RX ファミリ

Firmware Integration Technology とコード生成ツールを組み合わせて使用する場合の設定手順

R01AN2136JJ0110
Rev.1.10
2015.03.02

要旨

本アプリケーションノートでは、Firmware Integration Technology(FIT)とコード生成ツールを組み合わせて使用する場合の設定手順について説明します。説明は RX64M グループを代表としてします。RX64M グループ以外の製品をご使用の場合は、ご使用になる製品に置き換えてお読みください。

対象デバイス

・RX ファミリ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 概要

初期設定、クロックの設定に FIT の Board Support Package(BSP)モジュールを使用し、周辺機能の設定にコード生成ツールを組み合わせて使用する場合の以下 3 つの設定手順について説明します。

1. 新規プロジェクト立ち上げ時の設定手順

   新規プロジェクトの生成からビルドまでの手順を説明します。詳細は 4.1 章に示します。

2. 設定変更時の設定手順

   上記 1 で作成したプロジェクトのクロック設定や周辺機能の条件を変更する際の手順を説明します。
   詳細は 4.2 章に示します。

3. 周辺機能追加時の設定手順

   上記 1 で作成したプロジェクトにさらに周辺機能を追加する際の手順を説明します。
   詳細は 4.3 章に示します。

制限事項

・FIT の BSP モジュールとコード生成ツールの RTC を組み合わせて使用することはできません。

2. 動作確認条件

本アプリケーションノートで説明する設定手順は、下記の条件で動作できます。e2 studio および BSP モジュールのバージョンが変更された場合の動作は保証できません。

表2.1　動作確認条件

項目	内容
使用マイコン	R5F564MLCDFC (RX64M グループ)
統合開発環境	ルネサスエレクトロニクス製 e2 studio Version:3.0.1.07
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V2.01.00
BSP モジュールのバージョン	Version V2.80
iodefine.h のバージョン	Version 0.9
エンディアン	リトルエンディアン
動作モード	シングルチップモード
プロセッサモード	スーパバイザモード
使用ボード	Renesas Starter Kit+ for RX64M(製品型名:R0K50564MSxxxBE)

3. 関連アプリケーションノート

本アプリケーションノートに関連するドキュメントを以下に示します。併せて参照してください。

・Firmware Integration Technology ユーザーズマニュアル(R01AN1833JU0100)
・RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685EU0250)

4. 初期設定に BSP モジュール、周辺機能にコード生成ツールを使用した場合の設定手順

初期設定に BSP モジュール、周辺機能にコード生成ツールを使用する場合の、新規プロジェクト立ち上げ時、設定変更時、周辺機能追加時の各設定手順を 4.1～4.3 に示します。代表例として、RX64M グループの各設定手順を示します。RX64M グループ以外の製品をご使用の場合は、ご使用になる製品に置き換えてお読みください。

4.1 新規プロジェクト立ち上げ時の設定手順

新規プロジェクトの立ち上げから BSP モジュールでの初期設定、コード生成ツールでの周辺機能の設定、ビルドまでの設定手順を説明します。図 4.1 に新規プロジェクト立ち上げ時のフローを示します。また、各手順の詳細な処理方法を説明します。

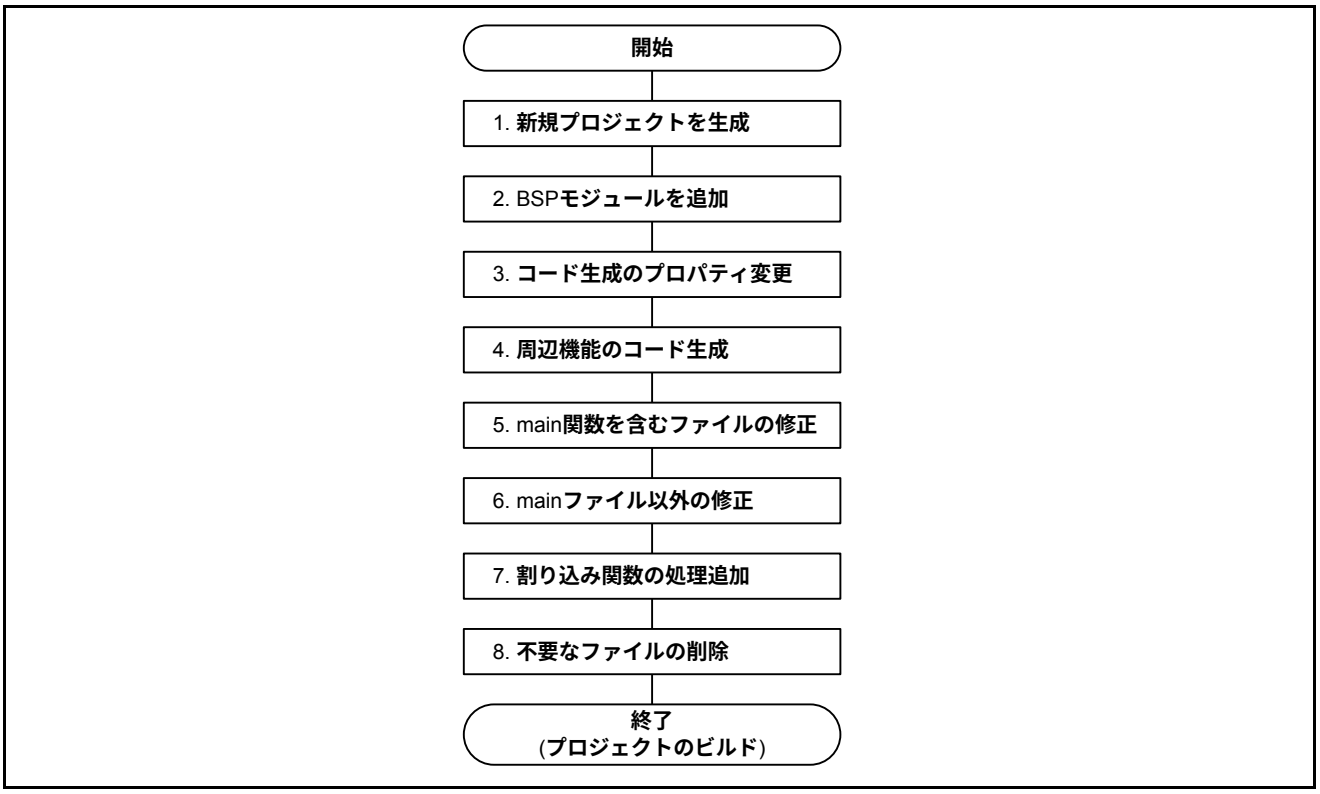
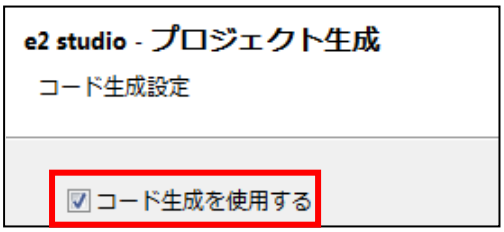


