RL78/G15,G16 グループ
フラッシュ・メモリ書き換え用サンプル・プログラム
Renesas Flash Sample Program Type01

R20AN0652JJ0121
Rev.1.21
2025.10.31

要旨

このアプリケーションノートは，RL78/G15,RL78/G16グループ用のRenesas Flash Sample Program Type01(RFSP Type01) 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。

対象デバイス

RL78/G15 グループ
RL78/G16 グループ

目次

1 概要

Renesas Flash Sample Program Type01(以降、本書では「RFSP Type01」と略します。) は、RL78/G15,RL78/G16 のフラッシュ・メモリ内のデータを書き換えるためのサンプル・プログラムです。

1.1 製品概要

RFSP Type01 はユーザ・プログラムから呼び出すことにより、コード・フラッシュ・メモリ、またはデータ・フラッシュ・メモリの消去/書き込みを実行します。

なお、本ユーザーズマニュアルは、対象の RL78/G15,RL78/G16 のユーザーズマニュアルと合わせてご使用ください。

1.1.1 目的

本書の目的は、RFSP Type01 に関する情報を記述することです。

1.2 製品内容

RFSP Type01 のサンプル API 関数をユーザ・プログラムから呼び出すことにより、コード・フラッシュ・メモリ、またはデータ・フラッシュ・メモリを書き換えることができます。

RFSP Type01 は、以下を含んでいます。

・本アプリケーションノート

・RL78/G15,RL78/G16 のコード・フラッシュ・メモリを操作する RFSP のサンプル・プログラム・ファイル。

・RL78/G15,RL78/G16 のデータ・フラッシュ・メモリを操作する RFSP のサンプル・プログラム・ファイル。

1.3 製品の特長

RFSP Type01 は、フラッシュ・メモリ制御回路用コマンドの処理フローに基づいて、フラッシュの書き換え操作を行います。それぞれのサンプル API 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。

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

RFSP Type01 では、複数のサンプル 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。フラッシュ操作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

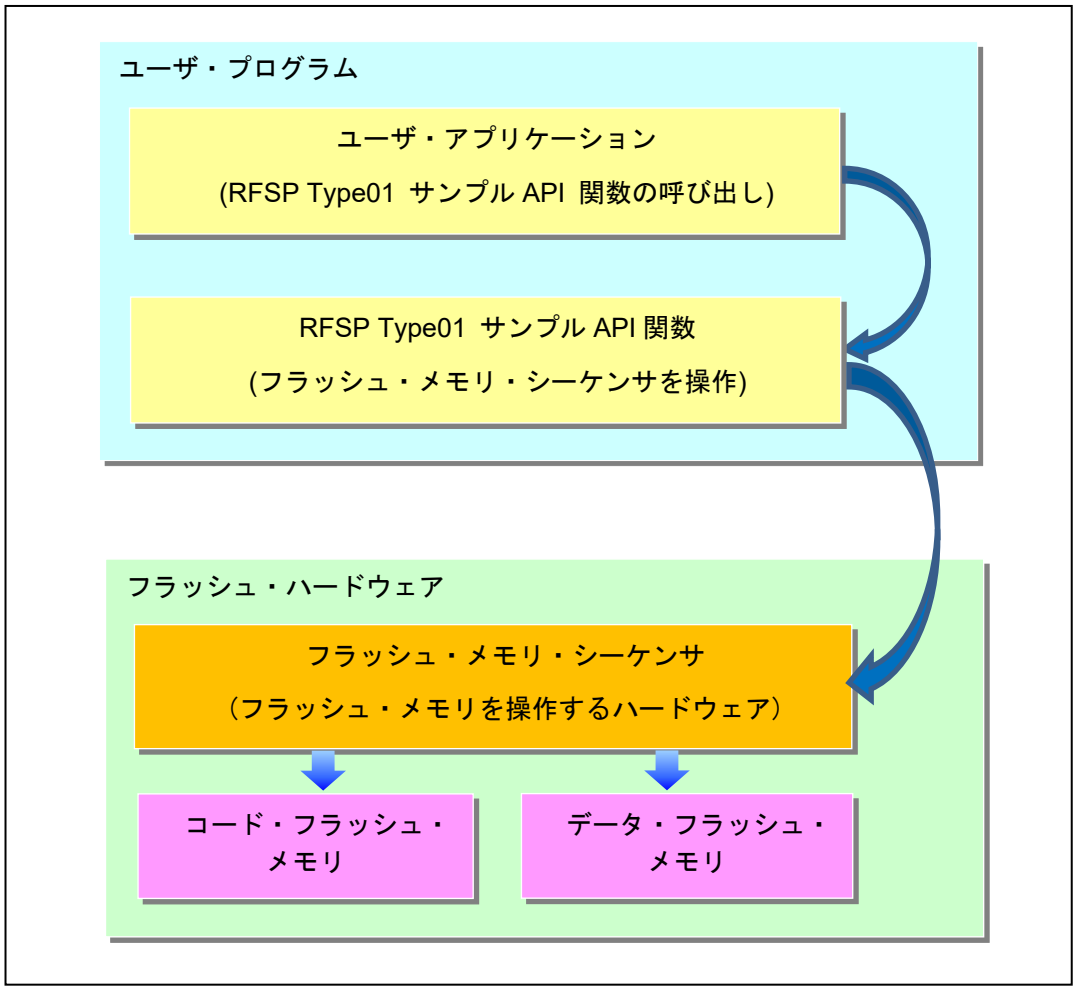


図 1-1 RFSP Type01 のサンプル API 関数を使ったフラッシュ・メモリの操作イメージ

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。（RFSP Type01 の開発は Windows10 Pro で実施）

- C コンパイラ・パッケージ

RFSP 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 Lite	Renesas Electronics

- ターゲット MCU

RL78/G15

RL78/G16

1.5 注意事項

(1)フラッシュ・メモリ書き換え操作のためのユーザ・プログラムの配置

コード/データ・フラッシュ領域の書き換えのためユーザ・プログラムをコード・フラッシュ領域に配置してください。RAM フェッチによるセルフ・プログラミングを禁止します。また、ブート領域、およびセルフ・プログラミングを実行するユーザ・プログラムが格納されているブロックに対しての書き換えは禁止です。

(2)セルフ・プログラミング・モード中の割り込み禁止

セルフ・プログラミング・モード設定前に、あらかじめ割り込み禁止をしてください。割り込みを禁止するためには、通常動作モード時と同様に、DI 命令によって IE フラグをクリア（0）してください。

(3)フラッシュ・メモリ・シーケンサへの CPU の動作周波数の設定

フラッシュ・メモリ・シーケンサを使用して、コード/データ・フラッシュ・メモリの書き換えなどの操作を実行する場合、FSSET レジスタの FSET4-0 ビットへ CPU の動作周波数を設定しておく必要があります。CPU の動作周波数が正しく設定されていない状態での書き換え動作は不定となり、書かれたデータは保証されませんのでご注意ください。（書き込み直後のフラッシュ・メモリのデータ値が期待値通りであっても、その値の保持期間を保証できません。）

(4)高速オンチップ・オシレータの動作設定

セルフ・プログラミングを実行する前に、高速オンチップ・オシレータを動作させておく必要があります。高速オンチップ・オシレータを停止させている場合は、高速オンチップ・オシレータ・クロック動作（HIOSTOP＝0）させ、30μs 経過以降にセルフ・プログラミングを実行してください。

(5)セルフ・プログラミング実行中の他操作の実施制限

セルフ・プログラミング実行フローの途中に、セルフ・プログラミングの手順と関係ない他の設定や他の命令実施をしないでください。

(6)フラッシュ・メモリ書き換え操作中のユーザ・プログラム動作

セルフ・プログラミングは、フラッシュ・メモリ・シーケンサを使用し、フラッシュ・メモリの書き換えを制御します。セルフ・プログラミングによる書き換え中は、CPU が停止してユーザ・プログラムの操作ができなくなりますのでご注意ください。

1.6 C コンパイラ定義

RFSP Type01 のヘッダ・ファイル(r_rfsp_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_rfsp_compiler.h"ファイルで認識し、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 COMPILER_CC (1)

- IAR コンパイラ:

" __IAR_SYSTEMS_ICC__"が定義されている場合

#define COMPILER_IAR (2)

- LLVM コンパイラの定義 ：

" __llvm__"が定義されている場合

#define COMPILER_LLVM (3)

