サウンドエフェクタ
ソフトウェア編(4)

～デバッグ、ゼロからプログラムを作成する場合に～

取扱説明書

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子
REV.1.0.0.0

ー目　次ー

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

表記の意味

警告 取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される

注意 取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

絵記号の意味

	**一般指示** 使用者に対して指示に基づく行為を強制するものを示します		**一般禁止** 一般的な禁止事項を示します
	**電源プラグを抜く** 使用者に対して電源プラグをコンセントから抜くように指示します		**一般注意** 一般的な注意を示しています

警告

以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

⚠注意

以下のことをされると故障の原因となる場合があります。

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

アクセスランプがある製品では、アクセスランプ点灯中に電源の切断を行わないでください。

製品の故障の原因や、データの消失の恐れがあります。

本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

概要

当社製品、サウンドエフェクタ向けのプログラムを作成する際の手順、注意点等に関して記載を行っている資料となります。製品のハードウェアに関しては、「サウンドエフェクタ　取扱説明書」に記載がありますので、そちらも合わせて参照頂きたく。

個別のプログラムに関しては、プログラム毎のドキュメント（アプリケーションノート）を参照ください。

ソフトウェア編の資料は、

・サウンドエフェクタ　ソフトウェア編(1)　取扱説明書[本書]　～波形を生成してみる～
・サウンドエフェクタ　ソフトウェア編(2)　取扱説明書　～入力信号を加工して出力してみる～
・サウンドエフェクタ　ソフトウェア編(3)　取扱説明書　～FFT/逆 FFT をライブラリ関数で処理してみる～
・サウンドエフェクタ　ソフトウェア編(4)　取扱説明書[本書]　～デバッグ、ゼロからプログラムを作成する場合に～

に分かれています。

本資料では、テンプレートで用意されている部分に関して説明します。

テンプレートを変更してカスタムしたい、入力バッファや出力バッファの構成を大きく変えたい等のケースでは、ソフトウェア編(1)~(3)では、ブラックボックス化していた部分に手を入れる必要があります。その様な場合、本書を参考にゼロからプログラムを構築してみてください。

参考文献

プログラムを作成するターゲットが、ルネサスエレクトロニクス製の RX651 マイコン(R5F5651EDDFM)となりますので、マイコンのハードウェアマニュアル

「RX65N グループ、RX651 グループ　ユーザーズマニュアル　ハードウェア編」

は、参照できる様にしておいてください。
（CS+のスマートマニュアルや、ルネサスエレクトロニクス Web からダウンロード可能です）

プログラム作成時、コードの自動生成機能である、「スマート・コンフィグレータ」を使用する場合は、スマート・コンフィグレータのマニュアルも参照ください。

また、音声データの送受信の部分に手を入れる際は、マイコンと通信を行う、オーディオ CODEC チップ（AK4554, 旭化成）のデータシートも参照頂きたく。

1. クロック設定

クロック設定は、マイコンを動作させるのに必要な基本的なタイミングを決める設定です。

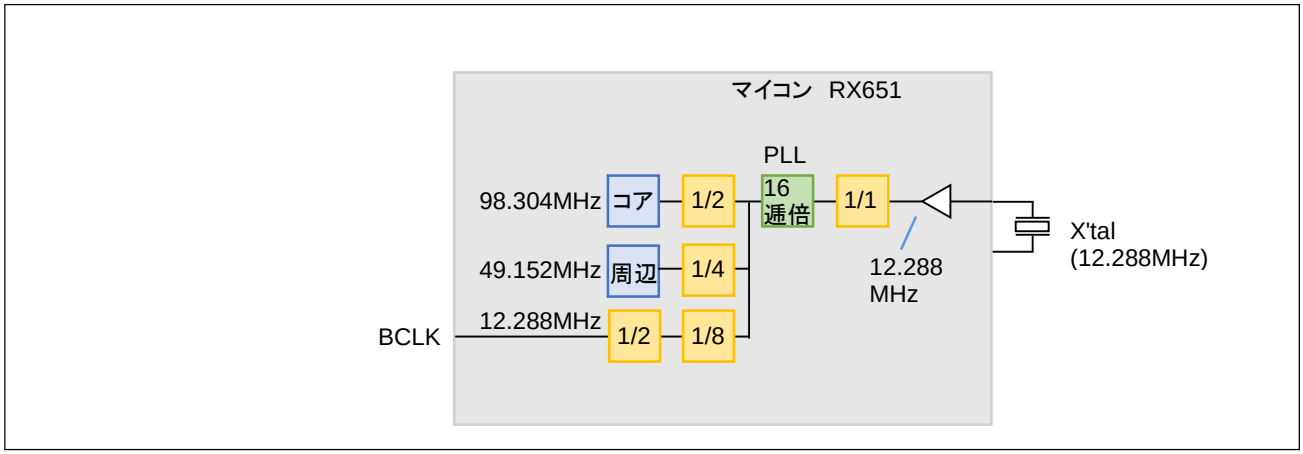


図　1-1　クロック系

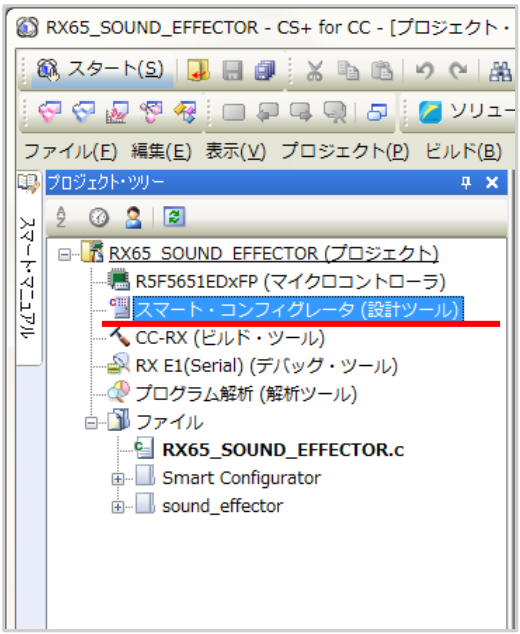
サウンドエフェクタでは、図 1-1 のクロック系となっています。オーディオのサンプリング周波数、48kHz をベースにし、オーディオ CODEC 側で、256fs(48kHz x 256 = 12.288MHz)のクロックが必要になりますので、マイコン側のクロック源として、オーディオ CODEC 上都合の良い 12.288MHz を採用しています。

12.288MHz を入力とし、マイコン内蔵の PLL で 16 逓倍し、その 1/2 のクロックをマイコンの CPU コアクロックとしています。マイコンの周辺回路を駆動するクロックは、コアクロックの 1/2。オーディオ CODEC に供給するマスタクロックは、PLL の出力を 1/8, 1/2 分周し生成する事とします。

CD 内の、
TEMPLATE¥RX65_SOUND_EFFECTOR_TEMPLATE
フォルダを PC 上のストレージにコピーし、

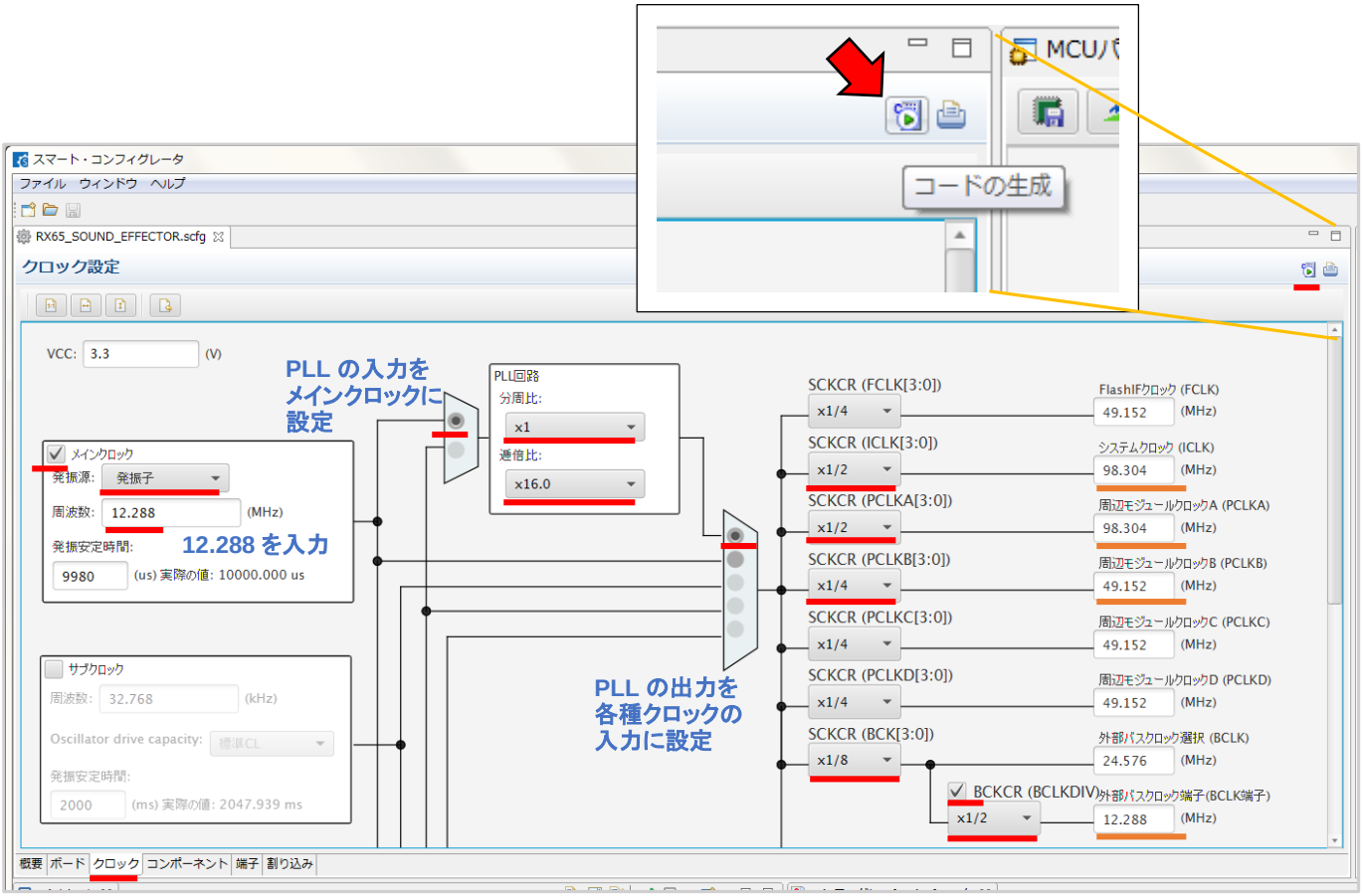
[path]¥RX65_SOUND_EFFECTOR_TEMPLATE¥RX65_SOUND_EFFECTOR.mtpj

をダブルクリックして、CS+を起動してください。



プロジェクトツリー内にある、スマート・コンフィグレータ（設計ツール）をダブルクリックで開いてください。

※予め「RX スマート・コンフィグレータ」「RX スマート・コンフィグレータ通信プラグイン」のインストールが必要です
製品の取扱説明書を参照の上、インストールを行ってください



クロックの設定は、クロックのタブを押してください。

搭載水晶振動子の値（本製品では、12.288MHz）を、メインクロックの周波数の欄に入力し、ラジオボタンやプルダウンメニューで各種クロック値を設定します。

本来マイコンのクロックの設定は、クロック関連のレジスタ値を決められた順番で設定する必要がありますが、スマート・コンフィグレータを使用すると、GUI で選択するだけで、設定が可能です。

設定後、右上の「コード生成」のボタンを押すと、設定した項目が反映されたソースコードが出力されます。

（非常に簡単に、クロックの設定を完了させる事ができます。）

CS+プロジェクトでのマイコン型名の選択に関して

本製品に搭載されているマイコンは、R5F5651EDDFM(64pin)となりますが、上記スマート・コンフィグレータの設定では、R5F5651EDxFP(100pin)のマイコンを選択して設定を行っています。

64pin のマイコンを選択した場合、スマート・コンフィグレータの設定で、BCLK 端子の設定が出てこない（マスクされます）ため、BCLK の分周の設定等はユーザが手動で設定する必要があります。100pin のマイコンを設定した場合、BCLK の設定を GUI で設定できるため、ここでは 100pin のマイコンを選択しています。

※100pin のマイコンを選択しているため、スマート・コンフィグレータでの、マイコンのピン番号の情報は正しくありません
端子名（PC7 等）で端子を識別する様にしてください

2. オーディオ CODEC チップとの通信

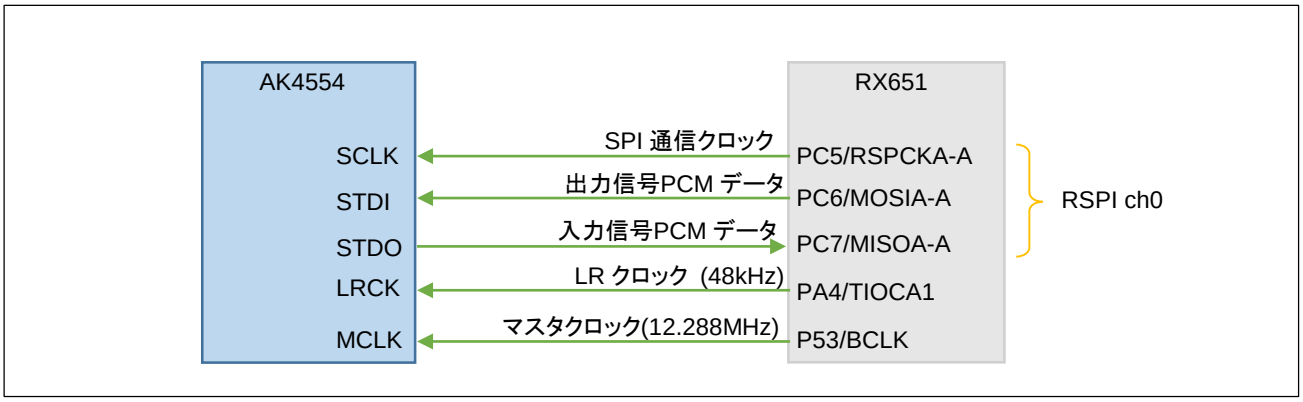


図 2-2 マイコンーオーディオ CODEC 間接続

音声信号の A/D 変換及び D/A 変換を行うオーディオ CODEC との通信は、

・マスタクロック(12.288MHz)の供給
・LR クロック(48kHz)の供給
・ディジタル化された音声データの送受信

から成り立っています。

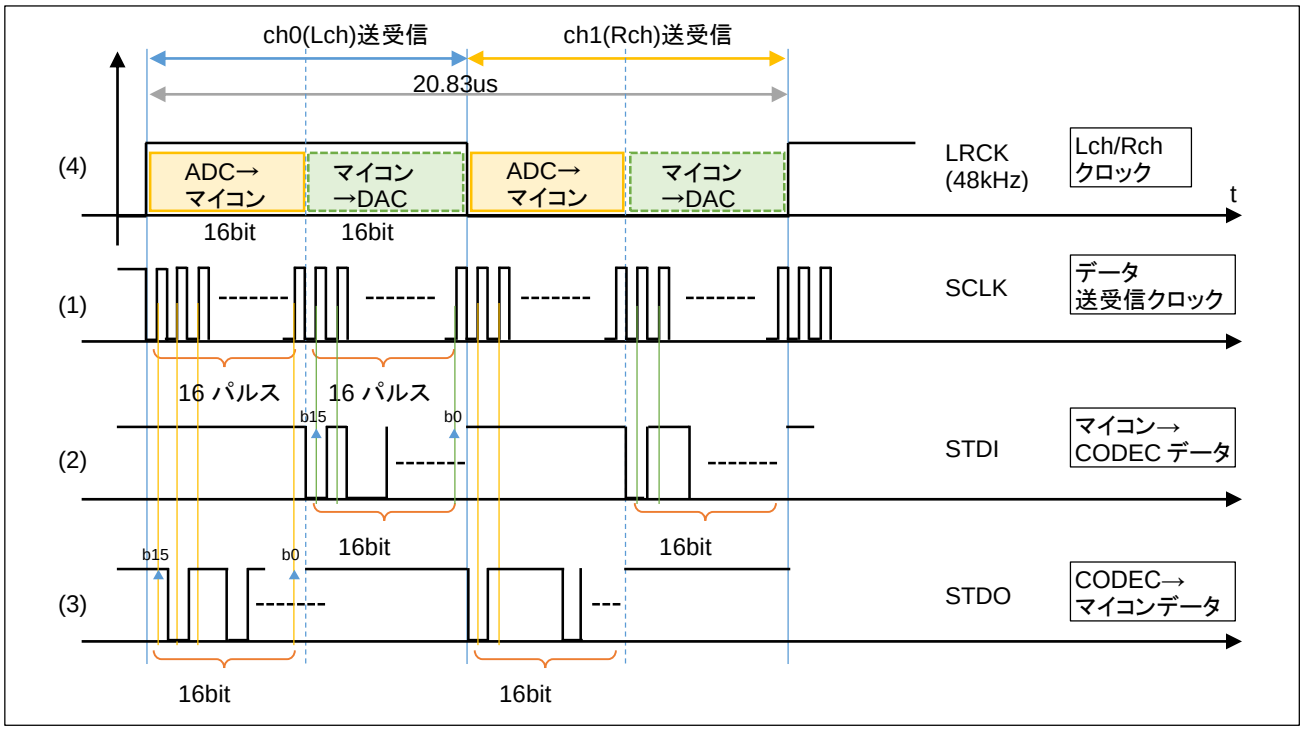


図 2-3 オーディオ CODEC データ送受信波形

マイコンとオーディオ CODEC のデータ送受信は、4 本の信号線を用い下記の様に行います。

・LRCK が H の期間は Lch のデータ送受信を行う
・LRCK が L の期間は Rch のデータ送受信を行う
・データ送受信クロックは、1 回, 1ch のデータ送信で、32 クロック（送信 16 クロック、受信 16 クロック）
・32 クロックの前半 16 クロックは、CODEC の ADC で変換したオーディオデータをマイコンに送信
・32 クロックの後半 16 クロックは、マイコンで演算したオーディオデータを CODEC に送信
・送受信するデータは 48kHz, 16bit リニア PCM のデータ（符号付き 16bit、2 補数表現）
・SCLK の rise エッジで CODEC, マイコン共データを取り込む動作
・データは、MSB ファーストで送受信（b15 が先頭、b0 が最後）

マイコン側のプログラムとしては、RSPI（ルネサス SPI）の機能を使用して、CODEC とデータの送受信を実現します。

(a)
マスタクロックは、図 2-3 には記載していませんが、オーディオ CODEC チップをサンプリング周波数 48kHz で使用する場合、12.288MHz の供給が必要となります。このクロックは、マイコンのバスクロック(BCLK)を供給する事により実現します。

(b)
LR クロックは、マイコンの TPU タイマを使用できる様、端子を割り当てていますので、TPU タイマで 48kHz を生成すれば良い事となります。

(c)
データ送受信を行う際、マイコン側は RSPI の機能を使用して、オーディオ CODEC のチップと通信を行います。48kHz の周期（20.83us）に 2 回、16bit のデータを送受信する様にします。

(d)
オーディオ CODEC から受信したデータを、受信バッファ（g_inbuf, 約 2 秒分）にコピーする。この部分は、マイコンの DMAC(Direct Memory Access Controller)の機能を使用する事とします。

よって、マイコン側は、(a)(b)(c)(d)の 4 点を実現するプログラムを作成する事となります。

(a)マスタクロックの供給

マスタクロックは、1 章のクロック設定の際、既に 12.288MHz の BCLK を生成する様設定済みです。
プログラム上記載する必要があるのは、P53/BCLK 端子を BCLK 出力端子に割付を行い、BCLK クロックを出力する部分となります。

当該端子は、P53（汎用 I/O ポート）／BCLK（バスクロック）兼用の端子となっていますので、BCLK 側の機能を使う様設定を行います。

具体的なコードは、sound_effector.c 内の、sound_effector_init()（初期化関数）内に記載があります。

sound_effector.c（抜粋）

```
//BLCK=12.288MHz出力
SYSTEM.PRCR.WORD = 0xA503;

SYSTEM.SCKCR.BIT.PSTOP1 = 1;
while(SYSTEM.SCKCR.BIT.PSTOP1 != 1) nop();

//BLCK強制出力
MPC.PFBCR0.BIT.BCLKO = 1;

SYSTEM.SCKCR.BIT.PSTOP1 = 0;
while(SYSTEM.SCKCR.BIT.PSTOP1 != 0) nop();

SYSTEM.PRCR.WORD = 0xA500;
```

SYSTEM.PRCR.WORD は、プロテクトレジスタとなります。クロック関連の設定は、容易に変更できない様保護されていますので、最初に保護を解除しています。

SYSTEM.SCKCR.BIT.PSTOP1 = 1; は、BCLK 端子出力を停止する設定です。

MPC.PFBCR0.BIT.BCLKO = 1; は、BCLK を常に出力する設定です。

本来 BCLK は、バスクロック、SDRAM や外部バス向けのクロックとなりますが、今回はバス（アドレスバス、データバスの信号を使い周辺回路にアクセスする機能）は使用せず、CODEC のクロック源として BCLK のみ出力したいので、BCLKO=1（バス機能の使用有無に拘わらず、BCLK 出力を有効化する設定）としています。

SYSTEM.SCKCR.BIT.PSTOP1 = 0; は、BCLK 端子を動かす設定です。

最後に、クロック関連の設定を変更できない様、プロテクトを有効化しています。

マスタクロックの供給に関するコードは、上記のみです。

BCLK クロックの使用に関して

本来 BCLK クロックは、バスクロックであり、外部に接続されたメモリ等に供給する目的で出力されます。

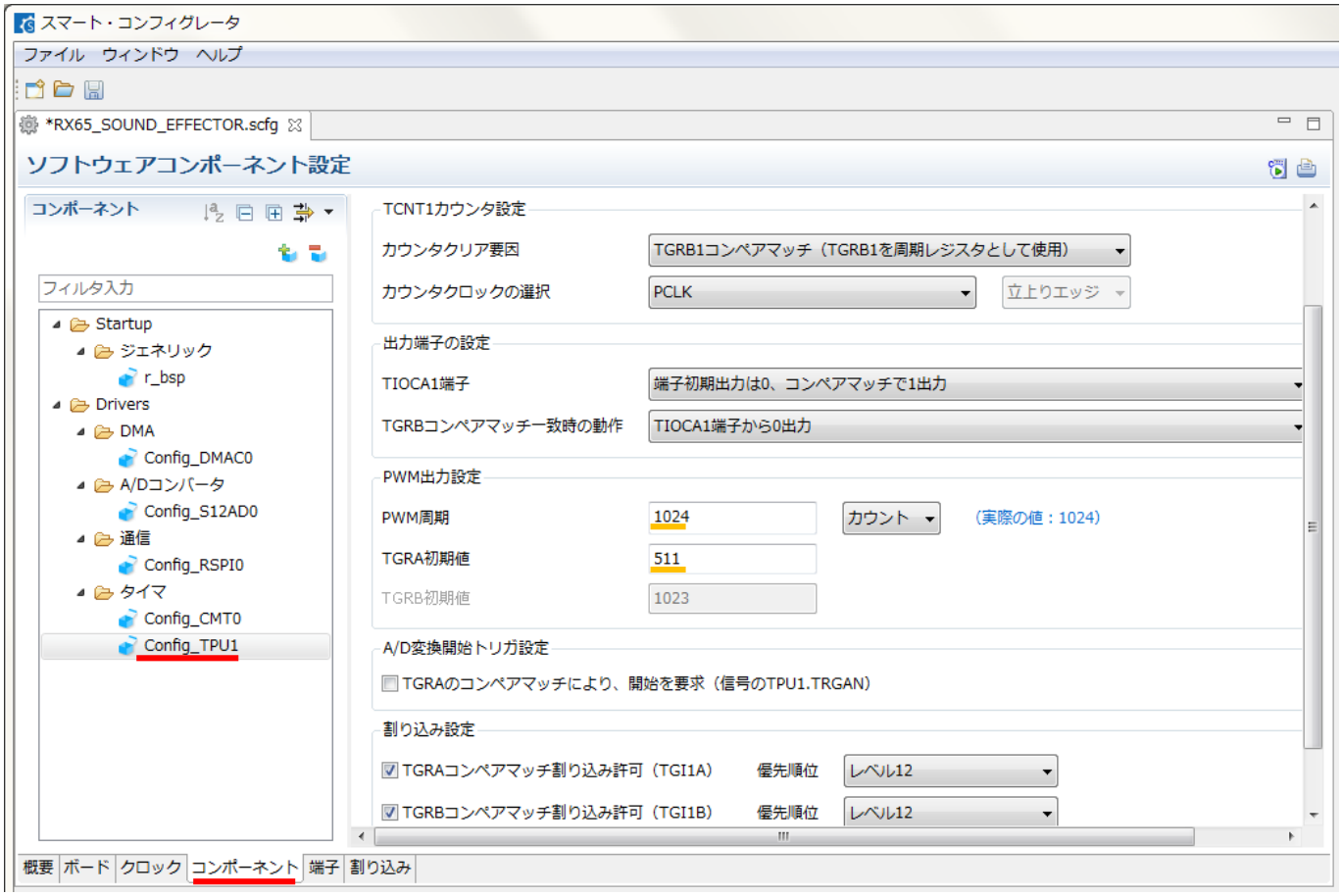
RX651 マイコンは、100pin 以上のピン数のマイコンでは、外部バス機能が使用可能です。