図 4.1 新規プロジェクト立ち上げ時の設定手順

1. 新規プロジェクトを生成

FIT を使用する際の新規プロジェクト作成をベースに作成します。

1) e2 studio のツールバー 「ファイル > 新規(N) > C Project」を選択
2) プロジェクト名を入力し、Toolchain は「Renesas RXC toolchain」を選択し「次へ(N)」をクリック
3) ターゲットを選択
4) 構成の選択から「リリース(デバッグしない)」のチェックをはずし、「次へ(N)」をクリック
5) 「コード生成を使用する」をチェックし、「次へ(N)」をクリック



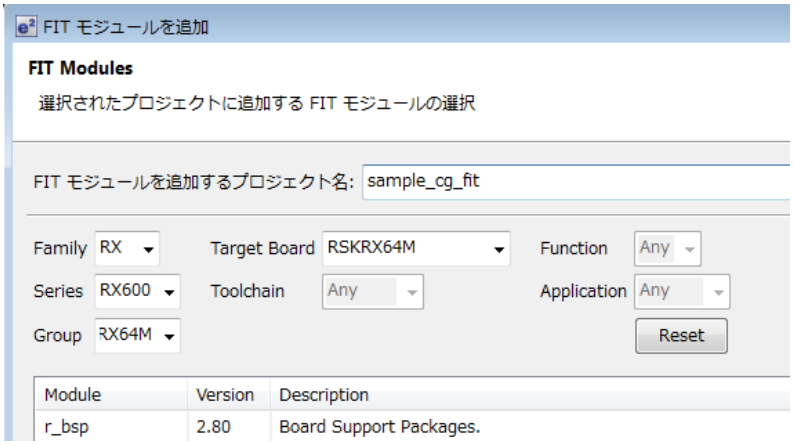
※ポイント：FIT のみを使用する場合の新規プロジェクト作成時には、チェックしませんが今回は周辺機能にコード生成ツールを使用するため、チェックします。

6) 「追加 CPU オプションの選択」「グローバル・オプション設定」は必要に応じて変更
7) 「標準ヘッダ・ファイル」ではライブラリ構成に『C(C99)』を選択し、「次へ(N)」をクリック
8) 「ユーザ・スタックの使用」「ヒープ・メモリの使用」「ベクタ定義ファイル」「I/O レジスタ定義ファイル」のチェックをすべて外し、「終了(F) 」をクリック
9) プロジェクトが作成されたら、src フォルダの 3 つのファイル(dbsct.c、<プロジェクト名>.c、typedefine.h)を削除する
10) プロジェクトを右クリックし「プロパティ(R)」から「C/C++ビルド > 設定 > Linker > セクション」を選択
11) PResetPRG セクション、PIntPRG セクションで「セクションの除去」をクリック
12) P セクションを「P*セクション」に変更
13) 「適用(L)」を設定

2. BSP モジュールを追加

1.で生成したプロジェクトに『BSP モジュールを追加』します。

1) e2 studio のツールバー「ファイル > 新規(N) > Renesas FIT Module」を選択
2) FIT モジュールを追加するプロジェクト名を選択
3) Family, Series, Group, Target Board を選択
4) r_bsp モジュールを選択
5) 「終了(F)」をクリック



6) r_bsp > board > rskrx64m > r_bsp_config_reference.h と r_bsp_interruput_config_reference.h を r_config フォルダにコピーし、ファイル名を「r_bsp_config.h」と「r_bsp_interruput_config.h」にリネームする

