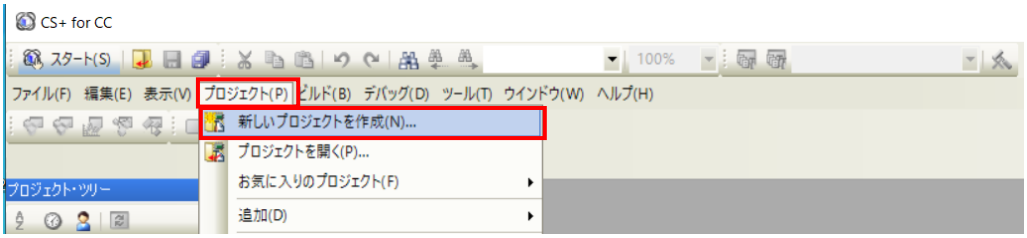
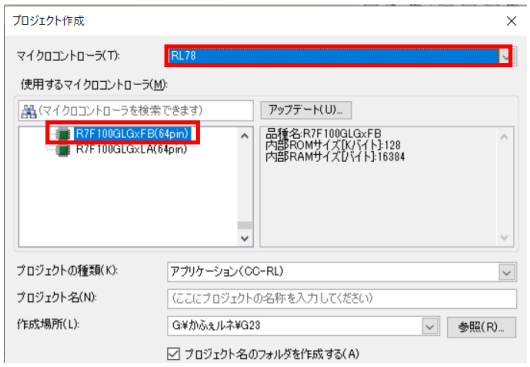
RL78/G23 のプロジェクト作成

CS+CC-RL およびスマート・コンフィグレータ(SC)がインストルされ、CS+に SC の exe ファイルパスが設定されている状態で、新しいプロジェクトを作成してみます。

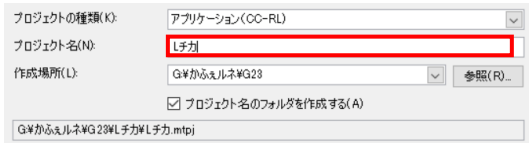
「プロジェクト(P)」のプルダウン・メニューで「新しいプロジェクトを作成(N)…」をクリックします。



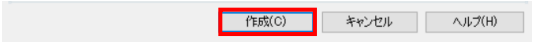
「マイクロコントローラ(T)」を「RL78」に設定し、「R7F100GLGxFB(64pin)」を選択します。



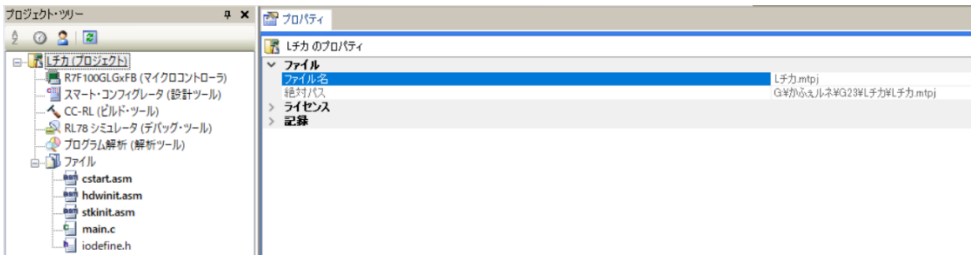
プロジェクトの名前は「L チカ」にしておきます。



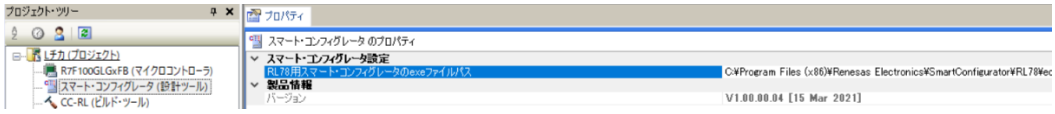
最後に、下の方の「作成(C)」をクリックして終了します。



これで、「L チカ」プロジェクトが作成されています。



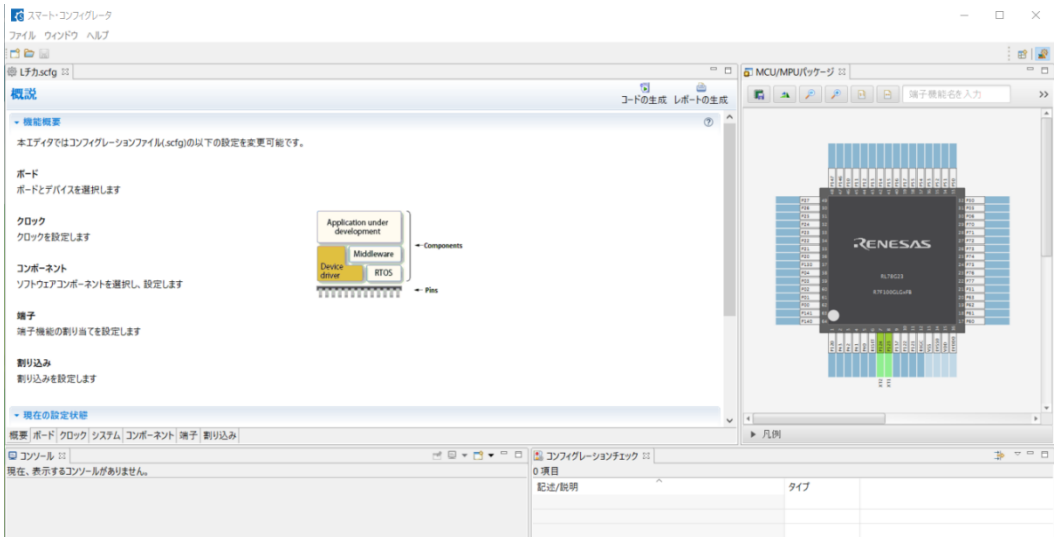
プロジェクト・ツリーの「スマート・コンフィグレータ(設計ツール)」を選択して、「スマート・コンフィグレータの設定」で「RL78 用スマート・コンフィグレータの exe ファイルパス」の設定を確認します。