本製品に搭載されている、64pin のマイコンでは、外部バス機能は端子数の都合上使用できないのですが、BCLK クロックは設定を行えば、クロック出力が行えます。(本製品では、外部バス機能は使用せず、本来外部バス機能に付随する BCLK 端子のみを使用している形となります。)

(b)LR クロックの供給

LR クロックは、CODEC チップに対し、サンプリングレートとなる、48kHz を供給する部分です。

今回は、マイコンの機能として、TPU という 16 ビットタイマを使用して、48kHz の信号を生成します。



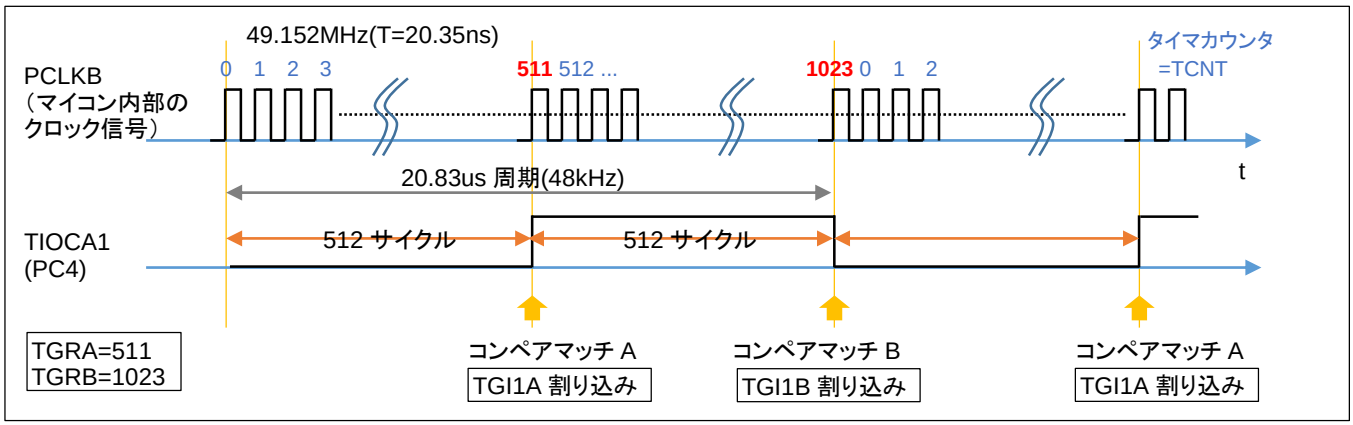


図 2-4 TPU 出力波形

TPU1 タイマ機能を上記の様に設定する事で、TPU1 で 48kHz の信号を生成し、TIOCA1 端子(PC4)からクロック波形を出力できます。

プログラムとして書き下す必要があるのは、以下となります。

sound_effector.c（抜粋）

```
#include "Config_TPU1.h"

（中略）

   R_Config_TPU1_Start();//48kHz
```

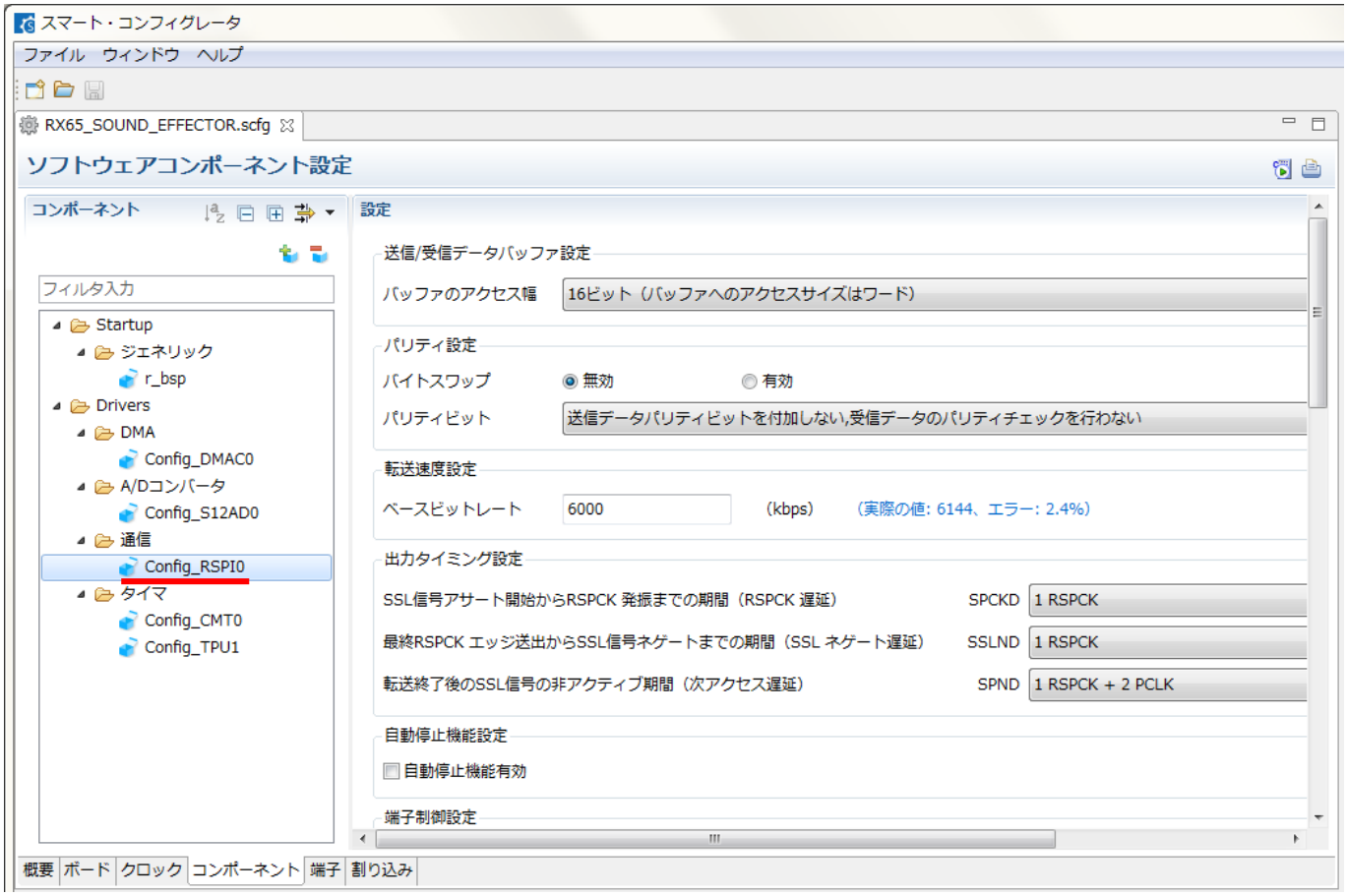
ユーザが書くコードは、タイマをスタートする記述のみで、他（タイマの周期やタイマに入力するクロックの設定等）は、GUI の設定に基づきスマート・コンフィグレータが生成してくれます。

なお、TPU1 の設定で、48kHz のクロックの切り替わりタイミングで、割り込み（TGI1A 及び TGI1B）が掛かる様設定しています。これは、次に説明する、RSPI を使用したデータ転送のトリガとして使用するためです。

(c)RSPI を使用したデータ送受信

RSPI は、ルネサス SPI モジュールで、一般的には、SPI(Serial Peripheral interface)と呼ばれる通信モジュールです。「クロック」「データ出力」「データ入力」「イネーブル信号」を用い、クロック同期でデータの送受信を行います。

RSPI は、マスタモードで使用し、今回は複数のスレーブ（周辺回路）が接続されるわけではないので、イネーブル信号は使用しません。



スマート・コンフィグレータでは、Config_RSPI0 のコンポーネントを追加し、

・データのビット数　16bit
・データフォーマット　MSB ファースト
・クロック位相　rise エッジで取り込み
等、オーディオ CODEC の仕様に合わせた設定を行っています。

また、
・データ受信：DMAC で処理
・データ送信完了割り込み設定
を行い、送信の処理が完了したタイミングで任意の処理ができる様にしています。

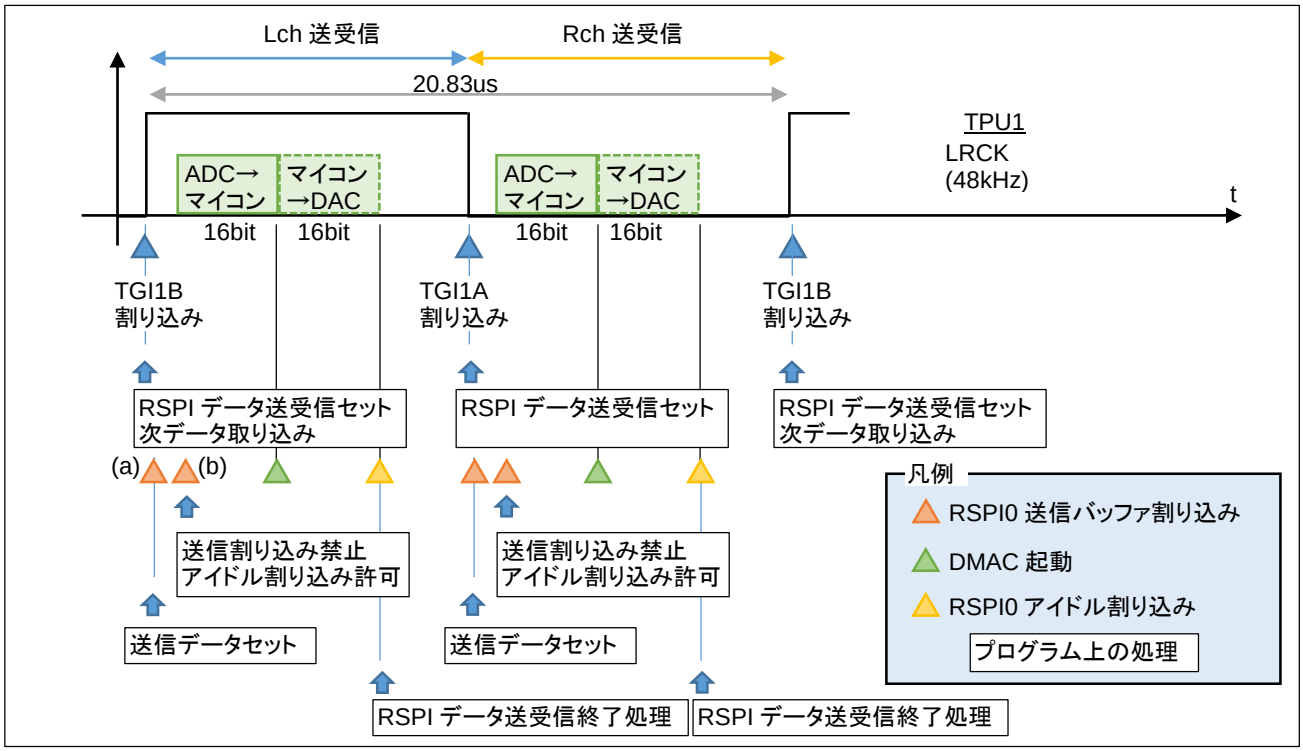


図 2-5 マイコン RSPI データ送受信

RSPI を使用したデータの送受信のタイミングは図 2-5 の様になります。

LR クロック(LRCK)切り替わりのタイミングで TPU1 の割り込み(TG1B, TG1A)が発生する様にしていますので、この割り込みをトリガとして、データの送受信（正確には受信＋送信）を行う様にしています。

TGI1A の割り込みで、RSPI のデータ送受信機能をキックし、TGI1B の割り込みで、RSPI のデータ送受信機能をキック及び、次データの取り込みを行っています。

RSPI のデータ送受信機能をキックすると、「送信バッファ割り込み(a)(b)」が入り、実際にデータの受信が完了したタイミングで「DMAC の起動」が発生し、データの送信バッファが空になったタイミングで「RSPI0 アイドル割り込み」が発生します。

TG1A 割り込みの処理は、
Config_TPU1¥Config_TPU1_user.c
内に記載します。

このファイルは、ツールが自動生成するファイルですが、

```
/* Start user code…
```
※ここにユーザコードを追加する
```
/* End user code.
```

決められた場所に、ユーザ側でコードを追加する事ができます。
※決められた場所以外にコードを追加すると、コードの自動生成の時に上書きされて消えてしまいます

・TGI1A 割り込み

Config_TPU1¥Config_TPU1_user.c

```
static void r_Config_TPU1_tgi1a_interrupt(void)
{
    /* Start user code for r_Config_TPU1_tgi1a_interrupt. Do not edit comment generated here */

    //ch0 send & receive

#if (OUT_CH == 2)
    g_rspi0_send_data = g_next_send_data.ch0;
#elif (OUT_CH == 1)
    g_rspi0_send_data = g_next_send_data;
#endif

    g_rspi0_flag = 0x1;

    setpsw_i();//multiple interrupt enable

    //RSPI-ch0 receive&send
    RSPI0.SPCR.BYTE |= 0xf0;

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

g_rspi0_send_data 変数に送信するデータをセット

フラグ変数をセット

多重割り込み許可

RSPI の送受信機能をキック

setpsw_i() は、多重割り込み許可の命令です。TGI1A 割り込み処理中でも優先度の高い割り込みを受け付ける処理です。

g_rspi0_flag は、フラグ変数です。

処理中:1 をセット
処理完了時:0 にクリア

となる様に使用しています。

・TGI1B 割り込み

Config_TPU1¥Config_TPU1_user.c

```
static void r_Config_TPU1_tgi1b_interrupt(void)
{
    /* Start user code for r_Config_TPU1_tgi1b_interrupt. Do not edit comment generated here */

//ch1 send & receive

#if (OUT_CH == 2)
    g_rspi0_send_data = g_next_send_data.ch1;
#endif

    g_rspi0_flag = 0x1;

    setpsw_i();//multiple interrupt enable

    //RSPI-ch0 receive&send
    RSPI0.SPCR.BYTE |= 0xf0;

    if(g_outbuf_rp != g_outbuf_wp)
    {
        g_next_send_data = g_outbuf[g_outbuf_rp];
        g_outbuf_rp++;
        if(g_outbuf_rp >= OUTBUF_SIZE) g_outbuf_rp = 0;
    }
    else
    {
        g_next_send_data = g_zero_data;
    }

    g_sampling_period_flag = 0;//flag clear
    g_sampling_counter++;//counter increment

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

g_rspi0_send_data 変数に送信するデータをセット（2ch 使用時）
※1ch 使用時は、ch0, ch1 で同じデータを出力するので、TGI1A で格納済みのデータをそのまま使用する

フラグ変数をセット

多重割り込み許可

RSPI の送受信機能をキック

送信バッファ(g_outbuf)の次に送信するデータを g_next_send_data にセット

送信バッファが空の場合は、ゼロデータをコピー

フラグのクリアとカウンタのインクリメント

TGI1B は、1 周期（0ch と 1ch のデータ送受信）が終わった後の処理が追加されています。

送信バッファ(g_outbuf)の読み取りと、送信バッファの読み出し位置(g_outbuf_rp)を進める処理。送信バッファに未送信データがない場合は、ゼロデータを次データにセット。

20.83us 周期のフラグのクリアと、カウンタのインクリメントの処理。

・RSPI0 送信バッファ（エンプティ）割り込み

Config_RSPI0¥Config_RSPI0_user.c

```
static void r_Config_RSPI0_transmit_interrupt(void)
{
    if(g_rspi0_flag == 0x1)                                  (a)の処理
    {
        RSPI0.SPDR.WORD.H = g_rspi0_send_data;               送信データをレジスタにコピー
        g_rspi0_flag = 0;                                    フラグクリア
    }
    else                                                     (b)の処理
    {
        RSPI0.SPCR.BIT.SPTIE = 0U;//disable RSPI-TX Interrupt    送信割り込み禁止
        RSPI0.SPCR2.BIT.SPIIE = 1U;//enable RSPI-IDLE Interrupt  アイドル割り込み許可
    }
}
```

送信バッファ割り込みは、RSPI の送信バッファが空になった時点で発生する割り込み（送信バッファエンプティ割り込み）です。

RSPI の送受信機能をキックした時点では、送信バッファは空なので、直ぐに 1 度目の割り込み(a)が入ります。ここでは、送信データを RSPI の送信バッファ（マイコンのレジスタ)にコピーします。マイコン内部の動作として、RSPI の送信バッファから送信用のシフトレジスタに転送を行い、RSPI の送信バッファが空になると、再度（2 度目の送信バッファエンプティ割り込み）(b)が入ります。

今度は、送信割り込みを禁止し、RSPI アイドル割り込みを許可します。

※今回、データは 1 ワード(16bit)ずつの送信ですので、2 回目の送信バッファエンプティ割り込みで、アイドルの処理に移ります（もし、複数ワードのデータ送信を行う場合は、ここで次のデータを RSPI の送信バッファにコピーします）

・RSPI0 アイドル割り込み

Config_RSPI0¥Config_RSPI0_user.c

```
void r_Config_RSPI0_idle_interrupt(void)
{
    /* Disable RSPI function */
    RSPI0.SPCR.BIT.SPE = 0U;                 RSPI 停止

    /* Disable idle interrupt */
    RSPI0.SPCR2.BIT.SPIIE = 0U;              アイドル割り込み禁止
}
```

RSPI のデータ送受信が完了した段階で、アイドル割り込みが入ります。ここでは、RSPI の機能を停止し、アイドル割り込みも禁止とします。

※注意

Config_RSPI0¥Config_RSPI0_user.c のファイルは、スマート・コンフィグレータの自動生成対象のファイルです。
基本的には、コード生成を行うと、上書きされ、
(1)r_Config_RSPI0_transmit_interrupt()
(2)r_Config_RSPI0_idle_interrupt()
の中身は上書きされ、ユーザ側で書いたコードは消えてしまいます。

スマート・コンフィグレータを使用する場合、(1)(2)の関数から呼ばれるコールバック関数の中には、

```
/* Start user code…
/* End user code.
```

があり、ユーザがコードを追加できる様になっていますが、(1)(2)の関数はユーザが書き下す事ができない様になっています。

そこで、裏技的な手法になるのですが、Config_RSPI0¥Config_RSPI0_user.c のファイルの先頭の、Pragma 関連のユーザコードを記載する部分に、選択コンパイルの#if 文を入れてしまい

```
/***********************************************************************************************************************
Pragma directive
***********************************************************************************************************************/
/* Start user code for pragma. Do not edit comment generated here */
#if 0
/* End user code. Do not edit comment generated here */
```

スマート・コンフィグレータで生成した部分が全て無効となる様に細工をしています。

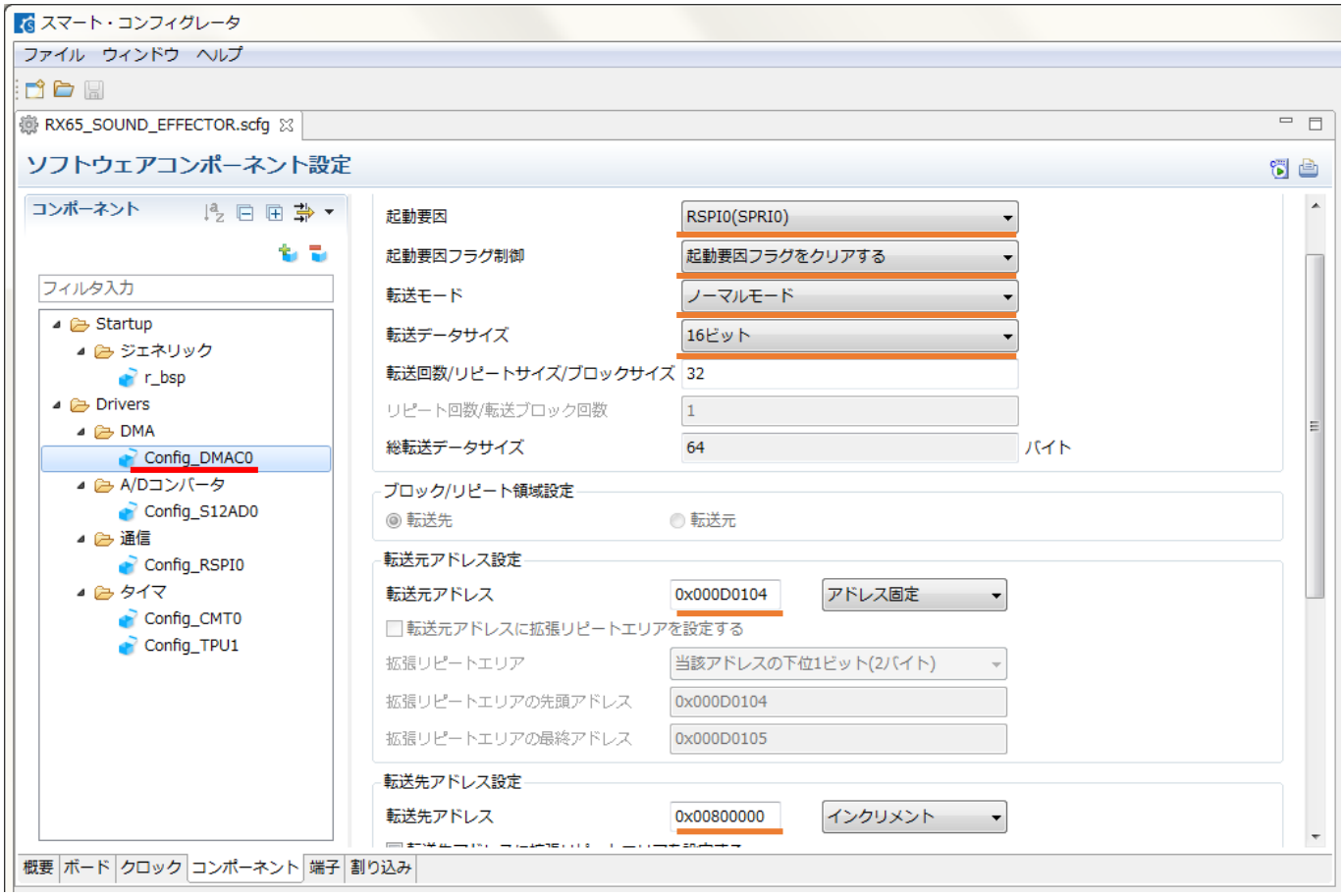
```
/***********************************************************************************************************************
* Function Name: r_Config_RSPI0_callback_error
* Description  : This function is a callback function when RSPI0 error occurs
* Arguments    : err_type -
*                    error type value
* Return Value : None
***********************************************************************************************************************/

static void r_Config_RSPI0_callback_error(uint8_t err_type)
{
    /* Start user code for r_Config_RSPI0_callback_error. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
#endif

//user code
[この部分に各種コードの記載]
/* End user code. Do not edit comment generated here */
```

また、スマート・コンフィグレータで生成した最後の関数のユーザ記載部分に、#endif 文を入れ、この行まで（スマート・コンフィグレータで生成した全体）を無効化し、その後に(1)(2)の関数を含むコードを記載しています。

この様にファイルを記載した場合、スマート・コンフィグレータでコード再生成した場合、上書きされるのは #if 0 ～ #endif で囲まれた部分（選択コンパイルで読み飛ばされる部分）となり、コード生成の度にコードを修正する手間を省く事ができます。
（裏技的な手法となるのですが、スマート・コンフィグレータで生成される部分のコードをユーザ側で書き下す場合に有効な手法です）

(d)DMAC を使用した受信データのメモリへの格納

マイコンの機能である、DMAC(Direct Memory Access Controller)を使用して、RSPI の受信データをマイコンの CPU コアを使わずにメモリに格納する処理を行っています。



DMAC の起動要因は、RSPI0 の SPRI0（受信）をトリガとします。転送モードは、ノーマルモードとし、転送データサイズは 16bit とします。転送回数は、設定を上書きする様にしているので、仮の値で構いません。

※sound_effector_userdef.h の INBUF_UNIT_SIZE が転送データサイズを決める様、コードを記載しています

転送元アドレスは、RSPI0 の受信レジスタ（アドレス固定）です。転送先アドレスは、拡張メモリのアドレスで（インクリメント）です。

決められたサイズ(INBUF_UNIT_SIZE)のデータを受信すると、割り込みが掛かる様に設定してあり、割り込みでアドレスの確認（384kB のバッファの最後まで来た場合、再度先頭に戻す）等の処理を行っています。

INBUF_UNIT_SIZE を 1 に設定すると、16bit のデータを 2 個(ch0, ch1)受信（20.83us）すると、割り込み処理が入ります。INBUF_UNIT_SIZE を 256 に設定すると、16bit のデータを 256×2 個受信（5.33ms）すると、割り込み処理が入ります。そのため、INBUF_UNIT_SIZE をできるだけ大きな数値とした方が、効率は良いです。
（但し、INBUF_UNIT_SIZE は、入出力間のレイテンシーに直結するので、適切な数値に設定ください）