7) r_bsp_config.h でクロック設定を行う
(クロックソース選択、メインクロック周波数設定、各クロック分周選択など)

```
/* Clock source select (CKSEL).
   0 = Low Speed On-Chip Oscillator  (LOCO)
   1 = High Speed On-Chip Oscillator (HOCO)
   2 = Main Clock Oscillator
   3 = Sub-Clock Oscillator
   4 = PLL Circuit
*/
#define BSP_CFG_CLOCK_SOURCE            (4)

/* Clock configuration options.
   The input clock frequency is specified and then the system clocks are set by specifying the multipliers used. The
   multiplier settings are used to set the clock registers in resetprg.c. If a 24MHz clock is used and the
   ICLK is 120MHz, PCLKA is 120MHz, PCLKB is 60MHz, PCLKC is 60MHz, PCLKD is 60MHz, FCLK is 60MHz, USB Clock is 48MHz,
   and BCLK is 120MHz then the settings would be:

   BSP_CFG_XTAL_HZ  = 24000000
   BSP_CFG_PLL_DIV  = 1        (no division)
   BSP_CFG_PLL_MUL  = 10.0     (24MHz x 10.0 = 240MHz)
   BSP_CFG_ICK_DIV  = 2      : System Clock (ICLK)        =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_ICK_DIV)  = 120MHz
   BSP_CFG_PCKA_DIV = 2      : Peripheral Clock A (PCLKA) =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKA_DIV) = 120MHz
   BSP_CFG_PCKB_DIV = 4      : Peripheral Clock B (PCLKB) =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKB_DIV) = 60MHz
   BSP_CFG_PCKC_DIV = 4      : Peripheral Clock C (PCLKC) =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKC_DIV) = 60MHz
   BSP_CFG_PCKD_DIV = 4      : Peripheral Clock D (PCLKD) =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_PCKD_DIV) = 60MHz
   BSP_CFG_FCK_DIV =  4      : Flash IF Clock (FCLK)      =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_FCK_DIV)  = 60MHz
   BSP_CFG_BCK_DIV =  2      : External Bus Clock (BCK)   =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_BCK_DIV)  = 120MHz
   BSP_CFG_UCK_DIV =  5      : USB Clock (UCLK)           =
                               (((BSP_CFG_XTAL_HZ/BSP_CFG_PLL_DIV) * BSP_CFG_PLL_MUL) / BSP_CFG_UCK_DIV)  = 48MHz
*/

/* XTAL - Input clock frequency in Hz */
#define BSP_CFG_XTAL_HZ                 (24000000)

/* The HOCO can operate at several different frequencies. Choose which one using the macro below.
   Available frequency settings:
   0 = 16MHz     (default)
   1 = 18MHz
   2 = 20MHz
*/
#define BSP_CFG_HOCO_FREQUENCY          (0)

/* PLL clock source (PLLSRCEL). Choose which clock source to input to the PLL circuit.
   Available clock sources:
   0 = Main clock (default)
   1 = HOCO
*/
#define BSP_CFG_PLL_SRC                 (0)

/* PLL Input Frequency Division Ratio Select (PLIDIV).
   Available divisors = /1 (no division), /2, /3
*/
#define BSP_CFG_PLL_DIV                 (1)

/* PLL Frequency Multiplication Factor Select (STC).
   Available multipliers = x10.0 to x30.0 in 0.5 increments (e.g. 10.0, 10.5, 11.0, 11.5, ..., 29.0, 29.5, 30.0)
*/
#define BSP_CFG_PLL_MUL                 (10.0)
```

8) プラットフォームを選択します。プラットフォームを選択するには、使用するボードの'#include'を非コメント化します。

(変更前)

```
/* RDKRX63N */
//#include "./board/rdkrx63n/r_bsp.h"

/* RSKRX64M */
//#include "./board/rskrx64m/r_bsp.h"

/* RSKRX210 */
//#include "./board/rskrx210/r_bsp.h"
```

(変更後)

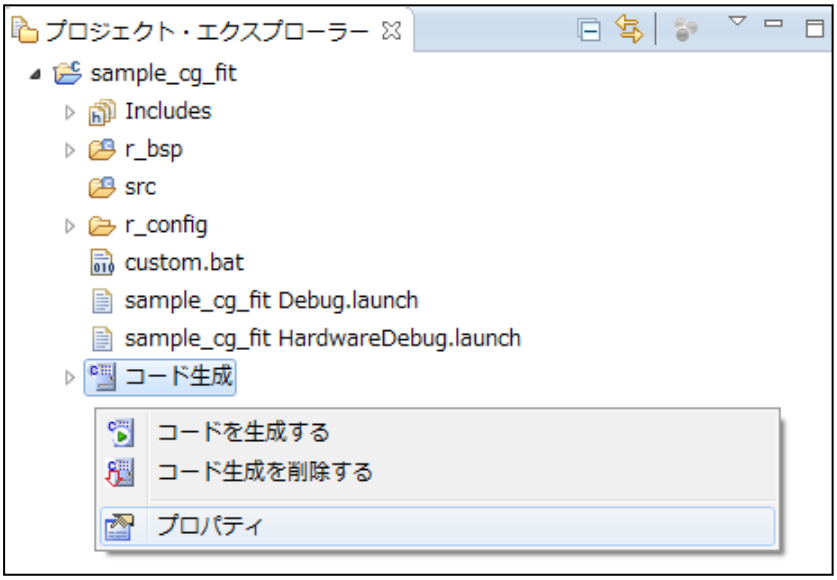
```
/* RDKRX63N */
//#include "./board/rdkrx63n/r_bsp.h"

/* RSKRX64M */
#include "./board/rskrx64m/r_bsp.h"

/* RSKRX210 */
//#include "./board/rskrx210/r_bsp.h"
```