<r_rfsp_compiler.h ファイル内の記述>

```
/* Compiler definition */
#define COMPILER_CC                             (1)
#define COMPILER_IAR                            (2)
#define COMPILER_LLVM                           (3)

#if defined (__llvm__)
    #define COMPILER COMPILER_LLVM
#elif defined (__CCRL__)
    #define COMPILER COMPILER_CC
#elif defined (__IAR_SYSTEMS_ICC__)
    #define COMPILER COMPILER_IAR
#else
    /* Unknown compiler error */
    #error   "Non-supported compiler."
#endif

/* Compiler dependent definition */
#if (COMPILER_CC == COMPILER)
    #define R_RFSP_FAR_FUNC                             __far
    #define R_RFSP_NO_OPERATION                         __nop
    #define R_RFSP_DISABLE_INTERRUPT                    __DI
    #define R_RFSP_ENABLE_INTERRUPT                     __EI
    #define R_RFSP_GET_PSW_IE_STATE                     __get_psw
    #define R_RFSP_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != ((u08_psw_ie_state) & 0x80u))
#elif (COMPILER_IAR == COMPILER)
    #define R_RFSP_FAR_FUNC                             __far_func
    #define R_RFSP_NO_OPERATION                         __no_operation
    #define R_RFSP_DISABLE_INTERRUPT                    __disable_interrupt
    #define R_RFSP_ENABLE_INTERRUPT                     __enable_interrupt
    #define R_RFSP_GET_PSW_IE_STATE                     __get_interrupt_state
    #define R_RFSP_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != ((u08_psw_ie_state) & 0x80u))
```

```
#elif (COMPILER_LLVM == COMPILER)
      #define R_RFSP_FAR_FUNC                          __far
      #define R_RFSP_NO_OPERATION                      __nop
      #define R_RFSP_DISABLE_INTERRUPT                 __DI
      #define R_RFSP_ENABLE_INTERRUPT                  __EI
      #define R_RFSP_GET_PSW_IE_STATE                  (uint8_t)__builtin_rl78_pswie
      #define R_RFSP_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != (u08_psw_ie_state))
#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=S2 -g -g_line -lang=c99

- [IAR(IAR Embedded Workbench)]

Major compile options:

--core s2 --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 -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wshadow -Waggregate-return -g -mcpu=s2 -mdisable-mda

2 システム構成

2.1 ファイル構成

2.1.1 フォルダ構成

この項では、RL78/G15 用のサンプル・プログラムを例に説明しています。RL78/G15 以外を使用する場合は、G15 を対象のデバイスに読みかえてください。

・RL78/G15 のサンプルのフォルダ名("RL78_G15")は、対象デバイスのフォルダ名に読み変えてください。

RL78/G16 を使用する場合のフォルダ名："RL78_G16"

RFSP Type01 のフォルダ構成を図 2-1 に示します。

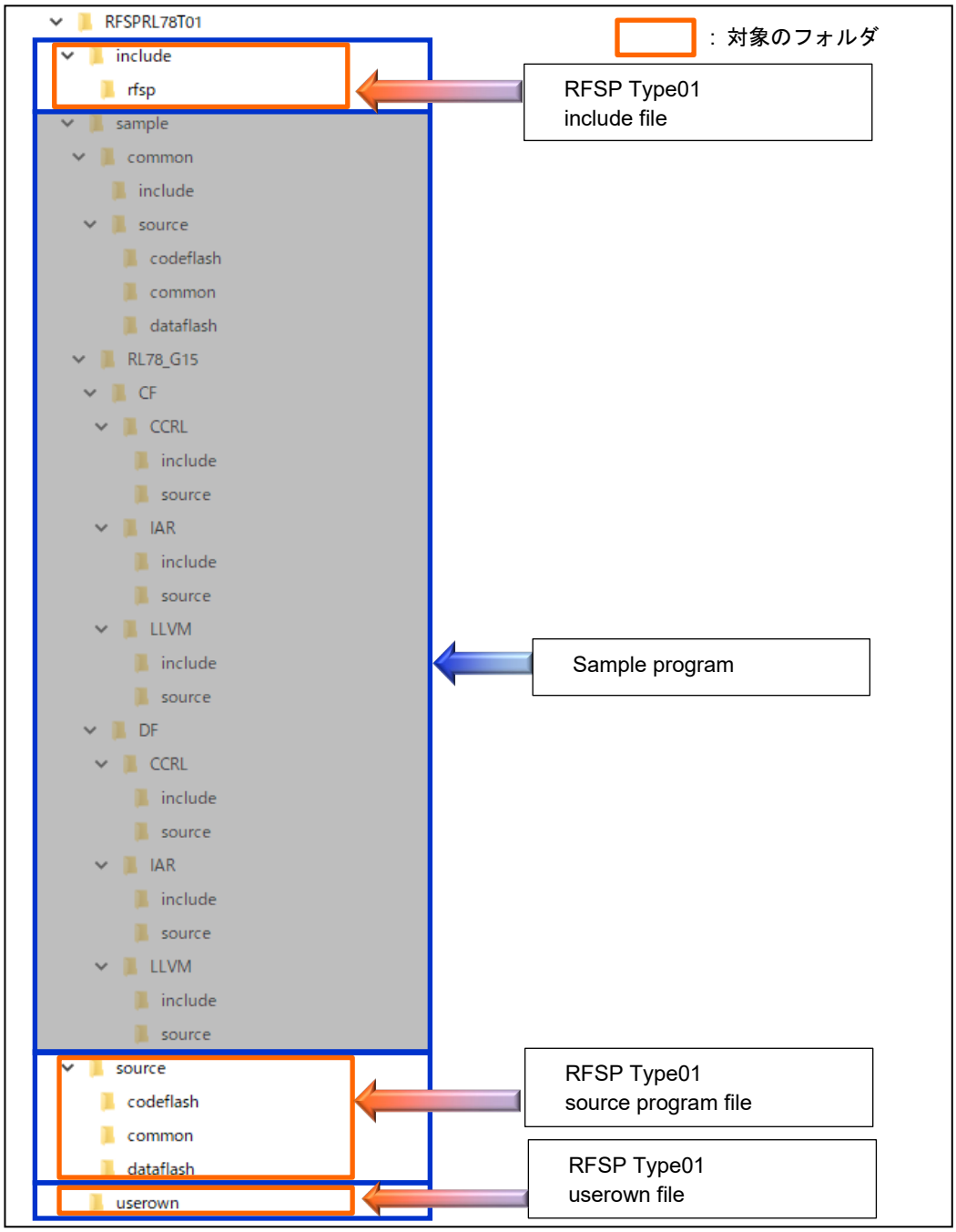


図 2-1 RFSP Type01 のフォルダ構成(Folder Structure)

2.1.2 ファイル・リスト

2.1.2.1 ソース・ファイル・リスト

"source\common\"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\common\"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_rfsp_common_api.c	フラッシュ・メモリ操作の共通設定 API 関数ファイル
2	r_rfsp_common_control_api.c	フラッシュ・メモリ操作の共通コマンド制御 API 関数ファイル

"source\codeflash\"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 "source\codeflash\"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_rfsp_code_flash_api.c	コード・フラッシュ・メモリ制御 API 関数ファイル

"source\dataflash\"フォルダ内のプログラム・ソース・ファイルを表 2-3 に示します。

表 2-3 "source\dataflash\"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_rfsp_data_flash_api.c	データ・フラッシュ・メモリ制御 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 2-4 に示します。

表 2-4 "userown\"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_rfsp_common_userown.c	RFSP Type01 内でユーザ処理を実施するためのフック関数ファイル

2.1.2.2 ヘッダ・ファイル・リスト

"include\rfsp\"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\rfsp\"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_rfsp.h	共通ヘッダ・ファイルを記述したファイル。 RFSP Type01 使用時にインクルードする必要があるファイル
2	r_rfsp_compiler.h	RFSP Type01 で使用するコンパイラごとに異なる定義等を記述したファイル
3	r_rfsp_device.h	RFSP 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
4	r_rfsp_types.h	RFSP Type01 で使用する変数の型を定義したファイル
5	r_typedefs.h	RFSP Type01 で使用するデータの型を定義したファイル

"include\"フォルダ内のプログラム・ヘッダ・ファイルを表 2-6 に示します。

表 2-6 "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_rfsp_code_flash_api.h	コード・フラッシュ・メモリ制御 API 関数のプロトタイプ宣言を定義したファイル
2	r_rfsp_common_api.h	フラッシュ・メモリ操作の共通設定 API 関数のプロトタイプ宣言を定義したファイル
3	r_rfsp_common_control_api.h	フラッシュ・メモリ操作の共通コマンド制御 API 関数のプロトタイプ宣言を定義したファイル
4	r_rfsp_common_userown.h	RFSP Type01 内でユーザ処理を実施するためのフック関数のプロトタイプ宣言を定義したファイル
5	r_rfsp_data_flash_api.h	データ・フラッシュ・メモリ制御 API 関数のプロトタイプ宣言を定義したファイル

2.2 RL78/G15,RL78/G16 リソース

2.2.1 メモリ・マップ

RL78/G15 のコード・フラッシュ:CF、データ・フラッシュ:DF、RAM のメモリ・マップを表 2-7 に示します。

表 2-7 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/G15)

RL78/G15	コード・フラッシュ:CF (1 ブロック:1Kbyte)	RAM
R5F120x8 (x=0,1,4,6)	8KB (0000H-1FFFH)	1KB (FFB00H-FFEFFH)
R5F120x7 (x=0,1,4,6)	4KB (0000H-0FFFH)	1KB (FFB00H-FFEFFH)
RL78/G15 共通	データ・フラッシュ:DF(1 ブロック:512byte)	
	1KB(9000H-93FFH)	

RL78/G16 のコード・フラッシュ:CF、データ・フラッシュ:DF、RAM のメモリ・マップを表 2-8 に示します。

表 2-8 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/G16)

RL78/G16	コード・フラッシュ:CF (1 ブロック:1Kbyte)	RAM
R5F121xC (x=1,4,6,7,B)	32KB (0000H-7FFFH)	2KB (FF700H-FFEFFH)
R5F121xA (x=1,4,6,7,B)	16KB (0000H-3FFFH)	2KB (FF700H-FFEFFH)
RL78/G16 共通	データ・フラッシュ:DF(1 ブロック:512byte)	
	1KB(9000H-93FFH)	

2.2.2 ブロックイメージ

RL78/G15 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を図 2-2、図 2-3 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。

R5F120x8(コード・フラッシュ 8Kbyte)

1FFFH	CF:ブロック 07H
1C00H	(1Kbyte)
1BFFH	CF:ブロック 06H
1800H	(1Kbyte)
17FFH	CF:ブロック 05H
1400H	(1Kbyte)
13FFH	CF:ブロック 04H
1000H	(1Kbyte)
0FFFH	CF:ブロック 03H
0C00H	(1Kbyte)
0BFFH	CF:ブロック 02H
0800H	(1Kbyte)
07FFH	CF:ブロック 01H
0400H	(1Kbyte)
03FFH	CF:ブロック 00H
0000H	(1Kbyte)

R5F120x7(コード・フラッシュ 4Kbyte)

0FFFH	CF:ブロック 03H
0C00H	(1Kbyte)
0BFFH	CF:ブロック 02H
0800H	(1Kbyte)
07FFH	CF:ブロック 01H
0400H	(1Kbyte)
03FFH	CF:ブロック 00H
0000H	(1Kbyte)

図 2-2 コード・フラッシュ のブロックイメージ

93FFH	DF:ブロック 01H
9200H	(512byte)
91FFH	DF:ブロック 00H
9000H	(512byte)

図 2-3 データ・フラッシュのブロックイメージ

2.2.3 レジスター覧(フラッシュ・メモリ・シーケンサ操作関連)

RFSP Type01 が使用する RL78/G15,RL78/G16 内蔵レジスタの一覧を表 2-9 に示します。

表 2-9 RFSP Type01 使用 RL78/G15,RL78/G16 内蔵レジスター覧

Base	Offset	Register Name	Size	Function name / Note
F0000H	BEH	FSSET	1byte	フラッシュ・メモリ・シーケンサ周波数設定レジスタ
	C0H	FLPMC	1byte	フラッシュ・プログラミング・モード・コントロール・レジスタ
	C1H	FSSQ	1byte	フラッシュ・メモリ・シーケンサ制御レジスタ
	C2H	FLAPL	1byte	フラッシュ・アドレス・ポインタ・レジスタ L
	C3H	FLAPH	1byte	フラッシュ・アドレス・ポインタ・レジスタ H
	C4H	FLSEDL	1byte	フラッシュ・エンド・アドレス・ポインタ・レジスタ L
	C5H	FLSEDH	1byte	フラッシュ・エンド・アドレス・ポインタ・レジスタ H
	C6H	FSASTL	1byte	フラッシュ・メモリ・シーケンサ・ステータス・レジスタ L
	C7H	FSASTH	1byte	フラッシュ・メモリ・シーケンサ・ステータス・レジスタ H
	C8H	FLWLL	1byte	フラッシュ・ライト・バッファ・レジスタ LL
	C9H	FLWLH	1byte	フラッシュ・ライト・バッファ・レジスタ LH
	CAH	FLWHL	1byte	フラッシュ・ライト・バッファ・レジスタ HL
	CBH	FLWHH	1byte	フラッシュ・ライト・バッファ・レジスタ HH

2.3 RFSP Type01 使用リソース

2.3.1 API 関数のコード・サイズとスタック・サイズ

RFSP Type01 の API 関数が使用するコード・サイズとスタック・サイズを表 2-10 に示します。

表 2-10 RFSP Type01 の API 関数が使用するコード・サイズとスタック・サイズ

API 関数名	コード・サイズ(Bytes)			スタック・サイズ(Bytes)		
	CC-RL	IAR	LLVM	CC-RL	IAR	LLVM
R_RFSP_Init	16	21	16	4	4	4
R_RFSP_SetFlashMemoryMode	21	29	31	4	8	8
R_RFSP_CheckCFDFSeqEndStep1	13	24	16	4	6	4
R_RFSP_CheckCFDFSeqEndStep2	8	19	11	4	6	4
R_RFSP_GetSeqErrorStatus	8	8	11	4	4	6
R_RFSP_ForceReset	2	2	2	4	4	4
R_RFSP_EraseCodeFlashReq	23	33	33	4	4	4
R_RFSP_WriteCodeFlashReq	33	40	49	8	6	4
R_RFSP_EraseDataFlashReq	22	38	33	4	6	4
R_RFSP_WriteDataFlashReq	33	40	49	8	6	6
R_RFSP_HOOK_EnterCriticalSection	9	9	11	4	4	4
R_RFSP_HOOK_ExitCriticalSection	11	10	9	4	4	4

3 RFSP Type01 API 関数

3.1 RFSP Type01 API 関数 一覧

3.1.1 共通フラッシュ制御 API 関数

RFSP Type01 の共通フラッシュ制御 API 関数一覧を表 3-1 に示します。

表 3-1 RFSP Type01 共通フラッシュ制御 API 関数一覧

	API 関数名	概要
1	R_RFSP_Init	引数で指定された周波数をフラッシュ・メモリ・シーケンサに設定し、RFSP Type01 の初期化を行います。
2	R_RFSP_SetFlashMemoryMode	引数で指定されたフラッシュ・セルフ・プログラミング・モードをフラッシュ・メモリ・シーケンサへ設定します。
3	R_RFSP_CheckCFDFSeqEndStep1	起動したフラッシュ・メモリ・シーケンサの動作終了を確認します。
4	R_RFSP_CheckCFDFSeqEndStep2	フラッシュ・メモリ・シーケンサ制御レジスタのクリアにより、コマンド動作が終了したかどうかを確認します。
5	R_RFSP_GetSeqErrorStatus	フラッシュ・メモリ・シーケンサ・コマンドにより、発生したエラー情報を取得します。
6	R_RFSP_ForceReset	CPU の内部リセットを発生させます。

3.1.2 コード・フラッシュ制御 API 関数

RFSP Type01 のコード・フラッシュ制御 API 関数一覧を表 3-2 に示します。

表 3-2 RFSP Type01 コード・フラッシュ制御 API 関数一覧

	API 関数名	概要
1	R_RFSP_EraseCodeFlashReq	フラッシュ・メモリ・シーケンサを起動し、コード・フラッシュ・メモリの消去(1 ブロック)を開始します。
2	R_RFSP_WriteCodeFlashReq	フラッシュ・メモリ・シーケンサを起動し、コード・フラッシュの書き込み(4byte)を開始します。

3.1.3 データ・フラッシュ制御 API 関数

RFSP Type01 のデータ・フラッシュ制御 API 関数一覧を表 3-3 に示します。

表 3-3 RFSP Type01 データ・フラッシュ制御 API 関数一覧

	API 関数名	概要
1	R_RFSP_EraseDataFlashReq	フラッシュ・メモリ・シーケンサを起動し、データ・フラッシュ・メモリの消去(1 ブロック)を開始します。
2	R_RFSP_WriteDataFlashReq	フラッシュ・メモリ・シーケンサを起動し、データ・フラッシュの書き込み(4byte)を開始します。

3.1.4 フック関数

RFSP Type01 のフック関数一覧を表 3-4 に示します。

表 3-4 RFSP Type01 フック関数一覧

	API 関数名	概要
1	R_RFSP_HOOK_EnterCriticalSection	割り込み許可フラグ(IE)の状態を退避し、割り込み禁止命令を実行します。
2	R_RFSP_HOOK_ExitCriticalSection	退避されていた割り込み許可フラグ(IE)の状態を復帰します。

3.2 データ型定義

3.2.1 データ型

RFSP Type01 のデータ型定義一覧を表 3-5 に示します。

表 3-5 RFSP 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
rBool_t	unsigned char	Boolean (false:0 / true:1)

3.2.2 グローバル変数

RFSP Type01 で使用するグローバル変数を以下に示します。

(1) sg_u08_psw_ie_state

型 / 名称	static uint8_t sg_u08_psw_ie_state
初期値	0x00 (R_RFSP_VALUE_U08_INIT_VARIABLE)
説明	PSW の割り込み許可フラグ(IE)の状態を退避/復帰するための保存データ - 割り込み禁止 : 0x00u - 割り込み許可 : 0x80u
定義ファイル	r_rfsp_common_userown.c

3.2.3 列挙型

- e_rfsp_flash_memory_mode (列挙変数名 : e_rfsp_flash_memory_mode_t)

フラッシュ・セルフ・プログラミング・モード

Symbol Name	Value	Description
R_RFSP_ENUM_FLASH_MODE_NONPROGRAMMABLE	0x08	非書き換えモード(Non-programmable mode)
R_RFSP_ENUM_FLASH_MODE_CODE_PROGRAMMING	0x02	コード・フラッシュ・プログラミング設定(Code flash memory programming setting)
R_RFSP_ENUM_FLASH_MODE_DATA_PROGRAMMING	0x22	データ・フラッシュ・プログラミング設定(Data flash memory programming setting)

- e_rfsp_ret (列挙変数名 : e_rfsp_ret_t)

戻り値

Symbol Name	Value	Description
R_RFSP_ENUM_RET_STS_OK	0x00	正常終了
R_RFSP_ENUM_RET_STS_BUSY	0x01	実行中
R_RFSP_ENUM_RET_ERR_PARAMETER	0x10	パラメータ・エラー
R_RFSP_ENUM_RET_ERR_MODE_MISMATCHED	0x11	モード不一致エラー

3.2.4 マクロ定義

3.2.4.1 RFSP グローバル・データ設定用マクロ

- 16bit/8bit データ・マスク用マクロ

データの指定サイズ外の bit を 0 で AND してマスクします。

Symbol Name	Value	Description
R_RFSP_VALUE_U08_MASK1_8BIT	0xFFu	8bit マスク値
R_RFSP_VALUE_U16_MASK1_16BIT	0xFFFFu	16bit マスク値

- データ 16bit/8bit シフト用マクロ

32bit のデータを 16bit/8bit シフト、16bit のデータを 8bit シフトします。

Symbol Name	Value	Description
R_RFSP_VALUE_U08_SHIFT_8BIT	8u	8bit シフト値
R_RFSP_VALUE_U08_SHIFT_16BIT	16u	16bit シフト値

- 初期設定マクロ

グローバル変数の初期値を定義。

Symbol Name	Value	Description
R_RFSP_VALUE_U08_INIT_VARIABLE	0x00u	グローバル変数の初期値

3.2.4.2 RL78/G15,RL78/G16 内蔵レジスタ設定用マクロ

- FSSQ(フラッシュ・メモリ・シーケンサ制御レジスタ)用マクロ 1

フラッシュ・メモリ・シーケンサ起動時の各コマンドを定義。

[bit7] SQST : シーケンサの動作開始/停止ビットです。SQST=1 でシーケンサは動作開始します。

[bit2-0] SQMD2-0 :フラッシュ・メモリ・シーケンサの各コマンド

対象レジスタ定義 : R_RFSP_REG_U08_FSSQ

Symbol Name	Value	Description
R_RFSP_VALUE_U08_FSSQ_WRITE	0x81u	フラッシュ・メモリの書き込みコマンド
R_RFSP_VALUE_U08_FSSQ_ERASE	0x84u	フラッシュ・メモリの消去コマンド
R_RFSP_VALUE_U08_FSSQ_CLEAR	0x00u	フラッシュ・メモリ・シーケンサ動作設定のクリア用設定値

- FLPMC(フラッシュ・プログラミング・モード・コントロール・レジスタ)用マクロ

セルフ・プログラミング・モードと非書き換えモードの移行制御に必要な値を定義。

[bit5] SELDFL : フラッシュ・プログラミングの対象領域を選択するビットです。SELDFL=0 でコード・フラッシュが、SELDFL=1 でデータ・フラッシュが選択されます。

[bit3] FWEDIS : フラッシュ・メモリの消去/書き込みの許可/禁止をソフトウェア的に制御するビットです。フラッシュ・メモリの書き込み/消去には FWEDIS=0 を設定しておく必要があります。

[bit1] FLSPM : フラッシュ・メモリの制御モードを操作するするビットです。FLSPM=1 でフラッシュ・メモリはセルフ・プログラミング・モードに移行します。

対象レジスタ定義 : R_RFSP_REG_U08_FLPMC

Symbol Name	Value	Description
R_RFSP_VALUE_U08_FLPMC_MODE_NONPROGRAMMABLE	0x08u	フラッシュ・メモリ・シーケンサが非実行
R_RFSP_VALUE_U08_FLPMC_MODE_CODE_FLASH_PROGRAMMING	0x02u	セルフ・プログラミング・モード(コード・フラッシュ領域選択)
R_RFSP_VALUE_U08_FLPMC_MODE_DATA_FLASH_PROGRAMMING	0x22u	セルフ・プログラミング・モード(データ・フラッシュ領域選択)

- FSASTH(フラッシュ・メモリ・シーケンサ・ステータス・レジスタ:High 8bit)用マクロ

フラッシュ・メモリ・シーケンサの終了ステータスを定義。

[bit6] SQEND : フラッシュ・メモリ・シーケンサの終了ステータスです。SQEND=1 でシーケンサは動作完了です。SQST ビットのクリアでクリアされます。

対象レジスタ定義 : R_RFSP_REG_U08_FSASTH

Symbol Name	Value	Description
R_RFSP_VALUE_U08_MASK1_FSASTH_SQEND	0x40u	フラッシュ・メモリ・シーケンサの終了比較値

- FSASTL(フラッシュ・メモリ・シーケンサ・ステータス・レジスタ:Low 8bit)用マクロ

フラッシュ・メモリ・シーケンサ終了時のエラー・ステータス・マスク値を定義。

[bit4] SEQER : フラッシュ・メモリ・シーケンサのエラーです。SEQER =1 でシーケンサ・エラーです。

[bit1] WRER : 書き込みコマンドのエラーです。WRER =1 で書き込みエラーです。

[bit0] ERER : ブロック消去コマンドのエラーです。ERER =1 で消去エラーです。

対象レジスタ定義 : R_RFSP_REG_U08_FSASTL

Symbol Name	Value	Description
R_RFSP_VALUE_U08_ MASK1_FSASTL_ERROR_FLAG	0x13u	フラッシュ・メモリ・シーケンサ終了時のエラー・ステータス・マスク値

- FSSET(フラッシュ・メモリ・シーケンサ周波数設定レジスタ)用マクロ

フラッシュ・メモリ・シーケンサの動作周波数範囲、および FSSET レジスタ設定値変換用補正値(-1)。

[bit4-0]FSET4-0 : 動作周波数-1 を入力します。(例 : 16MHz の場合、16-1 = 15[01111b]を入力します。)

対象レジスタ定義 : R_RFSP_REG_U08_FSSET

Symbol Name	Value	Description
R_RFSP_VALUE_U08_FREQUENCY_LOWER_LIMIT	1u	入力可能最小動作周波数(1MHz)
R_RFSP_VALUE_U08_FREQUENCY_UPPER_LIMIT	16u	入力可能最大動作周波数(16MHz)
R_RFSP_VALUE_U08_FREQUENCY_ADJUST	1u	FSSET レジスタ設定値変換用補正値 (-1)

- FLAPH/FLAPL, FLSEDH/FLSEDL(フラッシュ・アドレス・ポインタ・レジスタ HIGH/LOW)用マクロ

(1)コード・フラッシュ・メモリ消去(1 ブロック:1Kbyte)用の先頭/終了アドレスを定義。

FLAPH[bit4-0] : FLAP12-8 は、コード・フラッシュ・メモリ領域の先頭上位アドレス設定値。

FLAPL[bit7-0] : FLAP7-0 は、コード・フラッシュ・メモリ領域の先頭下位アドレス設定値。

FLSEDH[bit4-0] : EWA12-8 は、コード・フラッシュ・メモリ領域の終了上位アドレス設定値。

FLSEDL[bit7-2] : EWA7-2 は、コード・フラッシュ・メモリ領域の終了下位アドレス設定値。

対象レジスタ定義 : R_RFSP_REG_U08_FLAPH / R_RFSP_REG_U08_FLAPL

R_RFSP_REG_U08_FLSEDH / R_RFSP_REG_U08_FLSEDL

Symbol Name	Value	Description
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_LOW	0x00u	コード・フラッシュ・ブロック先頭の下位アドレス・マスク値(8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_HIGH	0x1Fu	コード・フラッシュ・ブロック先頭の上位アドレス・マスク値(8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_END_LOW	0xFCu	コード・フラッシュ・ブロック終了の下位アドレス設定値(8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_END_HIGH	0x03u	コード・フラッシュ・ブロック終了の上位アドレス設定値(8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_SHIFT_HIGH_ADDR	2u	ブロック番号からコード・フラッシュ領域オフセット算出用上位アドレス・シフト値

(2)データ・フラッシュ・メモリ消去(1 ブロック:512byte)用の先頭/終了アドレスを定義。

FLAPH[bit4-0] : FLAP12-8 は、データ・フラッシュ・メモリ領域の先頭上位アドレス設定値。

FLAPL[bit7-0] : FLAP7-0 は、データ・フラッシュ・メモリ領域の先頭下位アドレス設定値。

FLSEDH[bit4-0] : EWA12-8 は、データ・フラッシュ・メモリ領域の終了上位アドレス設定値。

FLSEDL[bit7-2] : EWA7-2 は、データ・フラッシュ・メモリ領域の終了下位アドレス設定値。

対象レジスタ定義 : R_RFSP_REG_U08_FLAPH / R_RFSP_REG_U08_FLAPL

R_RFSP_REG_U08_FLSEDH / R_RFSP_REG_U08_FLSEDL

Symbol Name	Value	Description
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_LOW	0x00u	データ・フラッシュ・ブロック先頭の下位アドレス・マスク値(8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_HIGH	0x01u	データ・フラッシュ・ブロック先頭の上位アドレス・マスク値(8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_END_LOW	0xFCu	データ・フラッシュ・ブロック終了の下位アドレス設定値(8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_END_HIGH	0x01u	データ・フラッシュ・ブロック終了の上位アドレス設定値(8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_SHIFT_HIGH_ADDR	1u	ブロック番号からデータ・フラッシュ領域オフセット算出用上位アドレス・シフト値

3.3 API 関数仕様

この章では、Renesas Flash Sample Program (RFSP) Type01 の API 関数の詳細仕様について説明します。

RFSP 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 RFSP 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_RFSP_Init()関数は、全ての RFSP 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。RFSP 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。

・データ・フラッシュを操作する場合、データ・フラッシュへのアクセスを許可した状態で RFSP Type01 の API 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、対象となる RL78 マイクロコントローラのユーザーズマニュアル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。

3.3.1 共通フラッシュ制御 API 関数仕様

RFSP Type01 の共通フラッシュ制御関数を示します。

3.3.1.1 R_RFSP_Init

Information

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_Init (unit8_t i_u08_cpu_freqency);	
Reentrancy	Non-reentrant	
Parameters (IN)	unit8_t i_u08_cpu_freqency	CPU 動作周波数 [1~16(MHz)]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK : 0x00 [正常終了：周波数が範囲内]
		R_RFSP_ENUM_RET_ERR_PARAMETER : 0x10 [パラメータ・エラー：周波数が範囲外]
Description	引数で指定された周波数をフラッシュ・メモリ・シーケンサに設定し、RFSP Type01 の初期化を行います。	
Preconditions	非書き換えモードで実行してください。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	全ての RFSP 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)が 1~16(MHz)の範囲内であるかを確認し、範囲内であれば、CPU 動作周波数 -1 を(g_u08_cpu_frequency)に設定するとともに、FSSET レジスタに設定します 。

備考：