・DMAC0 割り込み処理

Config_DMAC0¥Config_DMAC0_user.c

```
static void r_dmac0_callback_transfer_end(void)
{
    /* Start user code for r_dmac0_callback_transfer_end. Do not edit comment generated here */

    g_inbuf_wp += INBUF_UNIT_SIZE;
    if((unsigned long)DMAC0.DMDAR > BUF_RAM_END_ADDR)
    {
        DMAC0.DMDAR = (void *)BUF_RAM_START_ADDR;
        g_inbuf_wp=0;
    }
    DMAC0.DMCRA = INBUF_UNIT_SIZE*2;
    DMAC0.DMCNT.BIT.DTE = 1U;

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

受信バッファのインデックスを更新

メモリの最終アドレスに達した場合

格納先のアドレスを先頭に戻す
受信バッファのインデックスを
先頭に戻す

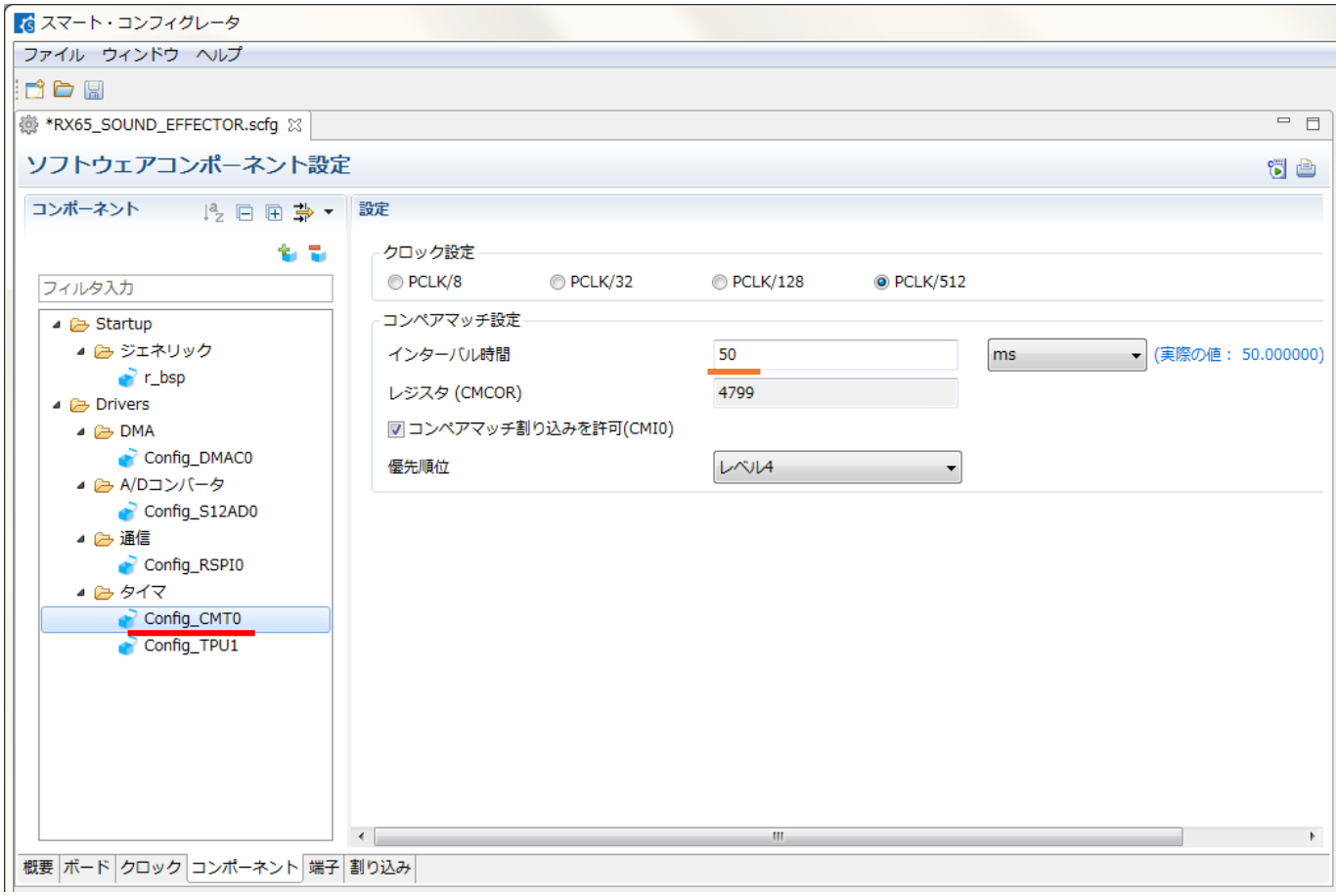
DMAC の転送サイズを設定
再度動作開始

3. スイッチとボリュームの読み取り

サウンドエフェクタには、ロック式のプッシュスイッチと 3 つのボリューム(VR1~VR3)があります。

スイッチとボリュームの読み取りには、以下の機能を使用しています。

・CMT0（コンペアマッチタイマ）



コンペアマッチタイマは、定期的な処理を行いたい場合に便利な機能です。一定時間毎に、割り込み関数を呼び出す処理を行います。

ここでは、周期を 50ms に設定し、周期毎に、スイッチとボリュームの回転角度をスキャンします。

・CMT0 割り込み

Config_CMT0¥Config_CMT0_user.c

```
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */

    setpsw_i();//multiple interrupt enable

    if(g_ad_flag == 0)
    {
        g_ad_flag=1;
        R_Config_S12AD0_Start();//A/D conversion start
    }

    g_sw_state = PORTE.PIDR.BYTE & 0x1;//PE0 read

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

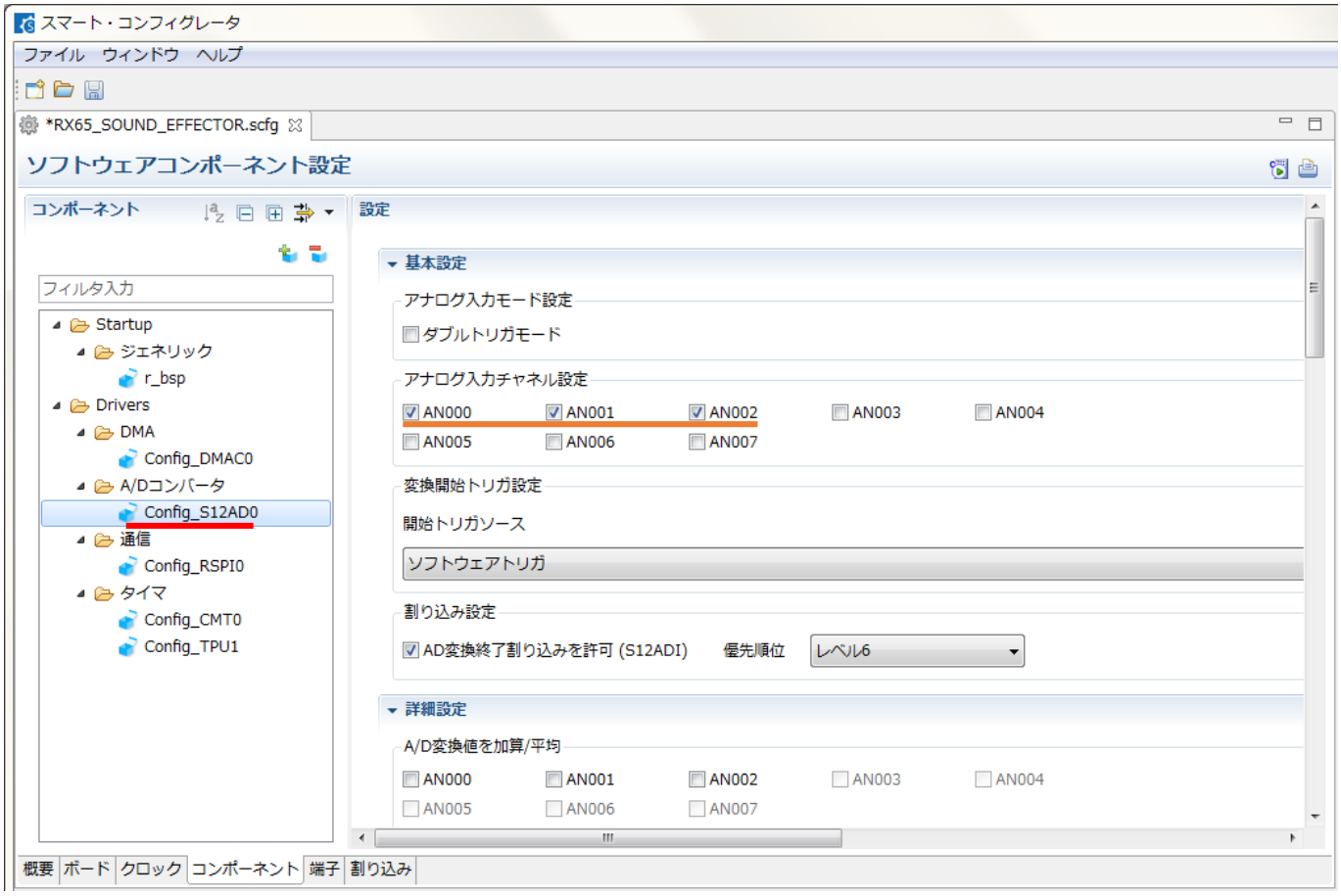
A/D 変換フラグセット
A/D 変換開始

Push-SW のポートの値を読み取り

g_ad_flag（A/D 変換処理中 1 にセットされる）がクリアされている場合、ボリューム読み取りのための A/D 変換処理を開始（R_Config_S12AD0_Start()）します。

g_sw_state 変数に、PE0（ロック式プッシュスイッチが接続されているポート）の読み取り結果をコピーします。

・A/D 変換



AN000~AN002（VR1~VR3 に対応）を、A/D 変換対象に設定し、A/D 変換処理が終わったタイミングで割り込みを掛ける様設定しています。

・A/D 変換割り込み

Config_S12AD0¥Config_S12AD0_user.c

```
static void r_Config_S12AD0_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */

    //R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &ad_val[0]);//function call

    g_ad_val[0] = S12AD.ADDR0;//equivalent to function call(direct resistor access)
    g_ad_val[1] = S12AD.ADDR1;
    g_ad_val[2] = S12AD.ADDR2;

    g_ad_flag=0;

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

A/D 変換結果をコピー

A/D 変換中フラグクリア

g_ad_val[0] = S12AD.ADDR0;　A/D 変換結果を変数にコピー

及び、A/D 変換処理中フラグのクリア(g_ad_flag=0)を行っています。

CMT0 割り込みと A/D 変換割り込みの組み合わせにより、50ms 毎に、現在のボリュームの回転角度に応じた値が、g_ad_val 変数に格納されます。

4. 初期化関数

sound_effector.c では、本機器の初期化と動作開始の関数が定義されています。

・初期化処理

src¥user¥sound_effector¥sound_effector.c

```
void sound_effector_init(void)
{
    //初期化関数

    //引数：なし

    //戻り値：なし

    unsigned long i;

    //変数初期化

#if (OUT_CH == 2)
    for(i=0; i<OUTBUF_SIZE; i++)
    {
        g_outbuf[i].ch0 = 0;
        g_outbuf[i].ch1 = 0;
    }
#elif (OUT_CH == 1)
    for(i=0; i<OUTBUF_SIZE; i++)
    {
        g_outbuf[i] = 0;
    }
#endif

    for(i=0; i<INBUF_SIZE; i++)
    {
        g_inbuf[i].ch0 = 0;
        g_inbuf[i].ch1 = 0;
    }

    g_outbuf_rp = 0;
    g_outbuf_wp = 0;

    g_inbuf_wp = 0;

    g_rspi0_send_data = 0;

#if (OUT_CH == 2)
    g_next_send_data.ch0 = 0;
    g_next_send_data.ch1 = 0;
    g_zero_data.ch0 = 0;
    g_zero_data.ch1 = 0;
#elif (OUT_CH == 1)
    g_next_send_data =0 ;
    g_zero_data = 0;
#endif
```

各種変数の初期化

```
    g_rspi0_flag = 0;

    g_sampling_period_flag = 0;
    g_sampling_counter = 0;

    g_sw_state = 0;

    g_ad_flag=0;

    for(i=0; i<3; i++) g_ad_val[i]=0;

    //PE0入力, pull-up設定
    PORTE.PDR.BYTE &= ~0x01;//PE0:input
    PORTE.PCR.BYTE |= 0x01;//PE0:pull-up
}
```

Push-SW のポート設定

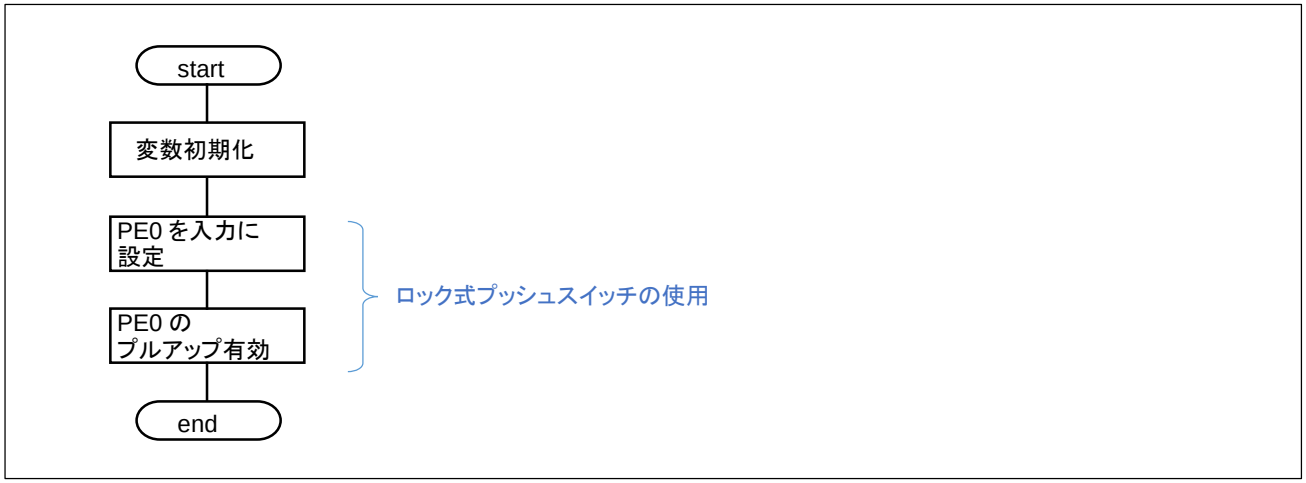
・動作開始処理

src¥user¥sound_effector¥sound_effector.c

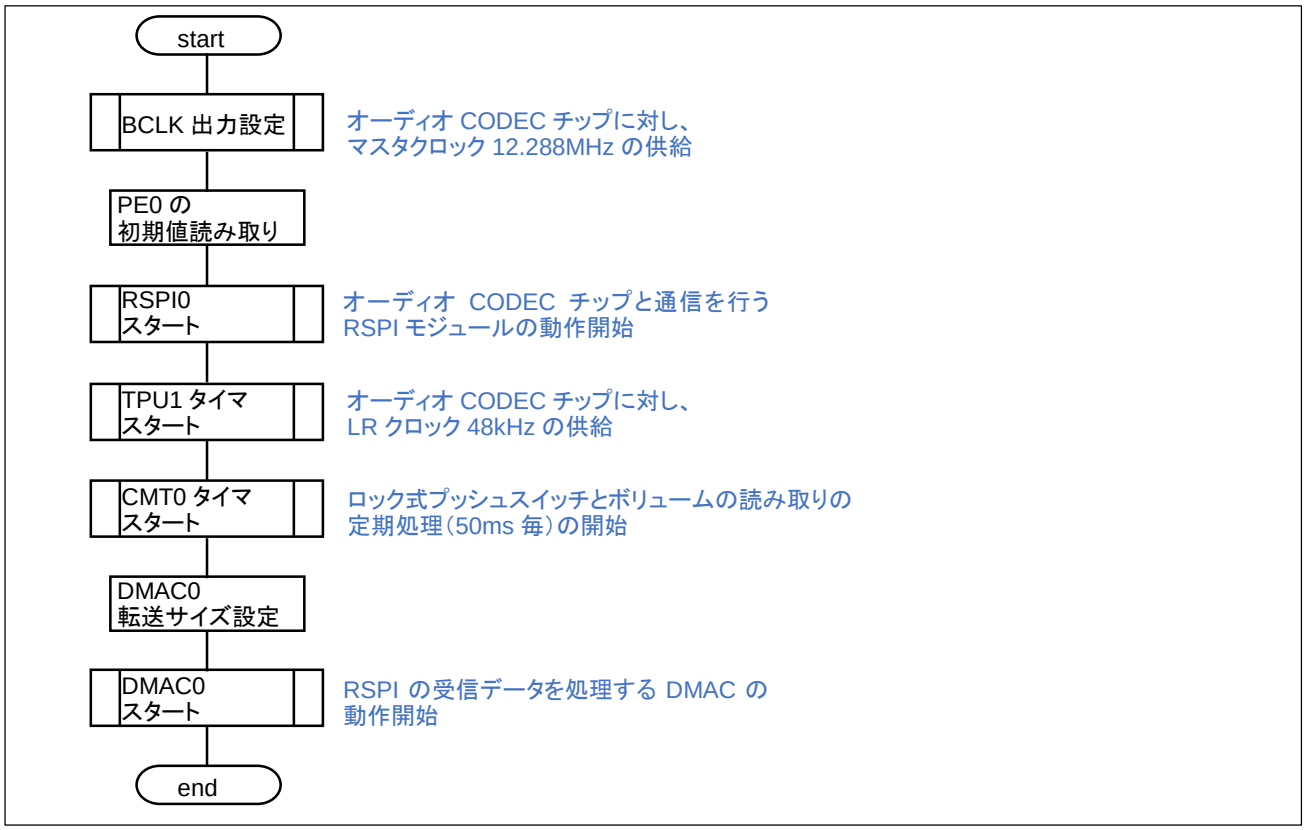
```
void sound_effector_start(void)
{
    //動作開始関数

    //引数：なし

    //戻り値：なし

    //BLCK=12.288MHz出力
    SYSTEM.PRCR.WORD = 0xA503;

    SYSTEM.SCKCR.BIT.PSTOP1 = 1;
    while(SYSTEM.SCKCR.BIT.PSTOP1 != 1) nop();

    //BLCK強制出力
    MPC.PFBCR0.BIT.BCLKO = 1;

    SYSTEM.SCKCR.BIT.PSTOP1 = 0;
    while(SYSTEM.SCKCR.BIT.PSTOP1 != 0) nop();

    SYSTEM.PRCR.WORD = 0xA500;

    g_sw_state = PORTE.PIDR.BYTE & 0x1;//PE0 初期値読み出し,設定

    R_Config_RSPI0_Start();//データ通信(RSPI0)