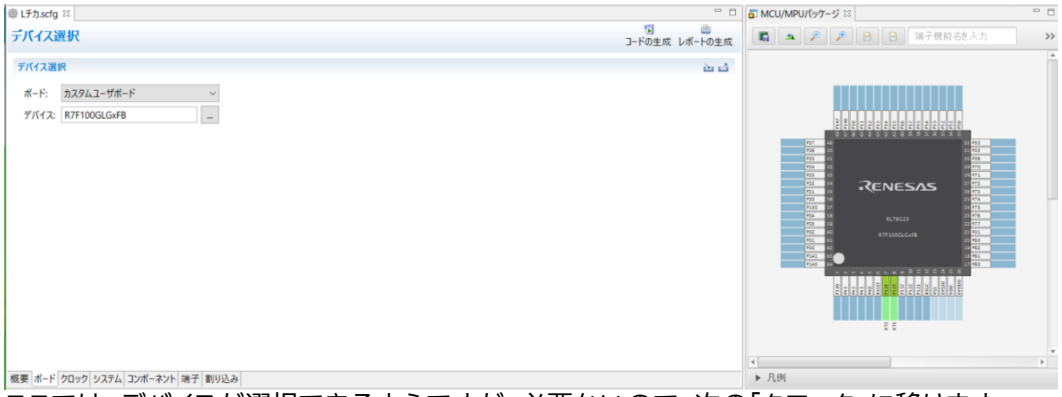
私の環境では、以下のような設定になっていました。

```
C:¥Program Files (x86)¥Renesas Electronics¥SmartConfigurator¥RL78¥eclipse¥SmartConfigurator.exe
```

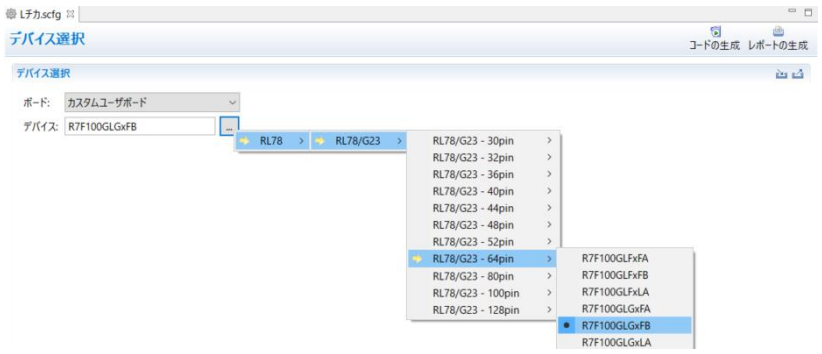
いよいよ、スマート・コンフィグレータを用いて、設定を行います。プロジェクト・ツリーの「スマート・コンフィグレータ（設計ツール）」をダブル・クリックして SC を起動します。起動すると下に示す概要ウィンドウが開きます。



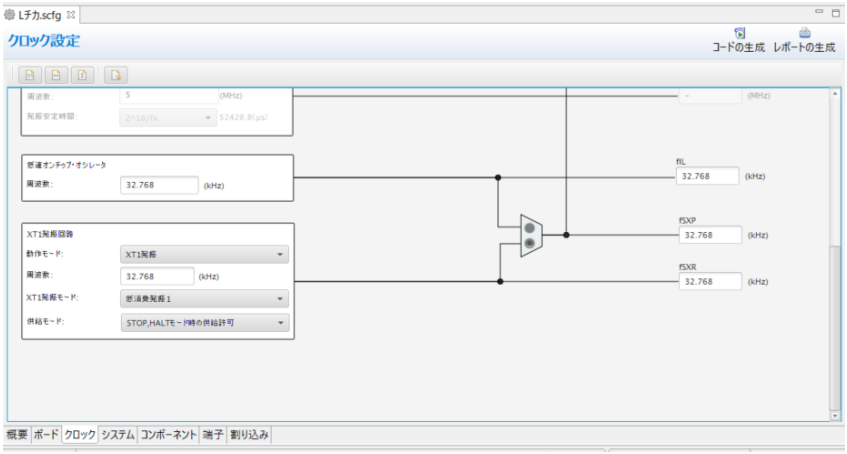
このウィンドウで下の「ボード」タグを選択すると、下に示す表示となります。



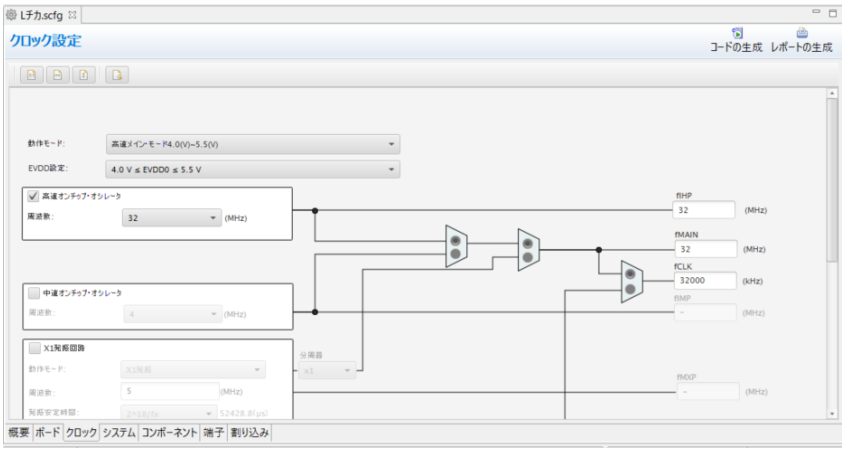
ここでは、デバイスが選択できるようですが、必要ないので、次の「クロック」に移ります。