・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。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。
※RFSP 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_freqency)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_freqency)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、対象となる RL78 マイクロコントローラのユーザーズマニュアルを参照してください。

・非書き換えモード以外で本関数を実行した場合、その後の動作は不定となります。

3.3.1.2 R_RFSP_SetFlashMemoryMode

Information

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_SetFlashMemoryMode (e_rfsp_flash_memory_mode_t i_e_pe_mode);	
Reentrancy	Non-reentrant	
Parameters (IN)	e_rfsp_flash_memory_mode_t i_e_pe_mode	フラッシュ・セルフ・プログラミング・モード R_RFSP_ENUM_FLASH_MODE_NONPROGRAMMABLE : 0x08 [非書き換えモード] R_RFSP_ENUM_FLASH_MODE_CODE_PROGRAMMING : 0x02 [セルフ・プログラミング・モード:コード・フラッシュ領域選択] R_RFSP_ENUM_FLASH_MODE_DATA_PROGRAMMING : 0x22 [セルフ・プログラミング・モード:データ・フラッシュ領域選択]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK : 0x00 [正常終了] R_RFSP_ENUM_RET_ERR_MODE_MISMATCHED : 0x11 [モード不一致エラー] (指定モードへ設定されなかった)
Description	引数で指定されたフラッシュ・セルフ・プログラミング・モードをフラッシュ・メモリ・シーケンサへ設定します。	
Preconditions	フラッシュ・メモリ・シーケンサのコマンドを実行していない状態で、本関数を実行してください。	
Remarks	-	

動作概要：

・引数(i_e_pe_mode)の値に応じて FLPMC レジスタ値を設定、指定されたフラッシュ・セルフ・プログラミング・モードに移行します。

備考：

・引数へフラッシュ・セルフ・プログラミング・モード以外の値を指定した場合は、その後の動作は不定となります。

・R_RFSP_Init 関数を実行せずに本関数を実行した場合、RFSP の各書き換え処理が正常に実施されても、そのデータの値は保証の対象外となります。RFSP Type01 を使用する場合、全ての RFSP 関数を使用する前に、必ず、R_RFSP_Init()関数を 1 回実行してください。

3.3.1.3 R_RFSP_CheckCFDFSeqEndStep1

Information

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_CheckCFDFSeqEndStep1 (void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK : 0x00 [正常終了]
		R_RFSP_ENUM_RET_STS_BUSY : 0x01 [シーケンサ・コマンド動作中]
Description	起動したフラッシュ・メモリ・シーケンサの動作終了を確認します。	
Preconditions	フラッシュ・メモリ・シーケンサの動作完了後に、本関数を実行してください。	
Remarks	戻り値が R_RFSP_ENUM_RET_STS_BUSY である間は、本関数を再度実行してください。本関数で R_RFSP_ENUM_RET_STS_OK を確認後、関数 R_RFSP_CheckCFDFSeqEndStep2()を実行してください	

動作概要：

・起動したフラッシュ・メモリ・シーケンサの動作が終了[SQEND(FSASTH の bit6) = 1]したかどうかを確認します。

・フラッシュ・メモリ・シーケンサの動作が終了していた場合、フラッシュ・メモリ・シーケンサ制御レジスタ(FSSQ)へ 0x00 を設定して SQST[bit7] をクリアし、R_RFSP_ENUM_RET_STS_OK を返します。終了していなかった場合は、R_RFSP_ENUM_RET_STS_BUSY を返します。

備考：

・戻り値が R_RFSP_ENUM_RET_STS_BUSY である間は、本関数を再度実行してください。

・フラッシュ・メモリ・シーケンサを起動するコマンド開始後以外で、本関数を実行した場合、正常に動作しません。

・本関数で R_RFSP_ENUM_RET_STS_OK を確認後、関数 R_RFSP_CheckCFDFSeqEndStep2()を実行してください。

3.3.1.4 R_RFSP_CheckCFDFSeqEndStep2

Information

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_CheckCFDFSeqEndStep2 (void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK : 0x00 [正常終了 ： シーケンサ動作終了]
		R_RFSP_ENUM_RET_STS_BUSY : 0x01 [シーケンサ・コマンド動作中]
Description	フラッシュ・メモリ・シーケンサ制御レジスタのクリアにより、コマンド動作が終了したかどうかを確認します。	
Preconditions	関数 R_RFSP_CheckCFDFSeqEndStep1()にて R_RFSP_ENUM_RET_STS_OK を確認した後に、本関数を実行してください。	
Remarks	戻り値が R_RFSP_ENUM_RET_STS_BUSY である間は、本関数を再度実行してください。	

動作概要：

・フラッシュ・メモリ・シーケンサ制御レジスタ(FSSQ)へ 0x00 を設定したことにより、フラッシュ・メモリ・シーケンサ・コマンドの動作が全て終了[SQEND(FSASTH の bit6) = 0]したかどうかを確認します。

・フラッシュ・メモリ・シーケンサ・コマンドの動作が終了していた場合、
R_RFSP_ENUM_RET_STS_OK を返します。
終了していなかった場合、R_RFSP_ENUM_RET_STS_BUSY を返します。

備考：

・戻り値が R_RFSP_ENUM_RET_STS_BUSY である間は、本関数を再度実行してください。

・R_RFSP_CheckCFSPFSeqEndStep1()で、R_RFSP_ENUM_RET_STS_OK 確認後以外で、本関数を実行した場合、正常に動作しません。

3.3.1.5 R_RFSP_GetSeqErrorStatus

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_GetSeqErrorStatus (uint8_t __near * onp_u08_error_status);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	uint8_t __near *onp_u08_error_status	発生したエラー情報を格納する変数へのポインタ
Return Value	N/A	
Description	フラッシュ・メモリ・シーケンサ・コマンドにより、発生したエラー情報を取得します。	
Preconditions	フラッシュ・メモリ・シーケンサの動作完了後に、本関数を実行してください。	
Remarks	-	

動作概要：

・FSASTL レジスタの値(8bit)を読み出し、引数ポインタ(onp_u08_error_status)が示す変数に格納します。

取得するエラー情報（FSASTL レジスタ bit 4, 1-0 の 3bit）

[bit7：(0) 予約ビット]

[bit6：(0) 予約ビット]

[bit5：(0) 予約ビット]

[bit4：フラッシュ・メモリ・シーケンサ・エラー]

[bit3：(0) 予約ビット]

[bit2：(0) 予約ビット]

[bit1：書き込みコマンド・エラー]

[bit0：ブロック消去コマンド・エラー]

3.3.1.6 R_RFSP_ForceReset

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_ForceReset(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	CPU の内部リセットを発生させます。	
Preconditions	-	
Remarks	-	

動作概要：

・意図的に不正命令(0xFF)の命令コードを実行し、CPU の内部リセットを発生させます。

備考：

・CPU の内部リセットが発生するため、この関数以降の処理は実行されません。

・FFH の命令コードによる内部リセット（不正命令の実行による内部リセット）については、対象となる RL78 マイクロコントローラのユーザーズマニュアルを参照してください。

・本関数によるリセットは、オンチップ・デバッグ・エミュレータによるエミュレーションでは発生しません。

3.3.2 コード・フラッシュ制御 API 関数仕様

RFSP Type01 のコード・フラッシュ制御関数を示します。

3.3.2.1 R_RFSP_EraseCodeFlashReq

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_EraseCodeFlashReq (uint8_t i_u08_block_number);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_block_number	消去対象ブロック番号[0~31] 例:RL78/G15 では、MAX 8KB[0~7] RL78/G16 では、MAX 32KB[0~31]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	フラッシュ・メモリ・シーケンサを起動し、コード・フラッシュ・メモリの消去(1 ブロック)を開始します。	
Preconditions	セルフ・プログラミング・モード(コード・フラッシュ領域を選択)で使用してください。 フラッシュ・メモリ・シーケンサのコマンドを実行していない状態で使用してください。	
Remarks	本関数実行後 R_RFSP_CheckCFDFSeqEndStep1()を実行してください。	

動作概要：

・フラッシュ・メモリ・シーケンサを起動し、コード・フラッシュ・メモリの消去する 1 ブロック(1Kbyte)のアドレスを設定します。

- 引数(i_u08_block_number)の消去対象ブロック番号から、コード・フラッシュ・メモリのスタート・アドレスとエンド・アドレス(1 ブロック分：1Kbyte)を計算し、FLAPL/H と FLSEDL/H へ設定します。

・FSSQ レジスタに R_RFSP_VALUE_U08_FSSQ_ERASE :0x84 を設定し、消去を開始します。
(SQST[bit7] = 1, SQMD[bit2-0] = 4[0b100], 他の bit は 0)

備考：

・引数(i_u08_block_number)は 8bit の上位 3bit を無効とした下位 5bit を使用します。デバイスに実装されているコード・フラッシュ・メモリのブロック数の範囲内の値であることが前提条件です。範囲外の値を指定した場合、その後の動作は不定となります。

・セルフ・プログラミング・モード(コード・フラッシュ領域選択)以外で本関数を実行した場合、その後の動作は不定となります。

・フラッシュ・メモリ・シーケンサのコマンド実行中に本関数を実行した場合、その後の動作は不定となります。

3.3.2.2 R_RFSP_WriteCodeFlashReq

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_WriteCodeFlashReq (uint16_t i_u16_start_addr, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	書き込み対象スタート・アドレス(4byte 境界) [コード・フラッシュ領域のアドレス]
	uint8_t __near * inp_u08_write_data	書き込みデータ変数へのポインタ [ポインタが指す書き込みデータは 4byte]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	フラッシュ・メモリ・シーケンサを起動し、コード・フラッシュの書き込み(4byte)を開始します。	
Preconditions	セルフ・プログラミング・モード(コード・フラッシュ領域選択)で使用してください。 フラッシュ・メモリ・シーケンサのコマンドを実行していない状態で使用してください。	
Remarks	本関数実行後 R_RFSP_CheckCFDFSeqEndStep1()を実行してください。	

動作概要：

・フラッシュ・メモリ・シーケンサを起動し、コード・フラッシュ・メモリの書き込みアドレスと書き込みデータ(4byte)を設定します。

- 引数(i_u16_start_addr)の書き込み対象のコード・フラッシュ・スタート・アドレスを FLAPL/H レジスタに設定します。
- 引数ポインタ(inp_u08_write_data)が指す変数(コード・フラッシュの書き込みデータ)の値(4byte)を FLWLL/LH/HL/HH レジスタに設定します。

・FSSQ レジスタに R_RFSP_VALUE_U08_FSSQ_WRITE:0x81 を設定し、書き込みを開始します。
(SQST[bit7] = 1, SQMD[bit2-0] = 1[0b001], 他の bit は 0)

備考：

・引数(i_u16_start_addr)は 16bit の上位 1bit と下位 2bit を'0'でマスクした 13bit を使用します。デバイスに実装されているコード・フラッシュ領域の範囲内の 4byte 境界のアドレスであることが前提条件です。範囲外の領域や 4byte 境界以外のアドレスを指定した場合、その後の動作は不定となります。

・引数(inp_u08_write_data)は、入力が 8bit データへのポインタです。このポインタを更新しながら継続して使用する場合、コード・フラッシュの書き込み単位 4byte ずつ更新する必要があるので、ご注意ください。

・セルフ・プログラミング・モード(コード・フラッシュ領域選択)以外で本関数を実行した場合、その後の動作は不定となります。

・フラッシュ・メモリ・シーケンサのコマンド実行中に本関数を実行した場合、その後の動作は不定となります。

3.3.3 データ・フラッシュ制御 API 関数仕様

RFSP Type01 のデータ・フラッシュ制御関数を示します。

3.3.3.1 R_RFSP_EraseDataFlashReq

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_EraseDataFlashReq (uint8_t i_u08_block_number);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_block_number	消去対象ブロック番号[0~1] 例:RL78/G15,RL78/G16 では、MAX 1KB[0~1]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	フラッシュ・メモリ・シーケンサを起動し、データ・フラッシュ・メモリの消去(1 ブロック)を開始します。	
Preconditions	セルフ・プログラミング・モード(データ・フラッシュ領域を選択)で使用してください。 フラッシュ・メモリ・シーケンサのコマンドを実行していない状態で使用してください。	
Remarks	本関数実行後 R_RFSP_CheckCFDFSeqEndStep1()を実行してください。	

動作概要：

・フラッシュ・メモリ・シーケンサを起動し、データ・フラッシュ・メモリの消去する 1 ブロック(512byte)のアドレスを設定します。

- 引数(i_u08_block_number)の消去対象ブロック番号から、コード・フラッシュ・メモリのスタート・アドレスとエンド・アドレス(1 ブロック分：512byte)を計算し、FLAPL/H と FLSEDL/H へ設定します。

・FSSQ レジスタに R_RFSP_VALUE_U08_FSSQ_ERASE :0x84 を設定し、消去を開始します。
(SQST[bit7] = 1, SQMD[bit2-0] = 4[0b100], 他の bit は 0)

備考：

・引数(i_u08_block_number)は 8bit の上位 7bit を無効とした下位 1bit を使用します。デバイスに実装されているデータ・フラッシュ・メモリのブロック数の範囲内の値であることが前提条件です。範囲外の値を指定した場合、その後の動作は不定となります。

・セルフ・プログラミング・モード(データ・フラッシュ領域選択)以外で本関数を実行した場合、その後の動作は不定となります。

・フラッシュ・メモリ・シーケンサのコマンド実行中に本関数を実行した場合、その後の動作は不定となります。

3.3.3.2 R_RFSP_WriteDataFlashReq

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_WriteDataFlashReq (uint16_t i_u16_start_addr, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	書き込むアドレス 0x9000 をベースアドレス"0x0000"とした書き込み対象スタート・アドレス(4byte 境界) [データ・フラッシュ領域のアドレス:0x0000-0x03FC]
	uint8_t __near * inp_u08_write_data	書き込みデータ変数へのポインタ [ポインタが指す書き込みデータは 4byte]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	フラッシュ・メモリ・シーケンサを起動し、データ・フラッシュの書き込み(4byte)を開始します。	
Preconditions	セルフ・プログラミング・モード(データ・フラッシュ領域選択)で使用してください。 フラッシュ・メモリ・シーケンサのコマンドを実行していない状態で使用してください。	
Remarks	本関数実行後 R_RFSP_CheckCFDFSeqEndStep1()を実行してください。	

動作概要：

・フラッシュ・メモリ・シーケンサを起動し、データ・フラッシュ・メモリの書き込みアドレスと書き込みデータ(4byte)を設定します。

- 引数(i_u16_start_addr)の書き込み対象のデータ・フラッシュ・スタート・アドレスを FLAPL/H レジスタに設定します。
- 引数ポインタ(inp_u08_write_data)が指す変数(データ・フラッシュの書き込みデータ)の値(4byte)を FLWLL/LH/HL/HH レジスタに設定します。

・FSSQ レジスタに R_RFSP_VALUE_U08_FSSQ_WRITE:0x81 を設定し、書き込みを開始します。
(SQST[bit7] = 1, SQMD[bit2-0] = 1[0b001], 他の bit は 0)

備考：

・引数(i_u16_start_addr)は 16bit の上位 6bit と下位 2bit を'0'でマスクした 8bit を使用します。デバイスに実装されているデータ・フラッシュ領域の範囲内の 4byte 境界のアドレスであることが前提条件です。範囲外の領域や 4byte 境界以外のアドレスを指定した場合、その後の動作は不定となります。

・引数(inp_u08_write_data)は、入力が 8bit データへのポインタです。このポインタを更新しながら継続して使用する場合、データ・フラッシュの書き込み単位 4byte ずつ更新する必要があるので、ご注意ください。

・セルフ・プログラミング・モード(データ・フラッシュ領域選択)以外で本関数を実行した場合、その後の動作は不定となります。

・フラッシュ・メモリ・シーケンサのコマンド実行中に本関数を実行した場合、その後の動作は不定となります。

3.3.4 フック関数仕様

RFSP Type01 のフック関数を示します。

3.3.4.1 R_RFSP_HOOK_EnterCriticalSection

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_HOOK_EnterCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	割り込み許可フラグ(IE)の状態を退避し、割り込み禁止命令を実行します。	
Preconditions	割り込み禁止状態で実行する処理の前に、割り込みを禁止する場合に実行します。	
Remarks	-	

動作概要：

・割り込みの禁止/許可状態を取得し、PSW の割り込み許可フラグ(IE)の保存データ(sg_u08_psw_ie_state)へ退避します。

・割り込み禁止マクロ命令 R_RFSP_DISABLE_INTERRUPT を実行します。

備考：

・割り込み禁止状態で実行する必要がある処理(クリティカル・セクション)の前に実行し、クリティカル・セクション終了後、R_RFSP_HOOK_ExitCriticalSection 関数を実行してください。

3.3.4.2 R_RFSP_HOOK_ExitCriticalSection

Information

Syntax	R_RFSP_FAR_FUNC void R_RFSP_HOOK_ExitCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	退避されていた割り込み許可フラグ(IE)の状態を復帰します。	
Preconditions	割り込み禁止状態で実行する処理の後に、割り込みを許可する場合に実行します。	
Remarks	-	

動作概要：

・PSW の割り込み許可フラグ(IE)の保存データ(sg_u08_psw_ie_state)の状態により、割り込み許可マクロ命令を実行します。

sg_u08_psw_ie_state の値:

- 0x00[bit7 = 0：割り込み禁止]の場合 ： 何も実行しません。
- 0x80[bit7 = 1：割り込み許可]の場合 ： 割り込み許可マクロ命令[R_RFSP_ENABLE_INTERRUPT]を実行して、割り込み許可状態(EI)へ復帰します。

備考：

・R_RFSP_HOOK_EnterCriticalSection 関数実行後、割り込み禁止状態で実行する必要がある処理(クリティカル・セクション)の終了後に実行します。

4 フラッシュ・メモリ・シーケンサ操作

RL78/G15,RL78/G16 のフラッシュ・メモリ・シーケンサを操作する前に、フラッシュ・メモリ・シーケンサ周波数設定レジスタへ CPU の動作周波数を設定しておく必要があります。

4.1 動作周波数の初期設定

R_RFSP_Init 関数の引数で入力された CPU の動作周波数[1~16(MHz)]の値(g_u08_cpu_frequency:整数値-1)をフラッシュ・メモリ・シーケンサ周波数設定レジスタ(FSSET)の FSET[bit4-0]へ設定します。

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。

本操作の対象関数：R_RFSP_Init, R_RFSP_SetFlashMemoryMode

≪操作方法≫

・フラッシュ・メモリ・シーケンサ周波数設定レジスタ(FSSET)への書き込みは、[bit7-5]を'0'、CPU の動作周波数[1~16(MHz)]の値の[整数値-1]を FSET にあたる[bit4-0]へ設定します。

注)フラッシュ・メモリ・シーケンサを使用して、コード/データ・フラッシュ・メモリへの書き換え操作を実行する場合、FSSET レジスタの FSET へ CPU の動作周波数を設定しておく必要があります。

CPU の動作周波数が正しく設定されていない状態での書き換え動作は不定となり、書かれたデータは保証されませんので、ご注意ください。(書き込み直後のフラッシュ・メモリのデータ値が期待値通りであっても、その値の保持期間を保証できません。)

FSSET レジスタ(リセット時:0x00):

7	6	5	4	3	2	1	0
0	0	0	FSET4	FSET3	FSET2	FSET1	FSET0
R	R	R	R/W	R/W	R/W	R/W	R/W

4.2 セルフ・プログラミング・モード及び領域の設定

RL78/G15,RL78/G16 はセルフ・プログラミング・モードを設定して、フラッシュ領域(コード・フラッシュ・メモリ、またはデータ・フラッシュ・メモリ)を書き換えることができます。

4.2.1 フラッシュ・セルフ・プログラミング・モード設定

フラッシュ・メモリのセルフ・プログラミング・モードは、フラッシュ・プログラミング・モード・コントロール・レジスタ(FLPMC レジスタ)で設定します。

FLPMC レジスタ(リセット時:0x08) :

7	6	5	4	3	2	1	0
0	0	SELDFL	0	FWEDIS	0	FLSPM	0
R	R	R/W	R	R/W	R	R/W	R

- SELDFL[bit5]は、フラッシュ・プログラミング領域を選択します。
  SELDFL = 0(リセット時) / 1 : コード・フラッシュ領域選択 / データ・フラッシュ領域選択
- FWEDIS[bit3]は、フラッシュ・メモリの[消去／書き込み]の許可／禁止を制御します。
  FWEDIS = 0 / 1(リセット時) : [書込み／消去]可能 / [書込み／消去]不可
- FLSPM[bit1]は、フラッシュ・プログラミング・モードを選択します。
  FLSPM = 0(リセット時) / 1 : リードモード(通常モード) / セルフ・プログラミング・モード

≪操作方法≫

- コード・フラッシュ 書き換え可能状態の設定
  FLPMC = 0x02 を設定します。
- データ・フラッシュ 書き換え可能状態の設定
  FLPMC = 0x22 を設定します。
- フラッシュ・メモリ書き換え不可状態の設定
  FLPMC = 0x08 を設定します。

4.3 フラッシュ・メモリ・シーケンサ

4.3.1 概要

RL78/G15,RL78/G16 のコード/データ・フラッシュ領域の書き換えには、フラッシュ・メモリ・シーケンサの専用コマンドを実行する必要があります。

4.3.2 フラッシュ・メモリ・シーケンサ・コマンド

RL78/G15,RL78/G16 のコード/データ・フラッシュ領域を書き換えるために、1 ワード[4byte]の書き込みコマンドとブロック消去コマンドが用意されています。コマンドの発行は、フラッシュ・メモリ・シーケンサ制御レジスタ(FSSQ)の SQMD2-0[bit2-0]へ対象のコマンド番号を入力するとともに、SQST[bit7]を'1'に設定します。コマンドは、「1.5 注意事項」を参照、ご理解いただいた上で実行してください。

FSSQ レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
SQST	0	0	0	0	SQMD2	SQMD1	SQMD0
R/W	R	R	R	R	R/W	R/W	R/W

フラッシュ・メモリ・シーケンサの専用コマンドを表 4-1 に示します。

表 4-1 フラッシュ・メモリ・シーケンサの専用コマンド

SQMD2-0	機能(専用コマンド) コマンドの説明
0h	初期値（コマンド非選択）
1h	書き込み FLAPH/FLAPLレジスタで指定されるフラッシュ・メモリのアドレスに、FLWHH/FLWHL/FLWLH/FLWLLレジスタで指定したデータを書き込みます。 - 書き込み単位(コード・フラッシュ領域) : 1ワード(4byte) [SELDFL ビットに0を設定時] - 書き込み単位(データ・フラッシュ領域) : 1ワード(4byte) [SELDFL ビットに1を設定時]
4h	ブロック消去 FLAPH/FLAPLレジスタで指定されるブロック先頭アドレスからFLSEDH/FLSEDLレジスタで指定されるブロック終了アドレスまでのブロック消去を行います。
その他の値	設定禁止

注)書き込み/ブロック消去コマンド共に、データ・フラッシュ領域が指定されている場合のアドレスは、先頭アドレス(0x9000)を 0x0000 とした相対アドレスを指定します。

・FLAPH/FLAPL レジスタ(フラッシュ・アドレス・ポインタ・レジスタ)

- 対象:RL78/G15

FLAPH レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
0	0	0	FLAP 12	FLAP 11	FLAP 10	FLAP 9	FLAP 8
R	R	R	R/W	R/W	R/W	R/W	R/W

FLAPL レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
FLAP 7	FLAP 6	FLAP 5	FLAP 4	FLAP 3	FLAP 2	FLAP 1	FLAP 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- 対象:RL78/G16

FLAPH レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
0	FLAP 14	FLAP 13	FLAP 12	FLAP 11	FLAP 10	FLAP 9	FLAP 8
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLAPL レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
FLAP 7	FLAP 6	FLAP 5	FLAP 4	FLAP 3	FLAP 2	FLAP 1	FLAP 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

・FLSEDH/FLSEDL レジスタ(フラッシュ・エンド・アドレス・ポインタ・レジスタ)

- 対象:RL78/G15

FLSEDH レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
0	0	0	EWA 12	EWA 11	EWA 10	EWA 9	EWA 8
R	R	R	R/W	R/W	R/W	R/W	R/W

FLSEDL レジスタ(リセット時:0x0000) :

7	6	5	4	3	2	1	0
EWA 7	EWA 6	EWA 5	EWA 4	EWA 3	EWA 2	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R	R

- 対象:RL78/G16

FLSEDH レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
0	EWA14	EWA13	EWA 12	EWA 11	EWA 10	EWA 9	EWA 8
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLSEDL レジスタ(リセット時:0x0000) :

7	6	5	4	3	2	1	0
EWA 7	EWA 6	EWA 5	EWA 4	EWA 3	EWA 2	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R	R

・FLWHH/ FLWHL/FLWLH/ FLWLL レジスタ(フラッシュ・ライト・バッファ・レジスタ)

FLWHH レジスタ(リセット時:0x00) :

15	14	13	12	11	10	9	8
FLW 31	FLW 30	FLW 29	FLW 28	FLW 27	FLW 26	FLW 25	FLW 24
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLWHL レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
FLW 23	FLW 22	FLW 21	FLW 20	FLW 19	FLW 18	FLW 17	FLW 16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLWLH レジスタ(リセット時:0x00) :

15	14	13	12	11	10	9	8
FLW 15	FLW 14	FLW 13	FLW 12	FLW 11	FLW 10	FLW 9	FLW 8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLWLL レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
FLW 7	FLW 6	FLW 5	FLW 4	FLW 3	FLW 2	FLW 1	FLW 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

4.3.2.1 コード・フラッシュ領域書き換えの操作

コード・フラッシュ領域の書き換えは、フラッシュ・プログラミング・モードをセルフ・プログラミング・モード(コード・フラッシュ領域選択)に設定してからフラッシュ・メモリ・シーケンサ・コマンドを実行します。各コマンド実行に必要な指定アドレスやデータをあらかじめ該当レジスタに設定してから、コマンドを開始する必要があります。

コード・フラッシュ領域書き換え時の消去ブロック単位/書き込み単位

- 消去ブロック単位：1**Kbyte**
- 書き込み単位：**1 ワード[4byte]**

本操作の対象関数：R_RFSP_EraseCodeFlashReq, R_RFSP_WriteCodeFlashReq

≪操作方法≫

対象コマンドは、コード・フラッシュのブロック消去、書き込みです。

・対象領域をコード・フラッシュへ設定してセルフ・プログラミング・モードに移行します。移行手順は「4.2.1 フラッシュ・セルフ・プログラミング・モード設定」を参照してください。
FLPMC = 0x02 を設定します。

・各コマンド実行前に、対象レジスタへ指定データを設定します。
(1)ブロック消去
FLAPH/FLAPL レジスタ：コード・フラッシュ・メモリのブロック先頭アドレス(例:0x0800)
FLSEDH/FLSEDL レジスタ：コード・フラッシュ・メモリのブロック終了アドレス(例:0x0BFF)