    R_Config_TPU1_Start();//48kHz

    R_Config_CMT0_Start();//50msに1回（スイッチ読み取りとA/D変換）

    DMAC0.DMCRA = INBUF_UNIT_SIZE*2;//DMAC:1回に処理するデータ

    R_Config_DMAC0_Start();//DMAC
}
```

－フローチャート－

初期化関数 sound_effector_init()



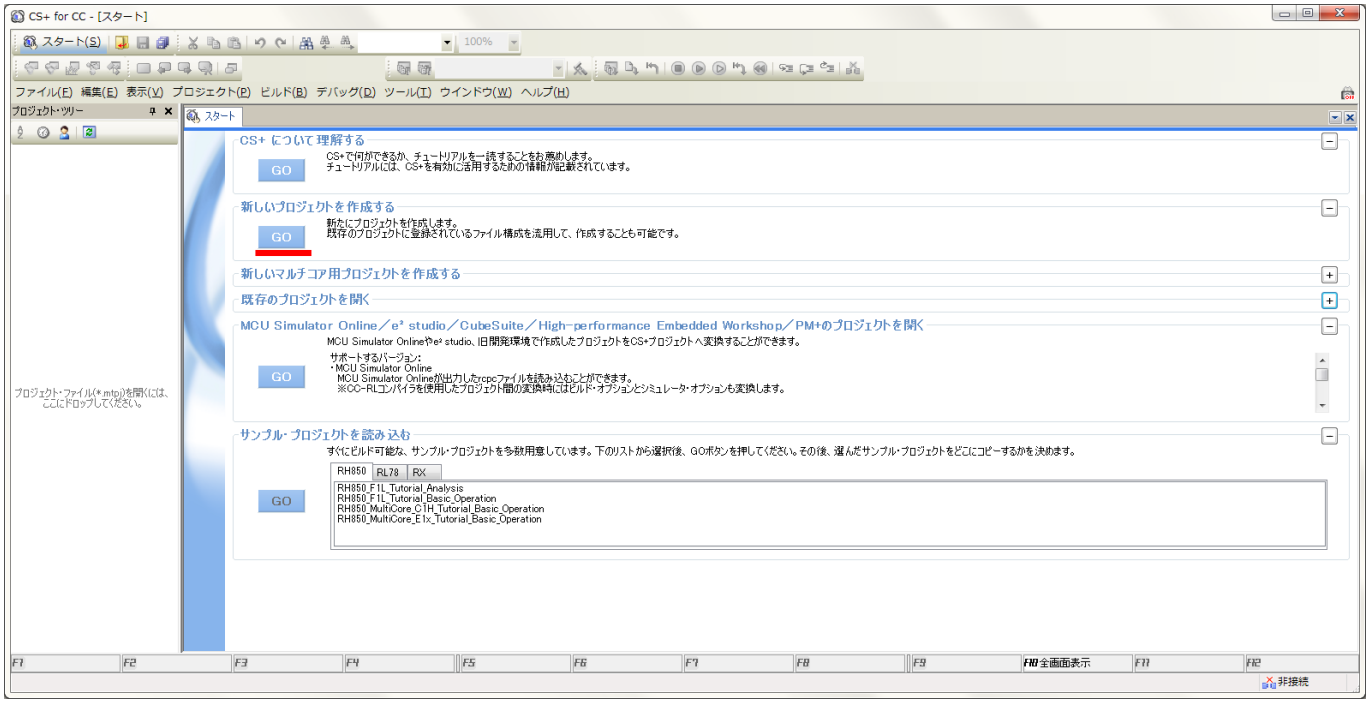
動作開始関数 sound_effector_start()



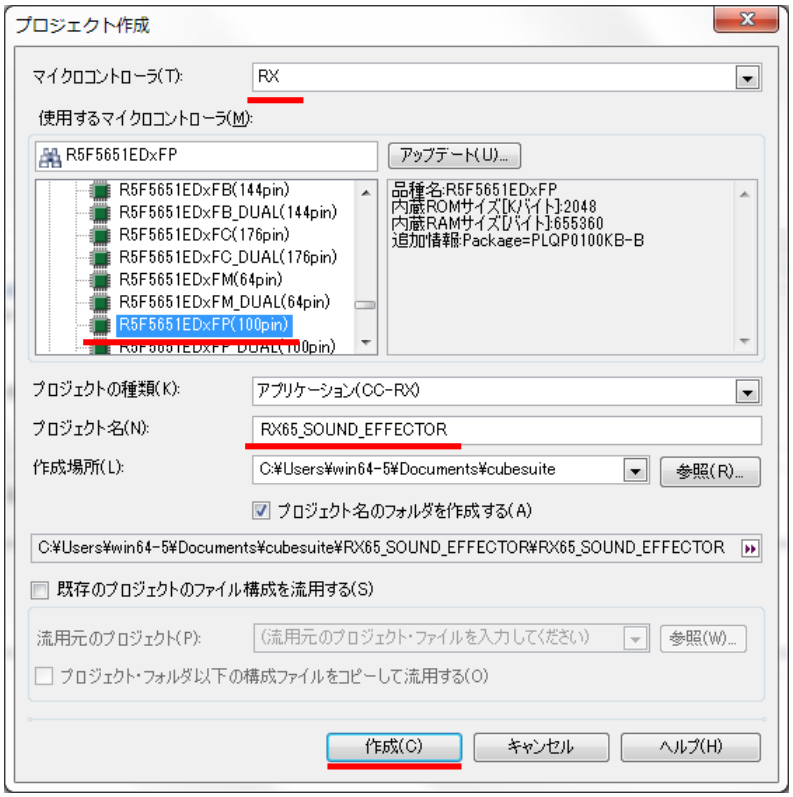
マイコンのクロックや周辺機能（RSPI, TPU, CMT, DMAC）の動作に必要なソースコードは、スマート・コンフィグレータが出力していますので、ユーザ側で記載するのは、動作開始関数を呼び出す程度で済みます

5. CS+のプロジェクトの作成手順

5.1. CS+のプロジェクトを新規に作成する手順



新しいプロジェクトを作成する「GO」ボタンを押す



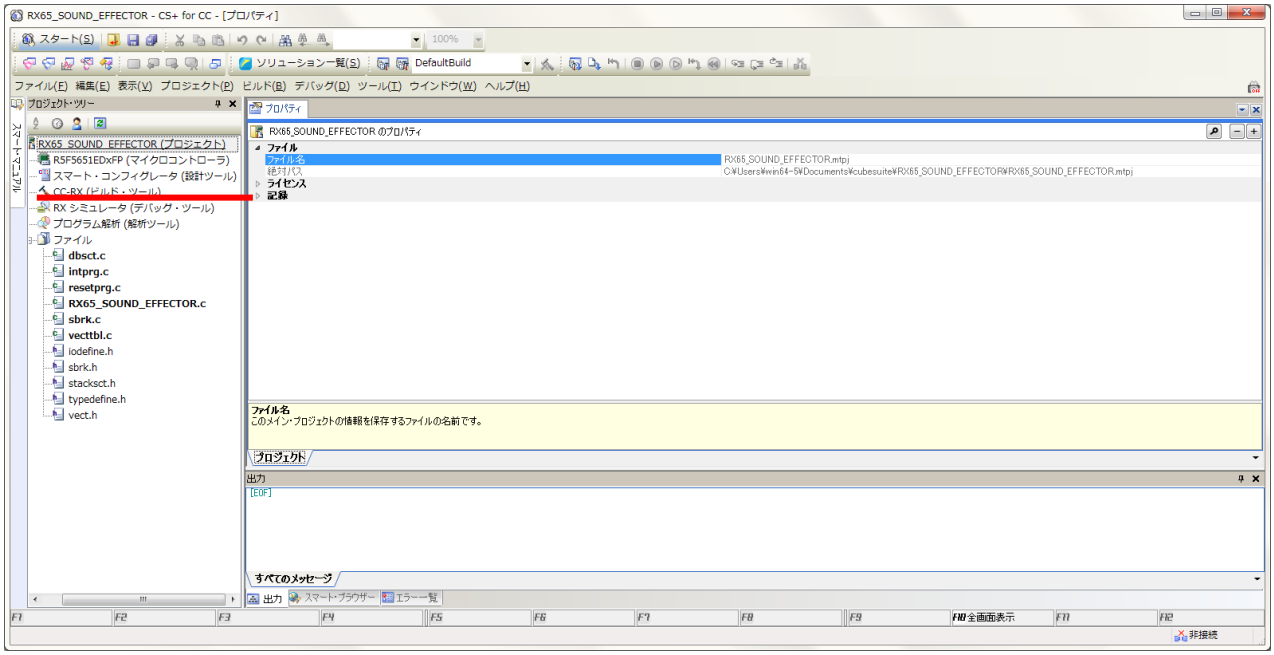
マイクロコントローラ　RX

使用するマイクロコントローラ　R5F5651EDxFP　（あえて 100pin のマイコンを選択しています）

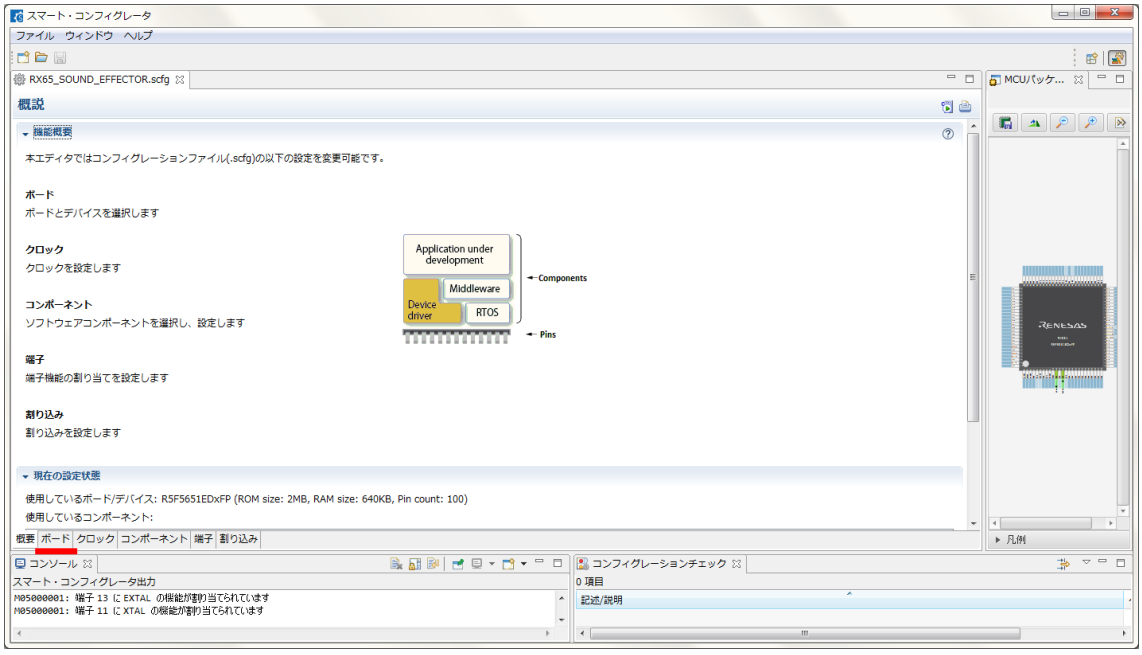
プロジェクト名　「任意の名称」を入力

（その他の項目は上記参照）

「作成」ボタンを押す

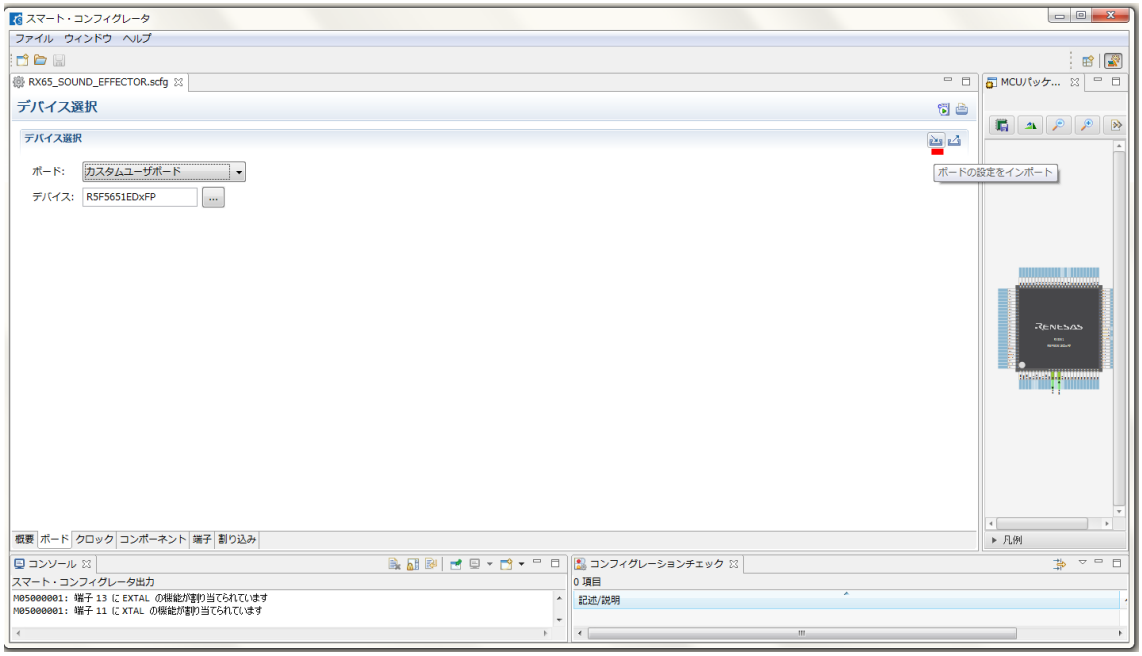


スマート・コンフィグレータ（設計ツール）をダブルクリック



「ボード」タブを選択

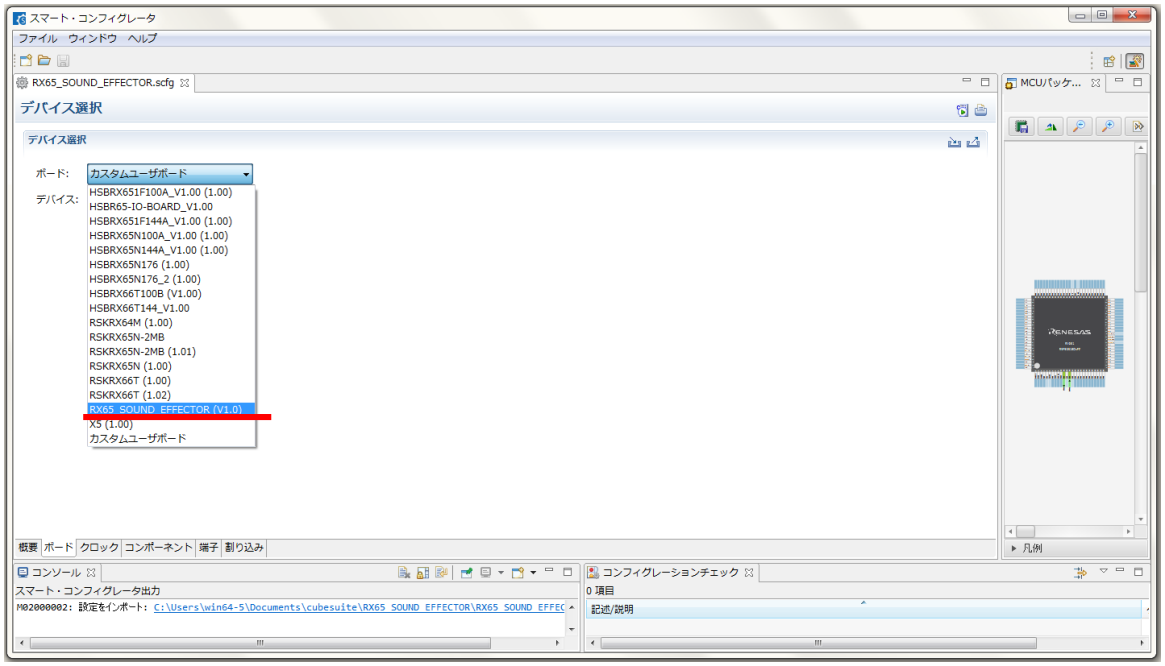
(*)BDF ファイルの読み込み



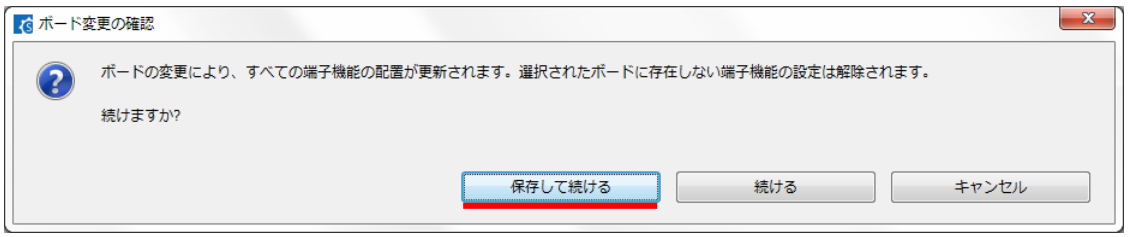
「ボードの設定をインポート」ボタンを押す

CD 内の、BDF¥RX65_SOUND_EFFECTOR_V1.0.bdf （バージョン番号は変わる可能性があります）

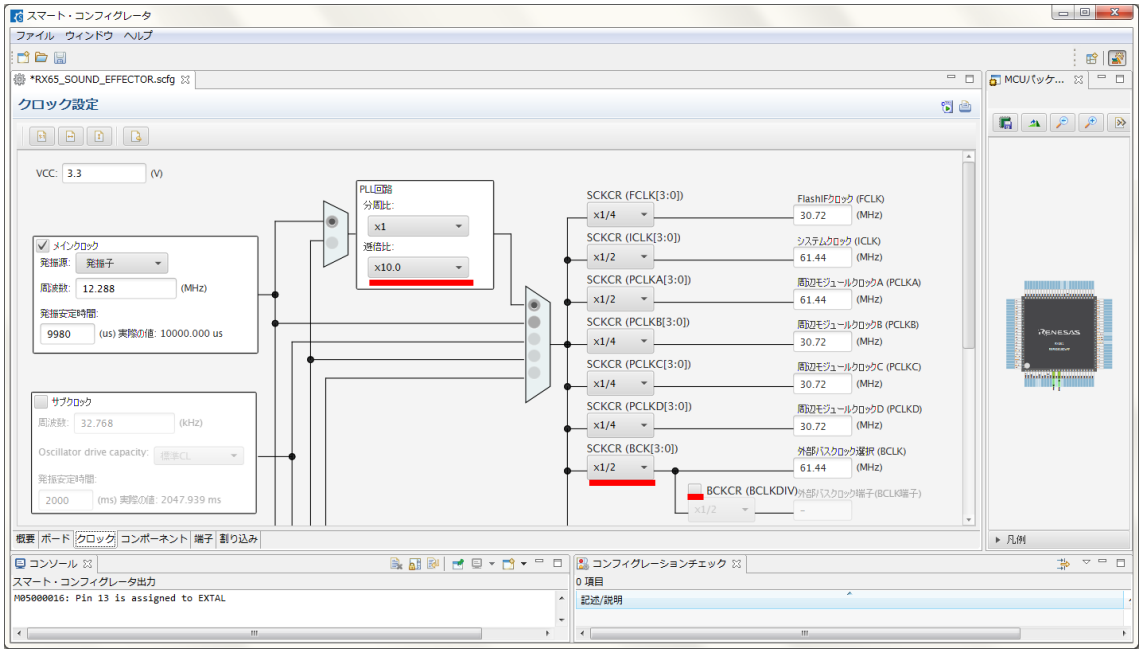
を選択して開く。



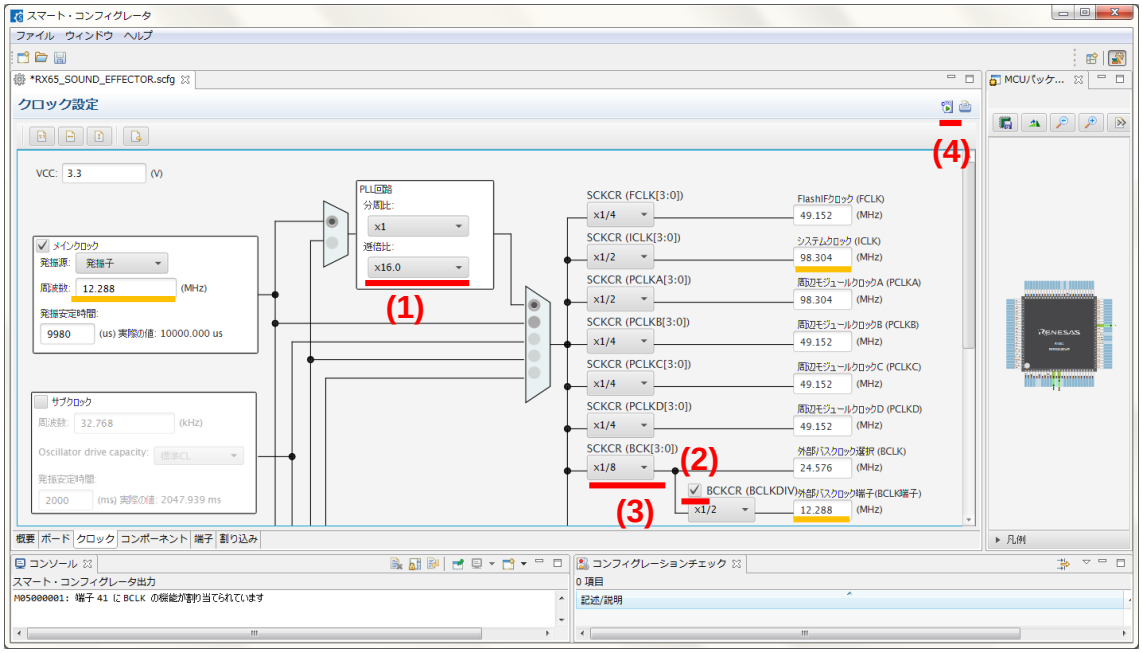
「ボード」プルダウンメニューから、RX65_SOUND_EFFECTOR(V1.0)を選択



「保存して続ける」を押す。



上記赤線部を変更

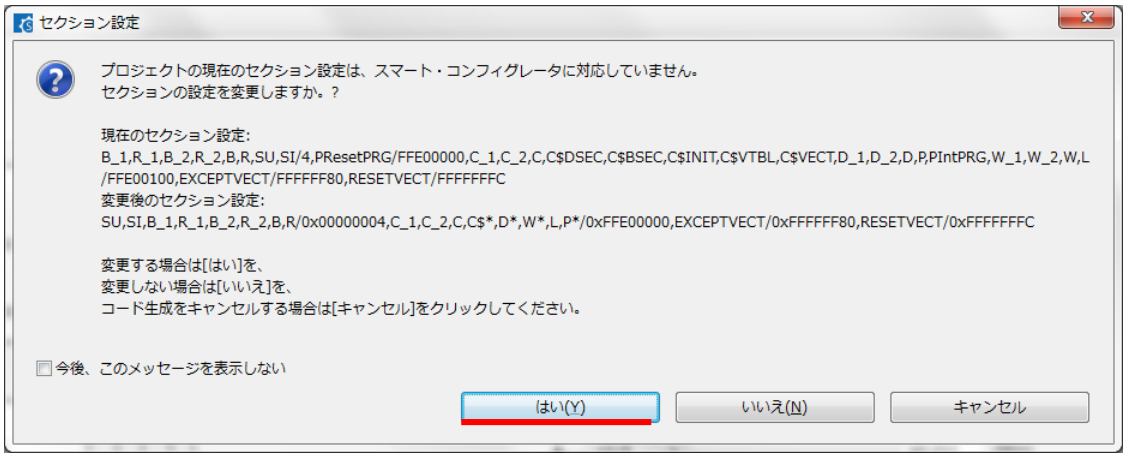


(1) x16.0 を選択
(2) BCKCR のチェックボックスにチェックを入れる
(3) SCKCR を 1/8 を選択

他は、デフォルト値で問題ありません。
※BDF ファイルの読み込み(*)を省略し、メインクロックの周波数値を手入力しても等価です

黄線部
メインクロック　周波数　12.288
システムクロック　98.304
外部バスクロック端子　12.288
に設定されている事を確認してください。

(4)「コード生成」ボタンを押してください



上記ダイアログが出た場合は、「はい」を押してください。

以上の手順で、初期設定は完了し、クロック設定等の基本的なプログラムコードが出力されました。

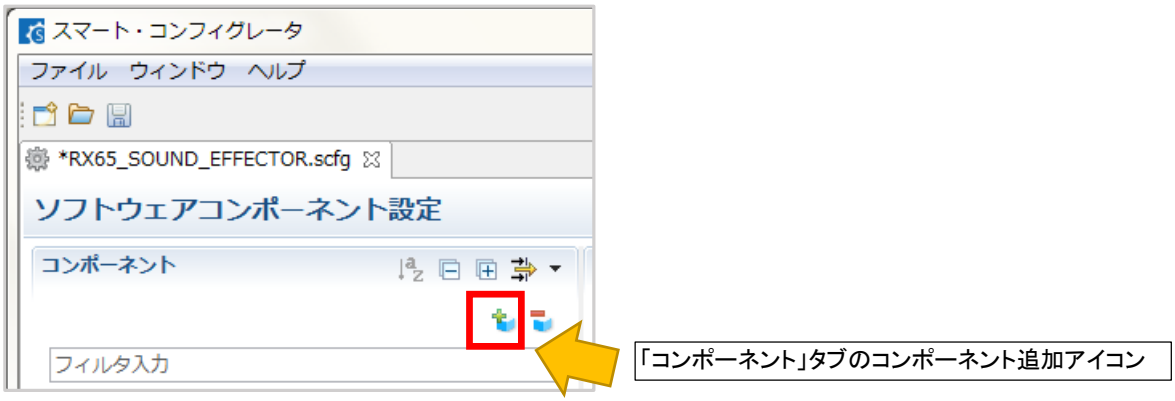
マイコンの周辺機能（RSPI や TPU タイマ等）は、次節

スマート・コンフィグレータでコンポーネントを追加する設定
スマート・コンフィグレータで端子の割り当てを変更する設定

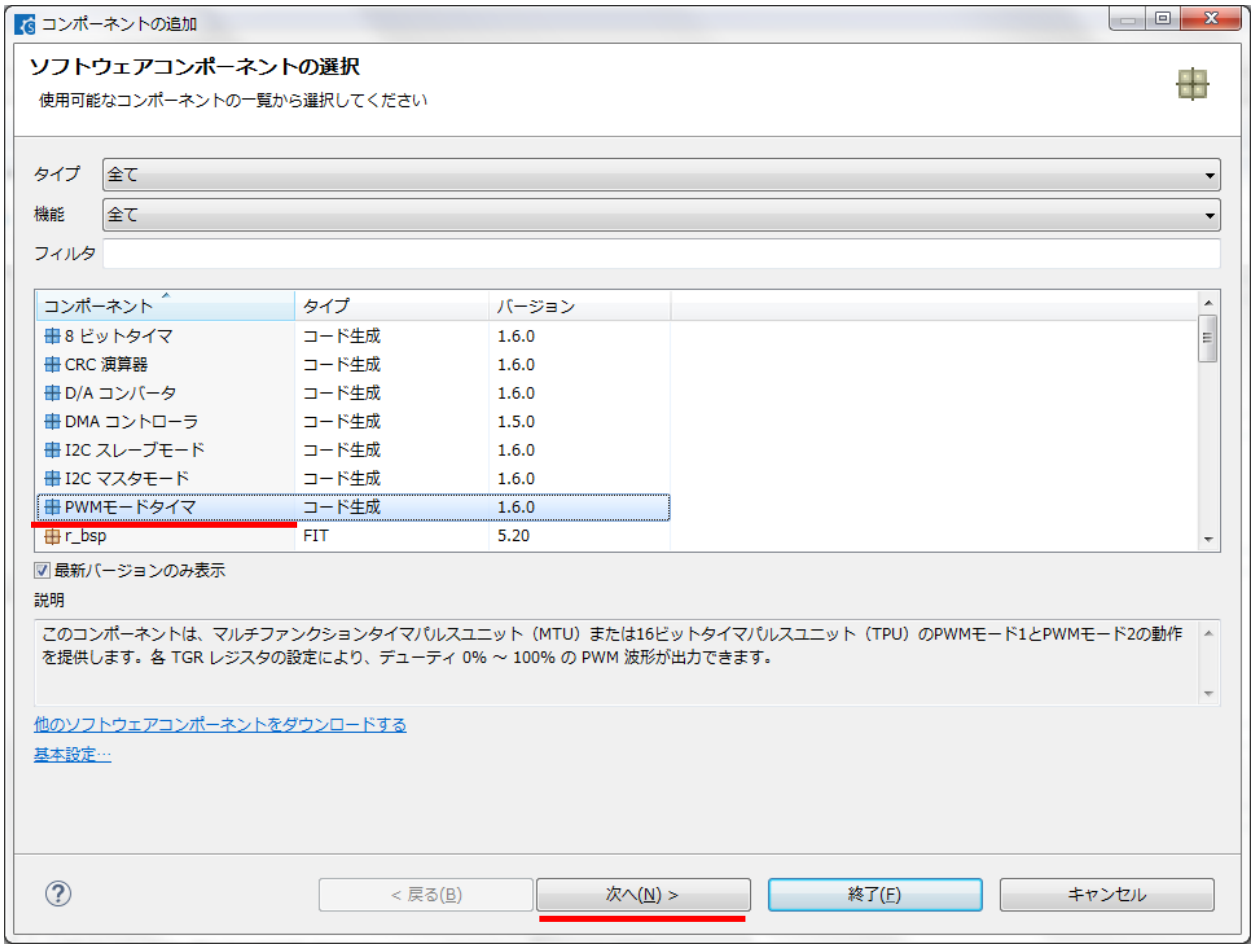
を参照ください。

5.2. スマート・コンフィグレータでコンポーネントを追加する設定

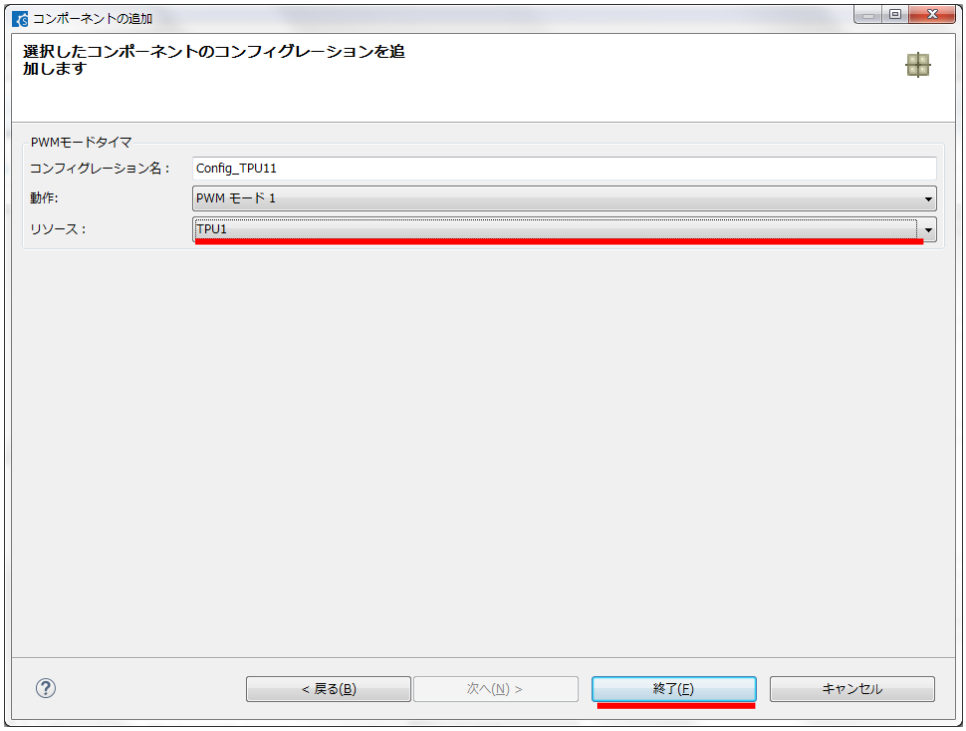
2 章では、既に TPU1 を使用する設定がスマート・コンフィグレータに追加されていましたが、プロジェクトを新規に作成した場合は、以下の手順でコンポーネント（マイコンのタイマ等の周辺機能）を追加できます。



「コンポーネント追加アイコン」を押す。



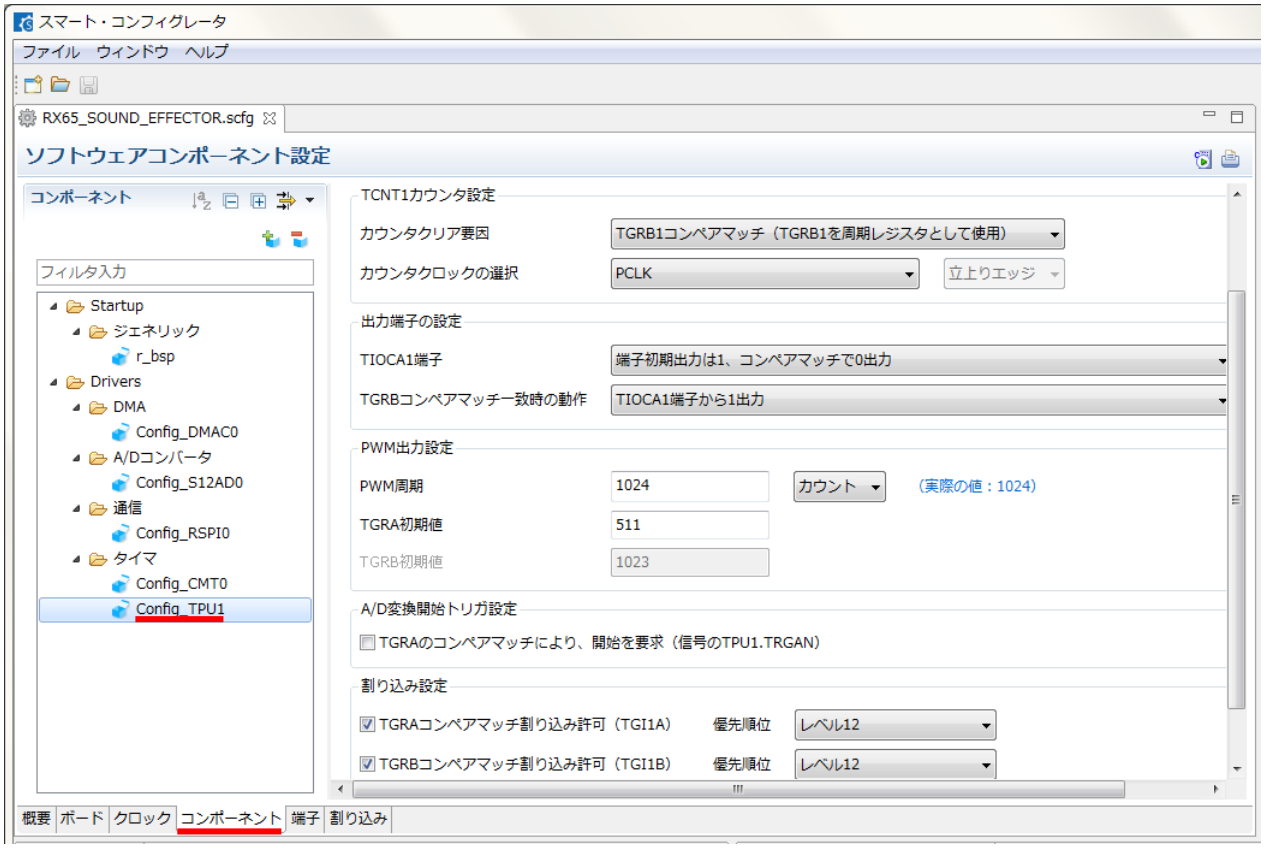
「PWM モードタイマ」を選択して、「次へ」



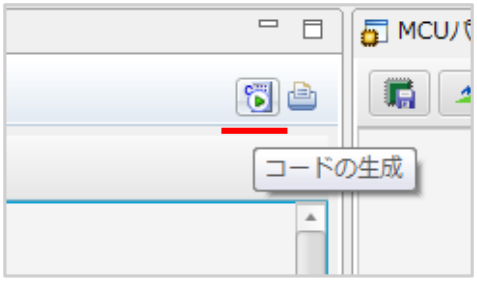
リソース「TPU1」を選択、動作「PWM モード 1」を選択。

「終了」を押す。

コンポーネントタブ、

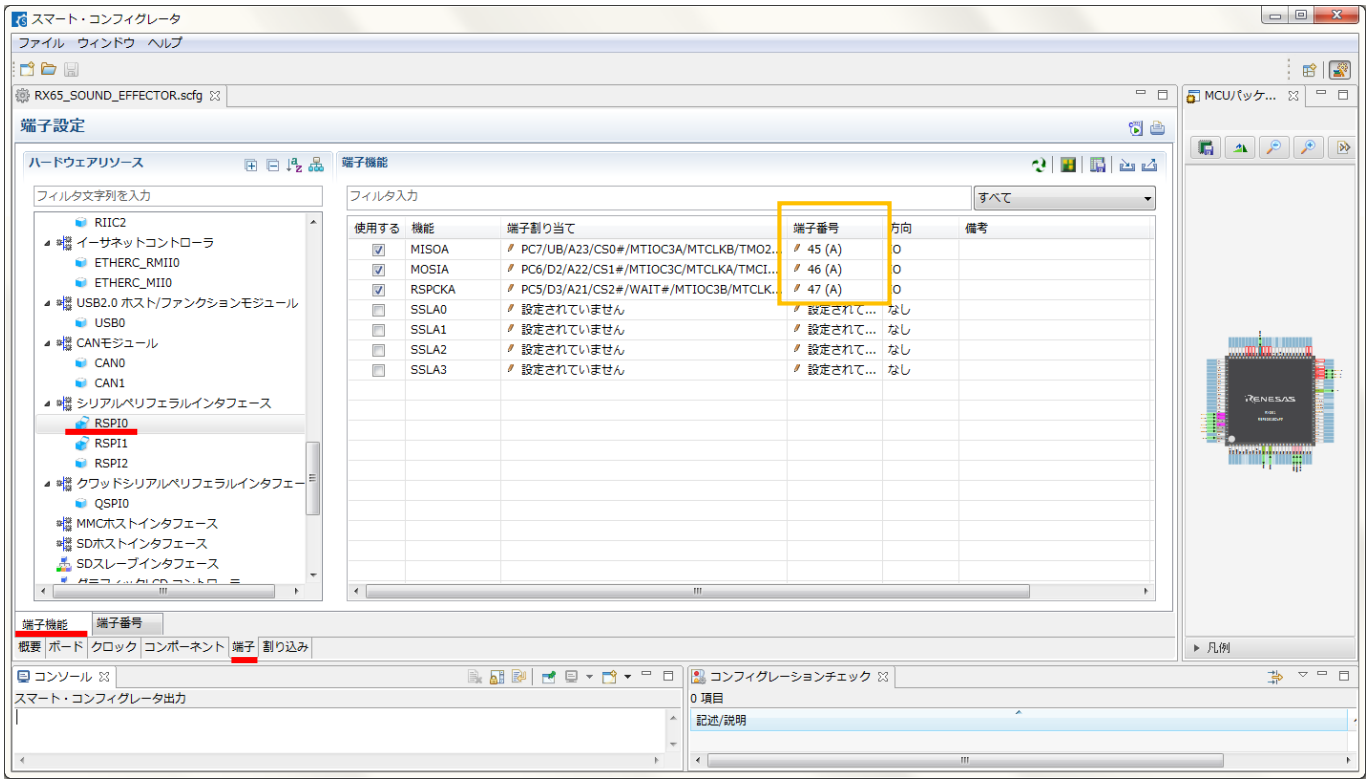


Config_TPU1 の設定を前記の様に設定。



コンポーネントの設定を変更した際は、必ず「コード生成」ボタンを押し、設定をソースコードに反映させるようにしてください。

5.3. スマート・コンフィグレータで端子の割り当てを確認する方法（変更する設定）



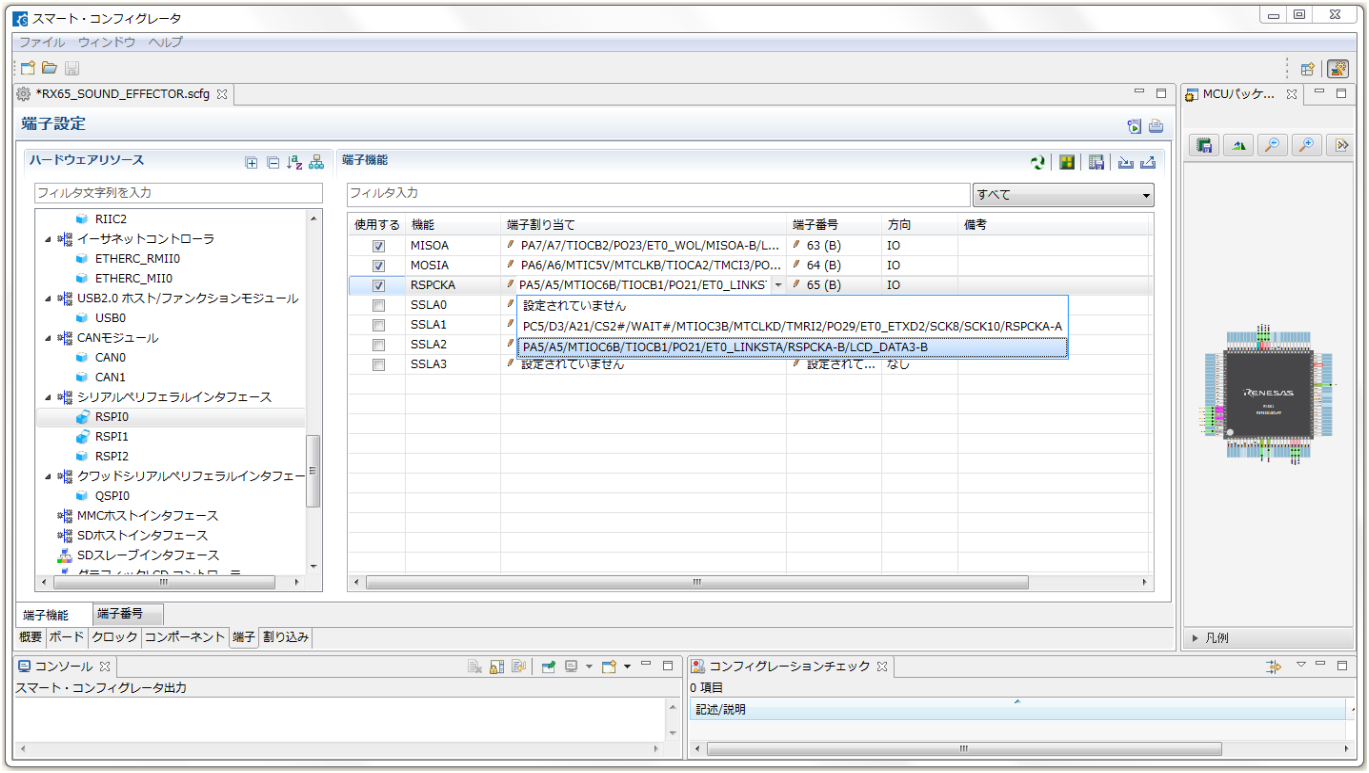
「端子」タブ「端子機能」タブ　RSPI0

上記を選択すると、MISOA, MOSIA, RSPCKA の端子がそれぞれ、PC7, PC6, PC5 に割り当てられている事が判ります。本機器で使用しているのは、PC7, PC6, PC5 なので、このままで問題ありません。

※ここで表示されている端子番号は、100pin のチップでの端子番号なので正しくありませんが問題ありません
端子名（PC7 等）が合っていれば問題ありません

[参考]

RX651 マイコンでは、RSPI0 等のマイコン内蔵周辺機能の端子割り当てを複数の端子から選択することができる様になっています。



端子割り当ての欄をクリックすると、プルダウンメニューで別な端子が選択可能です。端子割り当てを、PA7, PA6, PA5 等別な端子に割り当てる事が可能です。

もし、周辺機能が上手く動作していないということがあれば、スマート・コンフィグレータの端子割り当てを確認し、期待しているのと別な端子に機能が割り当てられていないかを確認してください。

（端子割り当て変更後は、「コード生成」を行うのを忘れないようにしてください。）

6. 割り込み関して

テンプレートのプログラムでは、以下の割り込みを使用しています。

優先度	割り込み名称	周辺機能	用途	備考
15	SPTI0	RSPI0	RSPI データ送信	高速割り込み指定 20.83us に 4 回
14	SPII0 SPEI0	RSPI0	RSPI アイドル（RSPI 通信終了） RSPI エラー	20.83us に 2 回
13	DMAC0I	DMAC	INBUF_UNIT_SIZE 受信後の DMAC 再起動	5.33ms に 1 回(*1)
12	TGI1A	TPU1	LR クロック rise 時の処理	20.83us に 1 回
12	TGI1B	TPU1	LR クロック fall 時の処理	20.83us に 1 回
6	S12ADI	S12AD	A/D 変換終了後の値の回収	50ms に 1 回
4	CMI0	CMT0	Push-SW, ボリュームの読み取り	50ms に 1 回

RX での割り込みは、0（割り込み禁止）, 1~15 の優先度があり、数値の大きな方の割り込みが優先されます。また、割り込みルーチン内で、

setpsw_i();

の構文があれば（setpsw_i()以降の処理は）、実行中の割り込みルーチンの優先度より高い優先度の割り込みであれば多重割り込みで処理されます。
（優先度 6 の割り込み実行中に、優先度 4 の割り込みが入ってきても後回しとされますが、優先度 14 の割り込みが入ってきた場合は、割り込みが実行されます。）

(*1)INBUF_UNIT_SIZE=256 の場合

7. メモリの割り当てに関して

RX651 マイコン（本機器に搭載されている型名のマイコン）では、

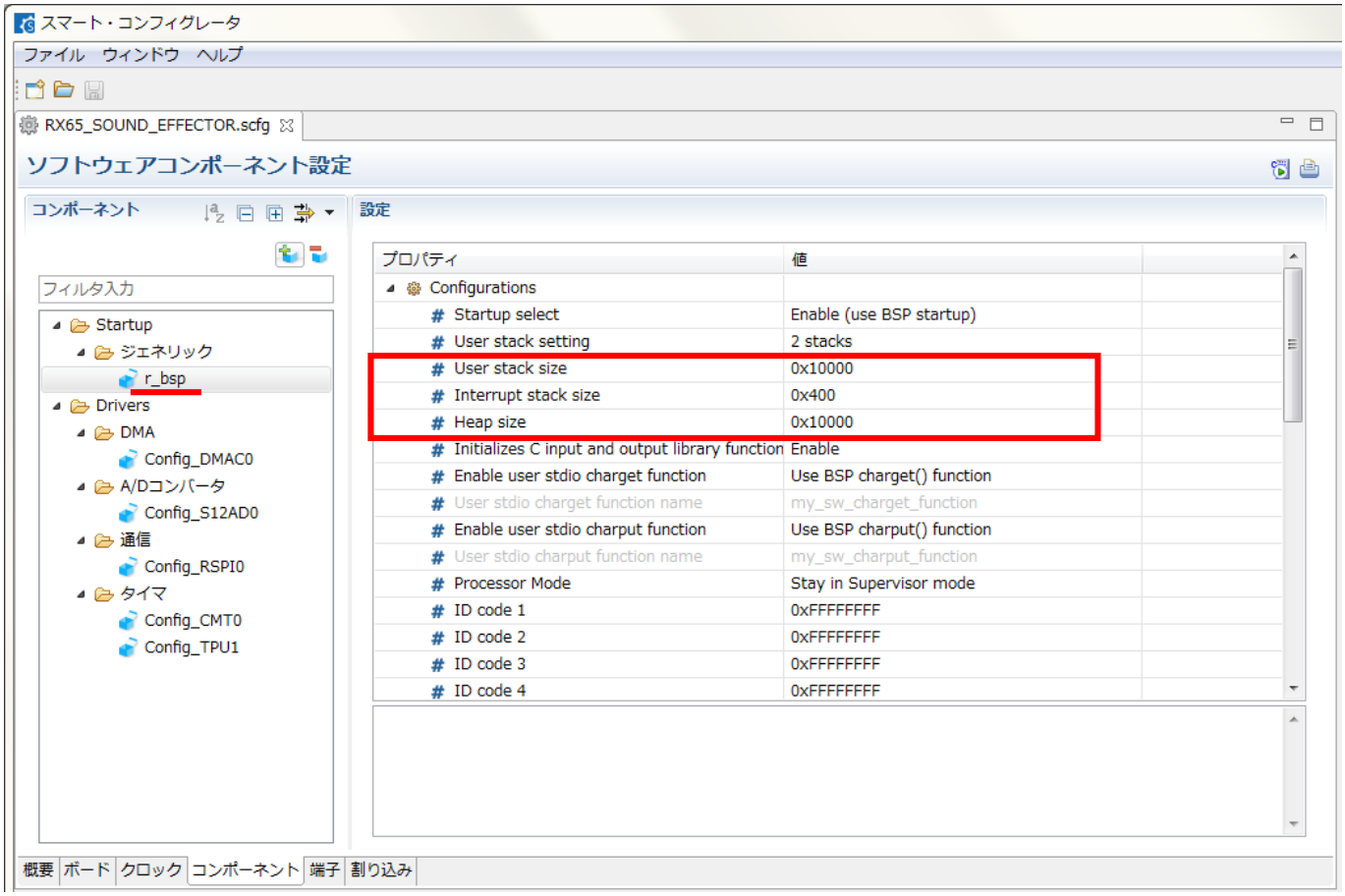
RAM（メインメモリ） 256kB　　（アドレス 0x0 ~ 0x0003 FFFF）