「クロック」タグが開くと、クロックの系統図が表示されます。なぜか、「XT1 発信回路」は禁止設定はできないようです。



ここでは、下図に示す上側の項目を設定します。

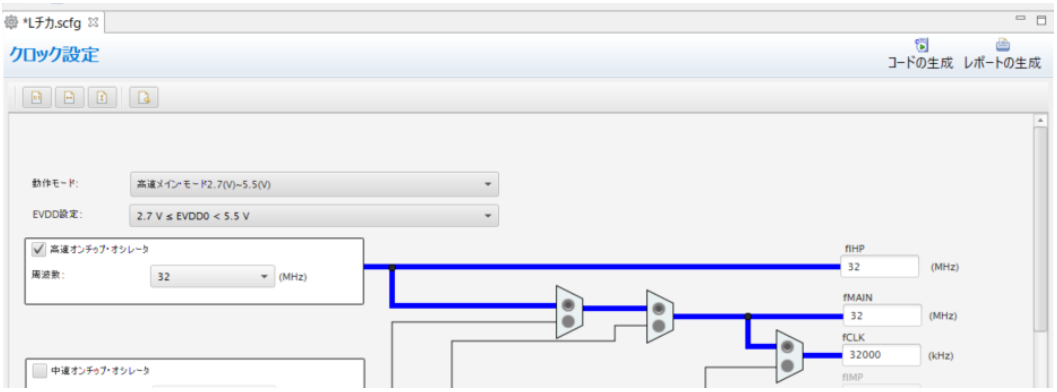


動作モードが「高速メイン・モード 4.0(V)～5.5(V)」になっているので、「高速メイン・モード 2.7(V)～5.5(V)」に変更します。ここらは、コード生成と同じ初期値のようです。

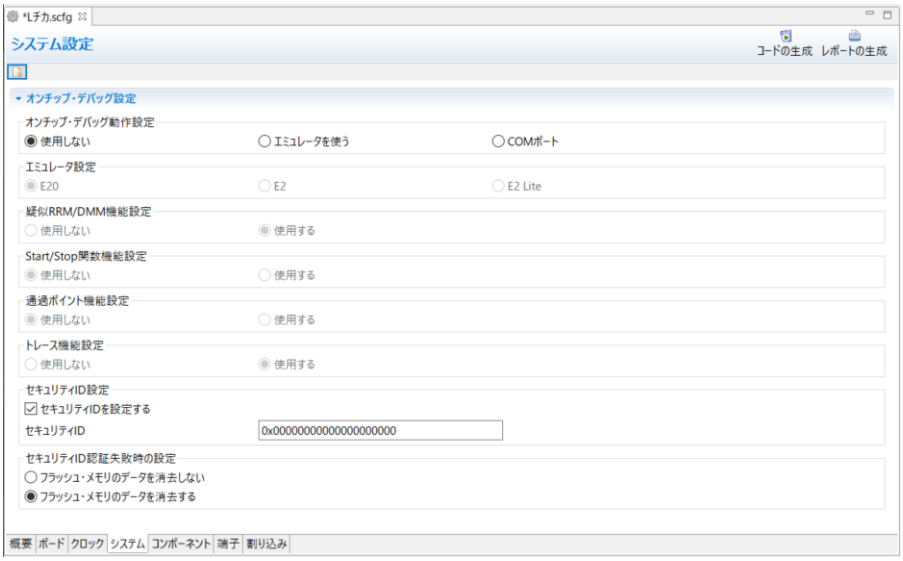
この変更を行うと、その下の「EVDD 設定」がエラーになるので、ここも「4.0V ≦ EVDD ≦ 5.5V」から「2.7V ≦ EVDD ≦ 5.5V」に変更します。

クロックの系統図で「高速オンチップ・オシレータ」にマウスカーソルを持っていくと、高速オンチップ・オシレータからのクロックが青い太線でどこに供給されるかが表示されます。