(2)書き込み:1 ワード[4byte]単位の為、アドレスは、[bit1-0]が'0'の 4 の倍数を設定する必要があります。
FLAPH/FLAPL レジスタ：対象のコード・フラッシュ・メモリの先頭アドレス(例:0x0800)
FLSEDH/FLSEDL レジスタ： ALL'0'、または未設定(例:0x000000)
FLWHH/FLWHL/FLWLH/ FLWLL レジスタ：書き込むデータ(1 ワード[4byte])

・FSSQ の SQMD2-0[bit2-0]へ対象コマンド番号を入力するとともに、SQST[bit7]を'1'に設定します。
ブロック消去：0x84、書き込み：0x81
注)セルフ・プログラミングによるコード・フラッシュ書き換え中は、CPU が停止してユーザ・プログラムの操作ができなくなりますのでご注意ください。

・フラッシュ・メモリ・シーケンサ・コマンドの完了を待ちます。コマンドの完了待ち手順は、「4.3.3 フラッシュ・メモリ・シーケンサ・コマンドの終了判定手順」を参照してください。

・コマンド実行後の処理
コマンド処理を継続する場合：
セルフ・プログラミング・モード(対象はコードフラッシュ領域)を設定したまま、対象レジスタのデータを更新してコード・フラッシュ領域の書き込みコマンド、またはブロック消去コマンドを実行することも可能です。
コマンド処理を完了する場合：
フラッシュ・メモリ書き換え不可状態に移行します。移行手順は「4.2.1 フラッシュ・セルフ・プログラミング・モード設定」を参照してください。

4.3.2.2 データ・フラッシュ領域書き換えの操作

データ・フラッシュ領域の書き換えは、フラッシュ・プログラミング・モードをセルフ・プログラミング・モード(データ・フラッシュ領域選択)に設定してからフラッシュ・メモリ・シーケンサ・コマンドを実行します。各コマンド実行に必要な指定アドレスやデータをあらかじめ該当レジスタに設定してから、コマンドを開始する必要があります。

データ・フラッシュ領域書き換え時の消去ブロック単位/書き込み単位

- 消去ブロック単位：**512byte**
- 書き込み単位：**1 ワード[4byte]**

本操作の対象関数：R_RFSP_EraseDataFlashReq , R_RFSP_WriteDataFlashReq

≪操作方法≫

対象コマンドは、データ・フラッシュのブロック消去、書き込みです。

・対象領域をデータ・フラッシュへ設定してセルフ・プログラミング・モードに移行します。移行手順は「4.2.1 フラッシュ・セルフ・プログラミング・モード設定」を参照してください。

FLPMC = 0x22 を設定します。

・各コマンド実行前に、対象レジスタへ指定データを設定します。

(1)ブロック消去

FLAPH/FLAPL レジスタ：データ・フラッシュ・メモリのブロック先頭アドレス

(例: 0x9000 を指定する場合、0x9000 をベースアドレスとして、相対値 0x0000 を設定します。)

FLSEDH/FLSEDL レジスタ：データ・フラッシュ・メモリのブロック終了アドレス

(例: 0x91FF を指定する場合、0x9000 をベースアドレスとして、相対値 0x01FF を設定します。)

(2)書き込み:1 ワード[4byte]単位の為、アドレスは、[bit1-0]が'0'の 4 の倍数を設定する必要があります。

FLAPH/FLAPL レジスタ：対象のデータ・フラッシュ・メモリの先頭アドレス

(例: 0x9000 を指定する場合、0x9000 をベースアドレスとして、相対値 0x0000 を設定します。)

FLSEDH/FLSEDL レジスタ： ALL'0'、または未設定(例:0x000000)

FLWHH/FLWHL/FLWLH/ FLWLL レジスタ：書き込むデータ(1 ワード[4byte])

・FSSQ の SQMD2-0[bit2-0]へ対象コマンド番号を入力するとともに、SQST[bit7]を'1'に設定します。

ブロック消去：0x84、書き込み：0x81

注)セルフ・プログラミングによるデータ・フラッシュ書き換え中は、CPU が停止してユーザ・プログラムの操作ができなくなりますのでご注意ください。

・フラッシュ・メモリ・シーケンサ・コマンドの完了を待ちます。コマンドの完了待ち手順は、「4.3.3 フラッシュ・メモリ・シーケンサ・コマンドの終了判定手順」を参照してください。

・コマンド実行後の処理

コマンド処理を継続する場合：

セルフ・プログラミング・モード(対象はデータフラッシュ領域)を設定したまま、対象レジスタのデータを更新して、データ・フラッシュ領域の書き込みコマンド、またはブロック消去コマンドを実行することも可能です。

コマンド処理を完了する場合：

フラッシュ・メモリ書き換え不可状態に移行します。移行手順は「4.2.1 フラッシュ・セルフ・プログラミング・モード設定」を参照してください。

4.3.3 フラッシュ・メモリ・シーケンサ・コマンドの終了判定手順

RL78/G15,RL78/G16 で起動したフラッシュ・メモリ・シーケンサ・コマンドを終了する場合、特定の終了判定手順を実行する必要があります。

フラッシュ・メモリ・シーケンサ・コマンドの終了判定では、FSASTH レジスタの SQEND[bit6]を参照、"1"にセットされたことを確認して、フラッシュ・メモリ・シーケンサ・コマンドの終了判定手順を実施します。終了判定実施後、コマンドごとのエラーの発生有無を FSASTL レジスタのエラービット(WRER[bit1], ERER[bit0])で確認します。

FSASTH レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
0	SQEND	0	0	0	0	0	0
R	R	R	R	R	R	R	R

FSASTL レジスタ(リセット時:0x00) :

7	6	5	4	3	2	1	0
0	0	0	SEQER	0	0	WRER	ERER
R	R	R	R	R	R	R	R

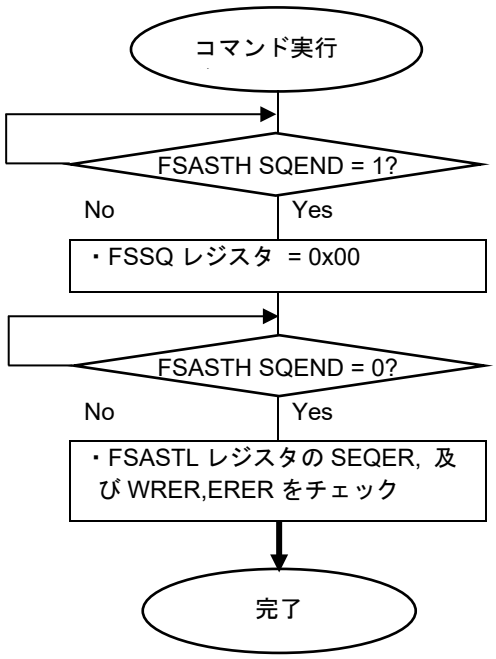
≪終了判定手順≫

(1)フラッシュ・メモリ・シーケンサ・コマンド起動後、FSASTH レジスタの SQEND[bit6]が自動的にセットされるまで待ちます。

(2)FSASTH レジスタの SQEND[bit6]のセット確認後、FSSQ レジスタの SQST[bit7] をクリアします。

(3)FSASTH レジスタの SQEND[bit6]が自動的にクリアされるまで待ち、クリアされたら終了します。

フラッシュ・メモリ・シーケンサ・コマンドの終了判定フローを図 4-1 に示します。



FSSQ レジスタの SQMD2-0,SQST ビットを設定し、書き込み、またはブロック消去コマンドが実行されたことが前提です。

・コマンド実行処理が終了したかを判別

・FSSQ レジスタに"0x00"を設定します。

・コマンドが完了したかを判別

・エラーが発生したかをチェック

図 4-1 フラッシュ・メモリ・シーケンサ・コマンドの終了判定フロー

4.4 フラッシュ領域書き換え時のコマンドの実行例

4.4.1 コード/データ・フラッシュ領域書き換え時のコマンド実行例

コード/データ・フラッシュ領域書き換え時のコマンド実行フローを図 4-2 に示します。

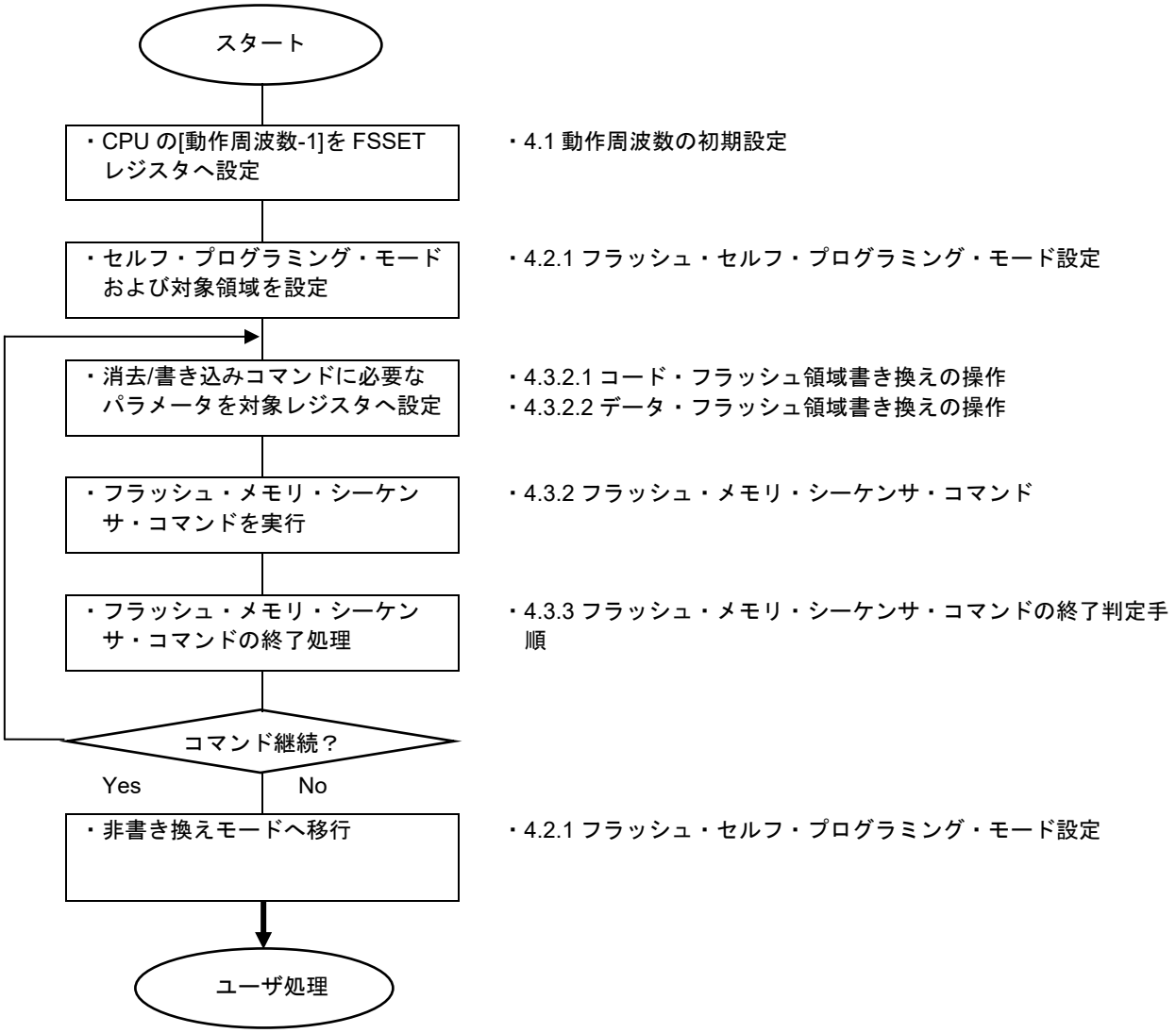


図 4-2 コード/データ・フラッシュ領域書き換え時のコマンド実行フロー

5 サンプル・プログラム

RFSP 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。

5.1 ファイル構成

5.1.1 フォルダ構成

この項では、RL78/G15 用のサンプル・プログラムを例に説明しています。RL78/G15 以外を使用する場合は、G15 を対象のデバイスに読みかえてください。

・RL78/G15 のサンプルのフォルダ名("RL78_G15")は、対象デバイスのフォルダ名に読み変えてください。

RL78/G16 を使用する場合のフォルダ名："RL78_G16"

サンプル・プログラムのフォルダ構成を図 5-1 に示します。

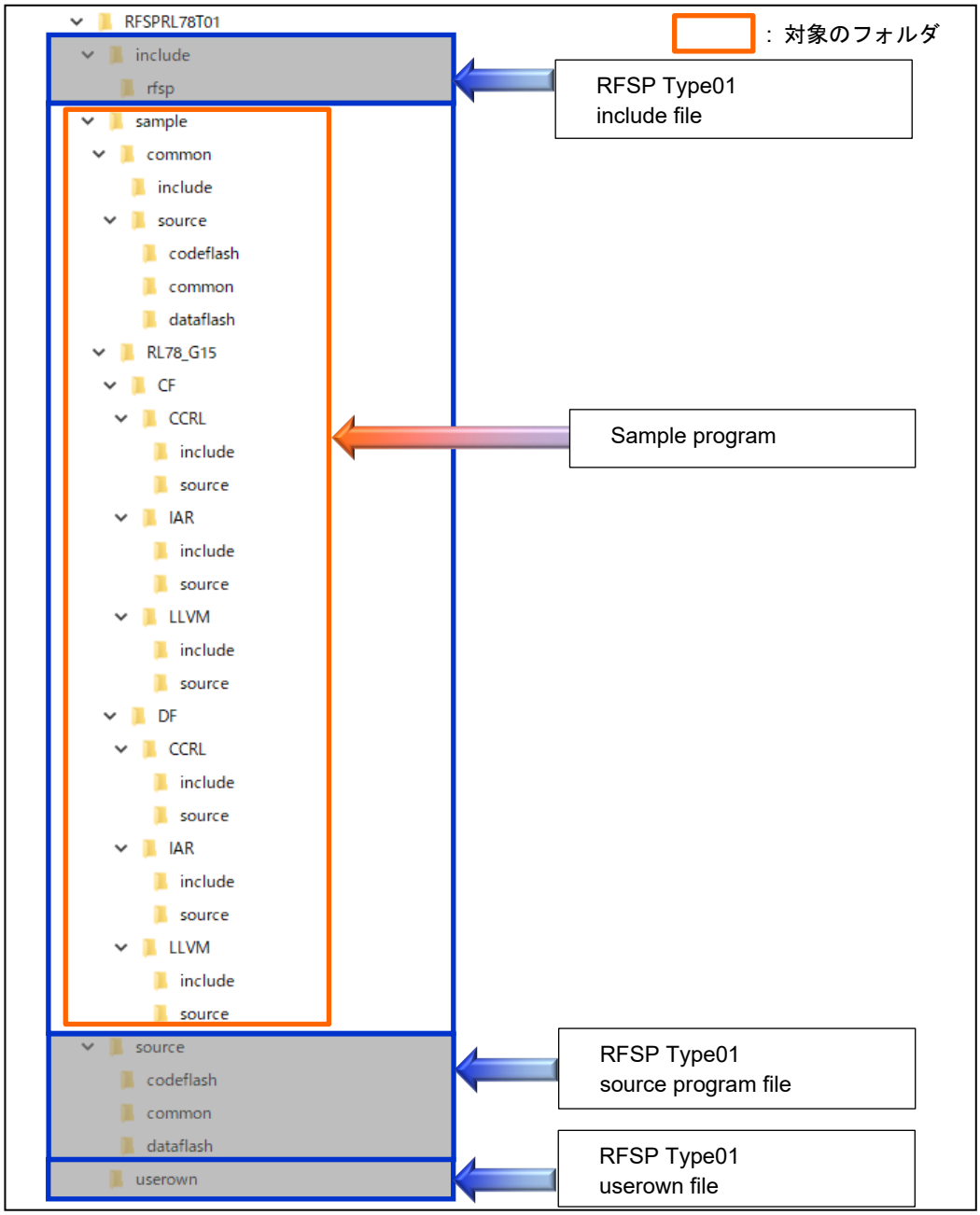


図 5-1 サンプル・プログラムのフォルダ構成

5.1.2 ファイル・リスト

5.1.2.1 ソース・ファイル・リスト

"sample\common\source\common\"フォルダ内のプログラム・ソース・ファイルを表 5-1 に示します。

表 5-1 "sample\common\source\common\"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_common.c	共通フラッシュ・メモリ制御用関数サンプル・ファイル

"sample\common\source\dataflash\"フォルダ内のプログラム・ソース・ファイルを
表 5-2 に示します。

表 5-2 "sample\common\source\dataflash\"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_data_flash.c	データ・フラッシュ・メモリ制御用関数サンプル・ファイル

"sample\common\source\codeflash\"フォルダ内のプログラム・ソース・ファイルを
表 5-3 に示します。

表 5-3 "sample\common\source\codeflash\"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_code_flash.c	コード・フラッシュ・メモリ制御用関数サンプル・ファイル

"sample\RL78_G15\"フォルダ内のコード・フラッシュ制御[CF],データ・フラッシュ制御[DF]の各メイン処理のプログラム・ソース・ファイルを表 5-4 に示します。

- コード・フラッシュ[CF]のメイン処理："sample\RL78_G15\CF\[コンパイラ名]\source\"フォルダ
- データ・フラッシュ[DF]のメイン処理："sample\RL78_G15\DF\[コンパイラ名]\source\"フォルダ

表 5-4 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 (for code flash)	コード・フラッシュ制御用メイン処理関数サンプル・ファイル
2	main.c (for data flash)	データ・フラッシュ制御用メイン処理関数サンプル・ファイル

5.1.2.2 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 5-5 に示します。

表 5-5 "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_common.h	共通フラッシュ・メモリ制御用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_control_data_flash.h	データ・フラッシュ・メモリ制御用関数サンプルのプロトタイプ宣言を定義したファイル
3	sample_control_code_flash.h	コード・フラッシュ・メモリ制御用関数サンプルのプロトタイプ宣言を定義したファイル
4	sample_defines.h	フラッシュ・メモリ制御用関数サンプルのマクロを定義したファイル
5	sample_types.h	サンプル・プログラム用列挙型返り値を定義したファイル

5.2 データ型定義

5.2.1 列挙型

- e_sample_ret (列挙変数名 : e_sample_ret_t)

フラッシュ・メモリ・シーケンサ実行結果(正常/エラー)と実行後ステータスを表 5-6 に示します。

表 5-6 フラッシュ・メモリ・シーケンサ実行結果(正常/エラー)と実行後ステータス

Symbol Name	Value	Description
SAMPLE_ENUM_RET_STS_OK	0x00u	ステータス(正常終了)
SAMPLE_ENUM_RET_ERR_PARAMETER	0x10u	パラメータ・エラー
SAMPLE_ENUM_RET_ERR_CONFIGURATION	0x11u	初期設定エラー
SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED	0x12u	モード不一致エラー
SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA	0x13u	書き込みデータ確認エラー
SAMPLE_ENUM_RET_ERR_CFDF_SEQUENCER	0x20u	フラッシュ・メモリ・シーケンサ・エラー
SAMPLE_ENUM_RET_ERR_ACT_ERASE	0x22u	消去動作エラー
SAMPLE_ENUM_RET_ERR_ACT_WRITE	0x23u	書き込み動作エラー
SAMPLE_ENUM_RET_ERR_CMD_ERASE	0x30u	消去コマンド・エラー
SAMPLE_ENUM_RET_ERR_CMD_WRITE	0x31u	書き込みコマンド・エラー

5.3 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 5-7 に示します。

表 5-7 サンプル・プログラム関数一覧

	API Name	Outline
1	main (for code flash)	コード・フラッシュ書き換え制御サンプル・プログラムのメイン処理
2	Sample_CodeFlashControl	コード・フラッシュ・メモリの書き換え処理を実行
3	main (for data flash)	データ・フラッシュ書き換え制御サンプル・プログラムのメイン処理
4	Sample_DataFlashControl	データ・フラッシュ・メモリの書き換え処理を実行
5	Sample_CheckCFDFSeqEnd	フラッシュ・メモリ・シーケンサ・コマンドの完了待ち処理

5.3.1 コード・フラッシュ書き換え制御サンプル・プログラム

RFSP Type01 のコード・フラッシュ書き換え制御サンプルでは、コード・フラッシュ領域のブロック 3(0x0C00)を消去し、ブロック 3 の先頭から 16 ワード(64byte)のデータを書き込みます。

動作条件：
- CPU 動作周波数:16MHz(メイン・システム・クロックに高速オンチップ・オシレータ・クロックを使用)
- コード・フラッシュ消去/書き込みアドレス:0x0C00
- 消去ブロック No.:0x03
- 書き込みデータ・サイズ:16 ワード(64byte)

RFSP Type01 のコード・フラッシュ書き換え制御サンプルのメイン処理実行フローを図 5-2 に示します。

5.3.1.1 main 関数

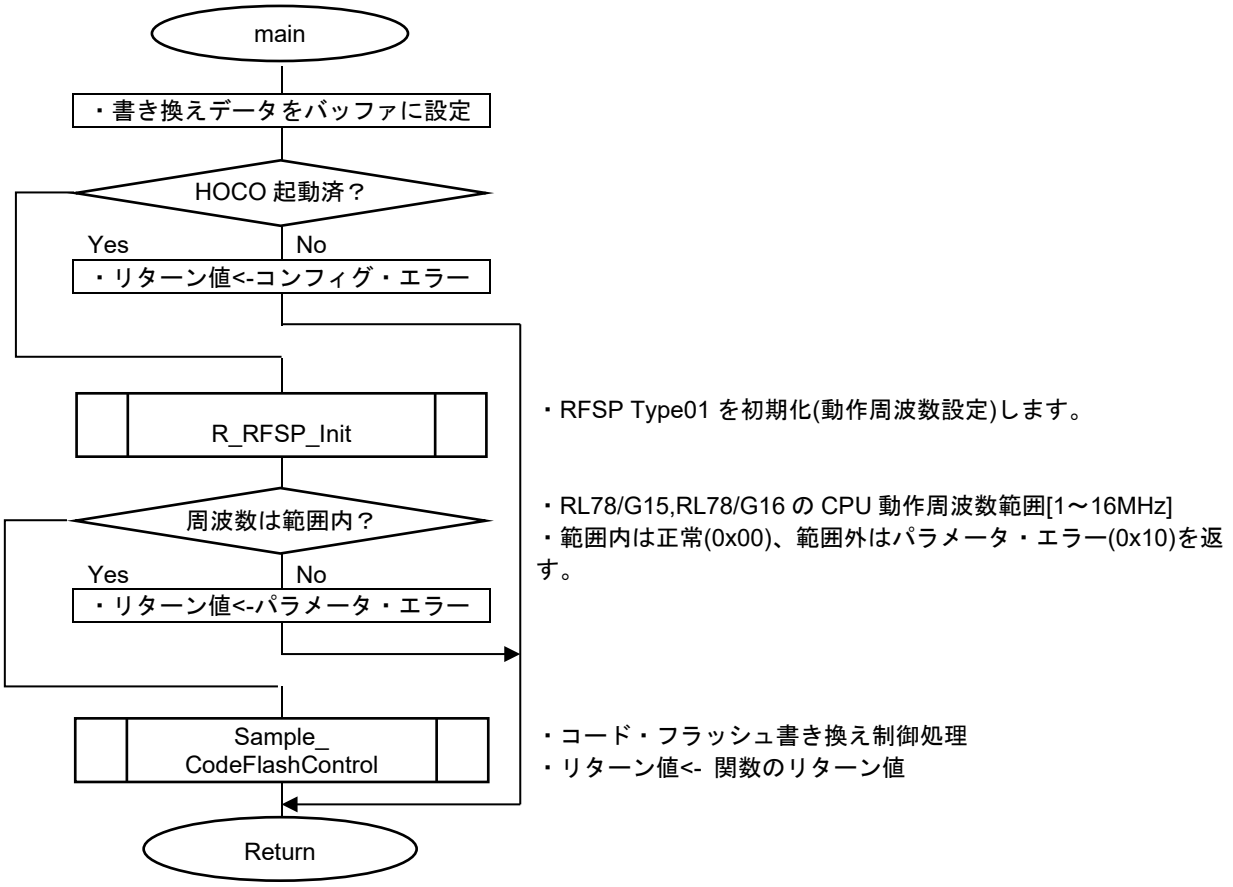


図 5-2 コード・フラッシュ書き換え制御サンプルのメイン処理実行フロー

5.3.1.2 Sample_CodeFlashControl 関数

・セルフ・プログラミング・モード(コード・フラッシュ領域選択)へ移行、ブロック消去を実行

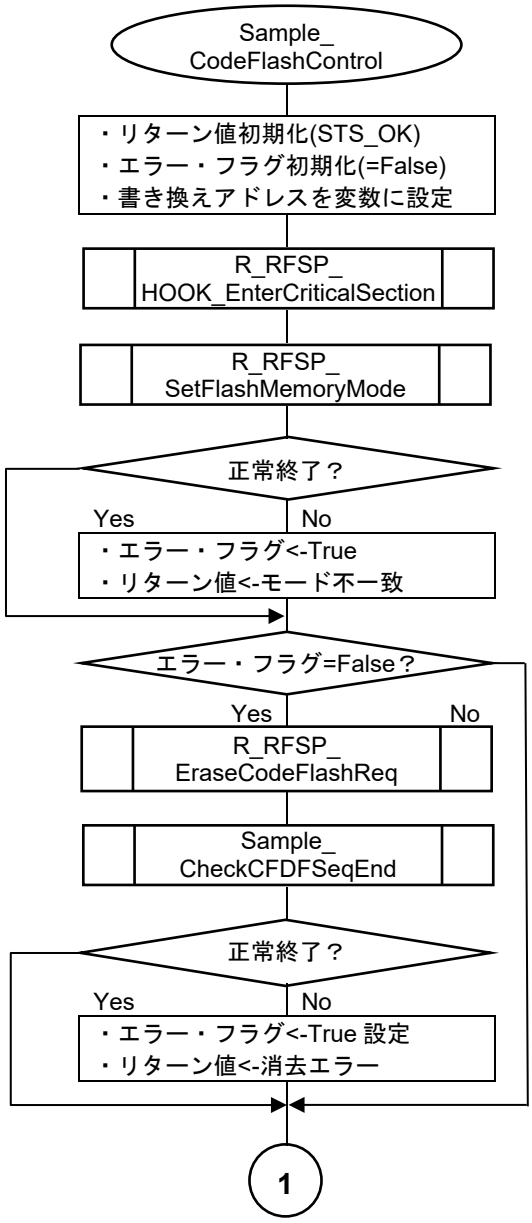


図 5-3 コード・フラッシュ書き換え制御サンプルの処理実行フロー(1/3)

・書き込みを実行

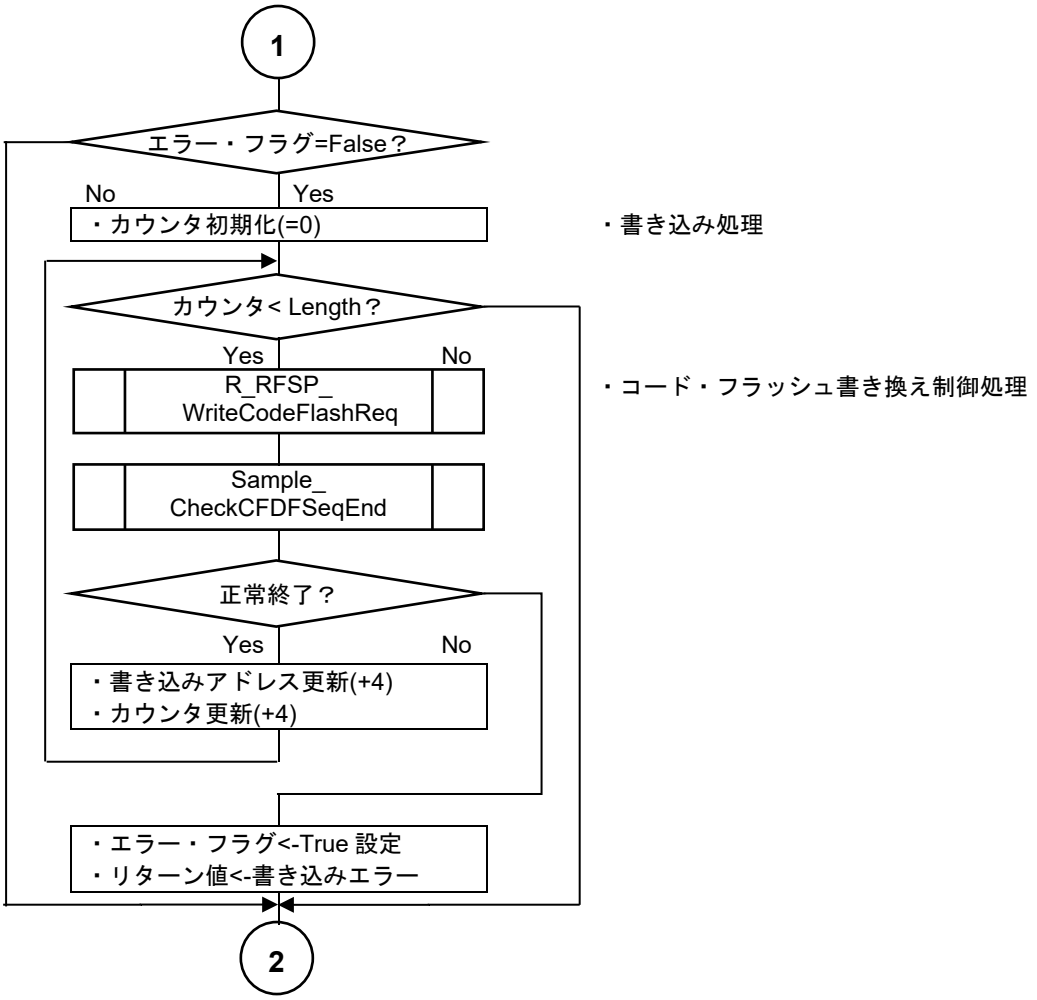
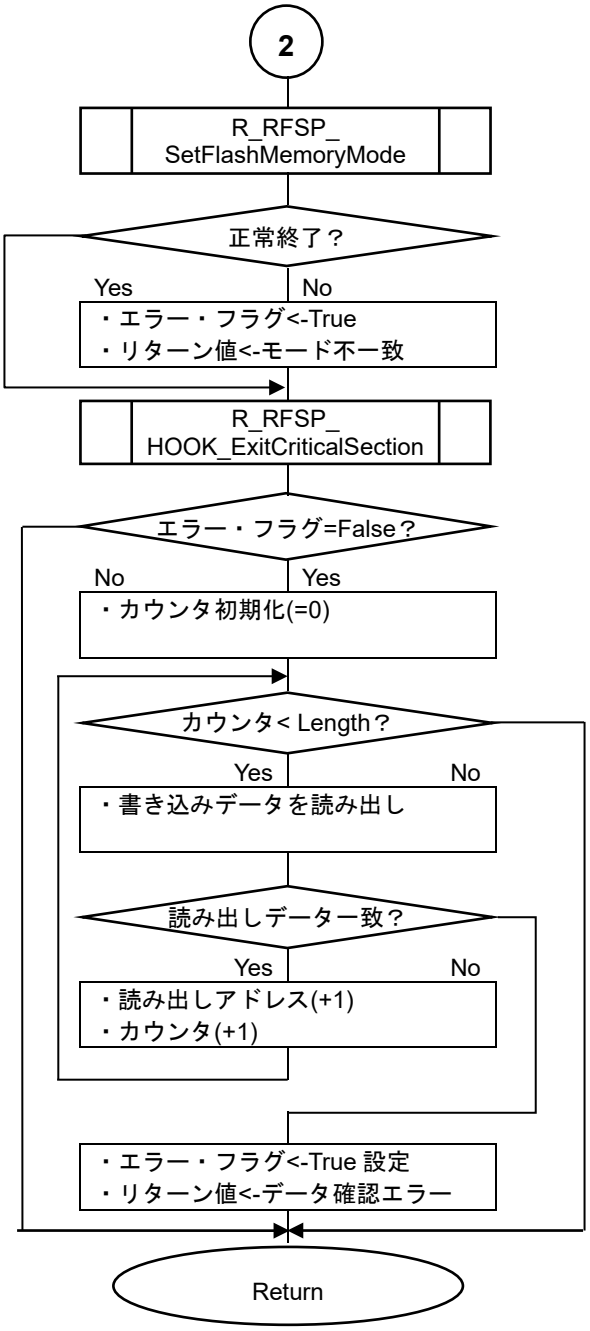


図 5-4 コード・フラッシュ書き換え制御サンプルの処理実行フロー(2/3)

・非書き換えモードへ移行、CPU 読み出しによるベリファイ・チェックを実行



・非書き換えモードを設定します。

・モード設定が正常(0x00)、モードが設定値と不一致(0x11)

・割り込みマスクを解除します。

・CPU 読み出しによるベリファイ・チェック

図 5-5 コード・フラッシュ書き換え制御サンプルの処理実行フロー(3/3)

5.3.2 データ・フラッシュ書き換え制御サンプル・プログラム

RFSP Type01 のデータ・フラッシュ書き換え制御サンプルでは、データ・フラッシュ領域のブロック 0(0x9000)を消去し、ブロック 0 の先頭から 16 ワード(64byte)のデータを書き込みます。

動作条件：
- CPU 動作周波数:16MHz(メイン・システム・クロックに高速オンチップ・オシレータ・クロックを使用)
- データ・フラッシュ消去/書き込みアドレス:0x9000
- 消去ブロック No.:0x00
- 書き込みデータ・サイズ:16 ワード(64byte)

RFSP Type01 のデータ・フラッシュ書き換え制御サンプルのメイン処理実行フローを図 5-6 データ・フラッシュ書き換え制御サンプルのメイン処理実行フローに示します。

5.3.2.1 main 関数

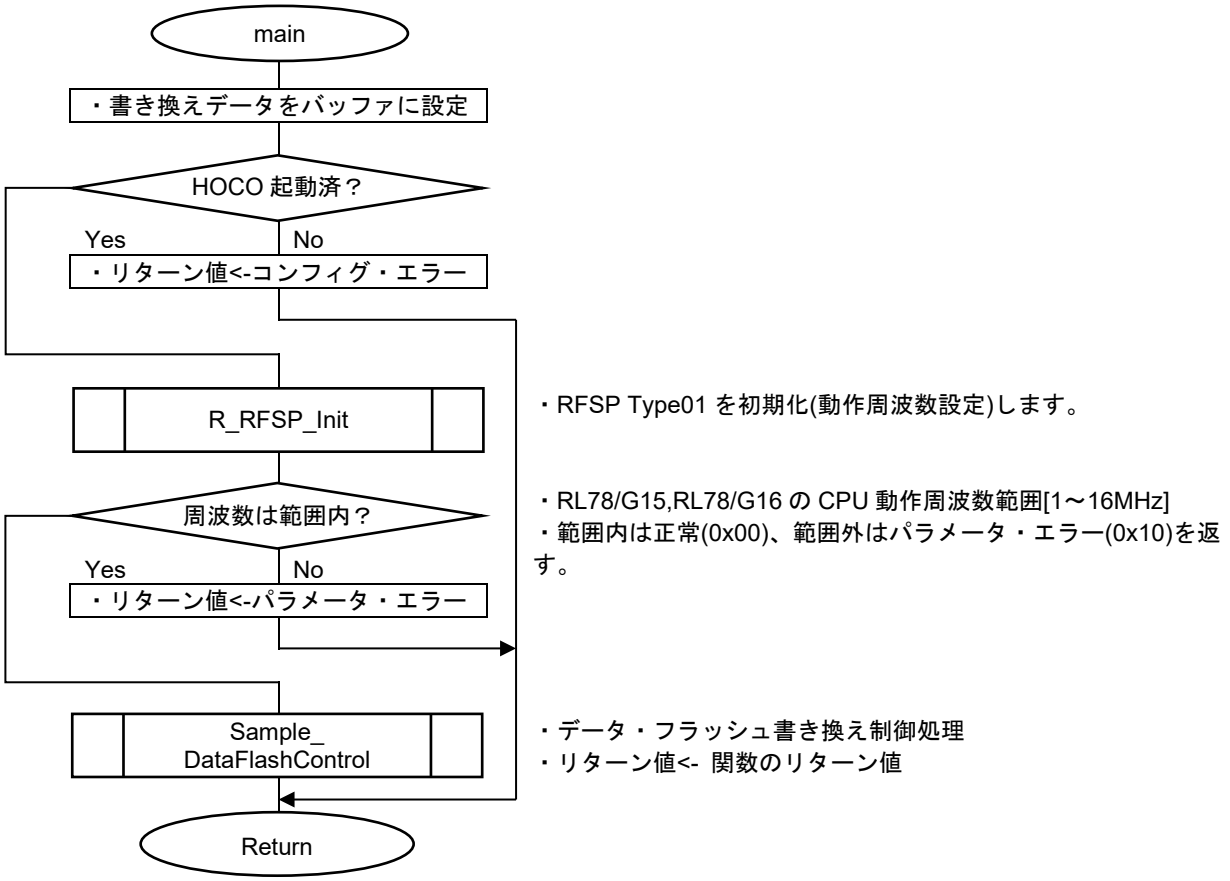
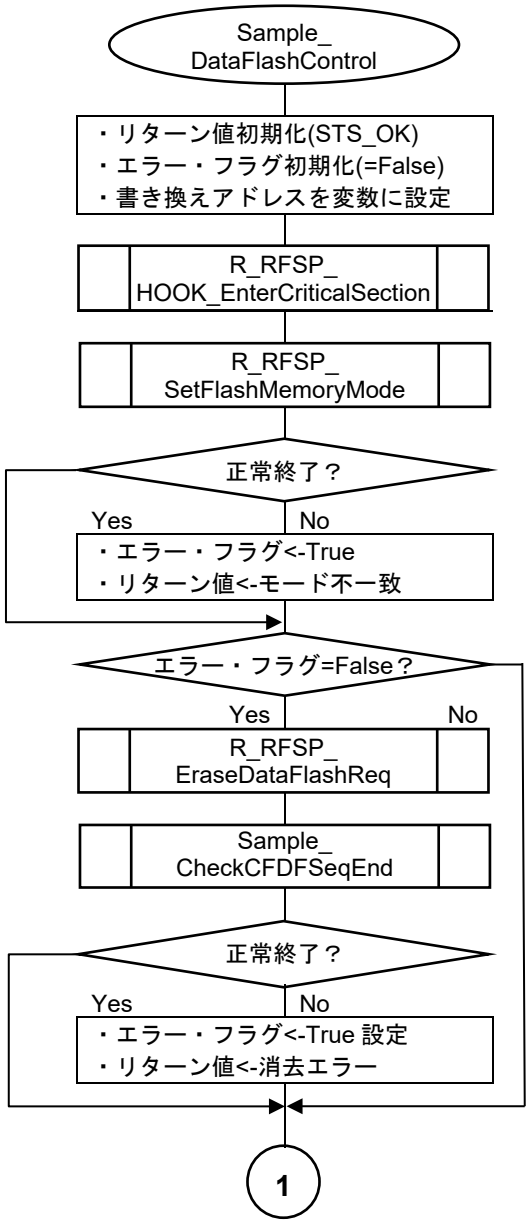


図 5-6 データ・フラッシュ書き換え制御サンプルのメイン処理実行フロー

5.3.2.2 Sample_DataFlashControl 関数

・セルフ・プログラミング・モード(データ・フラッシュ領域選択)へ移行、ブロック消去を実行



・割り込み要因をクリアし、割り込みをマスクします。

・セルフ・プログラミング・モードを設定します。
書き換える領域(データ・フラッシュ領域)を指定します。

・モード設定が正常(0x00)、モードが設定値と不一致(0x11)

・消去処理

図 5-7 データ・フラッシュ書き換え制御サンプルの処理実行フロー(1/3)

・書き込みを実行

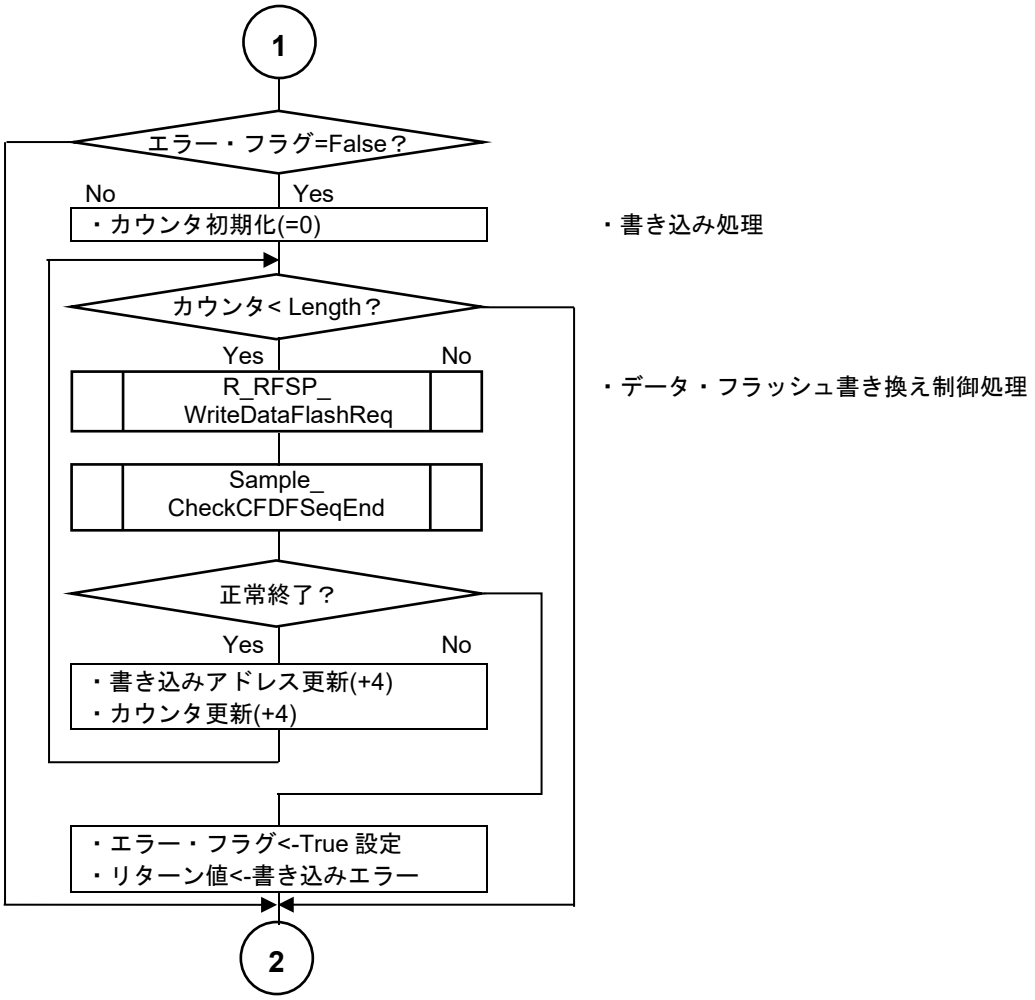
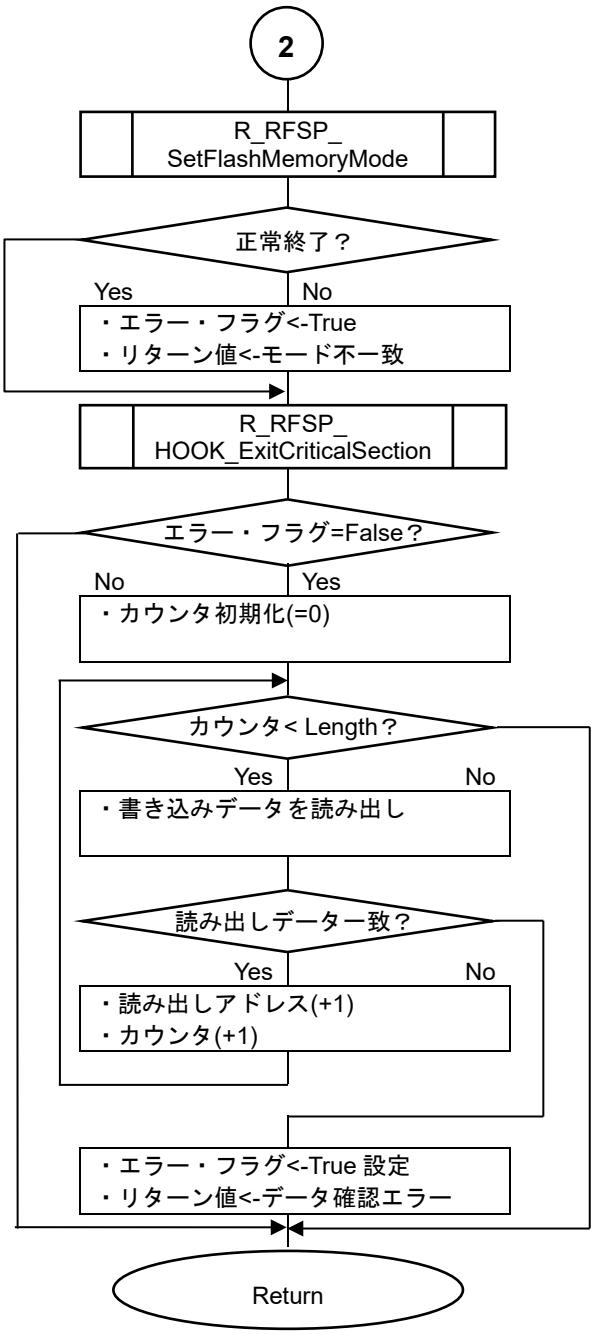


図 5-8 データ・フラッシュ書き換え制御サンプルの処理実行フロー(2/3)

・非書き換えモードへ移行、CPU 読み出しによるベリファイ・チェックを実行



・非書き換えモードを設定します。

・モード設定が正常(0x00)、モードが設定値と不一致(0x11)

・割り込みマスクを解除します。

・CPU 読み出しによるベリファイ・チェック

図 5-9 データ・フラッシュ書き換え制御サンプルの処理実行フロー(3/3)

5.3.3 共通サンプル・プログラムの関数仕様

5.3.3.1 Sample_CheckCFDFSeqEnd 関数

・起動したフラッシュ・メモリ・シーケンサの動作終了を確認し、実行結果を返します。

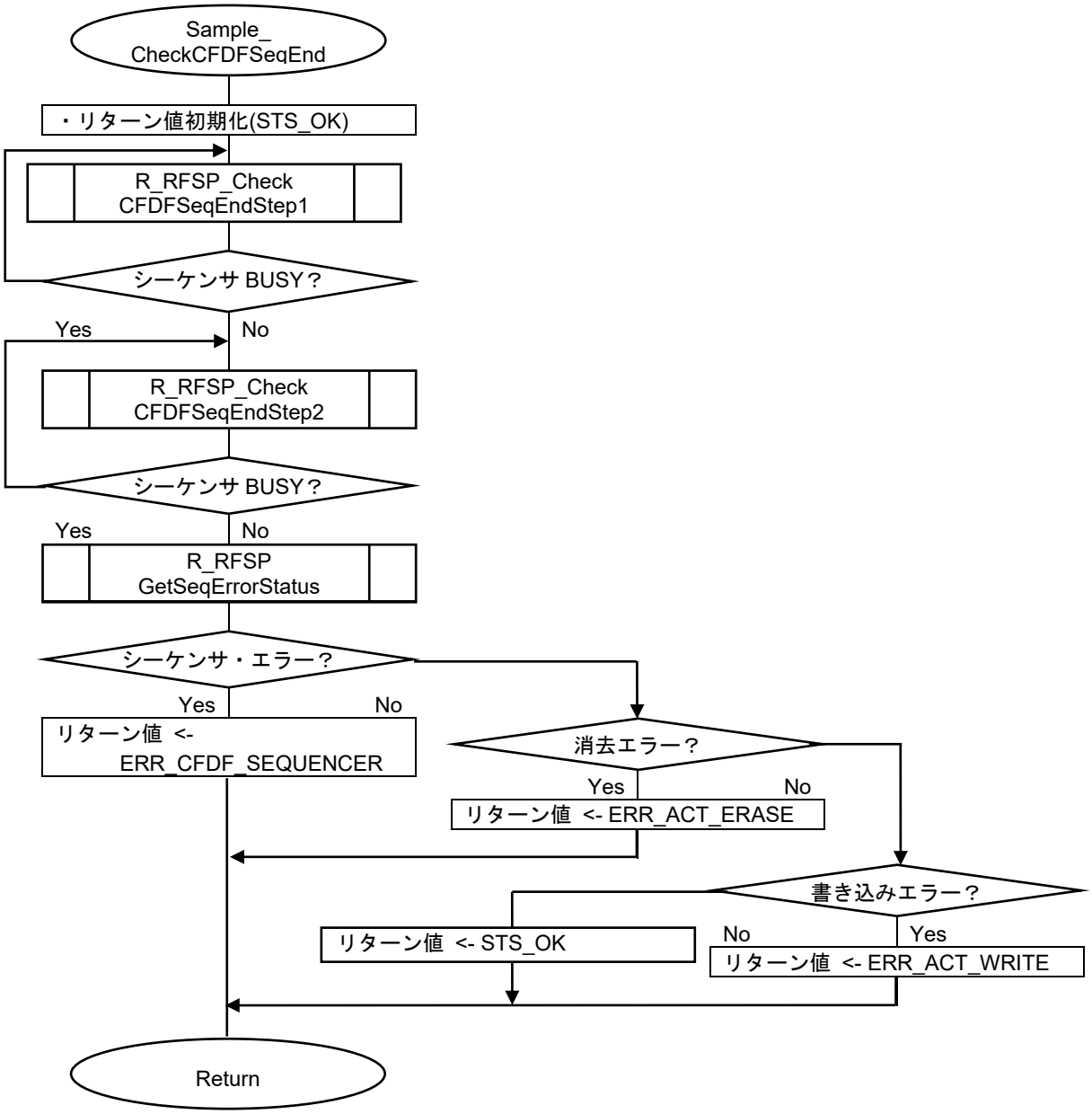


図 5-10 Sample_CheckCFDFSeqEnd 関数の処理実行フロー

5.4 サンプル・プログラム関数仕様

この章では、RFSP Type01 のサンプル・プログラム関数仕様について説明します。

RFSP Type01 のサンプル・プログラムは、コード・フラッシュ領域、およびデータ・フラッシュ領域を書き換える基本な処理例を示しています。各フラッシュ領域を書き換えるアプリケーションを開発する上で、各サンプル・プログラム関数を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。

5.4.1 コード・フラッシュ書き換え制御サンプル・プログラム

5.4.1.1 main

Information

Syntax	Int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [正常終了] SAMPLE_ENUM_RET_ERR_PARAMETER : 0x10 [パラメータ・エラー] SAMPLE_ENUM_RET_ERR_CONFIGURATION : 0x11 [初期設定エラー] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [モード不一致エラー] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [消去コマンド・エラー] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [書き込みコマンド・エラー] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [書き込みデータ確認エラー]
Description	コード・フラッシュ書き換え制御サンプル・プログラムのメイン処理	
Preconditions	非書き換えモードで実行してください。 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	-	

5.4.1.2 Sample_CodeFlashControl

Information

Syntax	R_RFSP_FAR_FUNC e_sample_ret_t Sample_CodeFlashControl (uint16_t i_u16_start_addrr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	書き換え領域の先頭アドレス
	uint16_t i_u16_write_data_length	書き換えデータ・サイズ
	uint8_t __near * inp_u08_write_data	書き換えデータ・バッファのポインタ
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [正常終了] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [モード不一致エラー] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [消去コマンド・エラー] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [書き込みコマンド・エラー] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [書き込みデータ確認エラー]
Description	コード・フラッシュ・メモリの書き換え処理を実行 - セルフ・プログラミング・モード(コード・フラッシュ領域選択)で、消去、書き込みの各コマンドを実行します。 - 非プログラマブル・モードで書き込まれたデータを読み出し、正しく書かれたかを確認します。	
Preconditions	非書き換えモードで実行してください。 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	-	

5.4.2 データ・フラッシュ書き換え制御サンプル・プログラム関数仕様

5.4.2.1 main

Information

Syntax	Int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [正常終了] SAMPLE_ENUM_RET_ERR_PARAMETER : 0x10 [パラメータ・エラー] SAMPLE_ENUM_RET_ERR_CONFIGURATION : 0x11 [初期設定エラー] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [モード不一致エラー] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [消去コマンド・エラー] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [書き込みコマンド・エラー] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [書き込みデータ確認エラー]
Description	データ・フラッシュ書き換え制御サンプル・プログラムのメイン処理	
Preconditions	非書き換えモードで実行してください。 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	-	

5.4.2.2 Sample_DataFlashControl

Information

Syntax	R_RFSP_FAR_FUNC e_sample_ret_t Sample_DataFlashControl (uint16_t i_u16_start_addrr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	書き換え領域の先頭アドレス
	uint16_t i_u16_write_data_length	書き換えデータ・サイズ
	uint8_t __near * inp_u08_write_data	書き換えデータ・バッファのポインタ
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [正常終了] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [モード不一致エラー] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [消去コマンド・エラー] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [書き込みコマンド・エラー] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [書き込みデータ確認エラー]
Description	データ・フラッシュ・メモリの書き換え処理を実行 - セルフ・プログラミング・モード(データ・フラッシュ領域選択)で、消去、書き込みの各コマンドを実行します。 - 非プログラマブル・モードで書き込まれたデータを読み出し、正しく書かれたかを確認します。	
Preconditions	非書き換えモードで実行してください。 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	-	

5.4.3 共通サンプル・プログラムの関数仕様

5.4.3.1 Sample_CheckCFDFSeqEnd

Information

Syntax	R_RFSP_FAR_FUNC e_sample_ret_t Sample_CheckCFDFSeqEnd(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [正常終了] SAMPLE_ENUM_RET_ERR_CFDF_SEQUENCER : 0x20 [フラッシュ・メモリ・シーケンサ・エラー] SAMPLE_ENUM_RET_ERR_ACT_ERASE : 0x22 [消去動作エラー] SAMPLE_ENUM_RET_ERR_ACT_WRITE : 0x23 [書き込み動作エラー]
Description	フラッシュ・メモリ・シーケンサ・コマンドの完了待ち処理	
Preconditions	セルフ・プログラミング・モード(コード・フラッシュ領域選択)、または、セルフ・プログラミング・モード(データ・フラッシュ領域選択)で使用してください。 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	-	

6 RFSP Type01 サンプル・プロジェクトの作成

RFSP Type01 は、コード・フラッシュ・メモリ領域とデータ・フラッシュ・メモリ領域の書き換えサンプル・プログラムが含まれます。RFSP Type01 で使用できるコンパイラは、CC-RL コンパイラ、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。

この項では、RL78/G15 用のサンプル・プログラムを例に説明しています。RL78/G15 以外を使用する場合は、G15 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。

注意　対象の統合開発環境、およびコンパイラは、RL78/G15,RL78/G16 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/G15,RL78/G16 が対象製品であることをご確認の上、ご使用ください。

6.1 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 RFSP 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。

6.1.1 サンプル・プロジェクトの作成例

（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

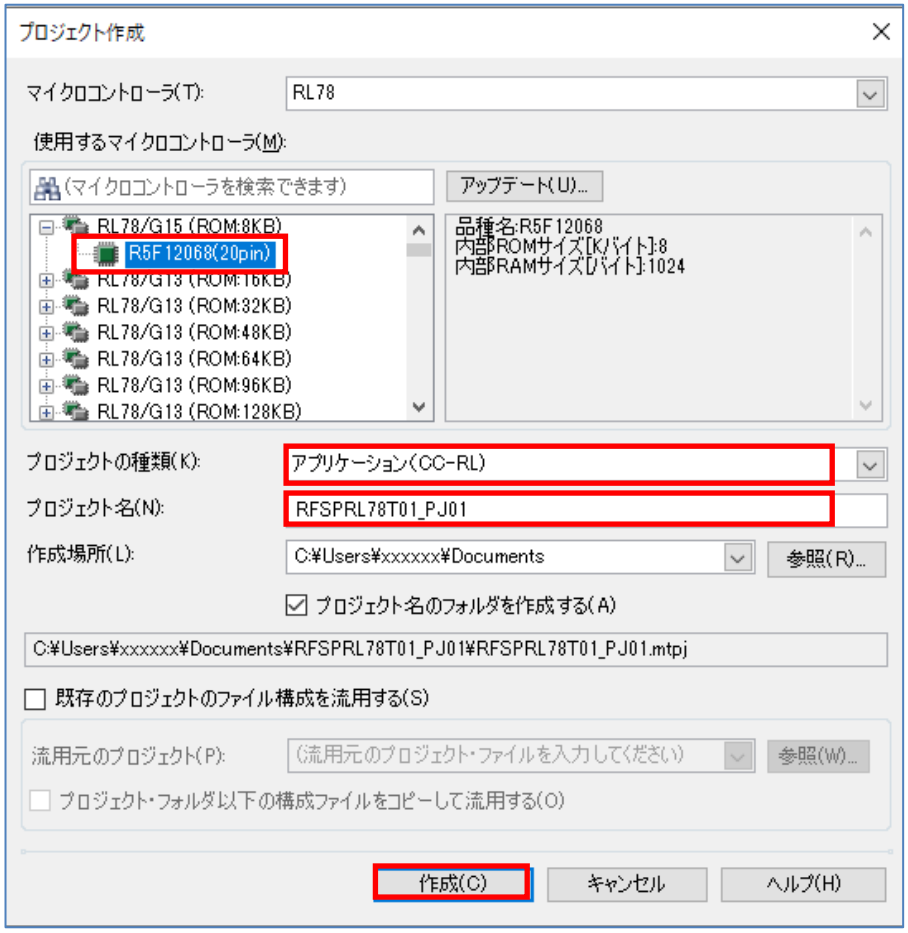
CS+を起動し、[プロジェクト]メニューの[新しいプロジェクトを作成]を選択し、以下に示す "プロジェクト作成"ウインドウを起動します。

・[使用するマイクロコントローラ]は、"RL78/G15 (ROM: 8KB)" - "R5F12068(20pin)"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。

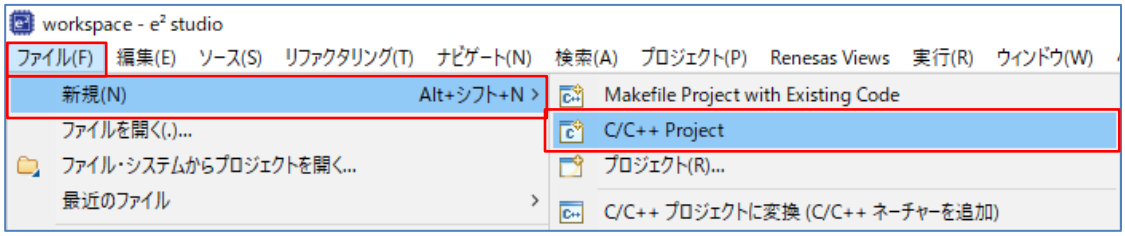
・ここでの[プロジェクト名]は、仮に"RFSPRL78T01_PJ01"とします。

・[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。

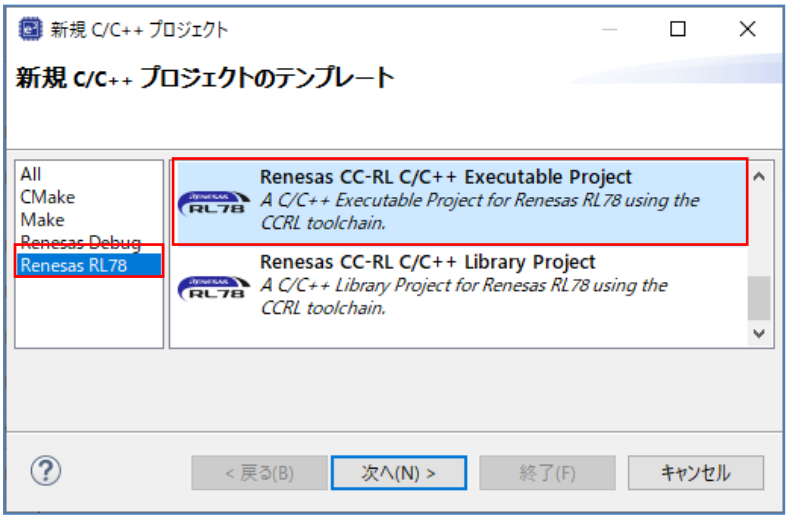


（２）統合開発環境 e² studio を使用したサンプル・プロジェクト作成例

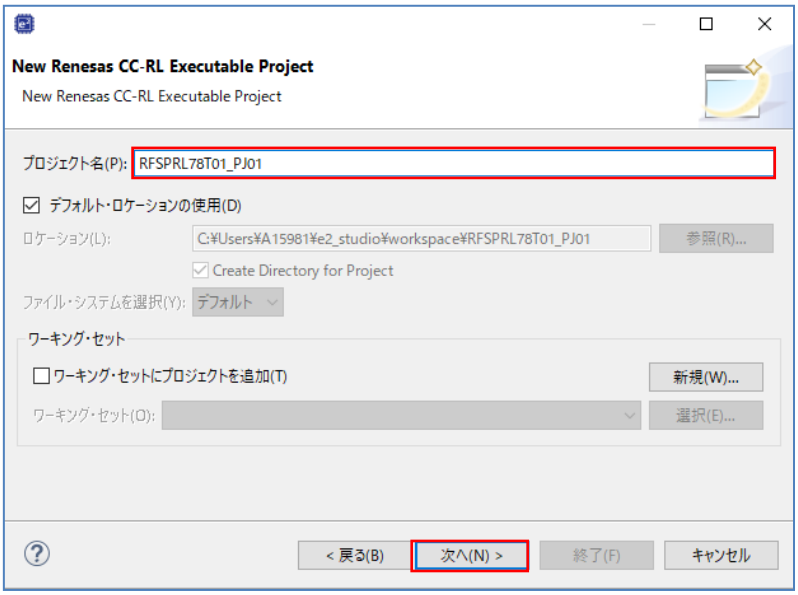
e² studio を起動し、[ファイル]メニューの[新規]から[C/C++ Project]を選択し、"新規 C/C++ プロジェクトのテンプレート"ウインドウを起動します。



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



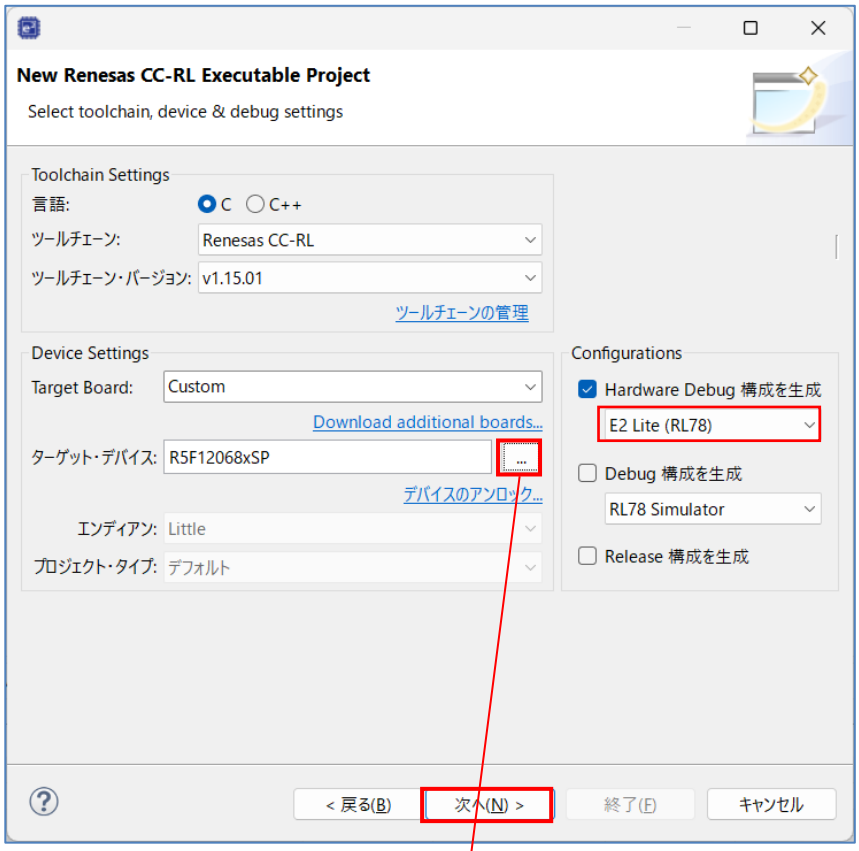
・"New Renesas CC-RL Executable Project"ウインドウで、プロジェクト名を入力して"次へ"ボタンを押します。（ここでは、仮に"RFSPRL78T01_PJ01"とします。）

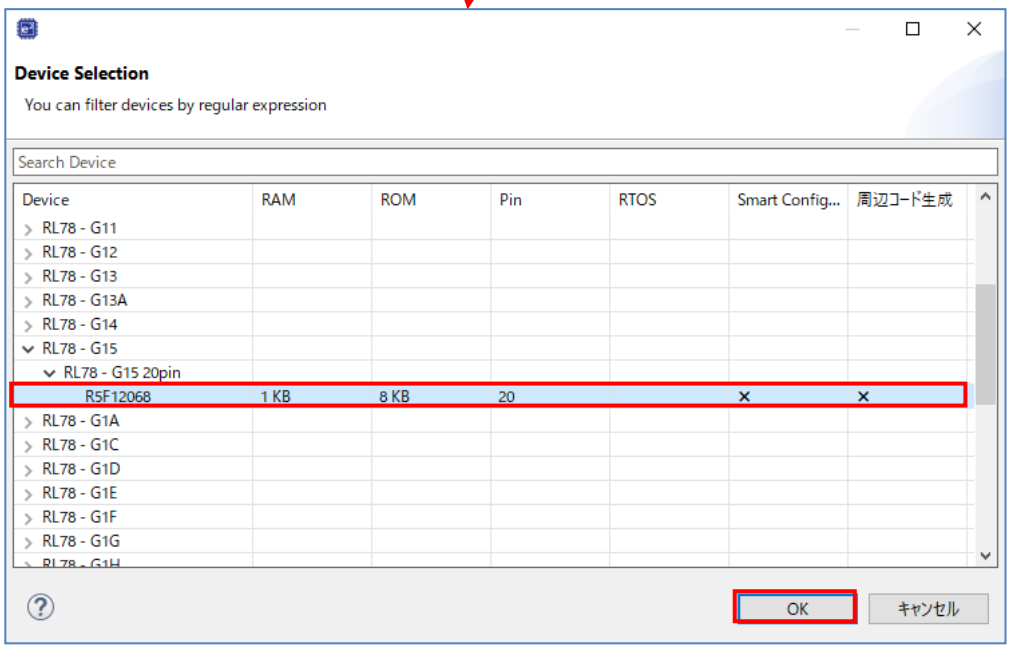


・[Device Settings]の[ターゲット・デバイスで、"RL78 - G15 20pin - R5F12068"を選択します。

・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。

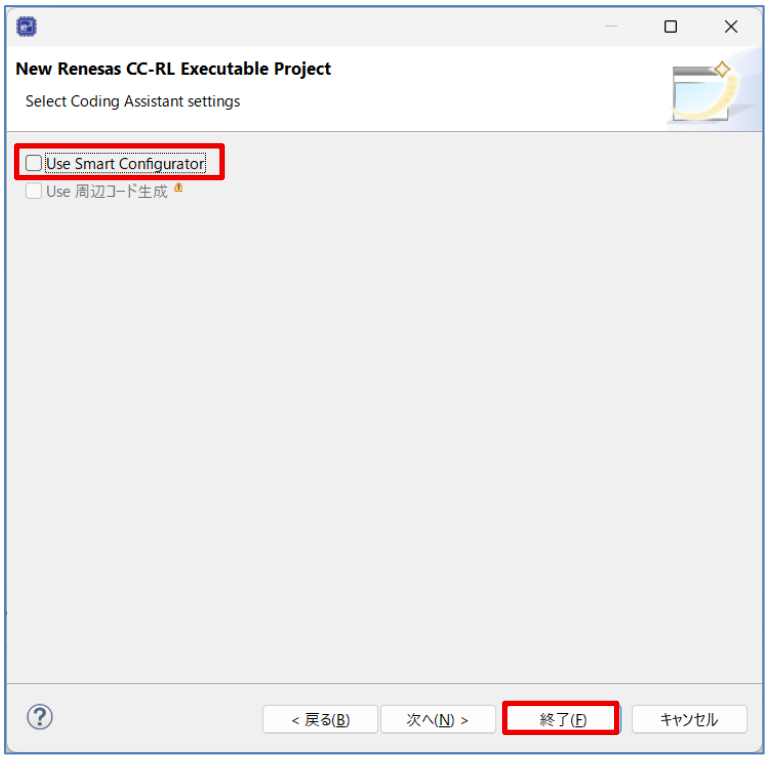
・[次へ]ボタンを押します。





・"Use Smart Configurator"のチェックを外します。

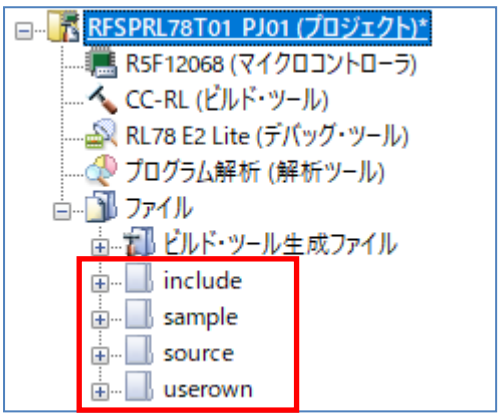
・[終了]ボタンを押します。



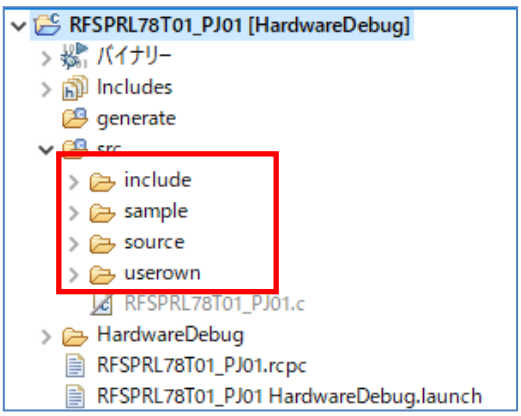
6.1.2 対象フォルダと対象ファイルの登録例

RFSP Type01 を使用して、各領域(1)コード・フラッシュ・メモリ、(2)データ・フラッシュ・メモリを書き換える場合に必要なファイルの登録例を記述します。 RFSP Type01 ソースプログラムファイルの各フォルダは、"include", "source", "userown", "sample"で、書き換える領域により各フォルダ内の対象ファイルを選択して登録します。

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 RFSP Type01 登録時のツリー画面



e^2 studio RFSP Type01 登録時のツリー画面

・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

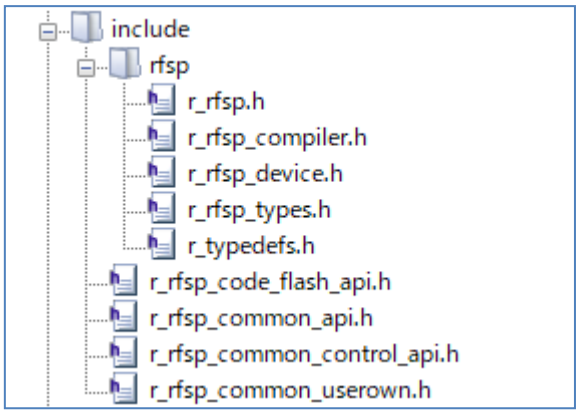
作成されたプロジェクトには、自動的に追加されるファイルがあります。これらと同様のファイルは、 RFSP Type01 の"sample"フォルダ内にも存在するため、ツリーで各ファイルを選択し、各統合開発環境の機能を使用して、プロジェクトから外します。

- 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 hdwinit.asm, main.c 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 hdwinit.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"RFSPRL78T01_PJ01.c")が対象。

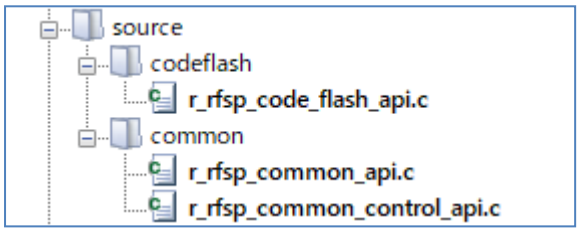
(1) コード・フラッシュ・メモリを書き換える場合の対象フォルダと対象ファイルの登録

RFSP 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

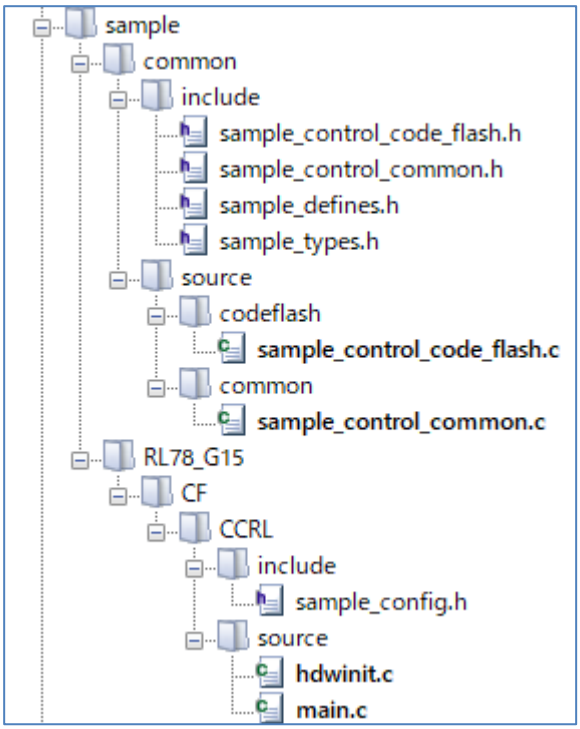


source フォルダ内



userown フォルダ内

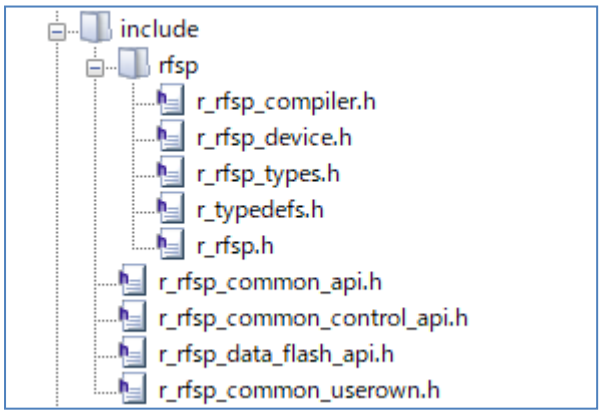
sample フォルダ内



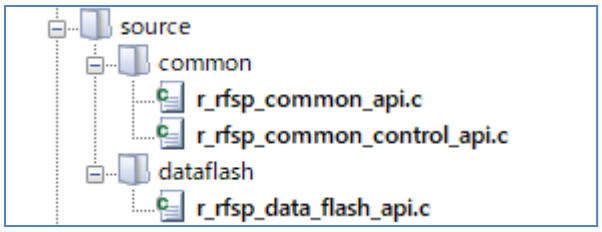
(2) データ・フラッシュ・メモリを書き換える場合の対象フォルダと対象ファイルの登録

RFSP 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

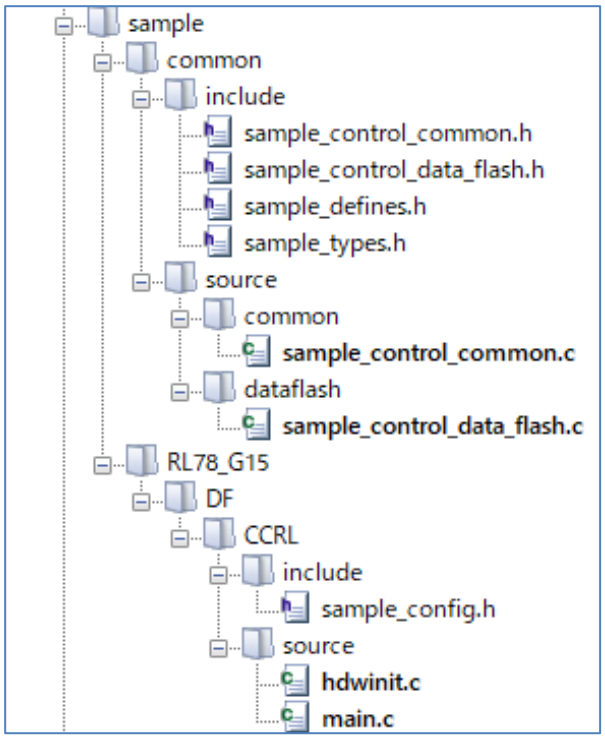


source フォルダ内



userown フォルダ内

sample フォルダ内



6.1.3 ビルド・ツールの設定

CC-RL コンパイラで RFSP 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、e2 studio ではツリーでプロジェクト(ここでは"RFSPRL78T01_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。

6.1.3.1 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(対象領域により変更)

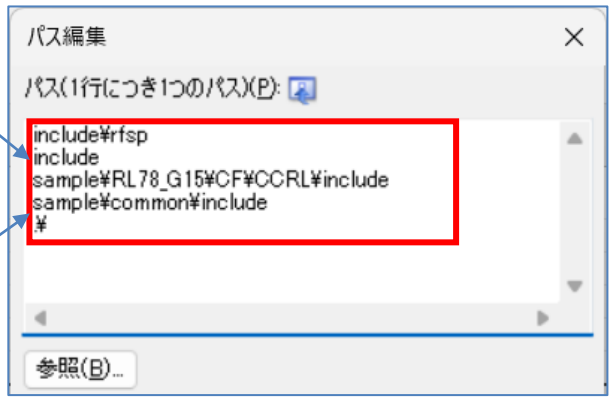
- [よく使うオプション(コンパイル)] – [追加のインクルード・パス]で"パス編集"ウインドウを表示して、インクルード・ファイルのパスを設定します。

(1)コード・フラッシュ・メモリ書き換え