拡張 RAM 384kB　　（アドレス 0x0080 0000 ~ 0x0085 FFFF）

となっています。

テンプレートのプログラムでは、音声データのバッファに拡張 RAM 側（384kB 全て）を割り当てています。

メインメモリ側は、ワーク変数やスタック領域として使用されます。

スマート・コンフィグレータで、RAM に関する設定があります。



	割り当て値（デフォルト値）	用途
User stack size	0x10000 (0x1000)	自動変数
Interrupt stack size	0x400 (0x400)	割り込みでのスタック
Heap size	0x10000 (0x400)	malloc 等のライブラリ関数

User stack size は、自動変数（通常の変数(*1)）で使用される領域です。テンプレートでは、0x10000(64kB)に増やしていますが、足りない場合は、変更が可能です。
(*1)main 関数や関数内で定義している変数
（メモリではなく CPU レジスタに割り当てられる場合もあります）
（最適化の結果、定数（即値）に置き換えられる場合もあります）

Heap size は、malloc や calloc で割り当てられた変数が使用する領域です。テンプレートでは、0x10000(64kB)に増やしていますが、足りない場合は、変更が可能です。

static を付けて定義した変数は、User stack, Heap で予約している領域とは別な領域に割付されます。（グローバル変数も同様です）

このテンプレートでは、User stack, Heap で、それぞれ 64kB の領域を予約していますので、
・自動変数として、64kB より大きなサイズを使用したい
・malloc で 64kB 以上のメモリを割り付けたい
・static 変数で大きなサイズを使用したい
というケースでは、上記値を適宜変更してください。（変更後は、「コード生成」ボタンで、ソースコードを更新する必要があります。）

拡張 RAM 側は、音声データのバッファとして使用しています。

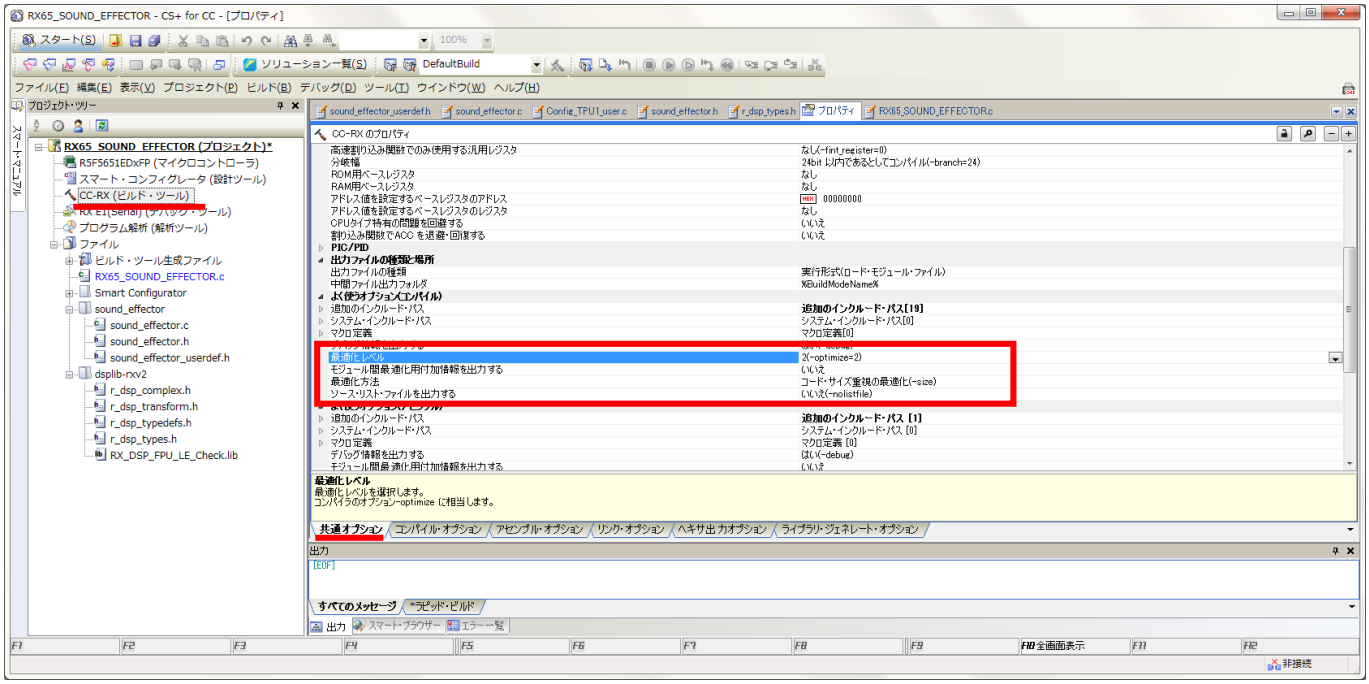
sound_effector.h
```
#define BUF_RAM_START_ADDR 0x00800000  //g_inbug を割り当てるアドレス
#define BUF_RAM_END_ADDR     0x0085FFFF
```

sound_effector.c
```
//アドレス割付（384kB の拡張 RAM 領域）
#pragma address g_inbuf=BUF_RAM_START_ADDR
```

※拡張 RAM は、入力を使用しないテンプレートでは、g_outbuf に割り当てており、入力を使用するテンプレートでは g_inbuf に割り当てています

上記の、ヘッダファイル、ソースファイルで割り当てを行っていますので、この部分を変更して割り当てサイズやアドレスを変更する事が可能です。

8. コンパイラの最適化に関して



CS+の、CC-RX（ビルド・ツール）－「共通オプション」タブ

最適化レベル　**2(-optimize=2)**
最適化方法　**コード・サイズ重視の最適化(-size)**

の部分で、最適化のレベルと、方法を選択可能です。

・最適化による速度の例
サンプル：RX65_SOUND_EFFECTOR_VOCORDER（FFT, IFFT を実行するプログラム）の 1 サイクル（1024 データを処理する時間）

	コード・サイズ重視の最適化 (-size)	実行性能重視の最適化 (-speed)
最適化 OFF(-optimize=0)	5.9ms	←
レベル 1(-optimize=1)	4.54ms	4.38ms
レベル 2(-optimize=2)	**3.88ms [デフォルト]** (*1)	3.80ms (*2)
レベル max(-optimize=max)	3.82ms	3.72ms

最適化は、CS+のデフォルトでは、有効(-optimize=2, -size)となっています。データのリアルタイム処理を行う場合、

最適化方法　**実行性能重視の最適化(-speed)**

を選択した方が有利な場合もありますので、最適化の設定が変えられるという事を認識願います。

なお、最適化レベル 2 で、「コード・サイズ重視の最適化」と「実行性能重視の最適化」を選んだ場合のプログラムサイズは以下の様になります。

	コード・サイズ重視の最適化 (-size) (*1)	実行性能重視の最適化 (-speed) (*2)
使用 ROM サイズ	0x3ed3 (15.7kB)	0x4293 (16.6kB)

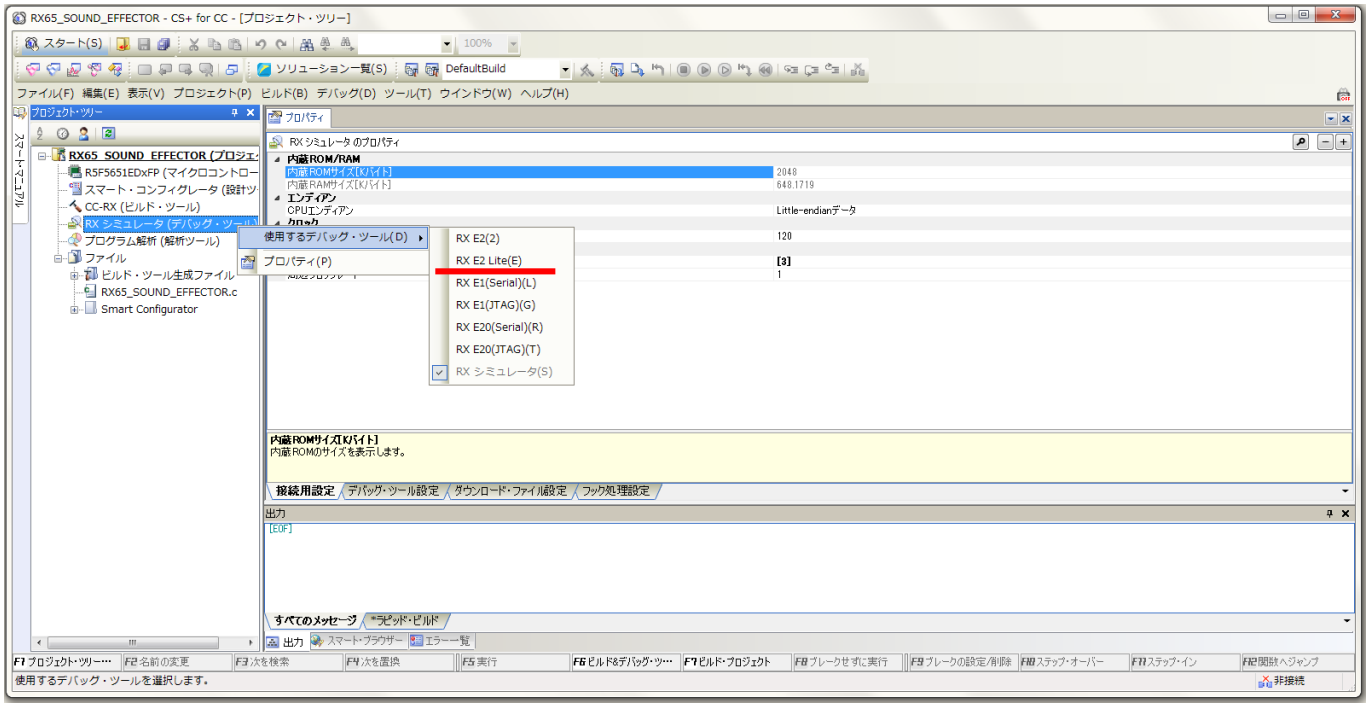
フリー版のツール(CC-RX)では、リンクサイズの上限は 128kB となります。このケースですと、上限サイズまでは十分に余裕がありますので、実行性能重視の設定を選択しても良いかと思います。

※なお、マイコンチップの ROM 容量は 2MB です
（製品版の CC-RX コンパイラや、GNURX を使用した場合、2MB までのプログラムを作成する事ができます）

9. デバッグに関して

9.1. デバッガ接続のための設定

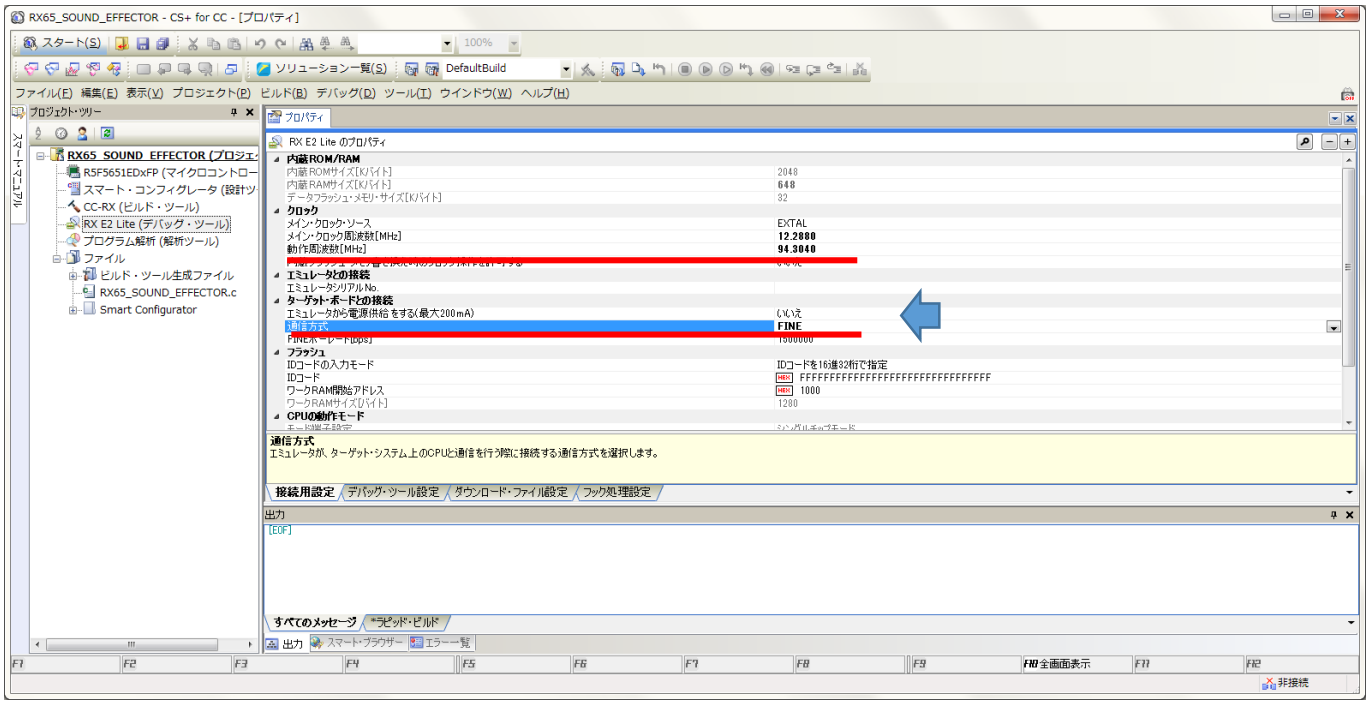
ルネサスのデバッガ(E1, E2, E2Lite, E20)を使用する場合、CS+で以下の設定を行う必要があります。



RX シミュレータ(デバッグ・ツール) を右クリックで使用するデバッガを選択します。ご利用されているデバッガを選択してください。

RX E1, RX E20 は、(Serial)と(JTAG)がありますが、(Serial)側を選択してください。

※RX シミュレータが選択されている場合は、実装置(サウンドエフェクタ)には、プログラムの転送等が行われません



メイン・クロック周波数[MHz] **12.288** を入力

動作周波数[MHz] **98.304** を入力

E2/E2Lite の場合は、

通信方式 **FINE** を選択

（E1, E20 の場合は、この選択項目はグレーアウトして変更できませんが、FINE となっている事を確認してください）

矢印の項目： エミュレータから電源供給をする **いいえ**（デフォルト値）を選択してください。

9.2. デバッグ向けに最適化を切る設定

プログラム実行中に、ソースレベルのステップ実行を行う場合や変数の値をモニタしたい場合、6 章の最適化の設定の、最適化レベルの設定を変更する事を推奨致します。

CC-RX（ビルド・ツール）「共通オプション」タブ　最適化レベル　**0(-optimize=0)**

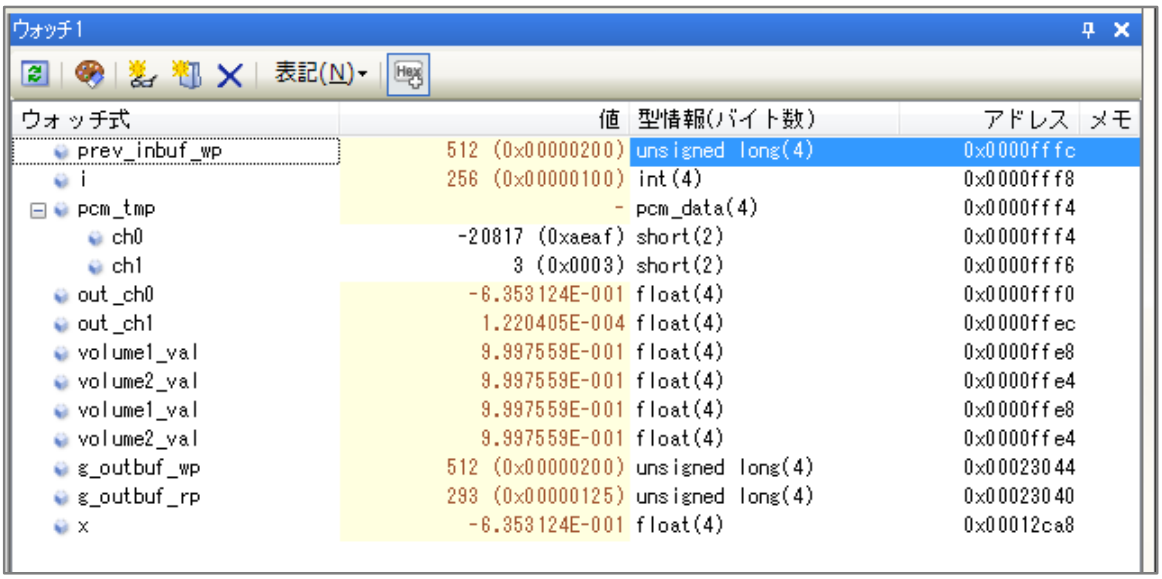
この設定は、コンパイル時の最適化に関する設定で、デフォルトでは最適化は有効です。最適化有効の場合、プログラムの実行順が変わったり、変数がまとめられるといった最適化が行われる事があり、デバッグ時に期待した実行結果が得られないケースがあります。

最適化を OFF とすると、ソースと、実行形式（バイナリ）の対応が容易になります。また、変数のモニタが容易になります。

最適化 ON 時は、変数はレジスタに割り付けられ、変数の使用が終わると、そのレジスタは別な用途に割り振られるため、変数の変化が追い難くなるケースがあります。

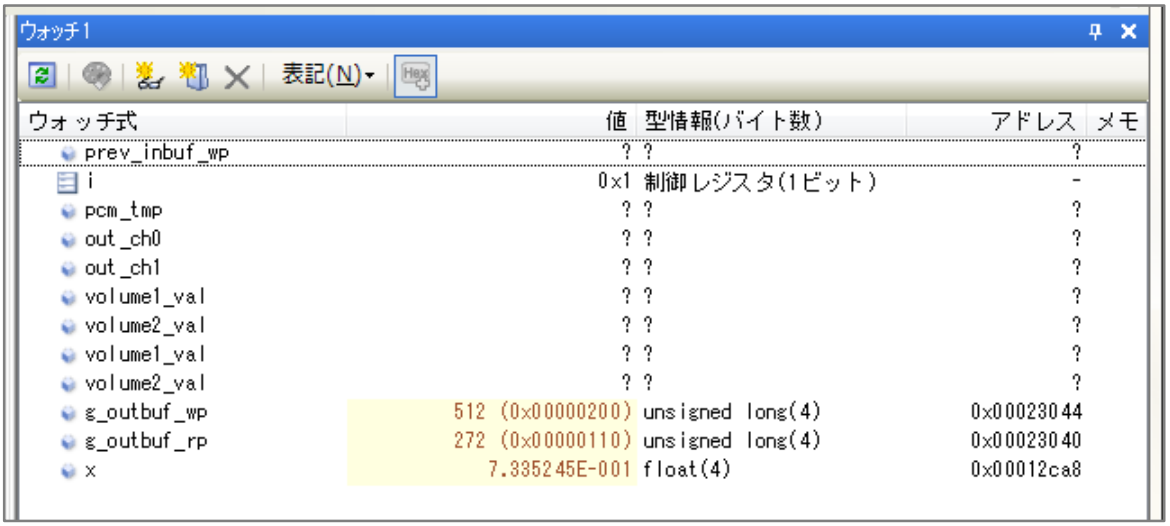
※最適化を OFF にすると、実行時間は増加（一般的には倍程度）となりますので、タイミングがシビアなアプリケーションでは、最適化の設定のために期待と異なる動作となる場合がありますので、ご注意ください。

・最適化：OFF(-optimize=0)設定



最適化を OFF とした場合、ほとんどの変数がメモリに割り付けられるため、プログラムをどこでブレークしても、変数の値がモニタできます。

・最適化：デフォルト(-optimize=2)設定



一方最適化を ON にすると、グローバル変数（g_で始まる変数）は、メモリに割り付けられているため、値がモニタできますが、それ以外の変数は?となって、モニタできていません。なお、上記では、x の値はモニタできていますが

