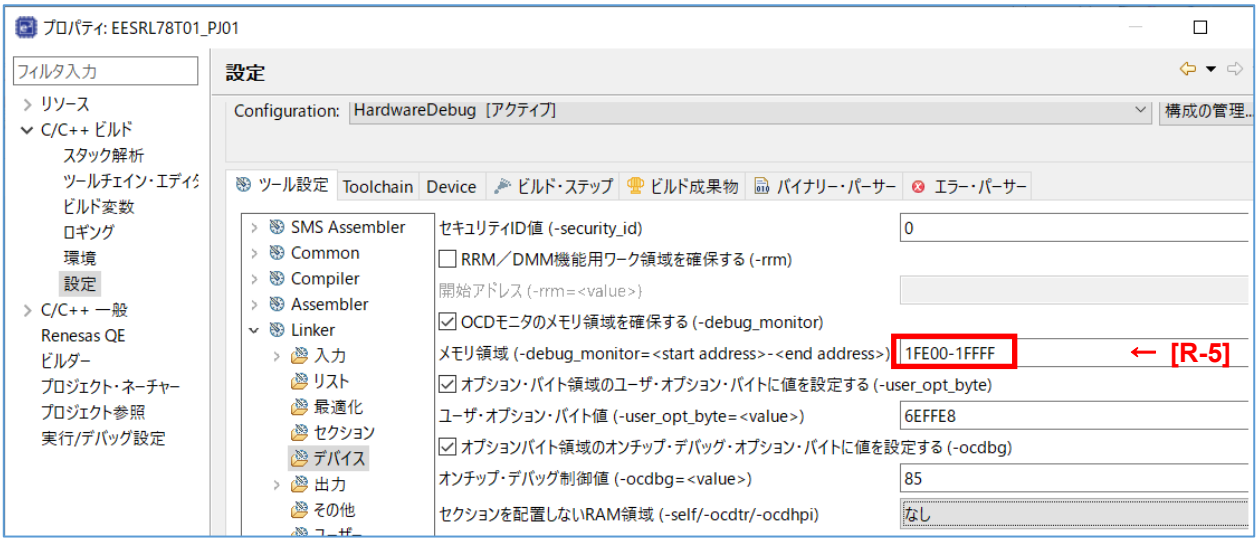


RL78/G22 用設定(ROM:64KB)　R7F102GGE の例

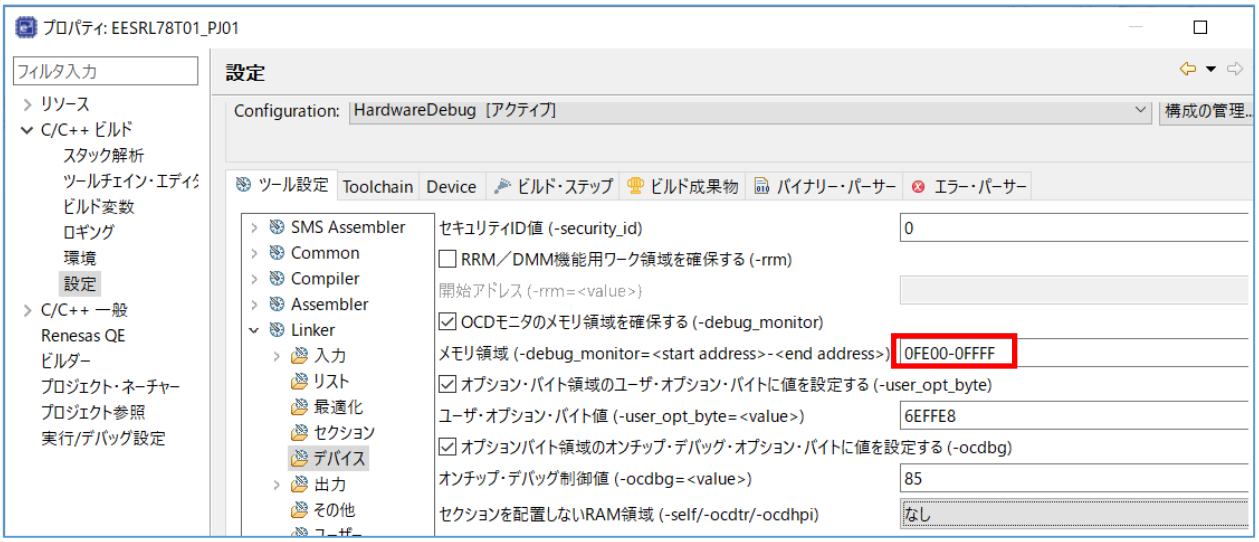


・e^2 studio での OCD・モニタのメモリ領域の設定は、“Linker”から［デバイス］を選択します。

RL78/G23 用設定(ROM:128KB)　R7F100GLG の例



RL78/G22 用設定(ROM:64KB)　R7F102GGE の例



7.4.2 IAR Embedded Workbench（IAR コンパイラ）を使用する場合の変更箇所

IAR コンパイラ環境(Embedded Workbench)を使用する場合の変更箇所と変更例を記載します。

7.4.2.1 対象デバイス用ヘッダ・ファイルの設定

EES RL78 Type01 で用意している main.c, low_level_init.c では、RL78/G23(R7F100GLG)用のヘッダ・ファイルをインクルードしています。その他の RL78/G23 製品や RL78/G22 製品を使用する場合は、インクルードするヘッダ・ファイルを使用するデバイス用のヘッダ・ファイルに変更する必要があります。

RL78/G23(R7F100GLG)用:

<main.c>

```
#include "ior7f100glg.h"
```

<low_level_init.c>

```
#include "ior7f100glg.h"
#include "ior7f100glg_ext.h"
```

RL78/G22(R7F102GGE)を使用する場合の例:

<main.c>

```
#include "ior7f102gge.h"
```

<low_level_init.c>

```
#include "ior7f102gge.h"
#include "ior7f102gge_ext.h"
```

※製品のデバイス型名については、対象 MCU リストの"Target MCU name" 列をご確認ください。

7.4.2.2 リンカ設定ファイルの設定

EES RL78 Type01 で提供しているサンプル・プログラム（RL78_G23 フォルダ）では、RL78/G23(R7F100GLG)のセクション（ROM, RAM, Data flash の範囲）が設定されています。その他の RL78/G23 製品や RL78/G22 製品を使用する場合は、セクション設定や、デバッガ使用時の TraceRAM 領域、デバッガ・モニタ領域の範囲が異なるため、EES RL78 Type01 の RL78/G23 用に提供されているサンプル用リンカファイル(sample_linker_file.icf)の内容を変更します。下記に変更箇所を赤文字で示していますので、対象 MCU リストを参照し、設定値を対象デバイス用に変更します。