```
include\rfsp
include
sample\RL78_G15\CF\CCRL\include
sample\common\include
.\
```

(2)データ・フラッシュ・メモリ書き換え

```
include\rfsp
include
sample\RL78_G15\DF\CCRL\include
sample\common\include
.\
```



・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(対象領域により変更)

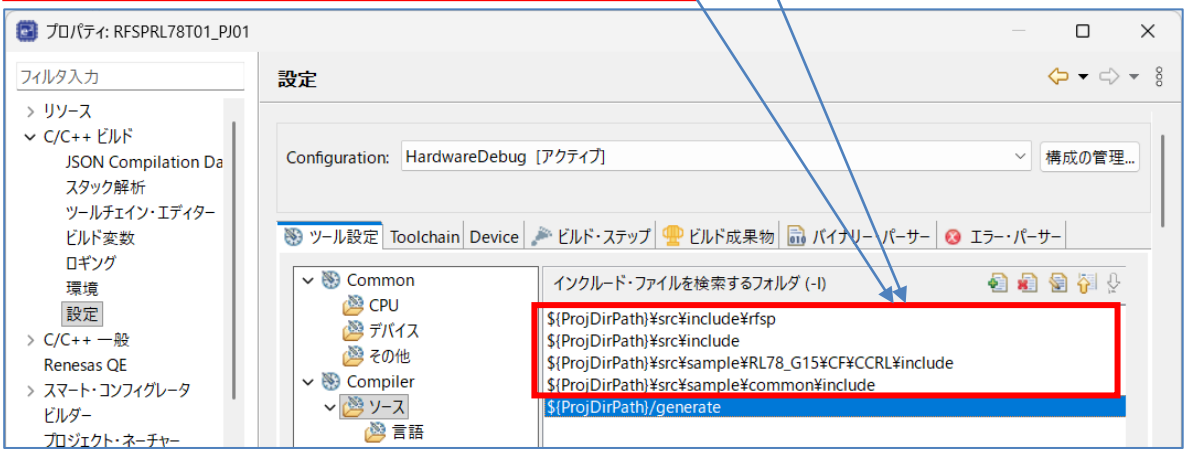
- "C/C++ビルド" [設定] – "Compiler" [ソース] で表示した画面でインクルード・ファイルのパスを追加します。

(1)コード・フラッシュ・メモリ書き換え