今回は、他のクロックは使用しないので、「クロック」タグはこれだけです。

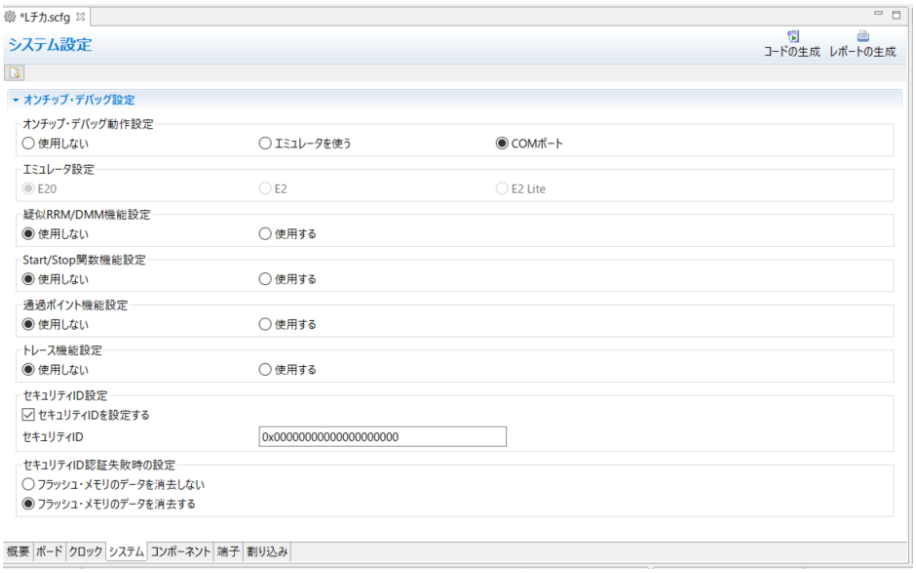


次は、「システム」タグです。

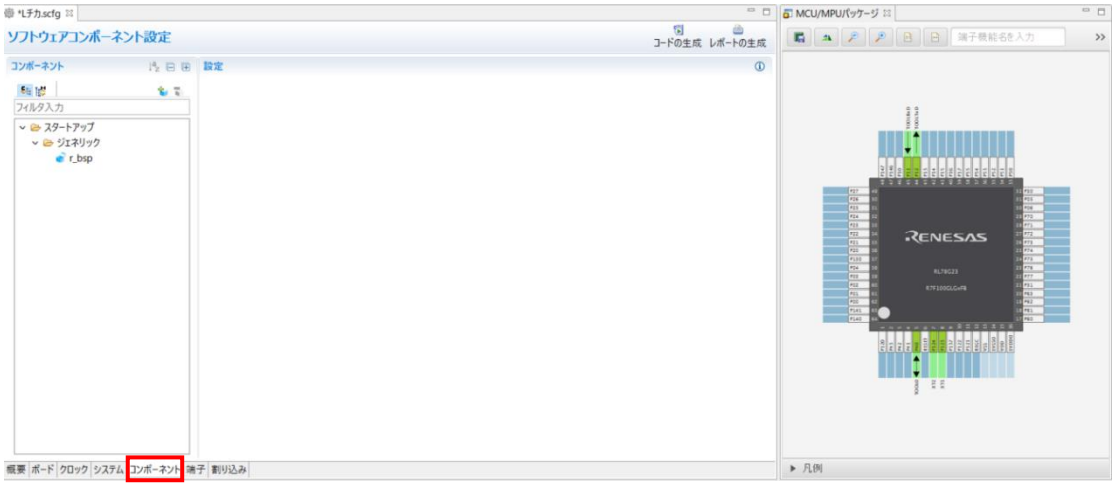


ディフォルトでは、オンチップ・デバッグは使用しないようになっているので、G23FPB では「COM ポート」を選択します。

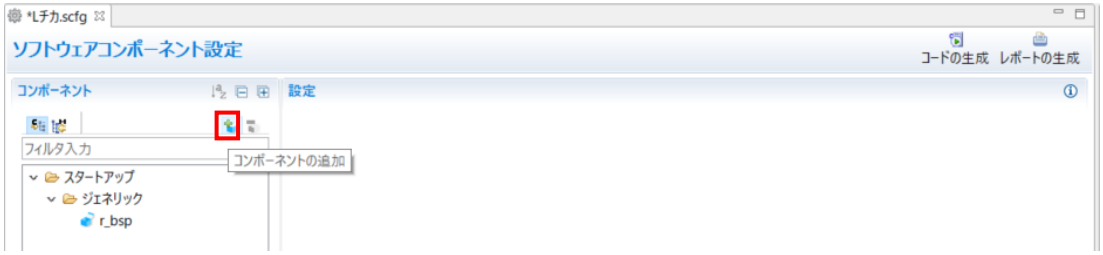
その他の項目は、ここではすべて、「使用しない」に設定しています。



次の「コンポーネント」タグが、実際に使用する内蔵周辺機能の設定になります。



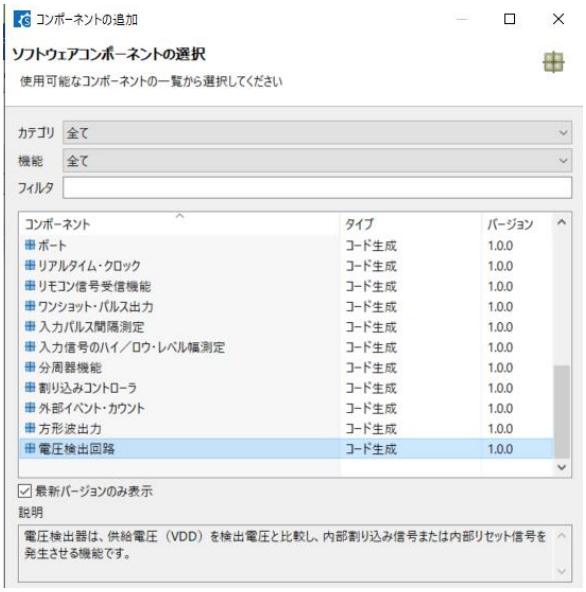
ここでは、左側の「コンポーネント」の「コンポーネントの追加ボタン」をクリックします。



「コンポーネントの追加ボタン」をクリックすると、「ソフトウェアコンポーネントの選択」ウィンドウ

が開きます。

コンポーネントの選択のスライダーを一番下まで動かすと、「電圧検出回路」があるので、これを選択します。

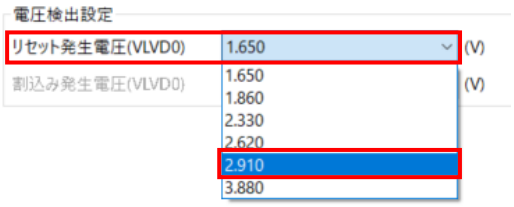


ウィンドウの下側にある「次へ(N)>」ボタンをクリックします。

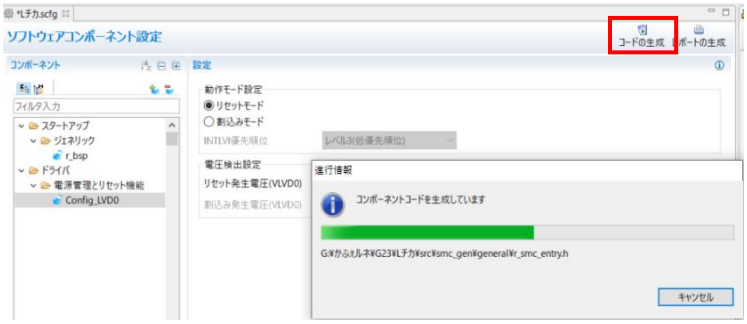
すると、電圧検出回路として LVD0 が選択されているので、このままウィンドウの下側にある「終了(F)」ボタンをクリックします。