非コメント化

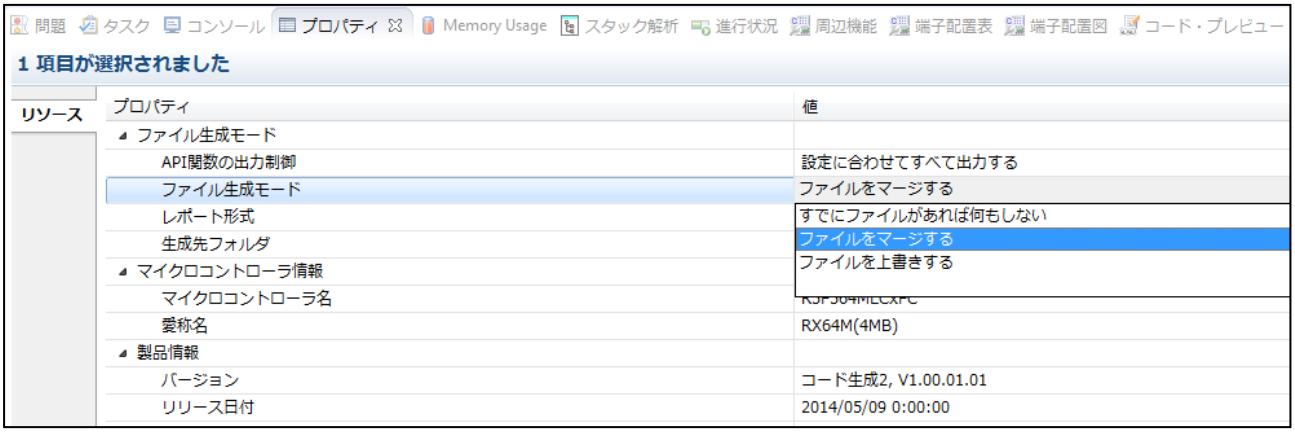
3. コード生成のプロパティ変更

コード生成のファイル生成モードを『ファイルをマージする』に変更します。

1) 作成したプロジェクト内のコード生成を右クリックし、「プロパティ」を選択



2) プロパティビューの「ファイル生成モード」を『ファイルをマージにする』に変更



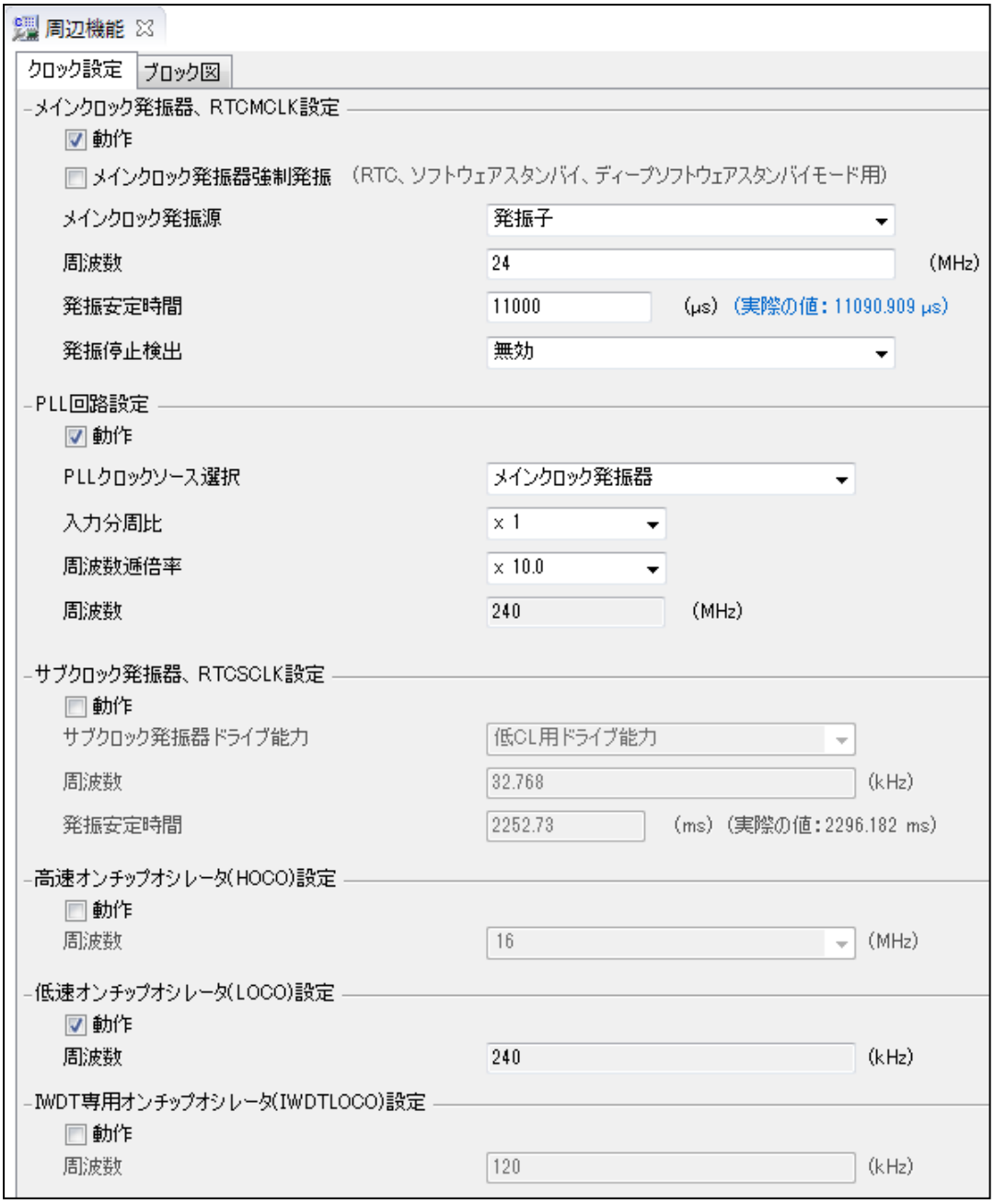
ファイ生成モードの 3 つの違いについて以下に示します。

ファイル生成モード	コード生成後の処理
すでにファイルがあれば何もしない	既存ファイルにファイル名が同一のものがある場合，該当ファイルの出力を行いません。
ファイルをマージする	指定のコメント間にコードを書いていれば、その箇所を残したままコードが更新されます。
ファイルを上書きする	既存ファイルにファイル名が同一のものがある場合，該当ファイルを上書きします。

4. 周辺機能のコード生成

コード生成ツールでクロックの設定と周辺機能の設定を行います。

1) プロジェクト内のコード生成 > 周辺機能 > クロック発生回路をクリックする
2) 2-7)で r_bsp_config.h に設定した内容と同じ設定を行う



※ポイント：周辺機能の設定(カウンタ値、ビットレートなど)はコード生成の設定から求められます。BSP モジュールの設定で正しく動作するために、コード生成のクロック発生回路の設定と BSP モジュールの設定を合わせます。

3) プロジェクト内のコード生成 > 周辺機能 > 使用したい周辺機能をクリックする
4) 使用したい条件を GUI で設定する
5) 『コードを生成する』をクリックする
6) プロジェクト内の src フォルダ内にソースコードが生成されたことを確認する