```
${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\CF\CCRL\include
${ProjDirPath}\src\sample\common\include
```

(2)データ・フラッシュ・メモリ書き換え

```
${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\DF\CCRL\include
${ProjDirPath}\src\sample\common\include
```



6.1.3.2 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(各領域共通)

- [デバイス] 項目を設定します。

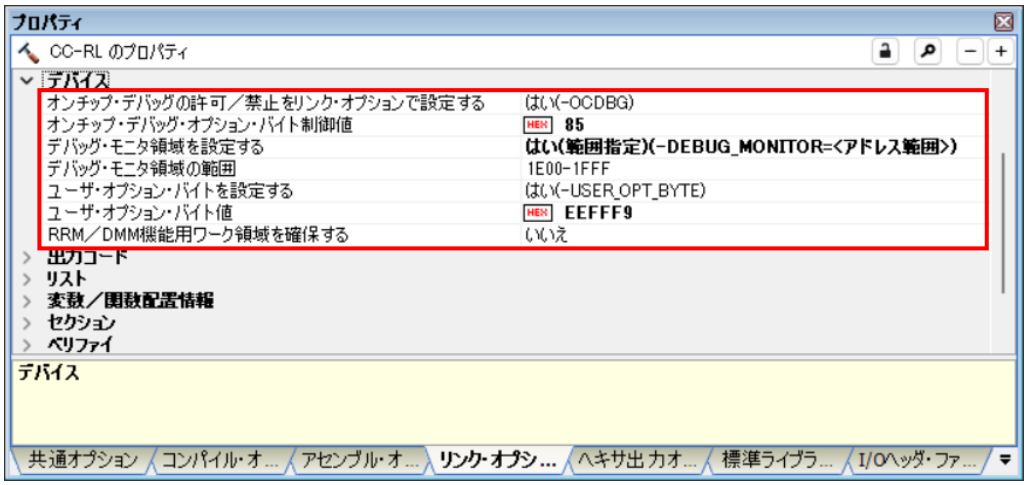
[オンチップ・デバッグの許可／禁止をリンク・オプション設定する]を"はい(-OCDBG)"に設定します。
注）オンチップ・デバッグを実施することを前提とした設定例です。

[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)(-DEBUG_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"1E00-1FFF"に設定します。[RL78/G15 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"EEFFF9"に設定します。
(WDT 停止,P125:RESET 入力, SPOR 検出電圧/2.16V/2.11V,16MHz [G15, G16 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(各領域共通)

- "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。

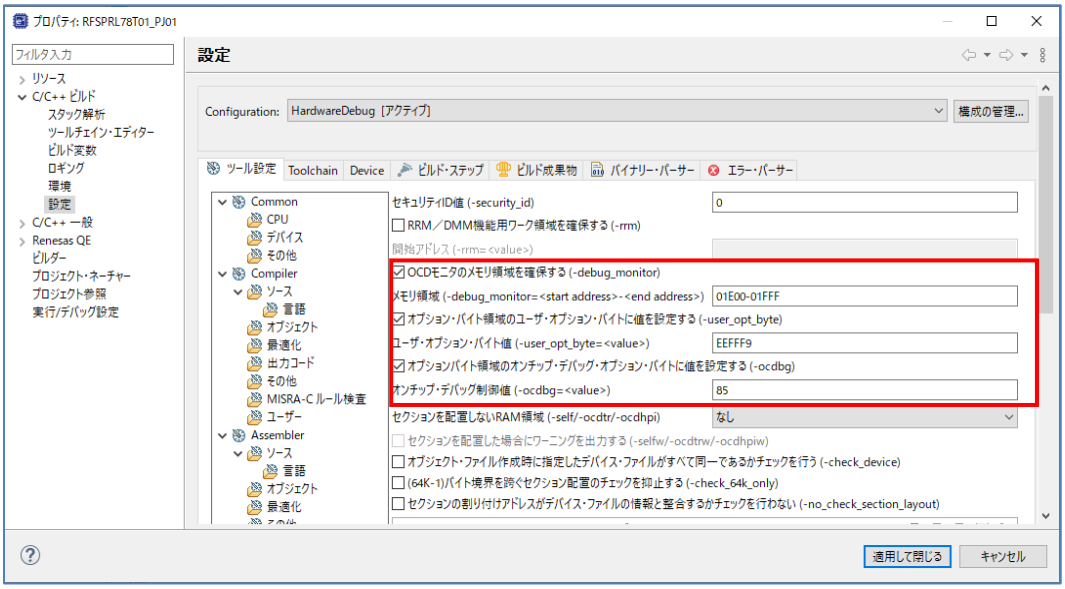
[メモリ領域(-debug_monitor=<start address>-<end address>)]を"1E00-1FFF"に設定します。[RL78/G15 の例]
注）対象デバイスのユーザーズマニュアルで「オンチップ・デバッグ機能」の章の「デバッグ用モニタ・プログラムが配置されるメモリ空間」をご確認いただき、使用する領域の範囲を書き込んでください。

[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"EEFFF9"に設定します。
(WDT 停止,P125:RESET 入力, SPOR 検出電圧/2.16V/2.11V,HS モード/16MHz [G15,G16 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。



6.1.4 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。選択後、再度マウス右クリックし、"プロパティ"を選択して"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、"RFSPRL78T01_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。

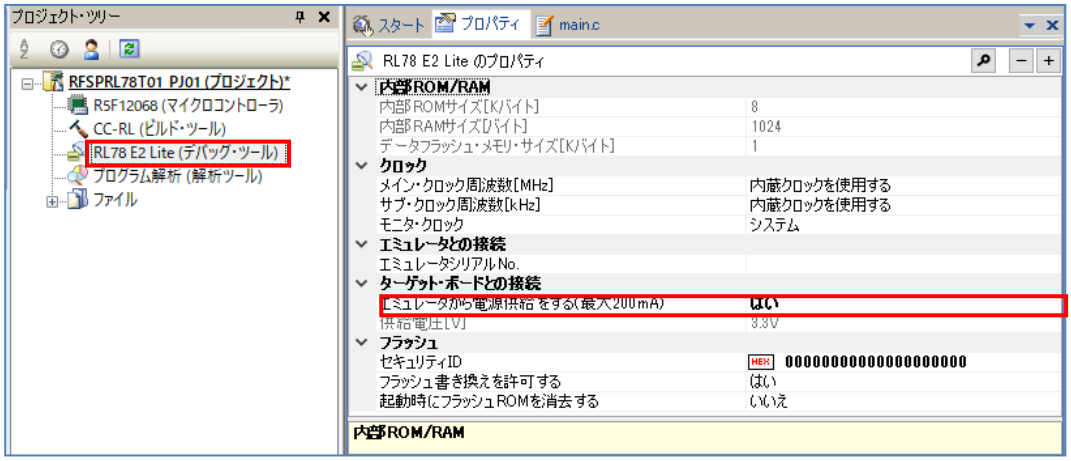
注）ターゲット・ボードに他の電源が供給されている場合や電源供給容量が不足するなど、E2 Lite を含むエミュレータからターゲット・ボードへの電源供給ができない場合があります。必ず、対象デバイス用のエミュレータのユーザーズマニュアル、およびユーザーズマニュアル別冊(RL78 接続時の注意事項)をご参照の上、ご使用ください。

6.1.4.1 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(各領域共通)

- [ターゲット・ボードとの接続] 項目

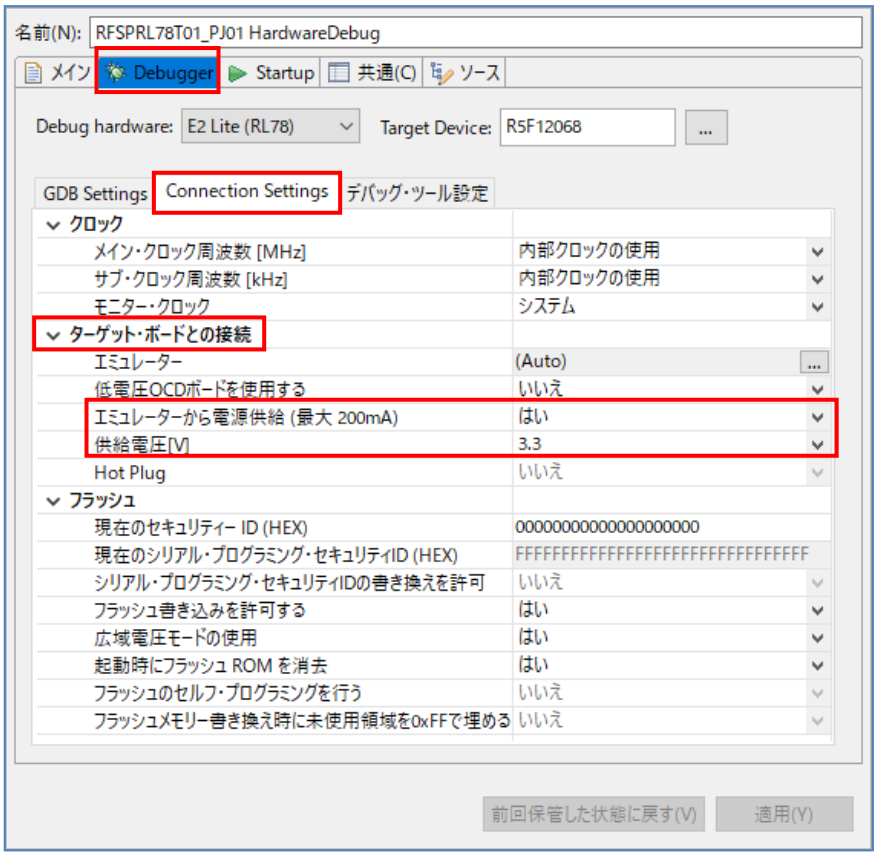
[エミュレータから電源供給をする(最大 200mA)]を"はい"に設定することで、E2 Lite からターゲット・ボードに電源供給(供給電圧:3.3V)することが可能です。



・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 Setting"タブで設定(各領域共通)

- [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。



6.2 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 RFSP 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。

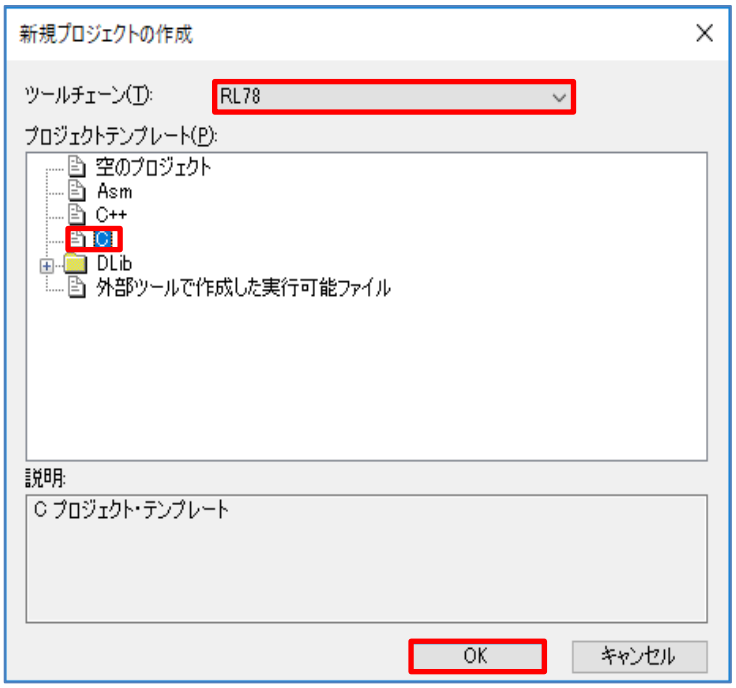
6.2.1 サンプル・プロジェクト作成例

（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。

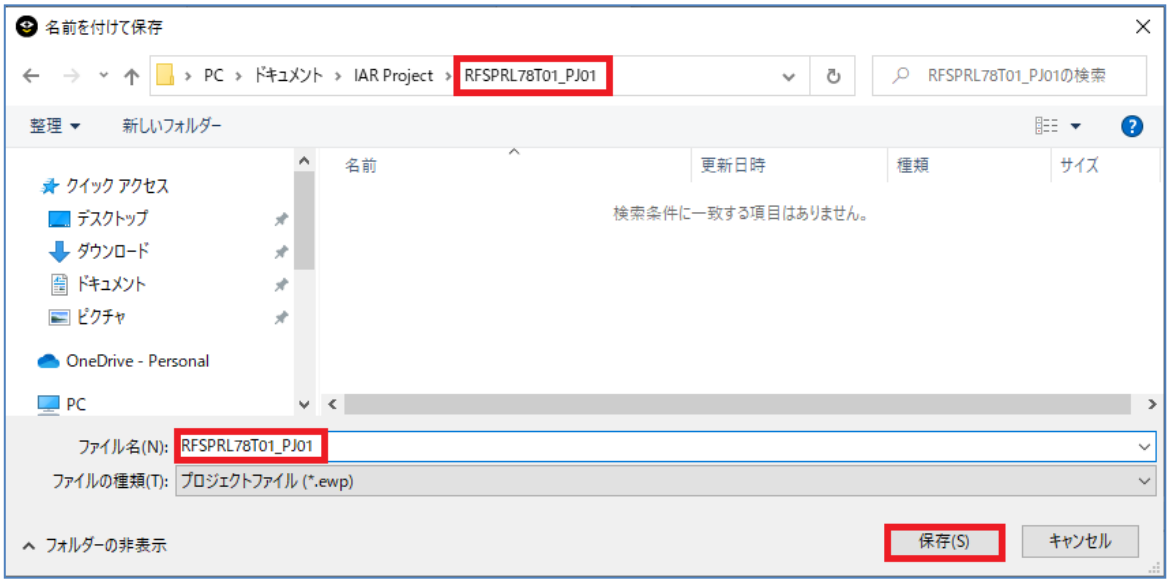
・[プロジェクトテンプレート]で、"C"を選択します。

・[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。



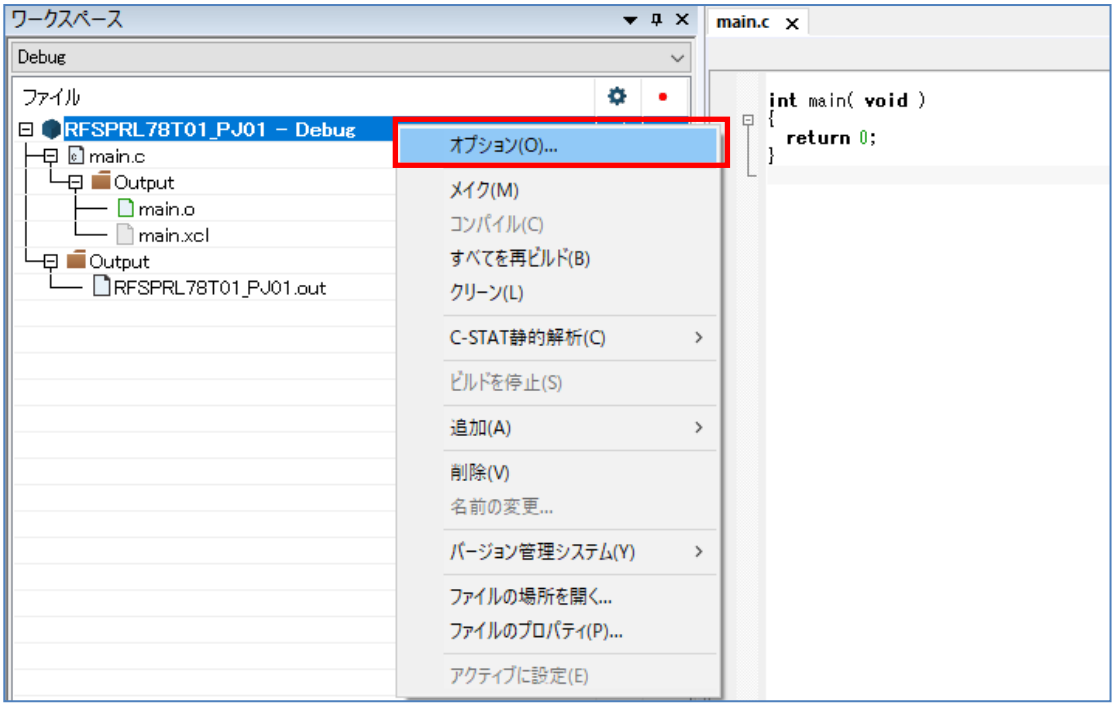
・ここでは、仮に"RFSPRL78T01_PJ01"フォルダを作成し、フォルダ内へ移動します。

・ここでの[プロジェクト名]は、仮に"RFSPRL78T01_PJ01"として保存します。



（２）対象デバイスの選択

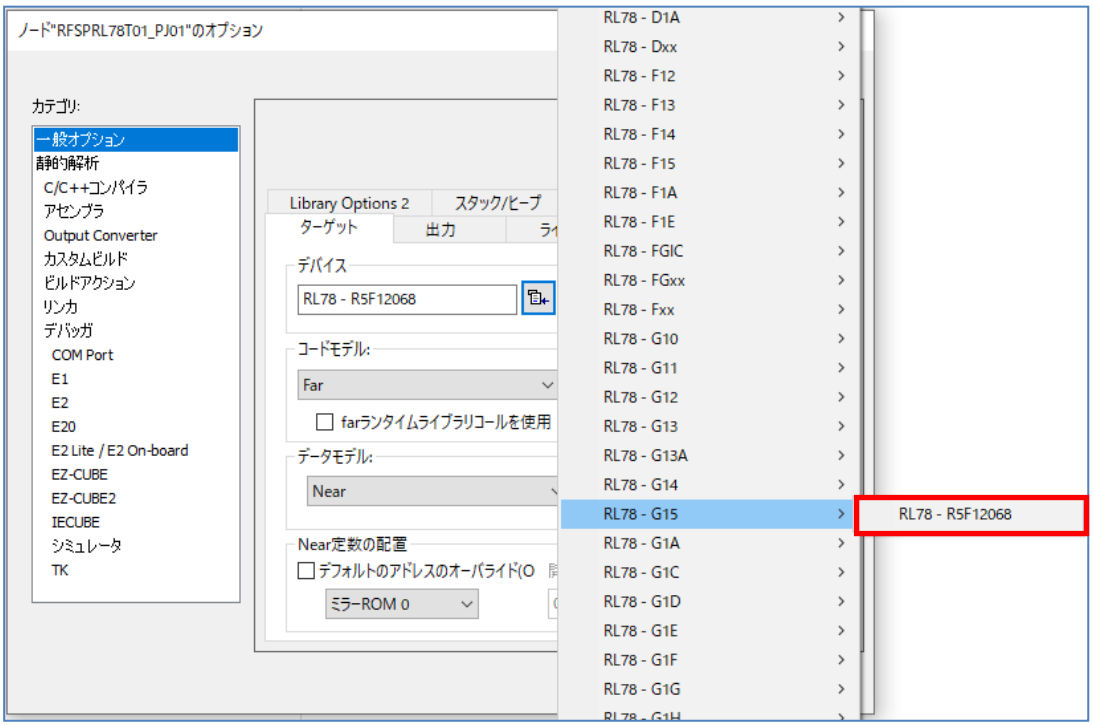
IAR Embedded Workbench ではツリーでプロジェクト(ここでは"RFSPRL78T01_PJ01 - Debug")のマウス右クリックで"オプション"を選択することにより、"オプション"の画面を表示します。



・"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"  "ボタンを押して、“RL78 – G15” - “RL78 – R5F12068”を選択します。

[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。

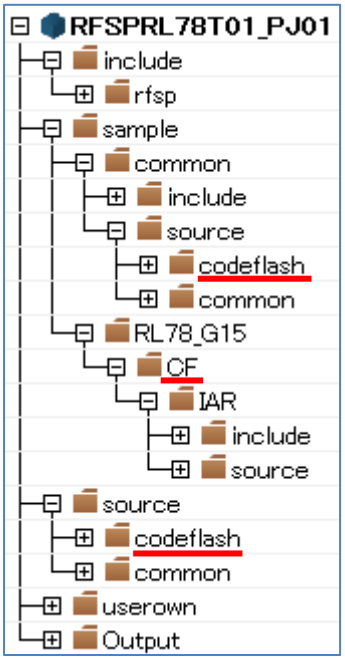


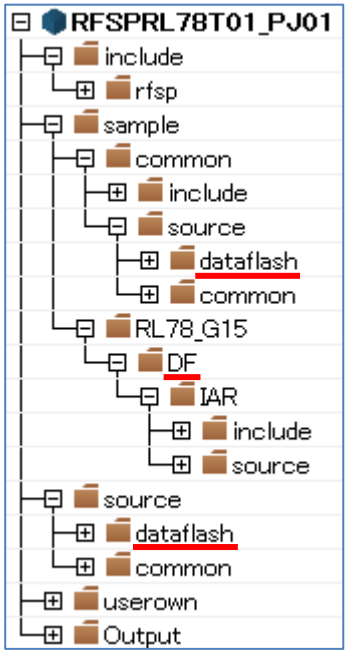
6.2.2 対象フォルダと対象ファイルの登録例

RFSP Type01 を使用して、各領域[(1)コード・フラッシュ・メモリ、(2)データ・フラッシュ・メモリ]を書き換える場合に必要なファイル、およびサンプル関数を含むファイルの登録例を記述します。 RFSP Type01 ソースプログラムファイルの各フォルダは、"include", "source", "userown", "sample"で、書き換える領域により各フォルダ内の対象ファイルを選択して登録します。

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、 RFSP Type01 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。）

各領域[(1)コード・フラッシュ・メモリ、(2)データ・フラッシュ・メモリ]のグループを追加した例を示します。（領域ごとに異なるグループ名を"＿"で示しています。）





(1)コード・フラッシュ・メモリ (2)データ・フラッシュ・メモリ

・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

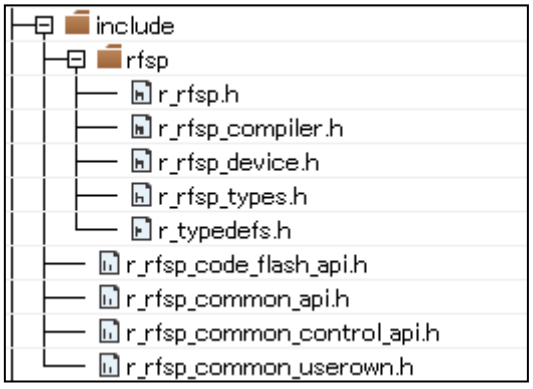
作成されたプロジェクトには、自動的に追加されるファイルがあります。これらと同様のファイルは、RFSP Type01 の"sample"フォルダ内にも存在するため、ツリーで各ファイルを選択し、各統合開発環境の機能を使用して、プロジェクトから外します。

- 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。

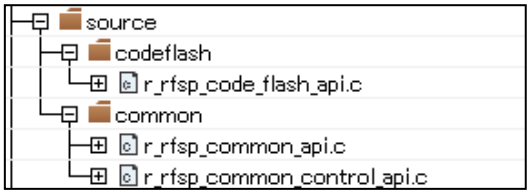
(1) コード・フラッシュ・メモリを書き換える場合の対象グループと対象ファイルの登録

RFSP Type01 ソースプログラムファイルの各グループ("include"、"source"、"userown"、"sample")と登録ファイルを以下に示します。

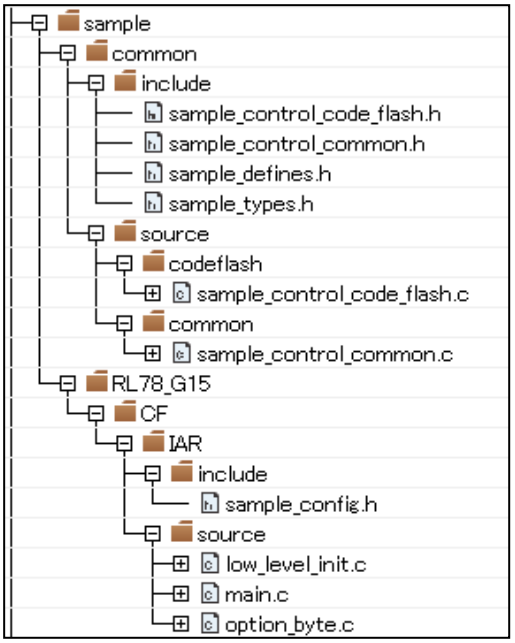
include グループ内



source グループ内



sample グループ内

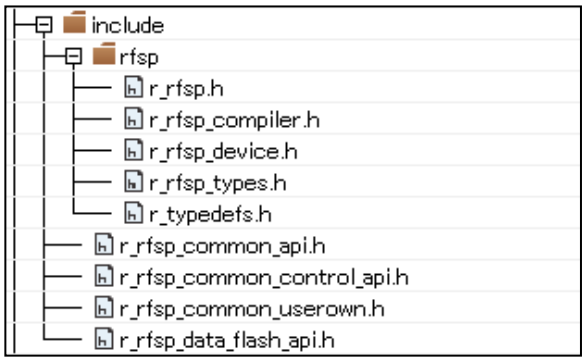


userown グループ内

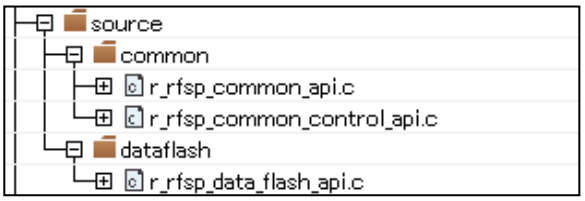
(2) データ・フラッシュ・メモリを書き換える場合の対象グループと対象ファイルの登録

RFSP Type01 ソースプログラムファイルの各グループ("include"、"source"、"userown"、"sample")と登録ファイルを以下に示します。

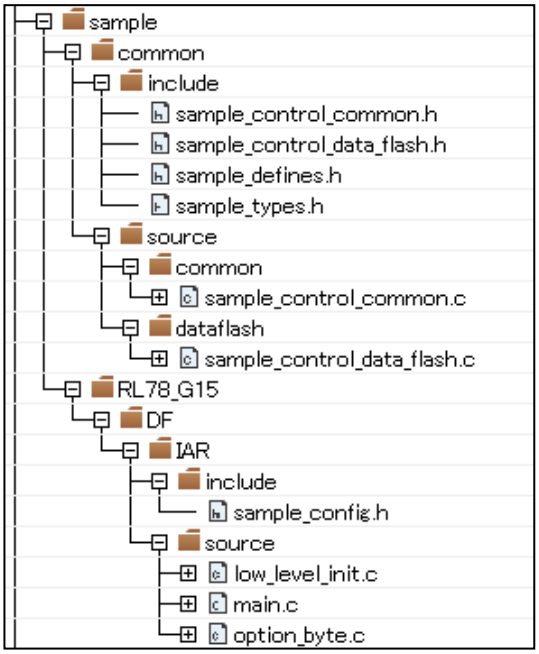
include グループ内



source グループ内



sample グループ内



userown グループ内

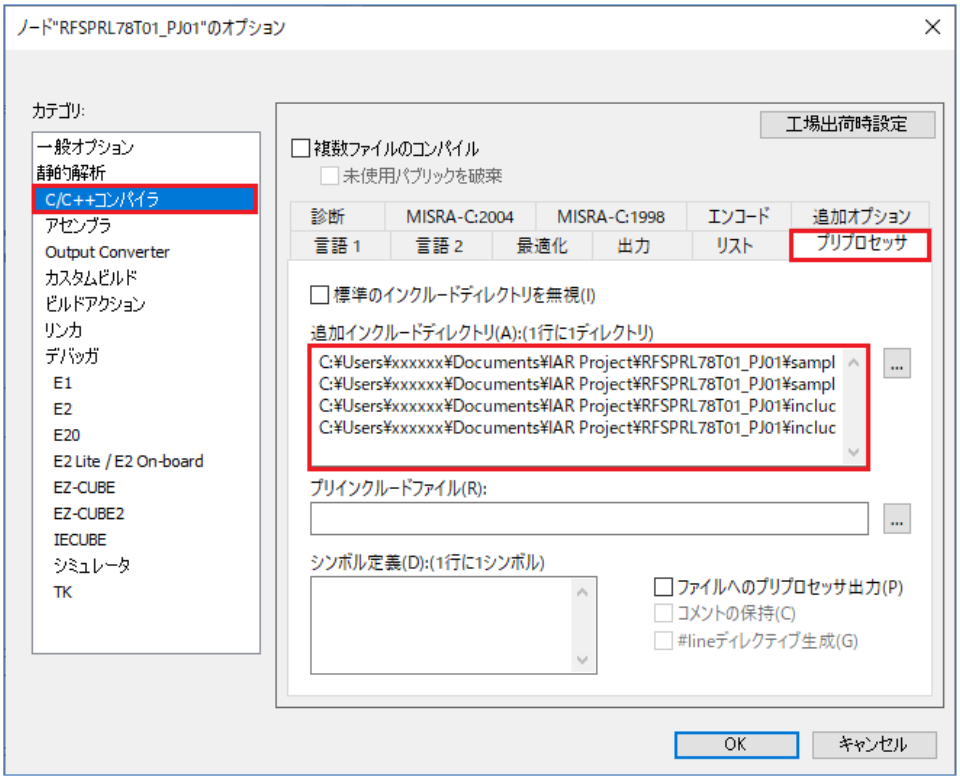
6.2.3 統合開発環境の設定

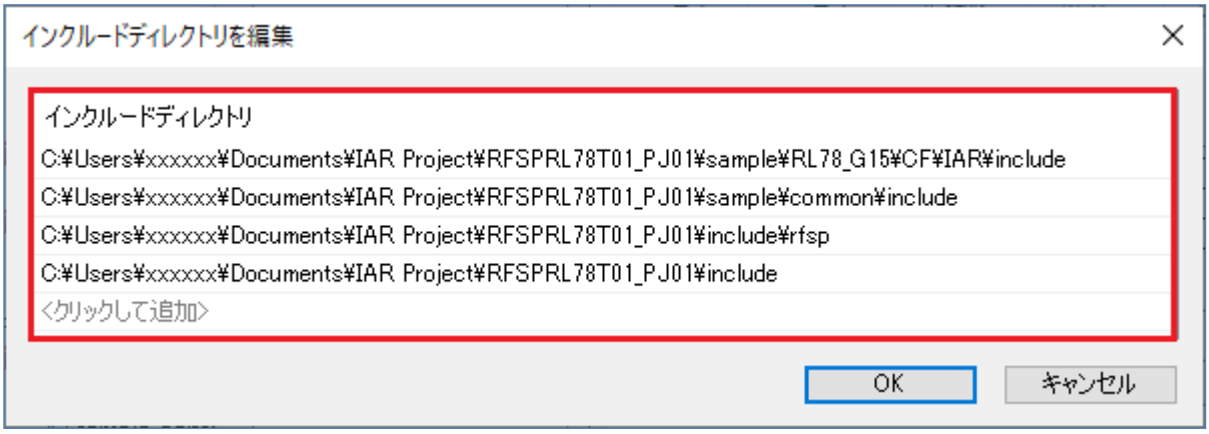
IAR コンパイラで RFSP Type01 をビルドして実行するための統合開発環境の設定を行います。IAR Embedded Workbench ではツリーで[プロジェクト]をマウス右クリックして"オプション"を選択、表示された画面内の"カテゴリ"を選択して各設定を行います。

6.2.3.1 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 (対象領域により変更) 。

- [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"パス編集"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\xxxxxxxx\Documents\IAR_Project\"に、RFSP Type01 ソースプログラムファイルの各フォルダ("include"、"source"、"userown"、"sample")を置いた場合の例です。

(1)コード・フラッシュ・メモリ書き換え

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\RL78_G15\CF\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include\rfsp

(2)データ・フラッシュ・メモリ書き換え

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\RL78_G15\DF\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\common\include

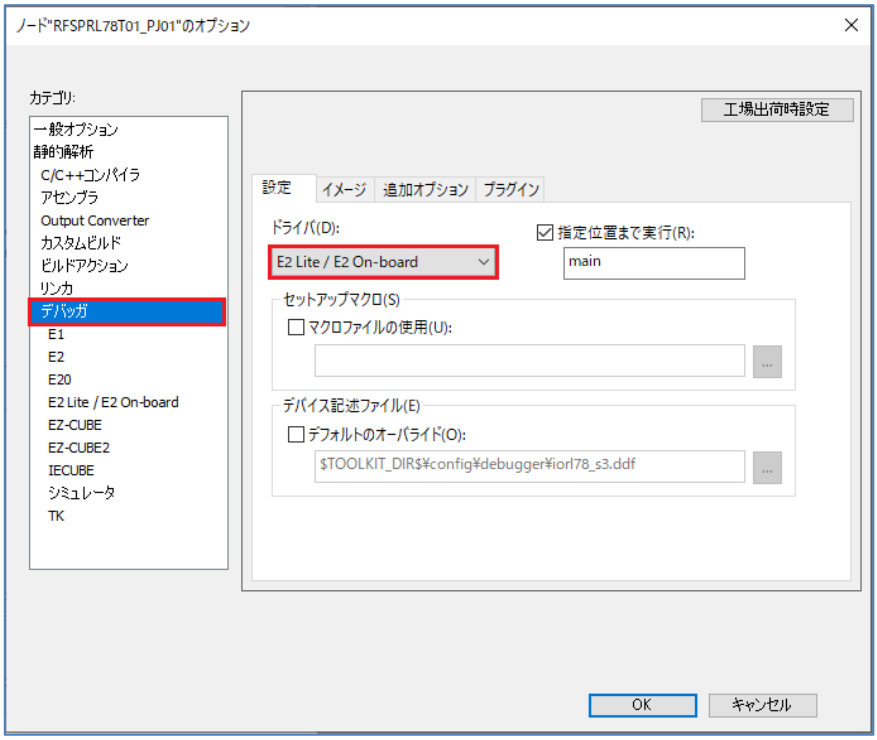
C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include\rfsp

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ご参照いただき、必要に応じて設定してください。

6.2.3.2 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。



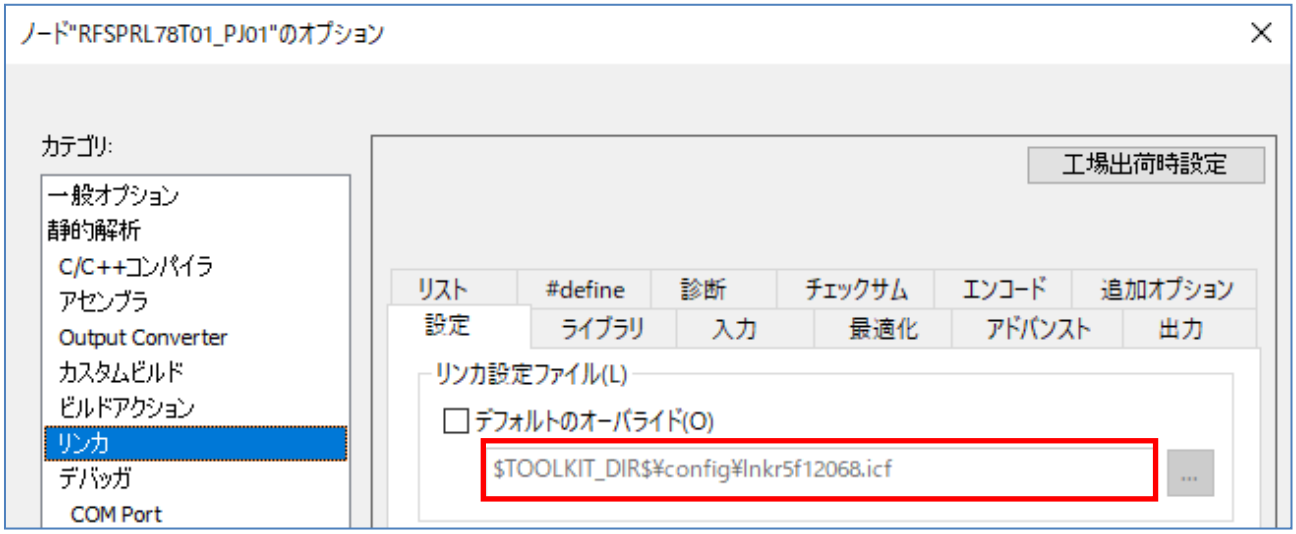
注)その他の設定項目については、IAR Embedded Workbench の[Help]から各リファレンスマニュアルをご参照いただき、必要に応じて設定してください。

6.2.4 リンカ設定ファイル(.icf)の設定

RFSP Type01 では、IAR Embedded Workbench でビルドする場合、対象デバイス用のリンク設定ファイル(*.icf)を使用しています。

対象デバイス用のリンク設定ファイル(*.icf)を指定するためには、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]-[設定]で設定されている"$TOOLKIT_DIR$\config\lnkrxxxxxxx.icf"を指定してください。

例) R5F12068[RL78/G15]では、"lnkr5f12068.icf"ファイルを指定します。



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。

6.2.4.1 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に記述されています。(例 R5F12068 [RL78/G15]では、"lnkr5f12068.icf")

RFSP 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。

IAR Embedded Workbench 付属の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 };
                          |
```