```
volatile static float x;
…
x = out_ch0;
```

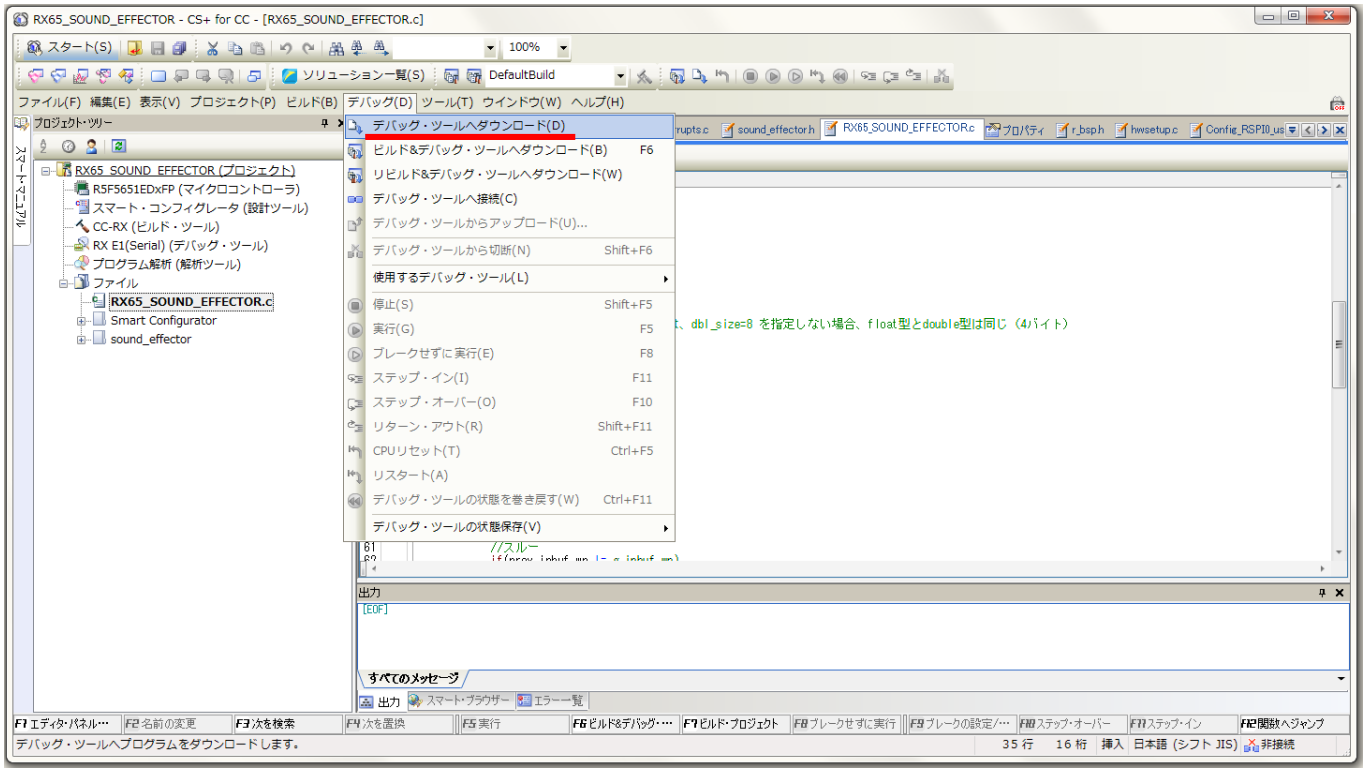
上記の様なコードとしています。volatile で最適化の対象外とした、static 指定の変数にモニタしたい値(out_ch0)をコピーしています。（最適化を有効にしつつ、変数値をモニタしたい場合、このような手法があります）

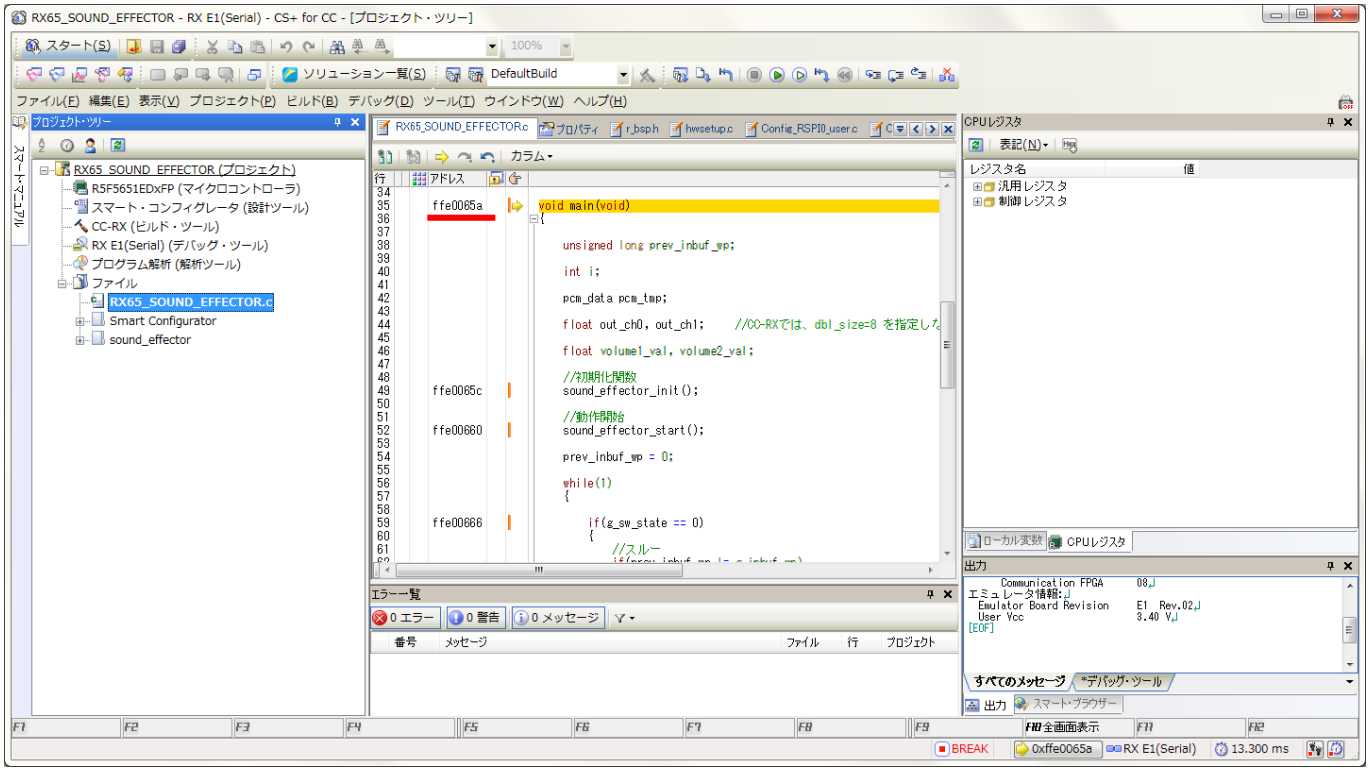
9.3. デバッグツールの使用法

本機器の 14P コネクタに、デバッガを接続し、電源（AC アダプタを）接続した状態で、

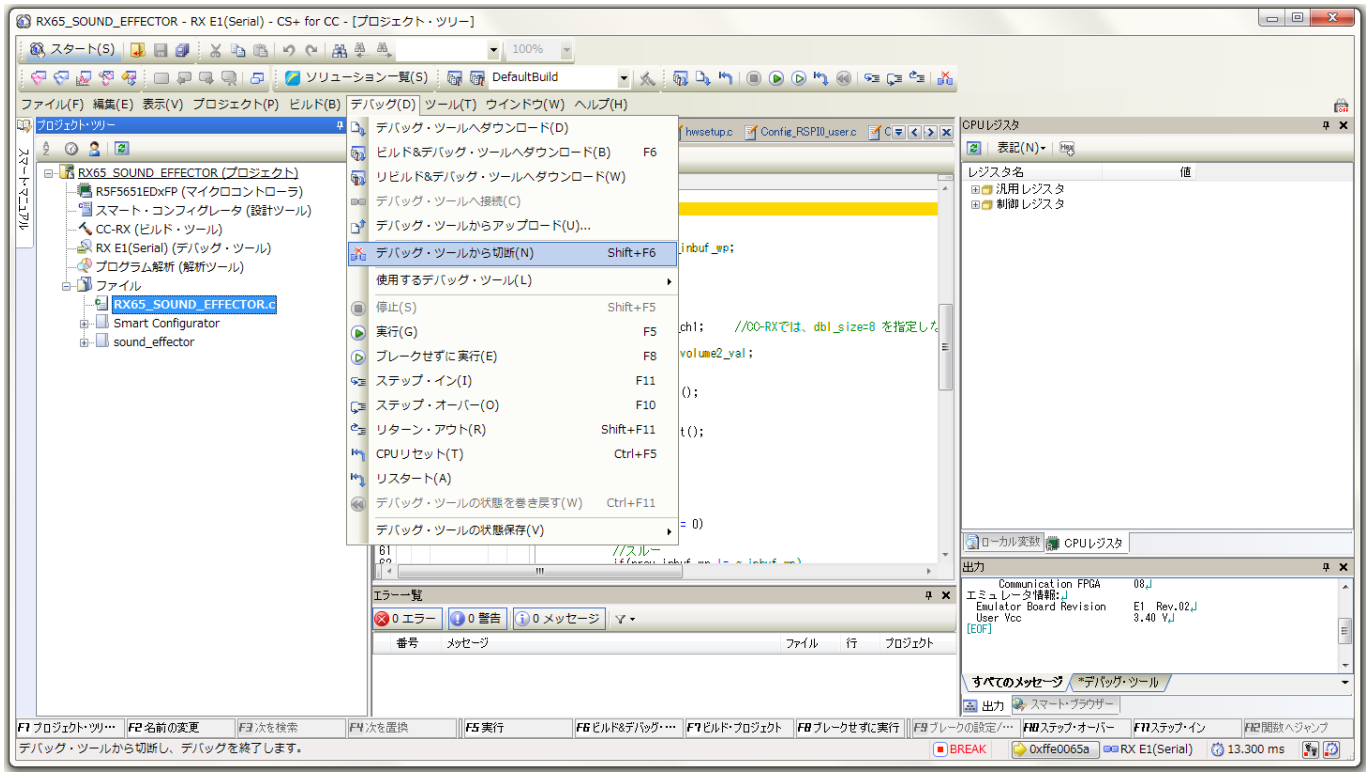
デバッグ－デバッグ・ツールへダウンロード（ビルド済みの場合）
デバッグ－ビルド＆デバッグ・ツールへダウンロード（ソースを変更した場合等）

でデバッガ経由で、マイコンにプログラムが転送されます。





アドレスの欄に数値が入り、void main(void)の行がオレンジになれば、デバッガの接続とプログラムのマイコンへのダウンロードは成功しています。

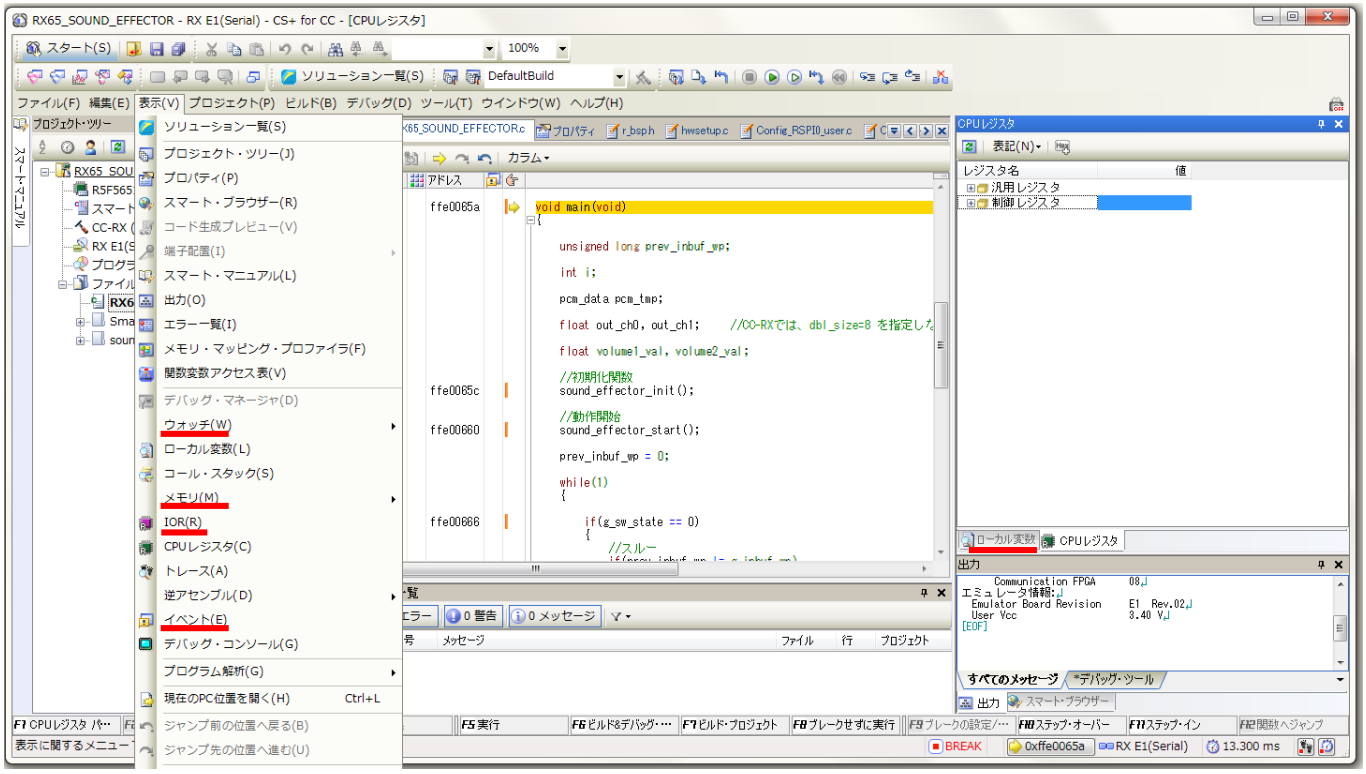


デバッグを終了させる際は、

デバッグ－デバッグ・ツールから切断

を実行してから、「電源を落とす」、「デバッガを外す」様にしてください。

9.4. 各種情報のモニタ

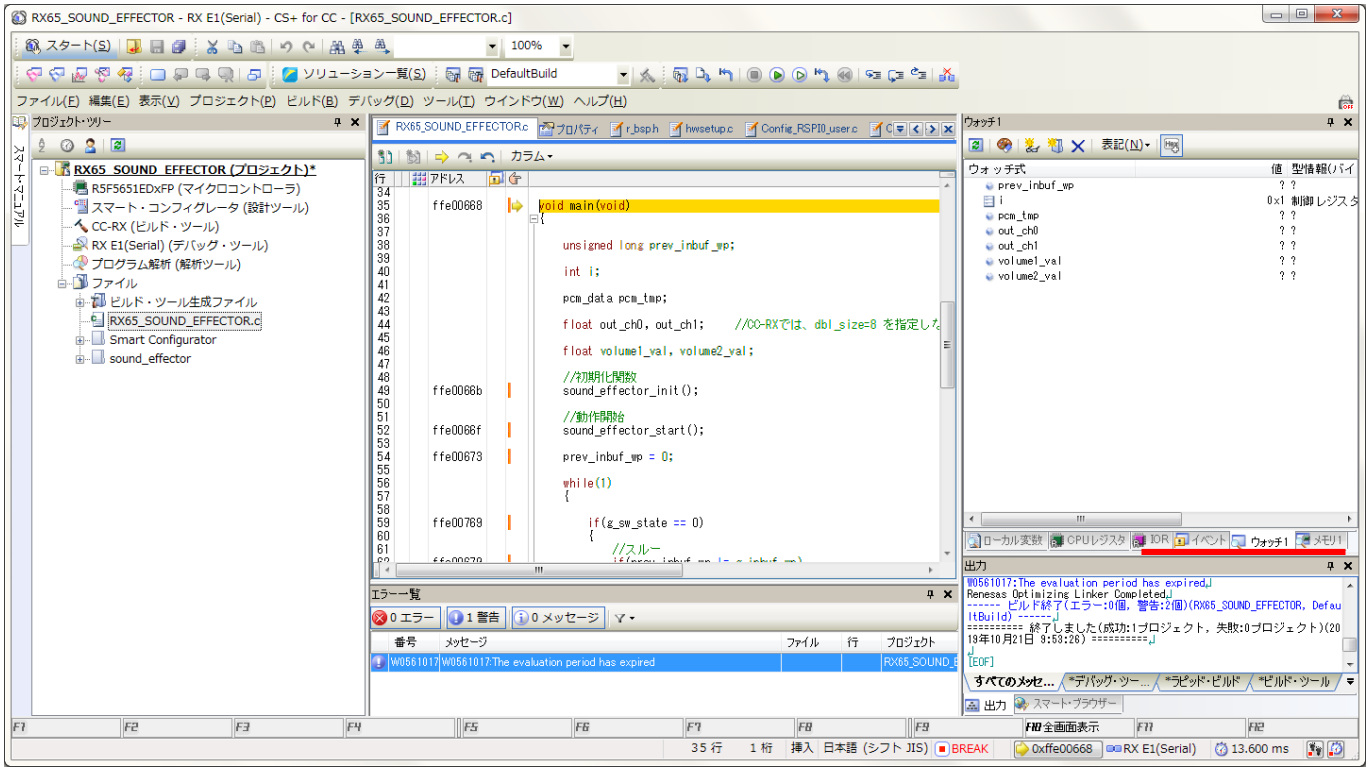


変数のモニタ等は、ローカル変数は、右側の「ローカル変数」タブ内に入ってきますので、そこでモニタが可能です。

観測する変数を追加する場合は、

表示ーウォッチーウォッチ(1)

で、ウォッチウィンドウを追加してください。同様に、メモリのダンプイメージを確認したい場合は、「メモリ」。マイコン内部のレジスタ値をモニタする場合は、「IOR」。ブレークポイントを確認したい場合は、「イベント」を追加してください。



追加した、各種情報は（デフォルトでは）右側の表示ウィンドウに追加されますので、タブを切り替えながら観測したい情報にアクセスしてください。ウォッチは、最初は何も変数が登録されていませんが、ユーザ側で任意の変数を追加できます。

ファンクションキーで、プログラムを実行したり、1 行ずつ実行したりといった操作が可能です。

良く使用するのは、下記です。（デバッグメニューからも同様の操作ができます）

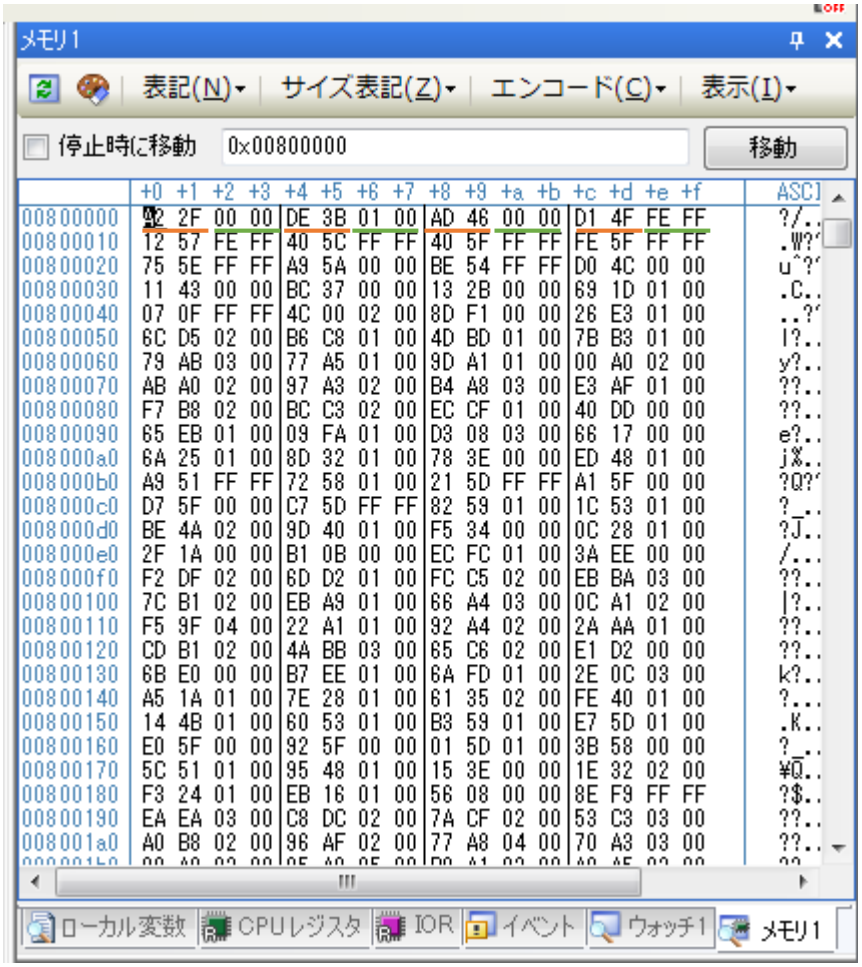
F5　実行

F10, F11 ステップ実行

Shift-F5　停止

Ctrl-F5　リセット

・ch0 に sin 波を入力, ch1 は信号入力なしのメモリダンプの例



0x0080 0000 ~ 0x0080 000F までのデータは、以下の様になっています。

ch0 のデータ　:0x2FA2 0x3BDE 0x46AD 0x4FD1

12194　15326　18093　20433

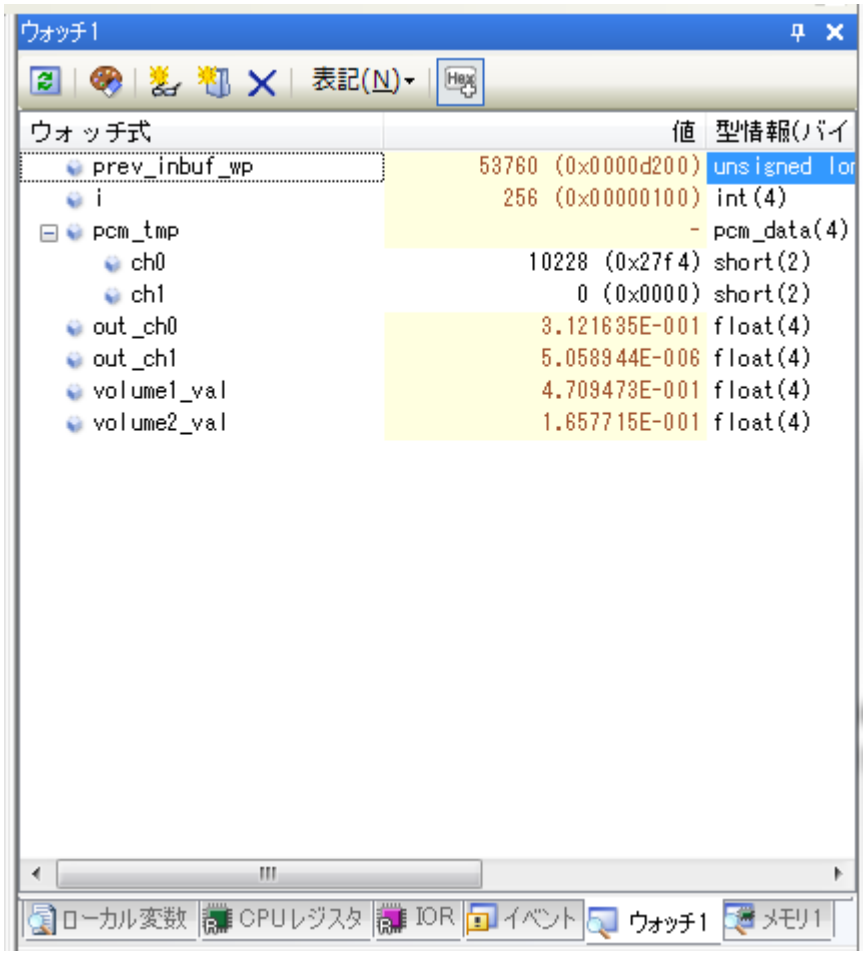
ch1 のデータ　:0x0000 0x0001 0x0000 0xFFFE

0　1　0　-2

1 つのデータは、2bytes(16bit)です。CPU のデータは、LittleEndian となっていますので、0x0080 0000 ~ 0x0080 0001 のデータは、0x2FA2 となります。符号付き 16bit、2 の補数表現のデータなので、0xFFFE は-2 を表します。（ch1 側のデータは、ノイズ等で完全に 0 にはなっていませんが、ほぼ 0 です）

※RX マイコンは、データに関しては、バイ・エンディアン仕様（BigEndian 選択可能）となっています。デフォルトは、LittleEndian で、テンプレートのプロジェクトも LittleEndian となっています。

・ウォッチの例



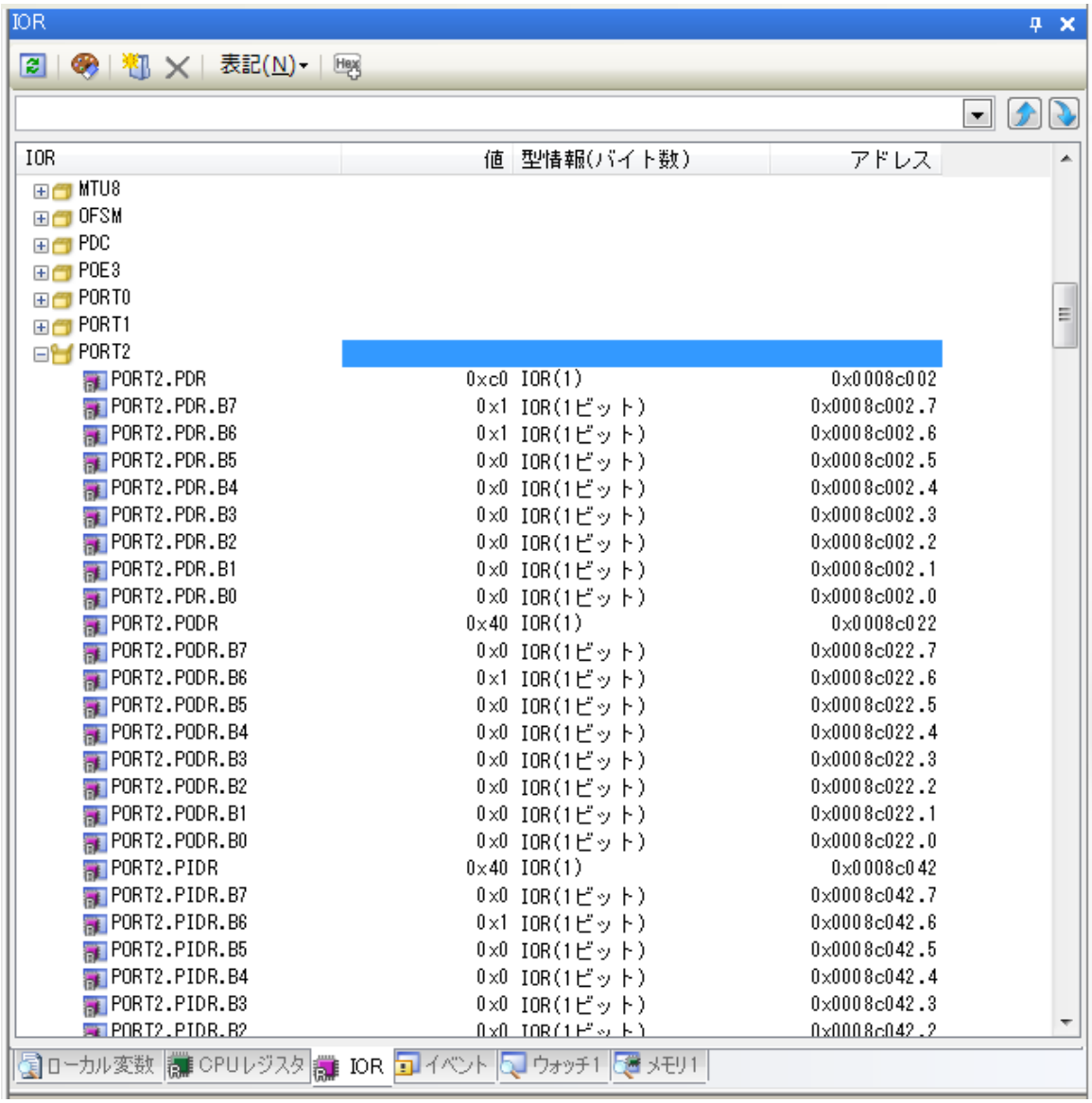
本プログラム(RX65_SOUND_EFFECTOR_TEMPLATE)では、volume1_val, volume2_val は 1 に正規化していますので、

volume1_val　47.1%の位置（右に目一杯回した場合が 1）

volume2_val　16.6%の位置

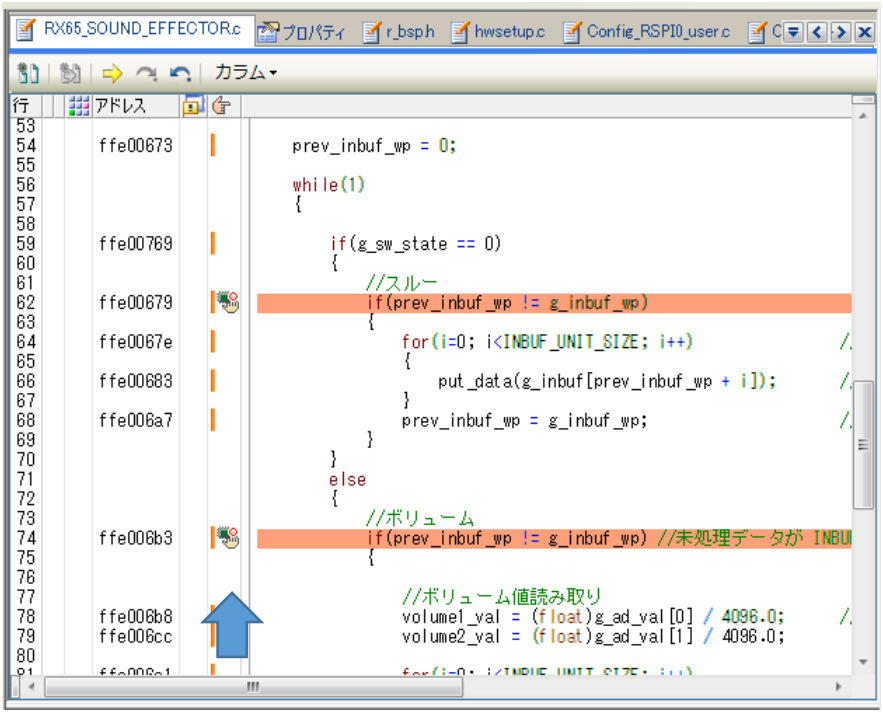
である事や、この瞬間の出力データ(pcm_tmp)の ch0 側が 10288 で、ch1 側が 0 であること等がモニタできます。

・IOR（IO レジスタ）表示例

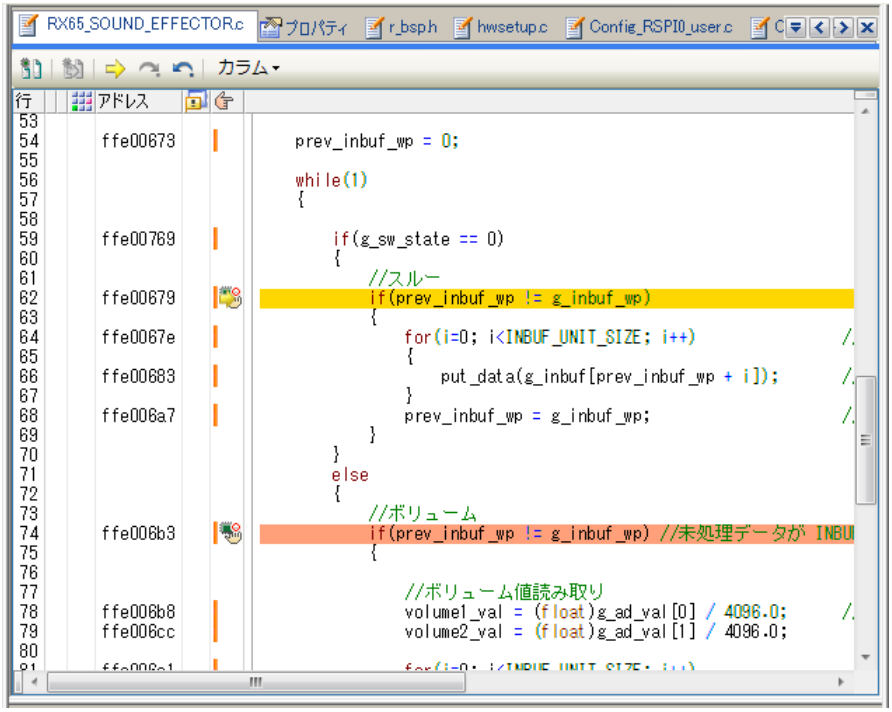


PORT2 の PDR（Port Direction Resistor，入出力方向）、PODR（Port Output Direction Resistor，出力の L/H）、PIDR（Port Input Direction Resistor，入力 L/H）の等、マイコンの制御レジスタ値のモニタ、変更が行えます。

9.5. ブレークポイントの設定



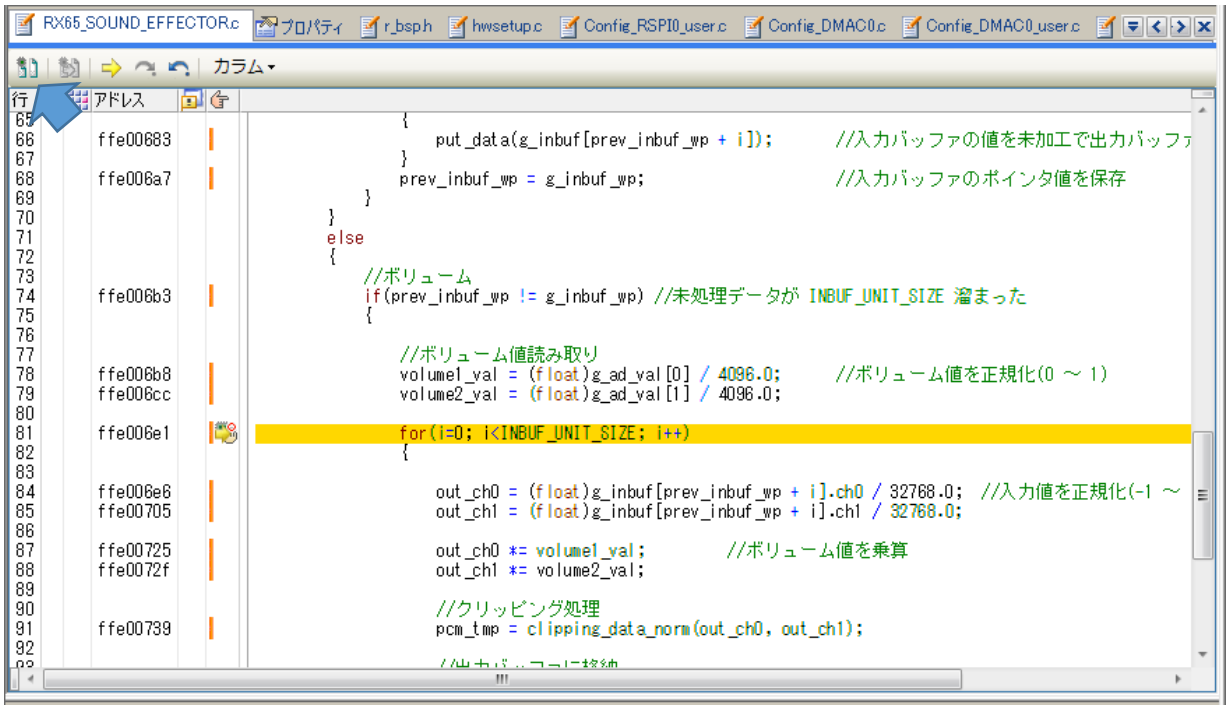
↑のカラムをマウスでクリックすると、ブレークポイントの設定が可能です。例えば、上記 2 行にブレークポイントを設定してプログラムを実行(F5)してみます。



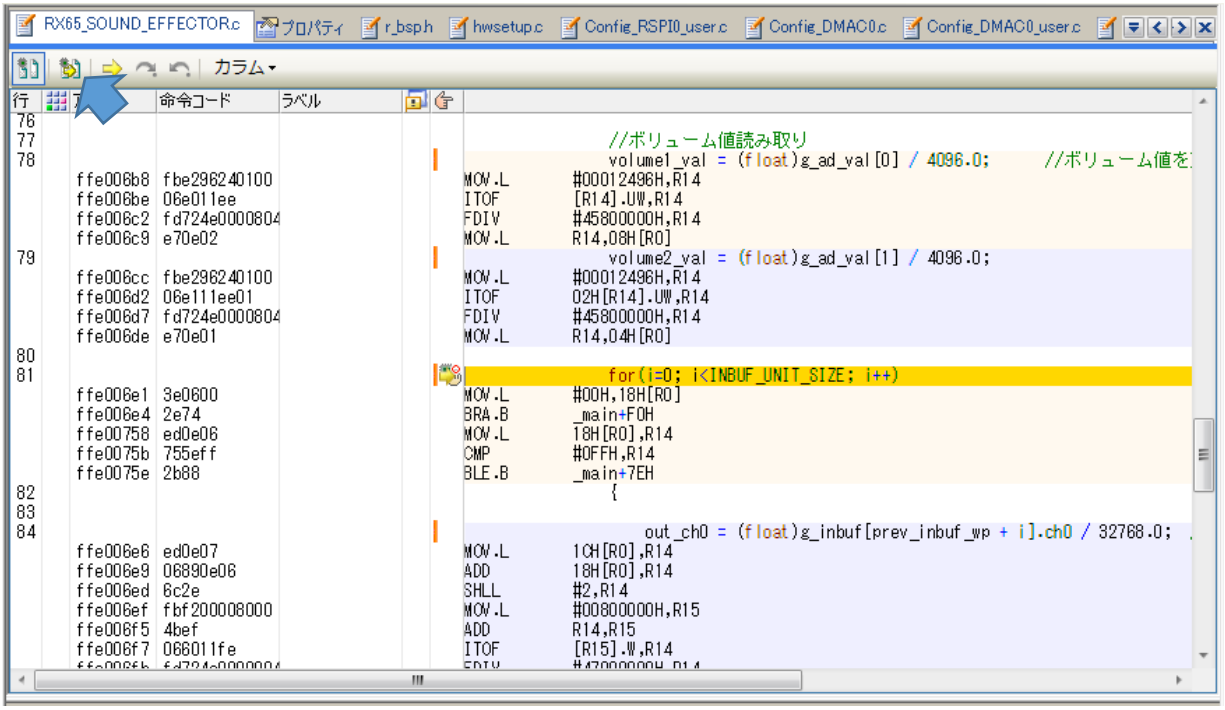