下に示すように「設定画面」になります。ここでは、リセット発生電圧を見直します。

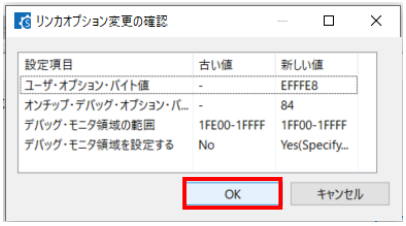
3.3V より低い電圧で近い電圧である 2.910(V)に設定します。



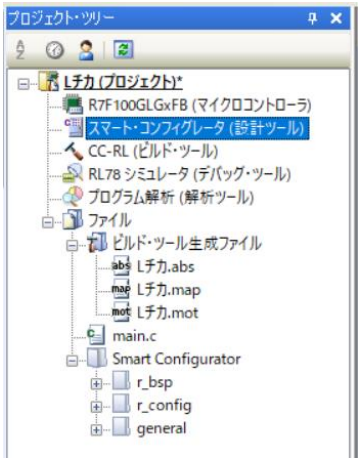
ここまでの設定は共通的な内容です。「コードの生成」ボタンをクリックすると、コードの生成が始まります。



「リンカオプション変更の確認」のポップアップ ウィンドウが開くので、「OK」をクリックします。



すると、CS+のプロジェクト・ツリーに以下の内容が生成されます。



さて、これからがSCを用いたプログラム作成です。まずは、L チカの仕様を決めることにします。G24FPB にはユーザ用の LED が 2 個とスイッチが 1 個搭載されています。これらを使って実現します。

P137 に接続されたスイッチを押すたびに、P53 に接続された LED1 は 0.125 秒→0. 25 秒→0. 5 秒→1 秒周期の順で、P52 に接続された LED2 は 1 秒→0. 5 秒→0. 25 秒→0.125 秒周期の順で点滅するものとします。このためにはこの半分の時間で点灯→消灯または消灯→点灯を切り替えていく必要があります。

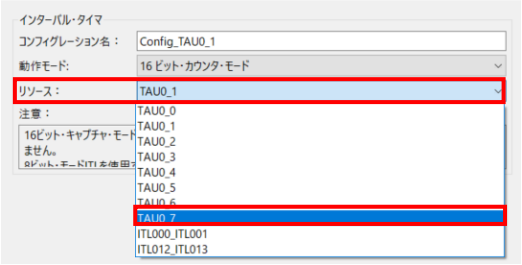
ソフトでタイマを作ってもいいのですが、RL78 ではタイマがいくつも使えるので、その中から 2 チャネルをインターバル・タイマとして使用します。最長の 1 秒周期を実現するには、0.5 秒周期での割り込みが必要ですが、TAU は 16 ビットのタイマなので、まずは 32MHz の fCLK をプリスケーラで分周しておくことになります。分周比はできるだけ小さくしたいので、プリスケーラの出力は、0.5 秒周期(2Hz)の 65536 倍の 131072Hz より低い 125kHz(fCLK を 256 分周したもの)に設定します。これを、タイマで 62500 分周すれば 2Hz(0.5 秒周期)が得られ、31250 分周すれば 4Hz(0.25 秒周期)が得られ、15625 分周で8Hz が得られ、7813 分周で 16Hz が得られます。これらの分周比(実際にはタイマに設定する値)を配列としてもち、スイッチを押した回数で選択することで、目的の点滅周期を得ます。

また、スイッチの状態のサンプリングのために 16 ミリ秒のインターバル・タイマも使用します。スイッチの状態を16ミリ秒ごとにサンプリングして、1 から 0 に変化したタイミングでスイッチが押されたと判断します。

以上の条件で、SC の設定を続けます。左側の「コンポーネント」の「コンポーネントの追加ボタン」をクリックして「ソフトウェアコンポーネントの選択」ウィンドウを開き、「インターバル・タイマ」を選択して、「次に(N)>」ボタンをクリックします。