"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] = {
    0xEE,   /* 11101110 */
            /*     |||| */
            /*     |||+-- Watchdog timer     */
            /*     ||||    operation stopped  */
            /*     ||||    in HALT/STOP mode  */
            /*     +++--- Watchdog timer     */
            /*     |       overflow time is   */
            /*     |       2^16 / fIL =       */
            /*     |       3799 ms         */
            /*     +------ Watchdog timer     */
            /*             operation disabled */

    0xFF,   /* 11111111 */
            /*     |||   */
            /*     |++-- SPOR 2.16V/2.11V */
            /*     +---- P125 input reset */
    0xF9,   /* HS mode 16 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"EEFFF9"です。

(WDT 停止,P125:RESET 入力,SPOR 検出電圧/2.16V/2.11V,16MHz [G15,G16 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)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。

6.2.5 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。

6.2.5.1 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。

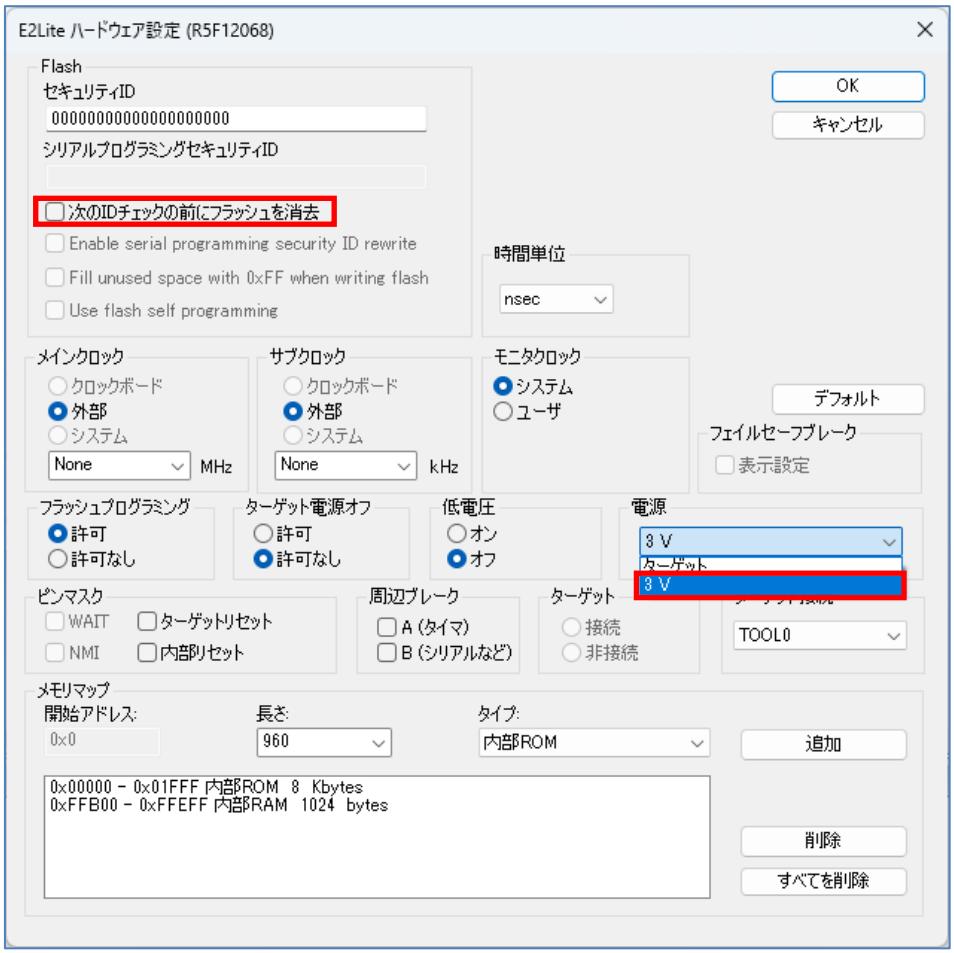
- ID コードが不一致の場合：

"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]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。



6.3 LLVM コンパイラを使用する場合のプロジェクトの作成

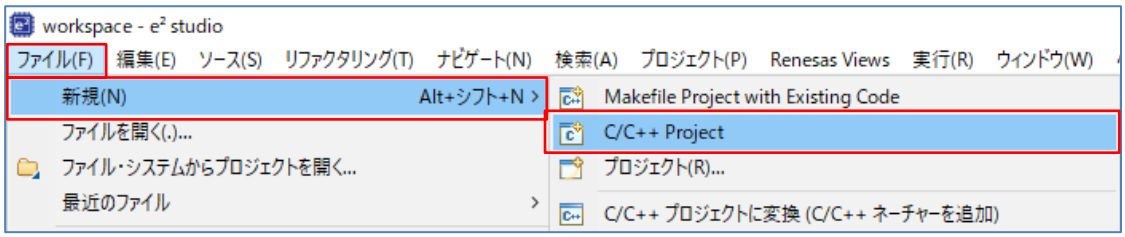
LLVM コンパイラは、統合開発環境として e^2 studio を使用して作成したプロジェクトへ RFSP Type01 を登録し、ビルドすることができます。e^2 studio を使用した場合のサンプル・プロジェクトの作成例を示します。