5. main 関数を含むファイルの修正

コード生成で作成された main 関数を含むファイル(初期設定時のファイル名は、r_cg_main.c)(以下、main ファイルと称す)を修正します。修正する際にコードを “ 追加 ” する際は以下注意を参照してください

【注意】コードを追加する場合

コードを追加する際は下記コメント間に必ず記載してください。下記コメント間以外にコードを追加した場合、コード生成を再度行った際にコードが削除されます。

/* Start user code for include. Do not edit comment generated here */

コードを追加する際はこのコメント間に追加してください。

/* End user code. Do not edit comment generated here */

1) r_cg_cgc.h **をインクルードするコードを削除**

2) platform.h **をインクルード**

(変更前)

```
/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_mtu3.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
```

削除

(変更後)

```
/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_mtu3.h"
/* Start user code for include. Do not edit comment generated here */
#include "platform.h"
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
```

追加

3) マクロ定義の追加

r_cg_macrodriver.h から以下の赤枠の箇所をコピーし、main ファイルに貼り付けてください。

(r_cg_macrodriver.h のコード 一部抜粋)

```
/***********************************************************************************************
Macro definitions
***********************************************************************************************/
#ifndef __TYPEDEF__                                                          コピー

/* Status list definition */
#define MD_STATUSBASE           (0x00U)
#define MD_OK                   (MD_STATUSBASE + 0x00U) /* register setting OK */
#define MD_SPT                  (MD_STATUSBASE + 0x01U) /* IIC stop */
#define MD_NACK                 (MD_STATUSBASE + 0x02U) /* IIC no ACK */
#define MD_BUSY1                (MD_STATUSBASE + 0x03U) /* busy 1 */
#define MD_BUSY2                (MD_STATUSBASE + 0x04U) /* busy 2 */

/* Error list definition */
#define MD_ERRORBASE            (0x80U)
#define MD_ERROR                (MD_ERRORBASE + 0x00U)  /* error */
#define MD_ARGERROR             (MD_ERRORBASE + 0x01U)  /* error argument input error */
#define MD_ERROR1               (MD_ERRORBASE + 0x02U)  /* error 1 */
#define MD_ERROR2               (MD_ERRORBASE + 0x03U)  /* error 2 */
#define MD_ERROR3               (MD_ERRORBASE + 0x04U)  /* error 3 */
#define MD_ERROR4               (MD_ERRORBASE + 0x05U)  /* error 4 */
#define MD_ERROR5               (MD_ERRORBASE + 0x06U)  /* error 5 */

#endif
```

(r_cg_main.c のコード)

```
/***********************************************************************************************
Global variables and functions
***********************************************************************************************/
/* Start user code for global. Do not edit comment generated here */          貼り付け

/* Status list definition */
#define MD_STATUSBASE           (0x00U)
#define MD_OK                   (MD_STATUSBASE + 0x00U) /* register setting OK */
#define MD_SPT                  (MD_STATUSBASE + 0x01U) /* IIC stop */
#define MD_NACK                 (MD_STATUSBASE + 0x02U) /* IIC no ACK */
#define MD_BUSY1                (MD_STATUSBASE + 0x03U) /* busy 1 */
#define MD_BUSY2                (MD_STATUSBASE + 0x04U) /* busy 2 */

/* Error list definition */
#define MD_ERRORBASE            (0x80U)
#define MD_ERROR                (MD_ERRORBASE + 0x00U)  /* error */
#define MD_ARGERROR             (MD_ERRORBASE + 0x01U)  /* error argument input error */
#define MD_ERROR1               (MD_ERRORBASE + 0x02U)  /* error 1 */
#define MD_ERROR2               (MD_ERRORBASE + 0x03U)  /* error 2 */
#define MD_ERROR3               (MD_ERRORBASE + 0x04U)  /* error 3 */
#define MD_ERROR4               (MD_ERRORBASE + 0x05U)  /* error 4 */
#define MD_ERROR5               (MD_ERRORBASE + 0x06U)  /* error 5 */

/* End user code. Do not edit comment generated here */
```

貼り付けは必ずこのコメント間にしてください。

4) 各周辺機能の初期化関数の追加

4-1)r_cg_hardware_setup.c の R_Systeminit 関数のコードをコピーし R_MAIN_UserInit 関数に貼り付け

```
/***********************************************************************************************************
* Function Name: R_Systeminit
* Description  : This function initializes every macro.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC and software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Initialize non-existent pins */
    PORT5.PDR.BYTE = 0x70U;

    /* Set peripheral settings */
    R_CGC_Create();
    R_MTU3_Create();

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

4-1) コピー

4-2) 未使用端子の処理と R_CGC_Create 関数のコールをコードから削除

```
/***********************************************************************************************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */

    /* Enable writing to registers related to operating modes, LPC, CGC and software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Initialize non-existent pins */
    PORT5.PDR.BYTE = 0x70U;

    /* Set peripheral settings */
    R_CGC_Create();
    R_MTU3_Create();

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;

    /* End user code. Do not edit comment generated here */
}
```

4-1) 貼り付け

4-2) 削除

4-2) 削除

貼り付けは必ずこのコメント間にしてください。

4-3)他処理の追加

そのほかに無限ループより前に行う処理は 4-2)に追加する
(たとえばタイマのスタート関数コール、LED の点灯など)

6. main ファイル以外の修正

コード生成で作成された main ファイル以外のファイルを修正します。

1) r_cg_macrodriver.h

1-1) iodefine.h のインクルードパスを変更する (../../r_bsp/mcu/rx64m/register_access)

(変更前)

```
/***********************
Includes
***********************
#include "../iodefine.h"
#include <machine.h>
```

(変更後)