対象ファイル名：sample_linker_file.icf

RL78/G23(R7F100GLG)から RL78/G22(R7F102GGE)へ変更する場合を例として示します。

- ROM 領域を 64KB[0x00000 – 0x0FFFF]の範囲に設定します。
- RAM 領域が 4KB[0x0FEF00 – 0x0FFEFF]のため、開始アドレスを"0xFEF00"に変更します。
- Data flash 領域が 2KB[0x0F1000 – 0x0F17FF]のため、終了アドレスを"0xF17FF"に変更します。

(1) セクション設定

- ROM, RAM, Data Flash のサイズの変更箇所

RL78/G23 用設定(ROM:128KB, RAM:16KB,DF:8KB) R7F100GLG の例

```
define region ROM_near = mem:[from 0x000D8 to 0x0FFFF];      ← [R-2]
define region ROM_far  = mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF];← [R-2], [R-3] 注 1
define region ROM_huge = mem:[from 0x000D8 to 0x1FFFF];      ← [R-2] or [R-3] 注 2
define region SADDR    = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xFBF00 to 0xFFE1F];
define region RAM_far  = mem:[from 0xFBF00 to 0xFFE1F];      ← [R-1]
define region RAM_huge = mem:[from 0xFBF00 to 0xFFE1F];      ← [R-1]
define region VECTOR   = mem:[from 0x00000 to 0x0007F];      ← [R-1]
define region CALLT    = mem:[from 0x00080 to 0x000BF];
define region EEPROM   = mem:[from 0xF1000 to 0xF2FFF];      ← [R-4]
```