インターバル・タイマの「動作モード」は「16 ビット・カウント・モード」を選択します。

「リソース」を「TAU07」に設定します。



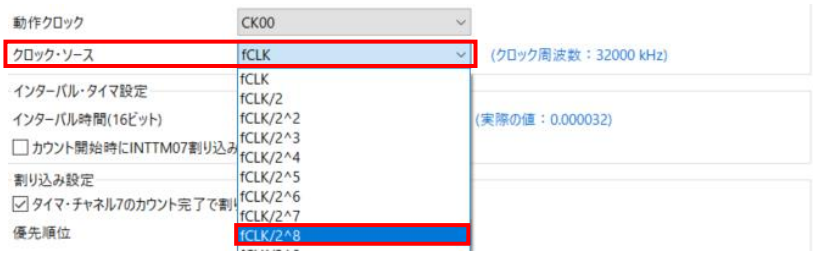
ウィンドウの下の「終了(F)」ボタンをクリックします。

これで、TAU0_7 の設定ウィンドウが表示されます。

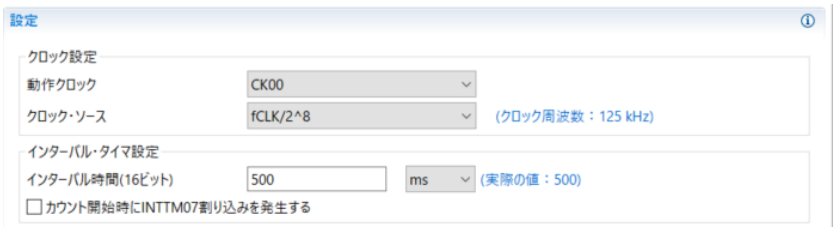
「インターバル・時間」の値を「500」に設定し、単位を「ms」に設定します。

すると、下に示すようにエラーとなります。ここらは、コード生成とは異なる動きです。

そこで、「クロック・ソース」を変更して、分周比を変更して、エラーが出ないようにします。ここでは「fCLK/2^8」を選択します。

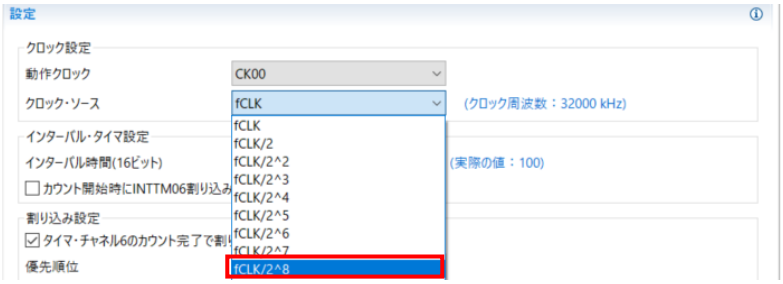


これで、エラーがなくなります。

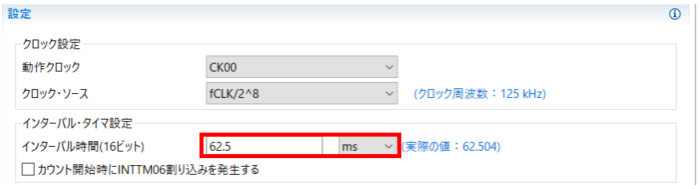


同様にして、TAU0_6 も「16 ビット・カウント・モード」に設定して追加します。

TAU0_6 の設定は、「クロック・ソース」を先ほどと同じく「fCLK/2^8」を選択します。なお、動作クロックは同じく CK00 のままです。

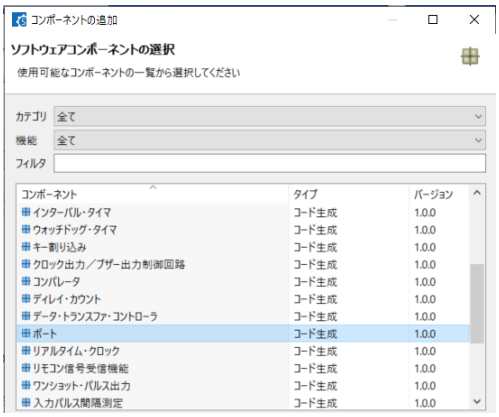


次に、「インターバル時間」を 0.0625 秒（62.5m 秒）に設定します。



タイマの最後にスイッチのサンプリング用に 16m秒のインターバル・タイマを同じようにして追加します。使用するタイマは TAU0_5 とします。ここまで終了した状態が、下の画面イメージです。

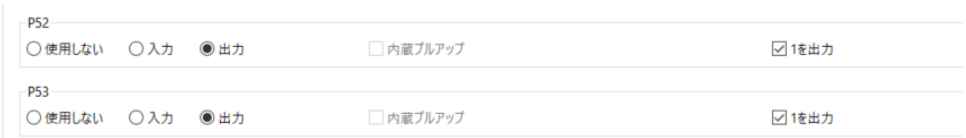
最後に、使用するポートの設定を行います。「コンポーネントの追加ボタン」をクリックして「ソフトウェアコンポーネントの選択」ウィンドウを開き、「ポート」を選択して、「終了(F)」ボタンをクリックします。



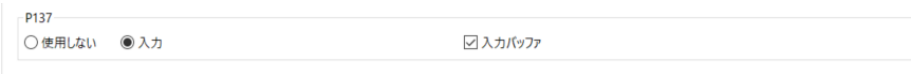
これで、使用するポートの選択画面が表示されます。

PORT5 と PORT13 にチェックを入れると「PORT5」と「PORT13」のタグが追加されます。

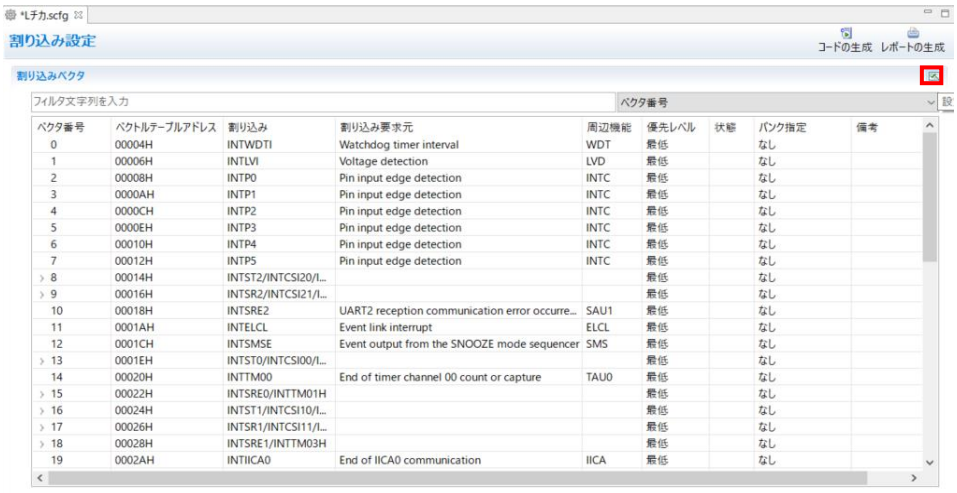
「PORT5」のタグを開いて、P52 と P53 を「出力」に設定して、「1 を出力」にチェックを入れます。