```
/***********************************************************************
Includes
***********************************************************************
#include "../../r_bsp/mcu/rx64m/register_access/iodefine.h"
#include <machine.h>
```

7. 割り込み関数の処理追加

周辺機能の割り込み処理は『r_cg_<周辺機能名>_user.c』で行います。
コード生成によって割り込み関数が生成されていますので、処理を追加してください。

8. 不要なファイルの削除

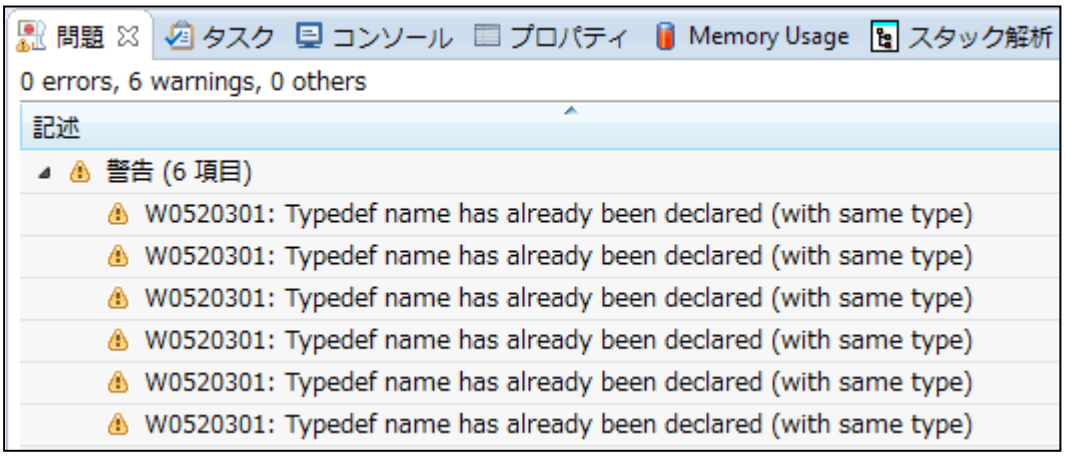
初期設定とクロック設定は BSP モジュールとコード生成の両方で設定できます。これらは競合するため、コード生成のファイルを e2 studio 上で削除してください。

r_cg_cgc_user.c、r_cg_cgc.c、r_cg_cgc.h、r_cg_dbsct.c、r_cg_hardware_setup.c、r_cg_intprg.c、r_cg_resetprg.c、r_cg_sbrk.c、r_cg_sbrk.h、r_cg_stacksct.h、r_cg_vect.h、r_cg_vecttbl.c

【注意】ビルド後に出るワーニングについて

同じファイルで<stdint.h>**と‘** r_cg_macrodriver.h **’をインクルードした場合、ビルド後、問題ビューに以下のワーニングが出力されます。**

このワーニングは、プロジェクト作成時に C99 **を選択していることにより生成される**<stdint.h>**とコード生成によって生成される‘** r_cg_macrodriver.h **’の両方で同様の** typedef **を宣言しているために出力されます。**



このワーニングを削除したい場合は、以下の手順を行ってください。
ただし、4.2 **設定変更時の設定手順、**4.3 **周辺機能追加時の設定手順でコード生成を再度行った際は再び以下の手順が必要です。**
そのため、すべてのコードが完成してからこの処理を行うことをお勧めします。

1) r_cg_macrodriver.h **の以下のコードを削除してください。**

```
/***********************************************************************
Typedef definitions
***********************************************************************/
#ifndef    TYPEDEF
typedef signed char          int8_t;
typedef unsigned char        uint8_t;
typedef signed short         int16_t;
typedef unsigned short       uint16_t;
typedef signed long          int32_t;
typedef unsigned long        uint32_t;
typedef unsigned short       MD_STATUS;
#define __TYPEDEF__
#endif
```

削除

2) r_cg_macrodriver.h **に** stdint.h **をインクルードしてください。**