注 1 ROM サイズが 64KB よりも大きい場合、ROM サイズが増加するごとに記載を変更する必要があります。記載内容については、次ページの“ROM_far の記載例”を参照してください。

注 2 対象 MCU リストの[R-3] にアドレス値が入力されている場合は[R-3] の値を使用します。[R-3] の値が、“ー”の場合は[R-2] の値を設定してください。

RL78/G22 用設定(ROM:64KB, RAM: 4KB,DF: 2KB) R7F102GGE の例

```
define region ROM_near = mem:[from 0x000D8 to 0x0FFFF];
define region ROM_far  = mem:[from 0x000D8 to 0x0FFFF];
define region ROM_huge = mem:[from 0x000D8 to 0x0FFFF];
define region SADDR    = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xFEF00 to 0xFFE1F];
define region RAM_far  = mem:[from 0xFEF00 to 0xFFE1F];
define region RAM_huge = mem:[from 0xFEF00 to 0xFFE1F];
define region VECTOR   = mem:[from 0x00000 to 0x0007F];
define region CALLT    = mem:[from 0x00080 to 0x000BF];
define region EEPROM   = mem:[from 0xF1000 to 0xF17FF];
```

・ROM_far の記載例

ROM サイズごとの ROM_far への記載例を示します。対象デバイスと同じ ROM サイズの行を参考に設定してください。色を付けている箇所は、[R-2]、または[R-3] に該当する値を示しています。

- ROM サイズが 64KB 以下の場合 ([R-3] が"ー"の場合)

ROM	[R-2]の値	mem:[from 0x000D8 to [R-2]];
32KB	0x07FFF	mem:[from 0x000D8 to 0x07FFF];
64KB	0x0FFFF	mem:[from 0x000D8 to 0x0FFFF];

- ROM サイズが 64KB 超の場合 ([R-3] が"ー"以外の場合)

ROM	[R-3]の値	mem:[from 0x000D8 to [R-2] | mem:[from 0x10000 to 0x1FFFF] |・・・省略・・・ | mem:[from 0xX0000 to [R-3]];
96KB	0x17FFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x17FFF];
128KB	0x1FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF];
192KB	0x2FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF];
256KB	0x3FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF];
384KB	0x5FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF] | mem:[from 0x40000 to 0x4FFFF] | mem:[from 0x50000 to 0x5FFFF];
512KB	0x7FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF] | mem:[from 0x40000 to 0x4FFFF] | mem:[from 0x50000 to 0x5FFFF] | mem:[from 0x60000 to 0x6FFFF] | mem:[from 0x70000 to 0x7FFFF];
768KB	0xBFFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF] | ・・・一部省略・・・ | mem:[from 0xA0000 to 0xAFFFF] | mem:[from 0xB0000 to 0xBFFFF];

(2) デバッグ設定

- デバッグ・モニタ領域の先頭アドレスは、ROM 領域の終了アドレスから"511byte(0x1FF)"を減算したアドレスを設定します。終了アドレスが"0x1FFFF"なら、"0x1FE00"を設定します。
- TraceRAM 領域の先頭アドレスは、RAM 領域の先頭アドレスに"1KB(0x400)"を加算したアドレスを設定します。先頭アドレスが"0xFBF00"なら、"0xFC300"を設定します。

RL78/G23(R7F100GLG)から RL78/G22(R7F102GGE)へ変更する場合を例として示します。

- デバッグ・モニタ領域の範囲を[from 0x0FE00 size 0x0200]に設定します。
- TraceRAM 領域の範囲を[from 0xFF300 size 0x0400]に設定します。