P137 については、「入力」選びます。「入力バッファ」の意味がよく分かりませんが、入力禁止に関連した内容と思われので、一応チェックを入れてあります。

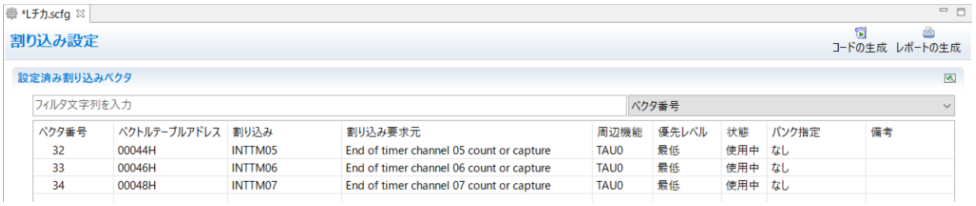


最後に、割り込みの設定です。何もしていない状態では全ての割り込みが表示されています。



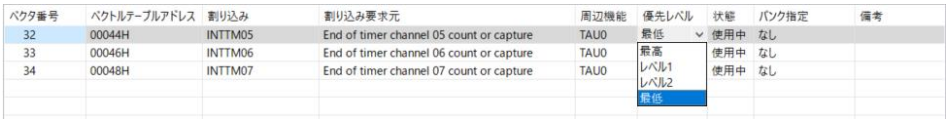
ベクタ番号	ベクトルテーブルアドレス	割り込み	割り込み要求元	周辺機能	優先レベル	状態	バンク指定	備考
0	00004H	INTWDTI	Watchdog timer interval	WDT	最低		なし	
1	00006H	INTLVI	Voltage detection	LVD	最低		なし	
2	00008H	INTP0	Pin input edge detection	INTC	最低		なし	
3	0000AH	INTP1	Pin input edge detection	INTC	最低		なし	
4	0000CH	INTP2	Pin input edge detection	INTC	最低		なし	
5	0000EH	INTP3	Pin input edge detection	INTC	最低		なし	
6	00010H	INTP4	Pin input edge detection	INTC	最低		なし	
7	00012H	INTP5	Pin input edge detection	INTC	最低		なし	
8	00014H	INTST2/INTCSI20/I...			最低		なし	
9	00016H	INTSR2/INTCSI21/I...			最低		なし	
10	00018H	INTSRE2	UART2 reception communication error occurre...	SAU1	最低		なし	
11	0001AH	INTELCL	Event link interrupt	ELCL	最低		なし	
12	0001CH	INTSMSE	Event output from the SNOOZE mode sequencer	SMS	最低		なし	
13	0001EH	INTST0/INTCSI00/I...			最低		なし	
14	00020H	INTTM00	End of timer channel 00 count or capture	TAU0	最低		なし	
15	00022H	INTSRE0/INTTM01H			最低		なし	
16	00024H	INTST1/INTCSI10/I...			最低		なし	
17	00026H	INTSR1/INTCSI11/I...			最低		なし	
18	00028H	INTSRE1/INTTM03H			最低		なし	
19	0002AH	INTIICA0	End of IICA0 communication	IICA	最低		なし	

これでは、わかりにくいので、右上の「設定した割り込みのみ表示」ボタンをクリックします。これで、使用する 3 つだけが表示されます。

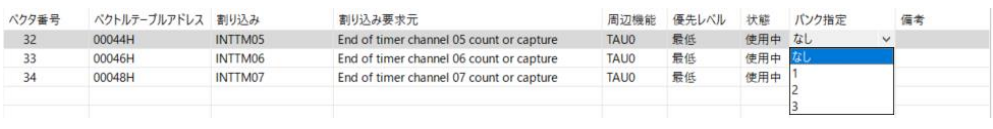


ベクタ番号	ベクトルテーブルアドレス	割り込み	割り込み要求元	周辺機能	優先レベル	状態	バンク指定	備考
32	00044H	INTTM05	End of timer channel 05 count or capture	TAU0	最低	使用中	なし	
33	00046H	INTTM06	End of timer channel 06 count or capture	TAU0	最低	使用中	なし	
34	00048H	INTTM07	End of timer channel 07 count or capture	TAU0	最低	使用中	なし	

ここで、選択できるのは、「優先レベル」(これは既に説明したコンポーネントの設定で設定可能でした)と



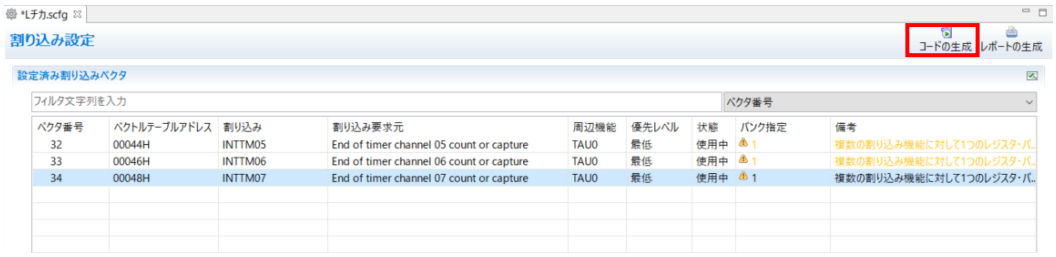
「バンク指定」です。使用するレジスタ・バンクの指定はここでしかできません(コード生成では、できなかったので、ここは進化しています)。



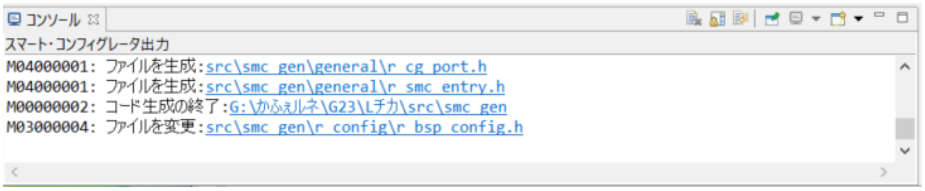
同じ優先順位の割り込みでは、同じレジスタ・バンクを使うのが当たり前なのですが、なぜか複数の割り込みに同じバンクを設定しようとするとワーニングが出ます。ここでは、ワーニングは無視しても、ちゃんと指定したレジスタバンクにはなるようです。ここでは全て「1」(レジスタ・バンク 1)に

設定しておきます。

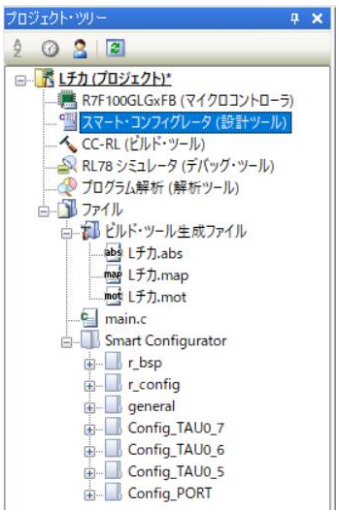
最後に、右上の「コードの生成」のアイコンをクリックして設定した結果をコードに反映させます。