```
/***********************************************************************
Includes
***********************************************************************/
#include "../../r_bsp/mcu/rx64m/register_access/iodefine.h"
#include <machine.h>
#include <stdint.h>
```

追加

4.2　設定変更時の設定手順

新規プロジェクト立ち上げ(上記 4.1)後に、クロックの設定や周辺機能の条件を変更する際の手順を説明します。図 4.2 に設定変更時のフローを示します。また、各手順の詳細な処理方法を説明します。

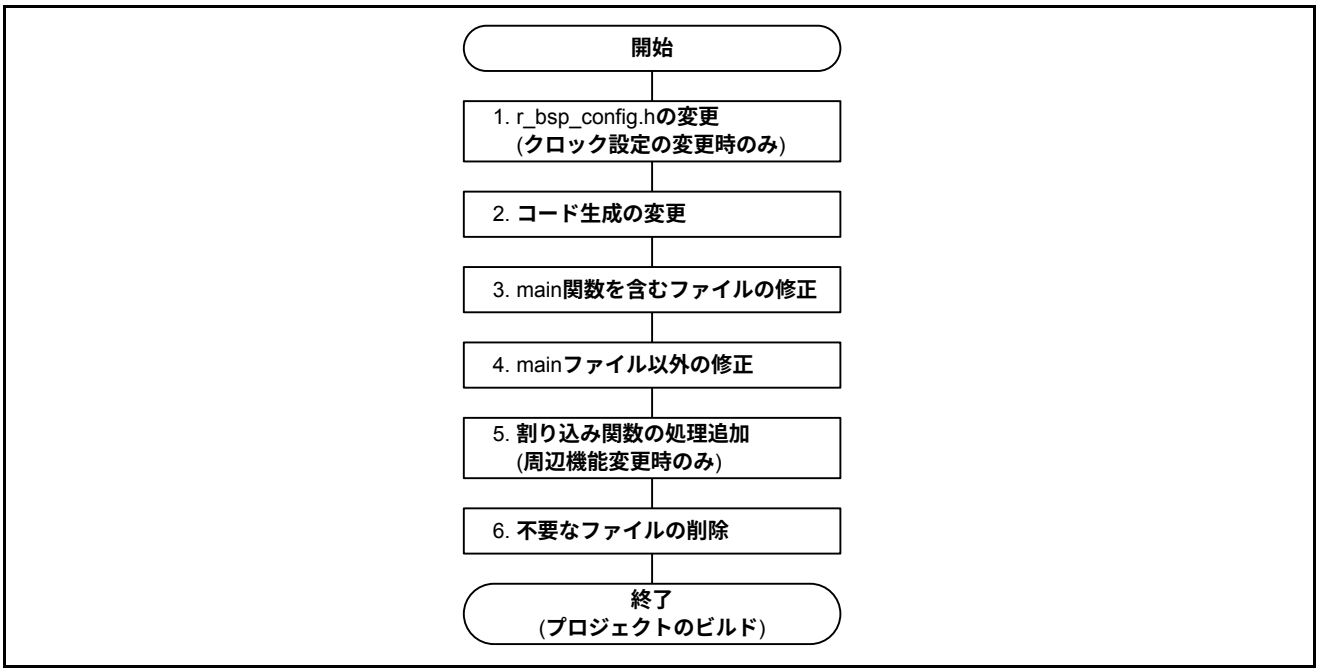


図 4.2　設定変更時の設定手順

1. r_bsp_config.h の変更

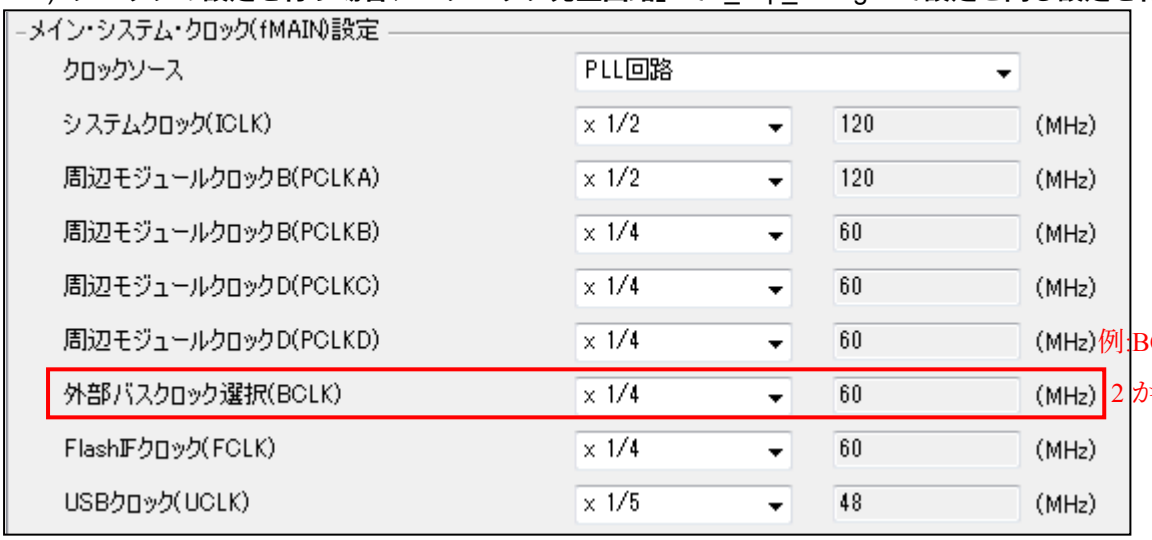
クロックの設定(PLL の分周、逓倍や PCLKB の分周など)を変更する場合、r_bsp_config.h を変更します。

```
/* Peripheral Module Clock D Divider (PCKD).
   Available divisors = /1 (no division), /2, /4, /8, /16, /32, /64
*/
#define BSP_CFG_PCKD_DIV                (4)

/* External Bus Clock Divider (BCLK).
   Available divisors = /1 (no division), /2, /4, /8, /16, /32, /64
*/
#define BSP_CFG_BCK_DIV                 (4)