注）TraceRAM はデバイスによっては対応していないため、対象デバイスのユーザーズマニュアルをご確認ください。

デバッガ使用時の TraceRAM 領域、デバッグ・モニタ領域の変更箇所

RL78/G23 用設定(ROM:128KB, RAM: 16KB, DF: 8KB) R7F100GLG の例

```
if (isdefinedsymbol(__RESERVE_OCD_ROM))
{
  if (__RESERVE_OCD_ROM == 1)
  {
    reserve region "OCD ROM area" = mem:[from 0x1FE00 size 0x0200];   ← [R-5]
  }
}
    .
    .    一部省略
    .
if (isdefinedsymbol(__RESERVE_OCD_TRACE_RAM))
{
  if (__RESERVE_OCD_TRACE_RAM == 1)
  {
    reserve region "OCD Trace RAM" = mem:[from 0xFC300 size 0x0400];   ← [R-6]
  }
}
```

RL78/G22 用設定(ROM: 64KB, RAM: 4KB, DF: 2KB) R7F102GGE の例

```
if (isdefinedsymbol(__RESERVE_OCD_ROM))
{
  if (__RESERVE_OCD_ROM == 1)
  {
    reserve region "OCD ROM area" = mem:[from 0x0FE00 size 0x0200];
  }
}
    .
    .    一部省略
    .
if (isdefinedsymbol(__RESERVE_OCD_TRACE_RAM))
{
  if (__RESERVE_OCD_TRACE_RAM == 1)
  {
    reserve region "OCD Trace RAM" = mem:[from 0xFF300 size 0x0400];
  }
}
```

7.4.3 LLVM コンパイラを使用する場合の変更箇所

LLVM コンパイラ環境(e² studio)を使用する場合の変更箇所と変更例を記載します。

7.4.3.1 リンカ・スクリプトファイルの設定

EES RL78 Type01 で提供しているサンプル・プログラム（RL78_G23 フォルダ）では、RL78/G23(R7F100GLG)のセクション（ROM, RAM, MIRROR 領域の範囲）が設定されています。また、その他の RL78/G23 製品や RL78/G22 製品を使用する場合は、セクション設定や、デバッガ使用時の TraceRAM 領域、デバッガ・モニタ領域(OCDROM)の範囲が異なるため、EES RL78 Type01 の RL78/G23 用に提供されているサンプル用リンカ・スクリプトファイル(sample_linker_file.ld)の内容を変更します。下記に変更箇所を赤文字で示していますので、対象 MCU リストを参照し、対象デバイス用に設定値を変更してください。

対象ファイル名：sample_linker_file.ld

RL78/G23(R7F100GLG)から RL78/G22(R7F102GGE)へ変更する場合を例として示します。

- OCDROM（デバッグ・モニタ領域）の先頭アドレスは、ROM 領域の終了アドレスから"511byte(0x1FF)"を減算したアドレスを設定します。ROM 領域の終了アドレスが"0x1FFFF"なら、OCDROM の ORIGIN には"0xFE00" [R-5]を設定します。
- ROM 領域のサイズは、"0xD8"から OCDROM の開始アドレスまでの領域を設定します。OCDROM の開始アドレスが"0xFE00"であれば、ROM の LENGTH には、OCDROM の開始アドレス"0xFE00"から"0xD8"を減算した値を 10 進数にした"64808"を設定します。
- MIRROR（ミラー領域）の先頭アドレス、およびサイズは、デバイスによって異なります。
  RL78/G22(R7F100GGE)の場合、MIRROR の ORIGIN には、ミラー領域の先頭アドレスである"0xF2000"を設定し、LENGTH には、ミラー領域の先頭アドレス"0xF2000"からミラー領域の終了アドレス"0xFEEFF"までの値を 10 進数にした"52992"を設定します。ミラー領域の詳細は、デバイスのハードウェアマニュアルをご確認ください。
- RAM 領域の ORIGIN には、RAM の先頭アドレス"0xFEF00" [R-1]を設定し、LENGTH には 4KB を 10 進数にした"4096"を設定します。
- TRACERAM 領域は、RAM の先頭アドレスに 1024 バイトを加算したアドレスから 1024 バイトの領域を使用するため、ORIGIN には"0xFF300" [R-6]を設定します。
  また、トレース機能を使用しない場合や、デバイスによっては使用できない場合があるため、TRACERAM 領域の詳細は、デバイスのハードウェアマニュアルをご確認ください。
  注）ただし、RL78/G22 ではトレース機能を使用できません。上記については設定例としてあげていますが、実際には対象の行が実行されないようにします。