SC のウィンドウの左下の「スマート・コンフィグレータ出力」に動作の状況が出力されます。



終わったら、ファイルを保存して、SC を終わります。CS+のプロジェクト・ツリーを見ると、SC で生成したコードが表示されています。



さてここからがプログラムの作成です。RL78/G23 のベアメタル環境で動作させるので、割り込みドリブンで処理を行うことにします。また、従来の RL78/G1x のコード生成の構成も引き継いでいます。基本的にソフトウェアを作成するというよりは、ハードウェアを制御するというのが近いかもしれません。

main.c の最初の部分は宣言です。SC が生成したヘッダファイルをインクルードし、タイマ関係の時間定数を定義し、プロトタイプ宣言をしています。

```
/***********************************************************************************
Includes
***********************************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_TAU0_7.h"
#include "Config_TAU0_6.h"
#include "Config_TAU0_5.h"

/***********************************************************************************
Pragma directive
***********************************************************************************/

/***********************************************************************************
Global variables and functions
***********************************************************************************/

#define Time500 ( 62500 )                   /* 0.5秒用のカウント値      */
#define Time250 ( 31250 )                   /* 0.25秒用のカウント値     */
#define Time125 ( 15625 )                   /* 0.125秒用のカウント値    */
#define Time63  (  7813 )                   /* 0.063秒用のカウント値    */

uint16_t const INTERVAL[2][4] =             /* インターバル時間テーブル */
{                                           /* LED1用のタイマデータ     */
    {Time63-1,Time125-1,Time250-1,Time500-1},
                                            /* LED2用のタイマデータ     */
    {Time500-1,Time250-1,Time125-1,Time63-1}
};


void R_MAIN_UserInit(void);

void main(void);
```

main 関数はプログラムとも言えないかもしれません。R_MAIN_UserInit 関数呼び出して、使用するタイマを起動し、あとは HALT 状態で割り込み待ちのループになっているだけです。

```
/***********************************************************************************
* Function Name: main
* Description  : This function main function.
* Arguments    : None
* Return Value : None
***********************************************************************************/
void main(void)
{
    R_MAIN_UserInit();                      /* タイマを起動する         */

    while (1)
    {
        HALT();                             /* HALTモードで割り込み待ち */
    }

}
```

最後が、タイマの起動処理を行っている R_MAIN_UserInit 関数です。

```
void R_MAIN_UserInit(void)
{
    R_Config_TAU0_5_Start();                /* スイッチ確認用タイマ起動 */
    R_Config_TAU0_6_Start();
    R_Config_TAU0_7_Start();
    EI();
}
```

次は、最もプログラムらしい INTTM05（スイッチのサンプリングおよび押下検出）処理です。

下に示す宣言部分は SC が生成したものです。

```
/***********************************************************************************************************************
Includes
***********************************************************************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_TAU0_5.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

/***********************************************************************************************************************
Pragma directive
***********************************************************************************************************************/
#pragma interrupt r_Config_TAU0_5_interrupt(vect=INTTM05, bank=RB1)
/* Start user code for pragma. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

プログラムとして宣言したのは以下の 3 行です。

```
/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
volatile    uint8_t  g_SW_state;        /* スイッチのステータス     */
volatile    uint8_t  g_SW_count;        /* スイッチの押下回数       */

extern uint16_t const INTERVAL[2][4];
```

下に示すのが INTTM05 の割り込み処理本体です。オンボードのスイッチの状態を読み出し、変数 g_SW_state に右から取り込んでいます。スイッチが押されたら、表示状態（変数 g_SW_count）を更新して、新しい LED 点滅用の時間を設定しています。

```
/***********************************************************************************************************************
* Function Name: r_Config_TAU0_5_interrupt
* Description  : This function is INTTM05 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
static void __near r_Config_TAU0_5_interrupt(void)
{
    /* Start user code for r_Config_TAU0_5_interrupt. Do not edit comment generated here */

    g_SW_state <<= 1;                   /* ステータスを左シフト     */

    if ( 0 != P13_bit.no7 )             /* スイッチの状態をチェック */

    {
        ++g_SW_state;                   /* 新しい1の状態を設定      */
    }

    if ( 0x02 == ( 0x03 & g_SW_state) ) /* スイッチが押下された     */
    {
        ++g_SW_count;                   /* 点滅状態を変更する       */
        g_SW_count &= 0x03;             /* 折り返しを行う           */

        TDR06 = INTERVAL[0][g_SW_count]; /* LED1の周期変更          */
        TDR07 = INTERVAL[1][g_SW_count]; /* LED2の周期変更          */

    }

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

LED を実際に点滅させている INTTM06 と INTTM07 は単純です。下は INTTM06 の例ですが、LED をドライブしているポートのデータを反転しているだけです。

```
static void __near r_Config_TAU0_6_interrupt(void)
{
    /* Start user code for r_Config_TAU0_6_interrupt. Do not edit comment generated here */

    P5_bit.no3 ^= 1;            /* LED1を反転                       */

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

最後に、P137 の「入力バッファ」の設定は、次に示すように、ポート・デジタル・インプット・ディス

エーブル(PDIDIS)レジスタをクリアしているだけのようです。初期値が 0 なので、意味はないかもしれませんが。

```
PDIDIS13 = _00_PDIDISn7_INPUT_BUFFER_ON;
```

ビルドして、G23FPB にダウンロードして実行させます。使用するデバッグ・ツールは「RL78 COM Port」を選択し、通信ポートとして COM3 を選択してあります。

ダウンロードして、実行すると、LED1 が早く点滅し、LED2 がゆっくり点滅しているのが確認できます。また、スイッチを押すたびに点滅速度が変わることが確認できます。

確認はできましたが、オンボードの LED ではあまり面白くありません。次は外部の LED を制御することにします。

以上