この場合は、上の行でプログラムの実行が止まった(黄色の→の行)ので、g_sw_state は 0 になっており、Push-SW は押されている(スルー)ということが判ります。

9.6. ソースとアセンブラ命令に関して

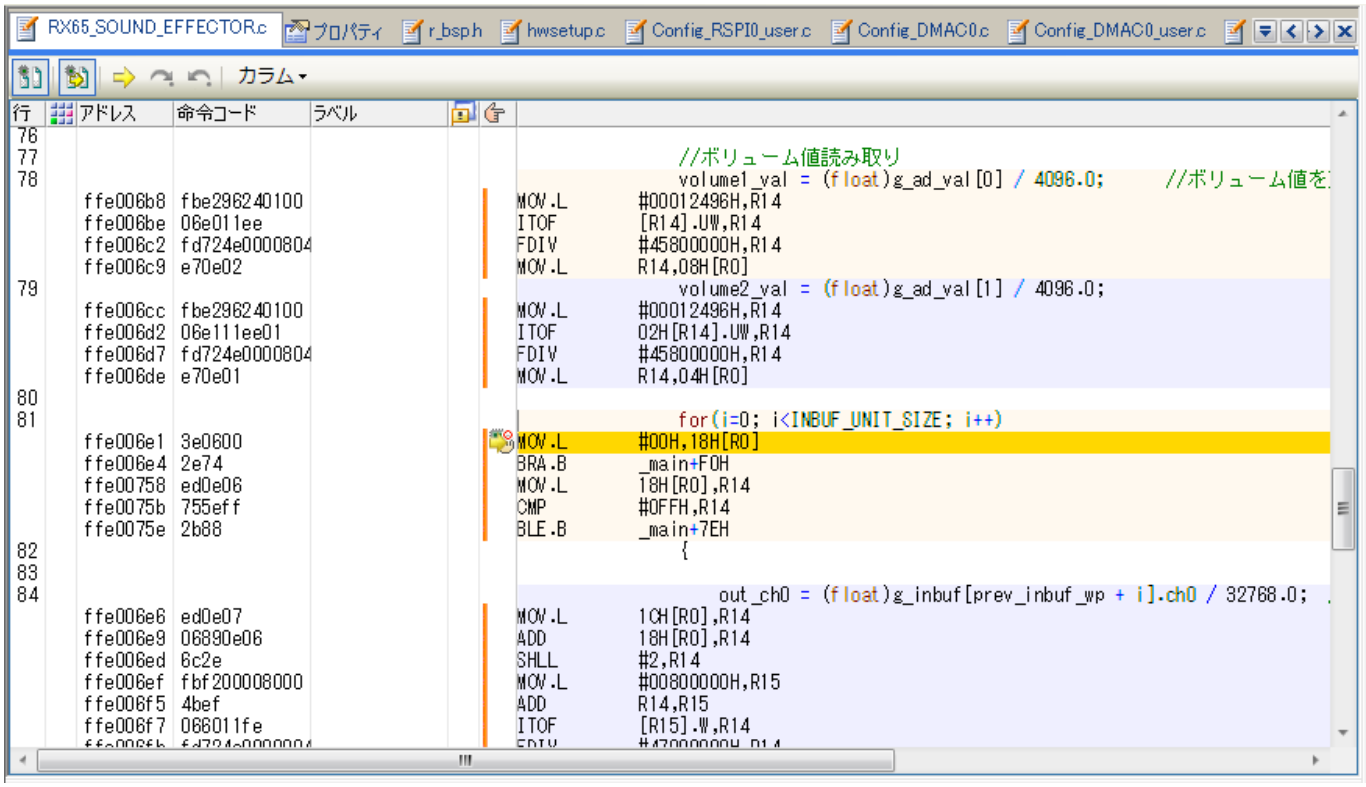
・ソースレベルの表示



・ソース＋アセンブラレベルの表示（前の図の矢印のボタンを押す）



・実行単位をアセンブラ命令レベルとする（前の図の矢印のボタンを押す）

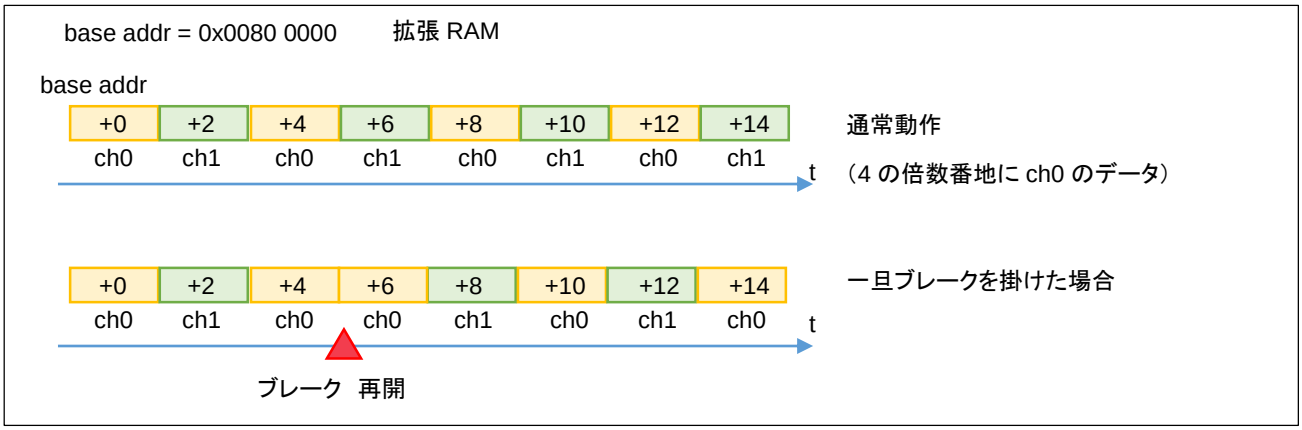


標準では、C 言語のソースレベルのデバッグとなりますが、アセンブラコードの表示や、ステップ実行の単位をアセンブラ命令レベルに切り替える事ができます。

9.7. ブレークの注意点

デバッガを接続した場合、任意の箇所でプログラムの停止を行う(ブレークポイントの設定)事ができますが、プログラムを停止させた場合、マイコンーオーディオ CODEC 間の通信が停止します。(音声の出力は停止します)

プログラムの実行を再開した場合、マイコンーオーディオ CODEC 間の通信が再開し、音声の出力も再開されますが、その際左右の音(ch0 と ch1)が逆になるケースがあります。



プログラムのブレーク(停止)、再開のタイミングによっては、通信が再開された後 ch0 と ch1 の位置が逆になる事が起こり得ます。(この場合でも、リセットして再度先頭からプログラムを実行すれば、逆になる現象は解消されます)

9.8. LED デバッグ

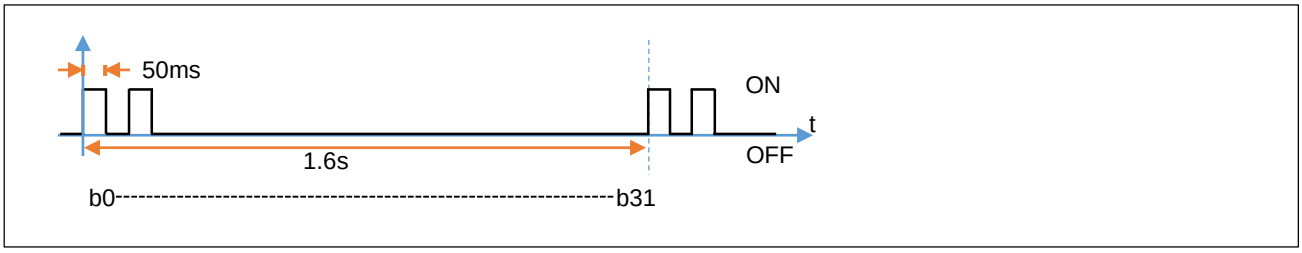
本機器には、Write/Err LED が搭載されています。この LED は、エラー時に点滅させてエラーを知らせる等の用途で使用可能です。

(1)sound_effector_userdef.h 内で、
#define DEBUG
を有効にする（デバッグ無効の時は、LED の制御ルーチンは有効になりません）

(2)g_err_led_patturn に点灯、点滅のパターンを代入する

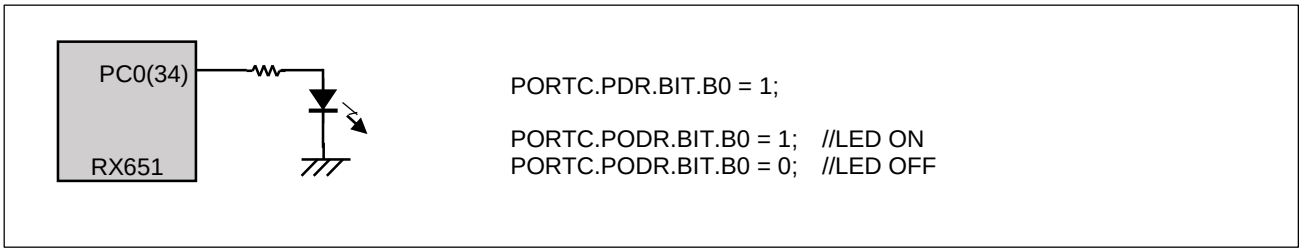
```
g_err_led_patturn = LED_ON;         //LED を ON にする
g_err_led_patturn = LED_OFF;        //LED を OFF にする
g_err_led_patturn = LED_BLINK;      //LED を点滅させる
g_err_led_patturn = LED_BLINK1;     //LED を点滅させる（高速）
g_err_led_patturn = LED_BLINK2;     //LED を点滅させる（2 回点滅）
g_err_led_patturn = 0x8880000;      //LED を任意のパターンで点滅させる（3 回点滅）
```

g_err_led_patturn は、32bit の変数で、0b00000000000000000000000000000101(0x5)の場合、



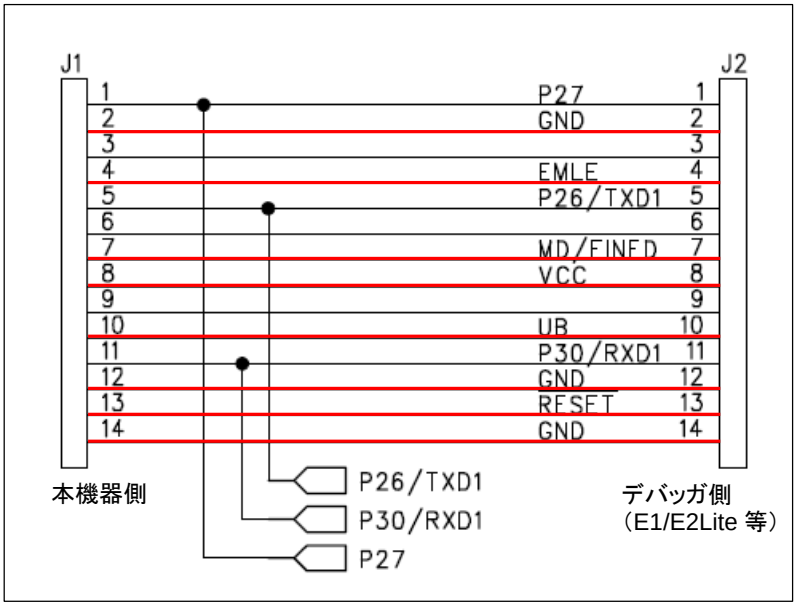
50ms を基本周期とし、ON-OFF-ON-OFF(x29)の点滅を繰り返します。（32bit パターンで 1.6 秒周期）
（見た目で簡単に判別できる、3 パターンの点滅を定数定義していますが、1 周期 32 個の ON/OFF パターンを任意に定義する事が可能です。

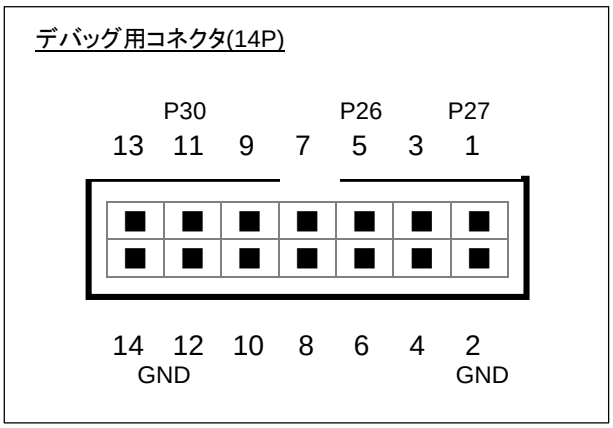
エラー時等に点滅や点灯させる事により、デバッグや状態のモニタに使用できます。



9.9. ポートデバッグ

信号のタイミングをモニタする場合、マイコンの汎用 I/O ポートをデバッグに使用する事ができます。(基本的には、要オシロスコープ)





デバッガ用コネクタ(14P)に、デバッグ時は使用しない端子が接続されています。P26/TXD1, P30/RXD1, P27 の 3 端子です。これらの 3 端子を外部に引き出し、オシロスコープに接続してください。GND は、2,12,14 のいずれかから引き出してください。

デバッガでは、2, 4, 7, 8, 10, 12, 13, 14 の 8 本の端子(上図赤線)を使用します。デバッガ使用時はこれらの端子はスルーでデバッガに接続してください。(これら 8 本の端子以外は、デバッガに接続しても、接続しなくてもどちらでも構いません)

ポートデバッグでは、とある処理にどの程度時間が掛かっているかを容易にモニタする事ができます。

ポートデバッグ機能を使用する際は、sound_effector_userdef.h 内で
#define DEBUG_PORT
を有効化(コメントアウトされていない状態)してください。

プログラムソース内では、

P27=1; (PORT2.PODR.BIT.B7 = 1; でも等価)
「時間をモニタしたい処理内容」
P27=0; (PORT2.PODR.BIT.B7 = 0; でも等価)

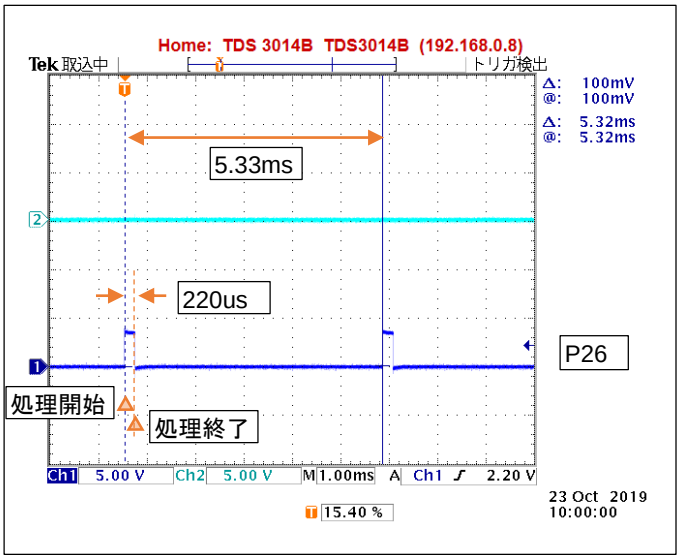
の様に記載し、P27 端子をモニタしてください。

・具体的な記載例

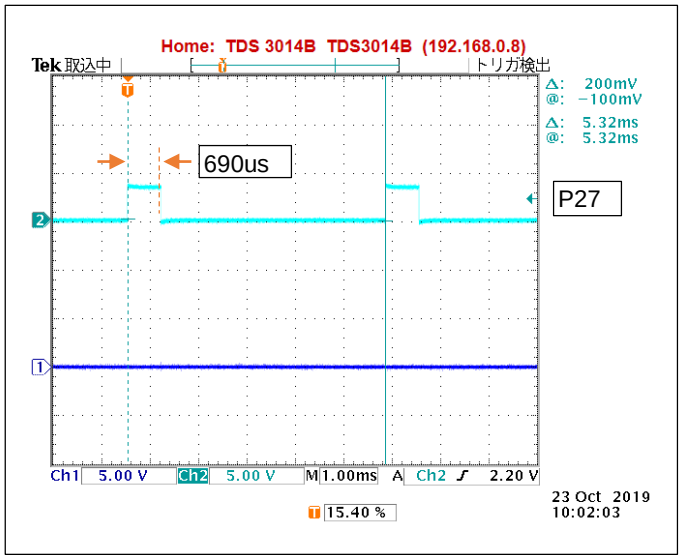
(TEMPLATE¥RX65_SOUND_EFFECTOR_TEMPLATE¥RX65_SOUND_EFFECTOR.c)