(1) MEMORY 設定

RL78/G23 用設定(ROM:128KB, RAM:16KB,DF:8KB) R7F100GLG の例

```
MEMORY
{
  VEC : ORIGIN = 0x0, LENGTH = 4
  IVEC : ORIGIN = 0x4, LENGTH = 188
  CALLT0 : ORIGIN = 0x80, LENGTH = 0x40
  OPT : ORIGIN = 0xC0, LENGTH = 4
  SEC_ID : ORIGIN = 0xC4, LENGTH = 10
  OCDSTAD : ORIGIN = 0xCE, LENGTH = 10
  OCDROM : ORIGIN = 0x1FE00, LENGTH = 512        ← [R-5]
  ROM : ORIGIN = 0xD8, LENGTH = 130344
  MIRROR : ORIGIN = 0xF3000, LENGTH = 36608
  SADDR : ORIGIN = 0xffe20, LENGTH = 0x000a0
  RAM : ORIGIN = 0xFBF00, LENGTH = 16384          ← [R-1]
  TRACERAM   : ORIGIN = 0xFC300, LENGTH = 1024    ← [R-6]
}
```

RL78/G22 用設定(ROM: 64KB, RAM: 4KB, DF: 2KB)　R7F102GGE の例

```
MEMORY
{
  VEC : ORIGIN = 0x0, LENGTH = 4
  IVEC : ORIGIN = 0x4, LENGTH = 188
  CALLT0 : ORIGIN = 0x80, LENGTH = 0x40
  OPT : ORIGIN = 0xC0, LENGTH = 4
  SEC_ID : ORIGIN = 0xC4, LENGTH = 10
  OCDSTAD : ORIGIN = 0xCE, LENGTH = 10
  OCDROM : ORIGIN = 0xFE00, LENGTH = 512
  ROM : ORIGIN = 0xD8, LENGTH = 64808
  MIRROR : ORIGIN = 0xF2000, LENGTH = 52992
  SADDR : ORIGIN = 0xffe20, LENGTH = 0x000a0
  RAM : ORIGIN = 0xFEF00, LENGTH = 4096
  /* TRACERAM : ORIGIN = 0xFF300, LENGTH = 1024 */
}
```

注）RL78/G22 ではトレース機能を使用できないためコメントにしていますが、トレース機能に対応しているデバイスでは値のみ変更してご使用ください。

(2) RAM 領域の開始アドレス設定

RL78/G23 用設定(ROM:128KB, RAM:16KB,DF:8KB) R7F100GLG の例

```
.data 0xFBF00 : AT(__mdata)                    ← [R-1]
{
    . = ALIGN(2);
    PROVIDE (__datastart = .);
    __data = .;
    *(.data)
    *(.data.*)
    . = ALIGN(2);
    /*INPUT_SECTION_FLAGS(!SHF_EXECINSTR,SHF_WRITE,SHF_ALLOC) *(*_n)*/
    __edata = .;
} >RAM
```

RL78/G22 用設定(ROM:64KB, RAM: 4KB,DF: 2KB) R7F102GGE の例

```
.data 0xFEF00 : AT(__mdata)
{
    . = ALIGN(2);
    PROVIDE (__datastart = .);
    __data = .;
    *(.data)
    *(.data.*)
    . = ALIGN(2);
    /*INPUT_SECTION_FLAGS(!SHF_EXECINSTR,SHF_WRITE,SHF_ALLOC) *(*_n)*/
    __edata = .;
} >RAM
```