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。

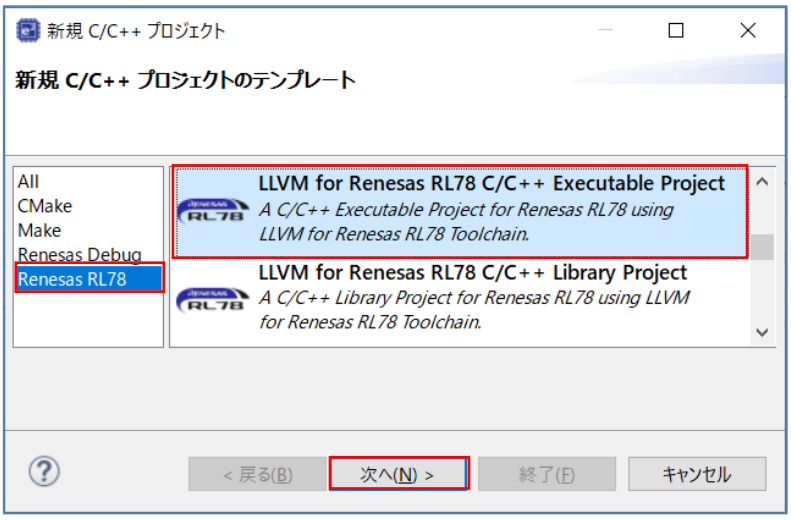
6.3.1 サンプル・プロジェクトの作成例

（１）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

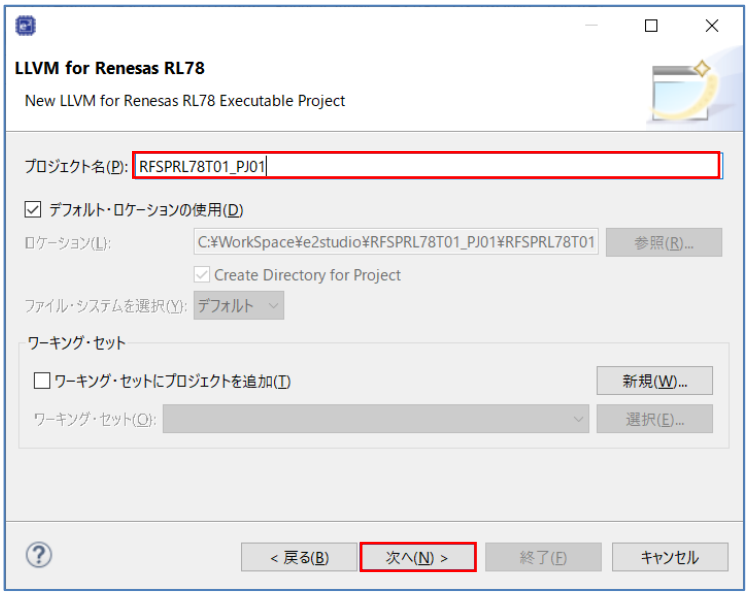
e² studio を起動し、[ファイル]メニューの[新規]から[C/C++ Project]を選択し、"新規 C/C++ プロジェクトのテンプレート"ウインドウを起動します。



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



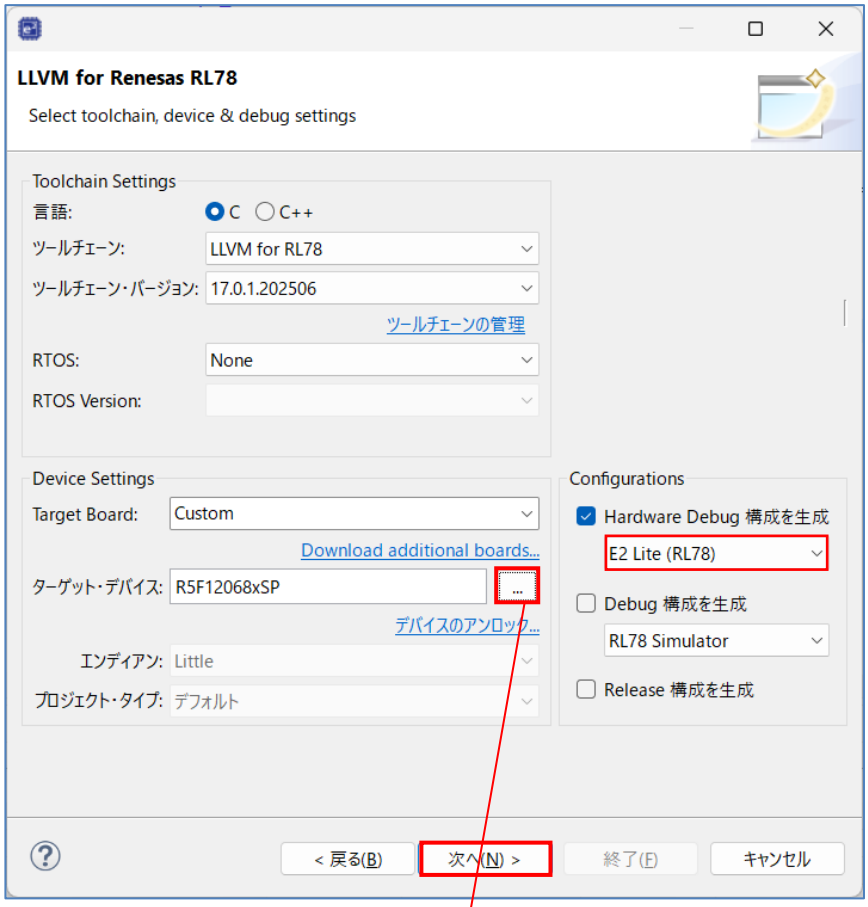
・"New LLVM for Renesas RL78 Executable Project"ウインドウで、プロジェクト名を入力して"次へ"ボタンを押します。（ここでは、仮に"RFSPRL78T01_PJ01"とします。）

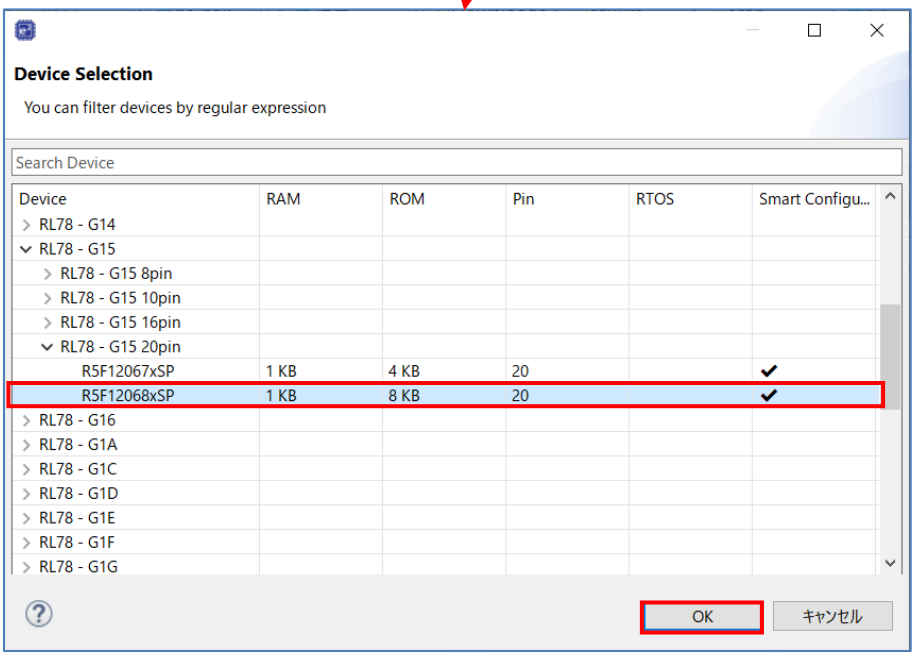


・[Device Settings]の[ターゲット・デバイス]で、"RL78 - G15 20pin - R5F12068xSP"を選択します。

・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。

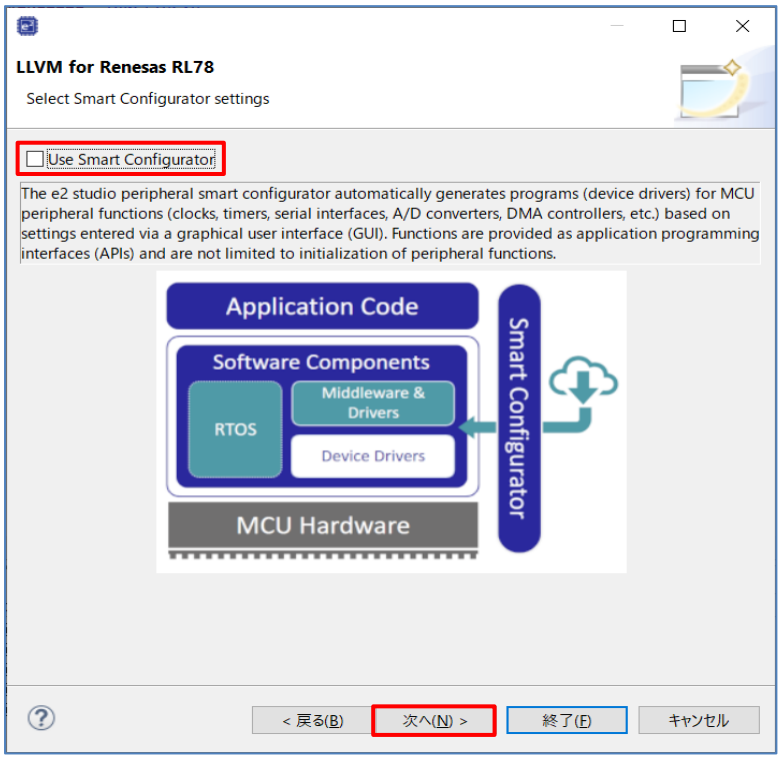
・[次へ]ボタンを押します。





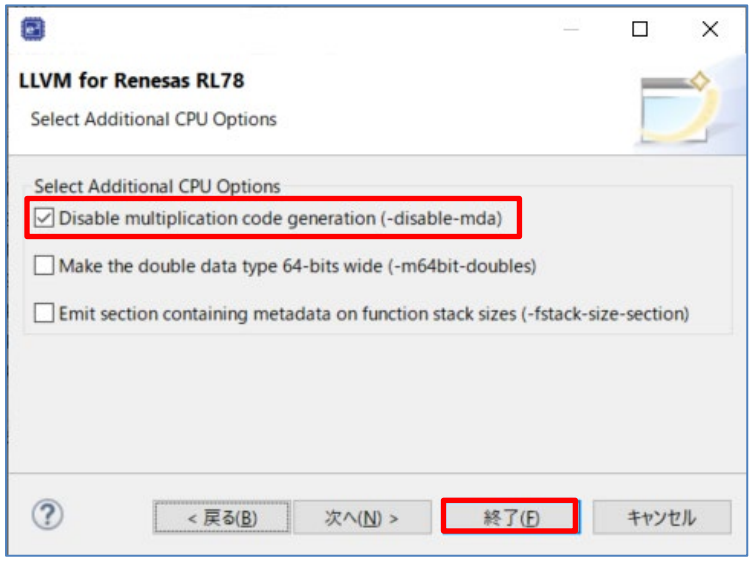
・"Use Smart Configurator"のチェックを外します。

・[次へ]ボタンを押します。



・"Disable multiplication code generation (-disable-mda)"にチェックを入れます。

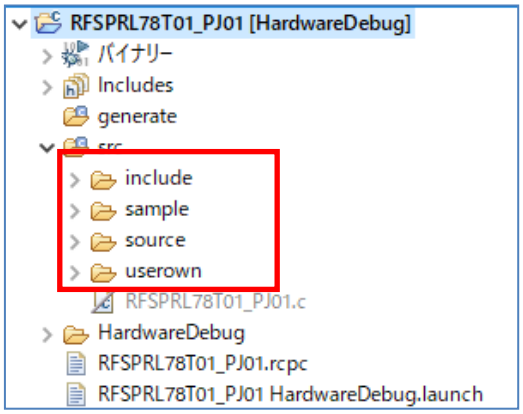
・"終了"ボタンを押します。



6.3.2 対象フォルダと対象ファイルの登録例

RFSP Type01 を使用して、各領域(1)コード・フラッシュ・メモリ、(2)データ・フラッシュ・メモリを書き換える場合に必要なファイルの登録例を記述します。 RFSP Type01 ソースプログラムファイルの各フォルダは、"include", "source", "userown", "sample"で、書き換える領域により各フォルダ内の対象ファイルを選択して登録します。

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 RFSP Type01 登録時のツリー画面

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

"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、RFSP Type01 に含まれている"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、"6.3.3.2 デバイス項目の設定"をご参照ください。

e² studio が"vects.c"を出力するフォルダ：

- e² studio：[プロジェクト名]/generate フォルダ

"vects.c"ファイルを入れ替え、もしくは上書きするフォルダ：

- コード・フラッシュ書き換え時："\[プロジェクト名]\sample\RL78_G15\CF\LLVM\source"
- データ・フラッシュ書き換え時："\[プロジェクト名]\sample\RL78_G15\DF\LLVM\source "

・e² studio の機能により自動的に追加されたファイルの除外

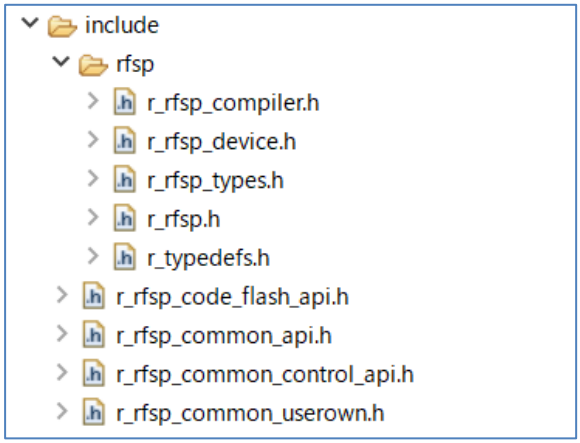
作成されたプロジェクトには、自動的に追加されるファイルがあります。これらと同様のファイルは、 RFSP Type01 の"sample"フォルダ内にも存在するため、ツリーで各ファイルを選択し、e² studio の機能を使用して、プロジェクトから外します。

- 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"hwinit.c", "vects.c"が対象
  - ・[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"RFSPRL78T01_PJ01.c")が対象。

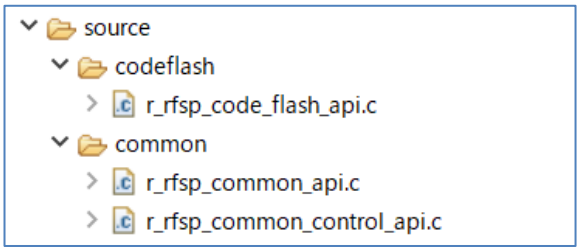
(1) コード・フラッシュ・メモリを書き換える場合の対象フォルダと対象ファイルの登録

RFSP 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。

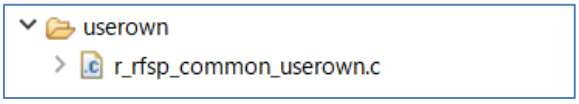
include フォルダ内



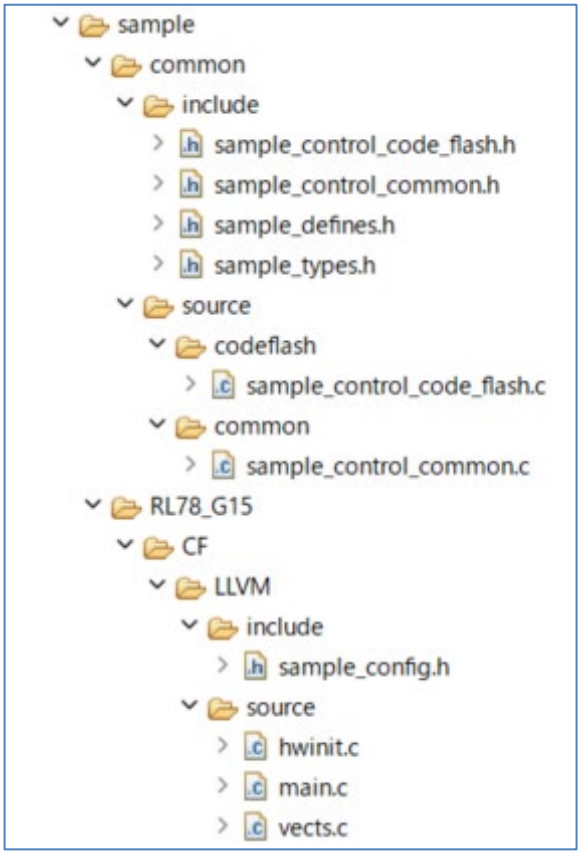
source フォルダ内



userown フォルダ内



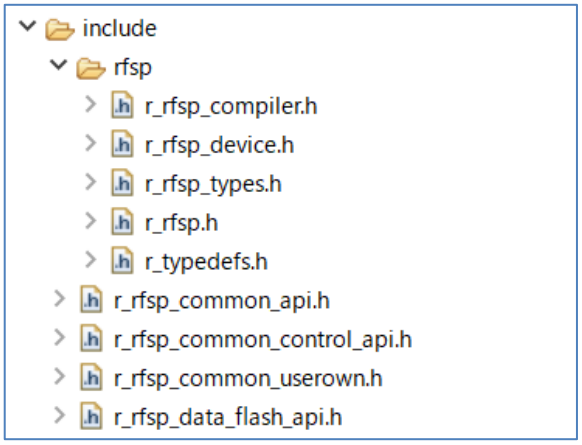
sample フォルダ内



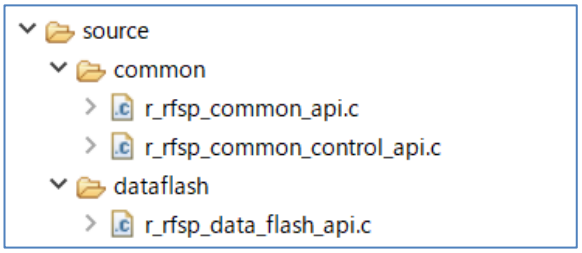
(2) データ・フラッシュ・メモリを書き換える場合の対象フォルダと対象ファイルの登録

RFSP 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

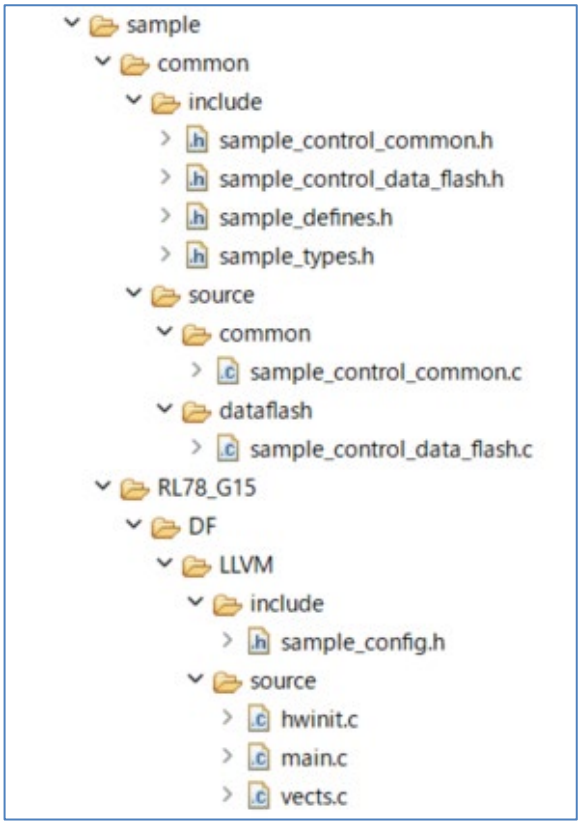


source フォルダ内



userown フォルダ内

sample フォルダ内



6.3.3 ビルド・ツールの設定

LLVM コンパイラで RFSP Type01 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。

e2 studio ではツリーでプロジェクト(ここでは"RFSPRL78T01_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。

6.3.3.1 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(対象領域により変更)

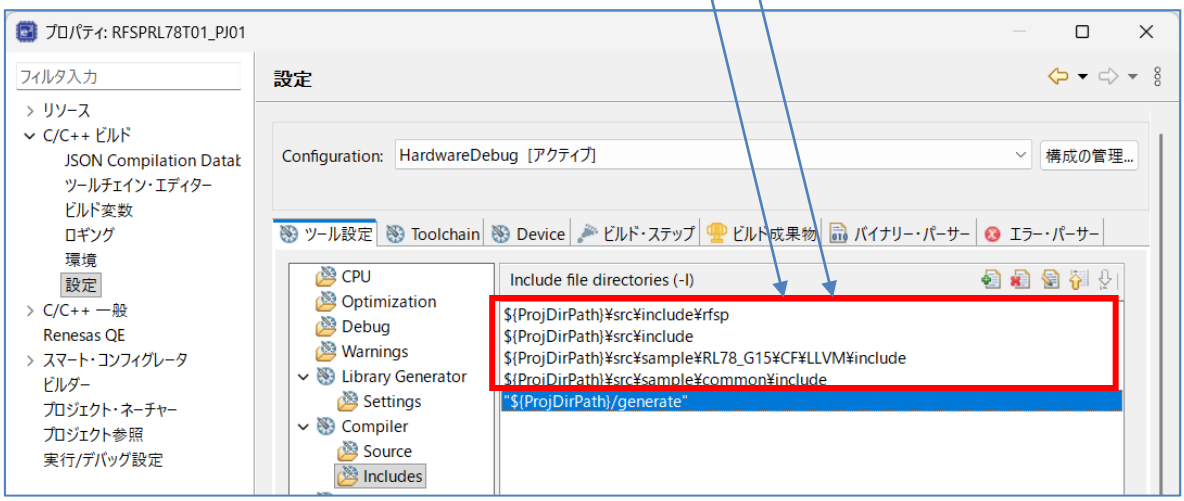
- "C/C++ビルド" [設定] – "Compiler" [Includes] で表示した画面でインクルード・ファイルのパスを追加します。

(1)コード・フラッシュ・メモリ書き換え

```
${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\CF\LLVM\include
${ProjDirPath}\src\sample\common\include
```

(2)データ・フラッシュ・メモリ書き換え

```
${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\DF\LLVM\include
${ProjDirPath}\src\sample\common\include
```



6.3.3.2 デバイス項目の設定

RFSP Type01 のオプション・バイト定義は、"vects.c"ファイルに記述されています。

"vects.c"ファイル内のオプション・バイト値の記述例

```
const unsigned char Option_Bytes[]  __attribute__ ((section (".option_bytes"))) = {
    0xee, 0xff, 0xf9, 0x85
};
```

- ユーザ・オプション・バイト値の説明：

" vects.c"ファイル内のユーザ・オプション・バイト(000C0H-000C2H)の値は"EEFFF9"です。

(WDT 停止,P125:RESET 入力,SPOR 検出電圧/2.16V/2.11V,16MHz [G15,G16 の例])

" vects.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)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。

6.3.4 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、"RFSPRL78T01_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。

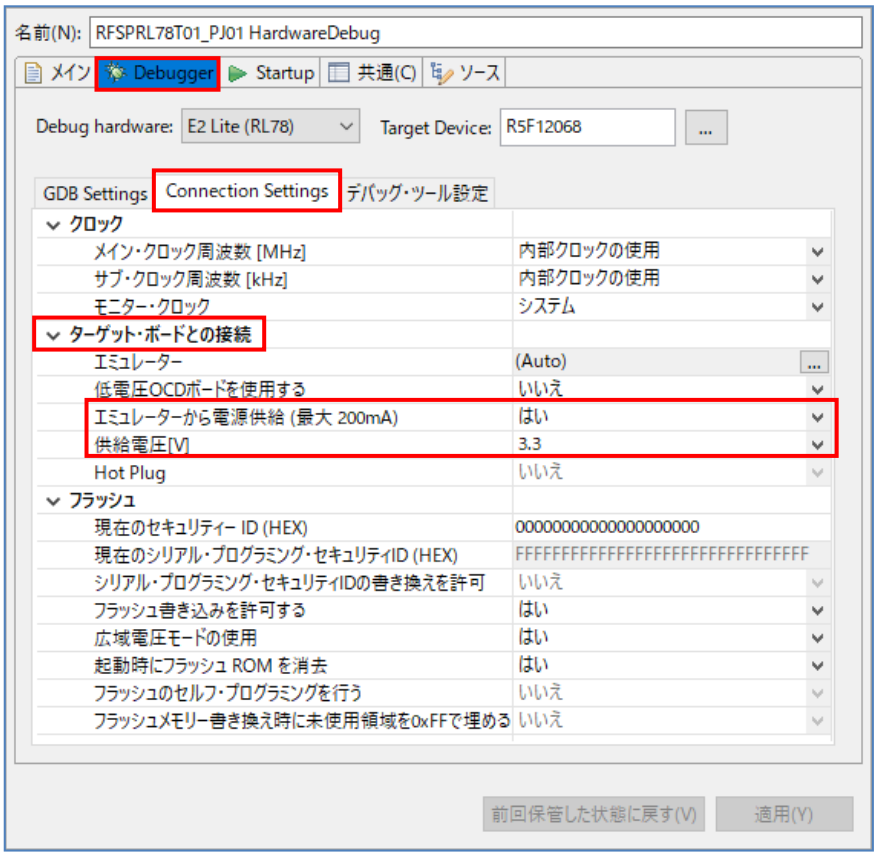
注）ターゲット・ボードに他の電源が供給されている場合や電源供給容量が不足するなど、E2 Lite を含むエミュレータからターゲット・ボードへの電源供給ができない場合があります。必ず、対象デバイス用のエミュレータのユーザーズマニュアル、およびユーザーズマニュアル別冊(RL78 接続時の注意事項)をご参照の上、ご使用ください。

6.3.4.1 ターゲット・ボードとの接続の設定

・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(各領域共通)

- [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 改定記録

7.1 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	2022.9.30	-	初版発行
1.10	2023.4.28	-	RL78/G16 を追加
1.20	2023.9.25	-	LLVM コンパイラを追加
1.21	2025.10.31	-	CC-RL コンパイラ、LLVM コンパイラを使用する場合に、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を優先的に使用できるようサンプルの構成を変更。
		-	軽微な誤記を修正、一部記述内容を改善。

製品ご使用上の注意事項

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

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュ・メモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。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。