/* Flash IF Clock Divider (FCK).
   Available divisors = /1 (no division), /2, /4, /8, /16, /32, /64
*/
#define BSP_CFG_FCK_DIV                 (4)
```

例:BCLK の分周を 2 から 4 分周に変更

2. コード生成の変更

1) クロックの設定を行う場合、「クロック発生回路」で r_bsp_config.h で設定と同じ設定を行います。



例:BCLK の分周を 2 から 4 分周に変更

2) 必要に応じて、周辺機能の変更を行います。

3) 『コードを生成する』をクリックします。

3. main 関数を含むファイルの修正

コード生成を再度行った場合、4-1. 新規プロジェクト立ち上げ手順 5 の【注意】に記載されているコメント間以外のコードは再生成されるため、以下の変更が必要です。この変更は新規プロジェクトの作成時にも行っていますが、コード生成を再度行うことで、再生成されます。

1) r_cg_cgc.h をインクルードするコードを削除

4. main ファイル以外の修正

コード生成を再度行った場合、以下の変更が必要です。これらは新規プロジェクトの作成時にも行っていますが、コード生成を再度行うことで、再生成されます。

1) r_cg_macrodriver.h

1-1) iodefine.h のインクルードパスを変更する (../../r_bsp/mcu/rx64m/register_access)

5. 割り込み関数の処理変更

必要に応じて割り込み関数の処理を変更してください。

6. 不要なファイルの削除

4-1. 新規プロジェクト立ち上げ手順 8 と同様の処理を行ってください。

4.3　周辺機能追加時の設定手順

新規プロジェクト立ち上げ後(上記 4.1)に、さらに周辺機能を追加する際の手順を説明します。図 4.3 に周辺機能追加時のフローを示します。また、各手順の詳細な処理方法を説明します。

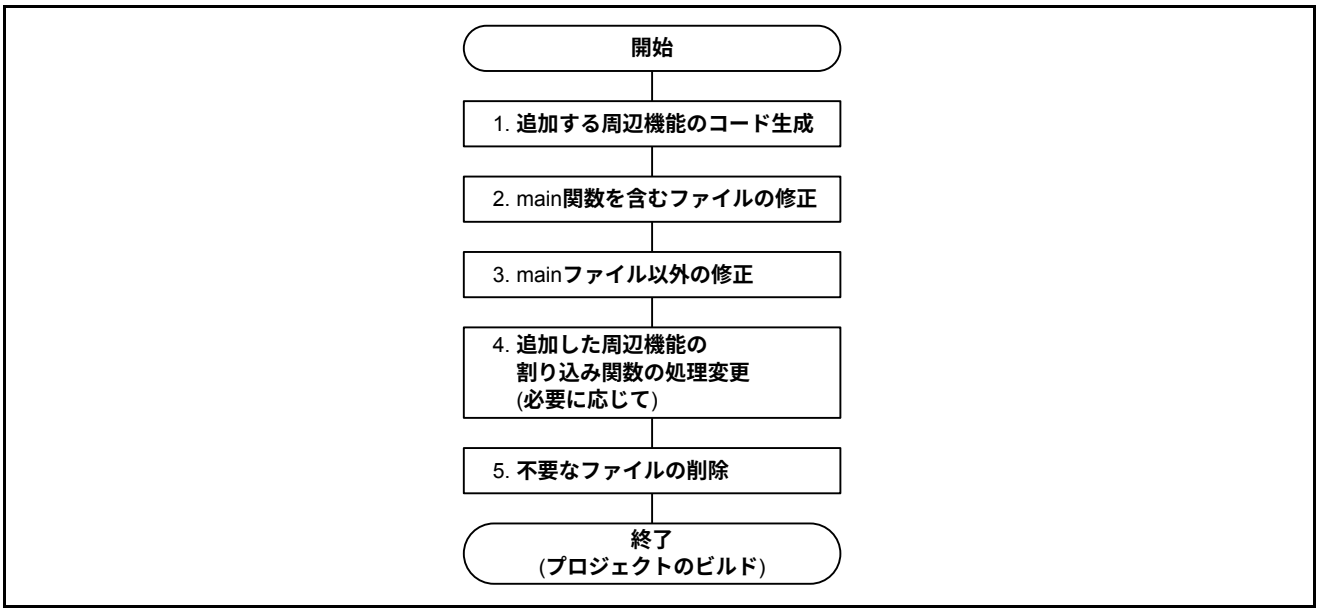


図 4.3　周辺機能追加時の設定手順

1. 追加する周辺機能のコード生成

追加する周辺機能のコードを生成します。

1) コード生成で追加する周辺機能の設定を行います

2) 『コードを生成する』をクリックします

2. main 関数を含むファイルの修正

追加した周辺機能のコード追加を行います。

1) r_cg_hardware_setup.c の R_Systeminit 関数で「追加された周辺機能の初期設定のみをコピーします

(周辺機能の初期設定関数名は『r_cg_<周辺機能>_Creat 関数』です)

```
/***********************************************************************************************
* Function Name: R_Systeminit
* Description  : This function initializes every macro.
* Arguments    : None
* Return Value : None
***********************************************************************************************/
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC and software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Initialize non-existent pins */
    PORT5.PDR.BYTE = 0x70U;

    /* Set peripheral settings */
    R_CGC_Create();
    R_LVD1_Create();
    R_MTU3_Create();

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

追加した周辺機能の初期設定のみをコピー

2) R_MAIN_UserInit 関数に、1)でコピーした関数を追加します。

```
/***********************************************************************************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */

    /* Enable writing to registers related to operating modes, LPC, CGC and software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Set peripheral settings */
    R_LVD1_Create();
    R_MTU3_Create();
    R_MTU3_C0_Start();
    LED0 = LED_ON;

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;

    /* End user code. Do not edit comment generated here */
}
```

コピーした関数の追加

これ以降の処理はコード生成ごとに行う処理です。

3) r_cg_cgc.h をインクルードするコードを削除

3. main ファイル以外の修正

コード生成を再度行った場合、以下の変更が必要です。これらは新規プロジェクトの作成時にも行っていますが、コード生成を再度行うことで、再生成されます。

1) r_cg_macrodriver.h

1-1) iodefine.h のインクルードパスを変更する (../../r_bsp/mcu/rx64m/register_access)

4. 追加した周辺機能の割り込み関数の処理変更

追加した周辺機能の割り込み関数がある場合、処理を追加してください。

周辺機能の割り込み処理は『r_cg_<周辺機能名>_user.c』で行います。

5. 不要なファイルの削除

4-1. 新規プロジェクト立ち上げ手順 8 と同様の処理を行ってください。

5. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RX64Mグループ　ユーザーズマニュアル ハードウェア編 Rev.1.00(R01UH0377JJ)
RX64M グループ以外の製品をご使用の場合は、それぞれ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。
(最新版をルネサス エレクトロニクスホームページから入手してください。)

テクニカルアップデート／テクニカルニュース

(最新の情報をルネサス エレクトロニクスホームページから入手してください。)

ユーザーズマニュアル：開発環境

RX ファミリ コンパイラ CC-RX v2.01.00 ユーザーズマニュアル RX コーディング編 Rev.1.00(R20UT2748JJ)
(最新版をルネサス エレクトロニクスホームページから入手してください。)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com

お問合せ先
http://japan.renesas.com/contact/

改訂記録	RX ファミリ　アプリケーションノート Firmware Integration Technology とコード生成ツールを 組み合わせて使用する場合の設定手順

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.12.26	—	初版発行
1.10	2015.03.02	全体	対象デバイスを RX64M グループから RX ファミリへ変更

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部 ROM、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、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/