7.4.4 サンプル・プログラムの変更(共通)

7.4.4.1 EES ブロックに使用するデータ・フラッシュ・メモリ数の変更

EES RL78 Type01 のサンプル・プログラムフォルダ「RL78_G23」で対象としている RL78/G23 と RL78/G22 では、データ・フラッシュ・メモリのブロック数が異なります。そのため、RL78/G22 で EEPROM エミュレーション・ブロック数を 3 ブロック以上(推奨)で使用するには、EES ブロックサイズを 512byte で使用する必要があります。

"r_ees_descriptor.h"ファイルの EES ブロックに使用するデータ・フラッシュ・メモリ数を RL78/G22 用に設定します。EES ブロックサイズを 4u (1024byte)から 2u (512byte) に変更します。

対象ファイル名：r_ees_descriptor.h

RL78/G23 用設定(DF: 8KB) R7F100GLG の例

```
/* Specifies the number of physical data flash blocks per one virtual block */
#define R_EES_EXRFD_VALUE_U08_PHYSICAL_BLOCKS_PER_VIRTUAL_BLOCK (4u)
```

RL78/G22 用設定(DF: 2KB) R7F102GGE の例

```
/* Specifies the number of physical data flash blocks per one virtual block */
#define R_EES_EXRFD_VALUE_U08_PHYSICAL_BLOCKS_PER_VIRTUAL_BLOCK (2u)
```

7.4.4.2 識別子(データ ID)毎のデータ・サイズ変更

EES ブロックサイズを 512byte に変更して使用する場合、EES RL78 Type01 のサンプル・プログラムを使用するとデータの合計が、推奨サイズ(251byte)を越えてしまうため、"r_ees_user_types.h"ファイルの"type_Z"のデータ・サイズを変更します。

推奨サイズの詳細については、「4.1 格納ユーザ・データ数とユーザ・データの合計サイズ」を参照してください。

対象ファイル名：r_ees_user_types.h

RL78/G23 用設定(DF: 8KB) R7F100GLG の例

```
typedef    uint8_t    type_A[2];
typedef    uint8_t    type_B[3];
typedef    uint8_t    type_C[4];
typedef    uint8_t    type_D[5];
typedef    uint8_t    type_E[6];
typedef    uint8_t    type_F[10];
typedef    uint8_t    type_X[20];
typedef    uint8_t    type_Z[255];
```

RL78/G22 用設定(DF: 2KB) R7F102GGE の例

```
typedef    uint8_t    type_A[2];
typedef    uint8_t    type_B[3];
typedef    uint8_t    type_C[4];
typedef    uint8_t    type_D[5];
typedef    uint8_t    type_E[6];
typedef    uint8_t    type_F[10];
typedef    uint8_t    type_X[20];
typedef    uint8_t    type_Z[30];
```

8 改定記録

8.1 本版で改定された主な箇所

Rev.	発行日	改定内容	
		Page	概要
1.00	2022.1.12	-	新規作成
1.01	2022.12.28	-	RL78/G22 追加サポート
1.10	2023.04.28	-	RL78/G24 追加サポート
			「7.1.3.2 ユーザ定義マクロの設定」を追加
			「7.2.3.2 ユーザ定義マクロの設定」を追加
			「7.3 デバイス変更に伴う設定」を更新
1.20	2023.09.28	-	LLVM コンパイラに対応
		105	「7.3 LLVM コンパイラを使用する場合のプロジェクトの作成」を追加
		129	「7.4.3 LLVM コンパイラを使用する場合の変更箇所」を追加

EEPROM エミュレーション・ソフトウェア RL78 Type01
ユーザーズマニュアル

発行年月日　2023年 09月 28日　　Rev.1.20

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）

EEPROM エミュレーション・ソフトウェア
RL78 Type01

ルネサス エレクトロニクス株式会社

R01UT5008JJ0120