```
void main(void)
{

「中略」

    while(1)
    {

        if(g_sw_state == 0)
        {
            //スルー
            if(prev_inbuf_wp != g_inbuf_wp)
            {
                P26 = 1; //このコードを追加[時間観測開始]
                for(i=0; i<INBUF_UNIT_SIZE; i++)//未処理データが INBUF_UNIT_SIZE 溜まった
                {
                    put_data(g_inbuf[prev_inbuf_wp + i]);//入力バッファの値を未加工で出力バッファに格納
                }
                prev_inbuf_wp = g_inbuf_wp;//入力バッファのポインタ値を保存
                P26 = 0; //このコードを追加[時間観測開始]
            }
        }
        else
        {
            //ボリューム
            if(prev_inbuf_wp != g_inbuf_wp)//未処理データが INBUF_UNIT_SIZE 溜まった
            {

                P27 = 1; //このコードを追加[時間観測開始]
                //ボリューム値読み取り
                volume1_val = (float)g_ad_val[0] / 4096.0;//ボリューム値を正規化(0 ～ 1)
                volume2_val = (float)g_ad_val[1] / 4096.0;

                for(i=0; i<INBUF_UNIT_SIZE; i++)
                {

                    out_ch0 = (float)g_inbuf[prev_inbuf_wp + i].ch0 / 32768.0; //入力値を正規化(-1 ～ 1)
                    out_ch1 = (float)g_inbuf[prev_inbuf_wp + i].ch1 / 32768.0;

                    out_ch0 *= volume1_val;//ボリューム値を乗算
                    out_ch1 *= volume2_val;

                    //クリッピング処理
                    pcm_tmp = clipping_data_norm(out_ch0, out_ch1);

                    //出力バッファに格納
                    put_data(pcm_tmp);
                }

                prev_inbuf_wp = g_inbuf_wp;
                P27 = 0; //このコードを追加[時間観測終了]

            }
        }
    }//while
}
```

・スルー選択時



・ボリューム選択時



RX65_SOUND_EFFECTOR_TEMPLATE のプロジェクトは、Push-SW を押し込むと、「スルー」「ボリューム」の動作の切り替えとなりますが、その動作をポートデバッグで観測した場合上記の様になります。（コンパイラの最適化は OFF 設定）

このプログラムでは、1 回の動作で、256 個のデータを処理(INBUF_UNIT_SIZE=256 設定）ですので、5.33ms 毎に、正極性（山）のパルスが 1 つ出力されるのが期待値です。

スルーの動作を選択した場合、256 個のデータを入力バッファから読み取り、出力バッファに格納する事となりますが、この処理に約 220us 掛かっている事が判ります。

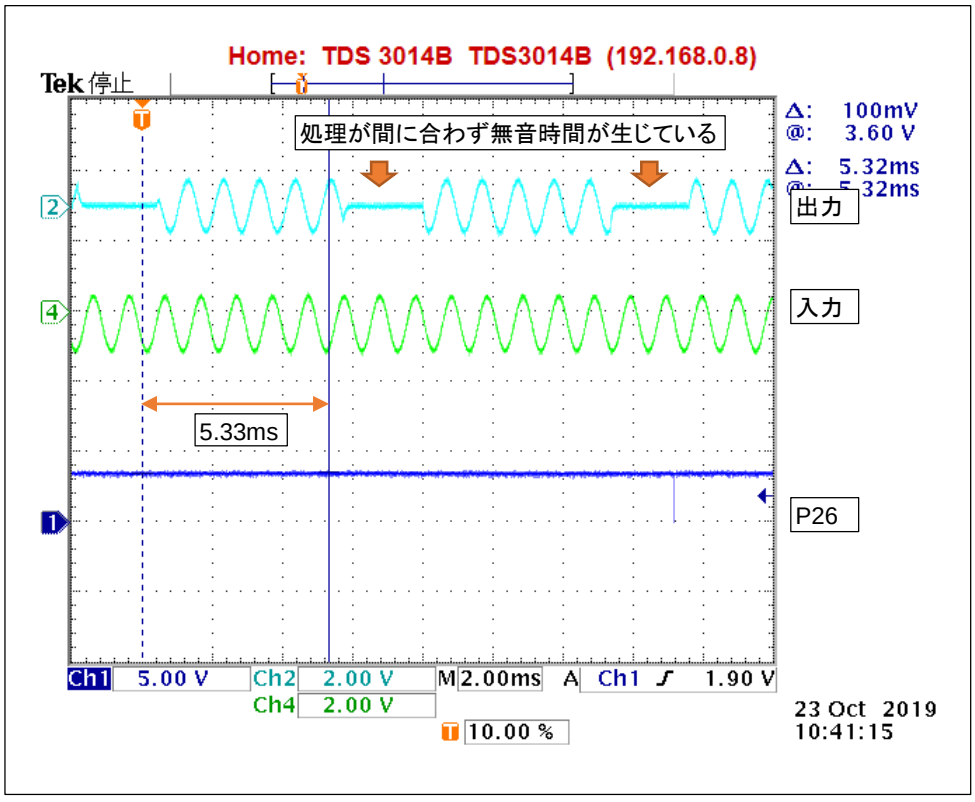
ボリュームの動作を選択した場合、256 個のデータを入力バッファから読み取り-1~1 に正規化、ボリューム値の乗算、-32768~32767 に正規化、出力バッファに格納する処理で、690us 掛かっている事が判ります。

このケースでは、5.33ms 以内に処理が終わらないと、データの取りこぼし（出力バッファのデータが空になり、無音データが出力される時間が生じる）が発生します。

オシロスコープ、またはパルス幅を測定できる装置が必要となりますが、データのリアルタイム処理が上手くいっているかを観測する有効なデバッグ方法となります。

・意図的に処理が間に合わない様なコードを作成した場合

```
            //スルー
            if(prev_inbuf_wp != g_inbuf_wp)
            {
                P26 = 1; //このコードを追加[時間観測開始]
                for(i=0; i<INBUF_UNIT_SIZE; i++)//未処理データが INBUF_UNIT_SIZE 溜まった
                {
                    put_data(g_inbuf[prev_inbuf_wp + i]);//入力バッファの値を未加工で出力バッフ
ァに格納
                }
                prev_inbuf_wp = g_inbuf_wp;//入力バッファのポインタ値を保存

                for(x=0; x<50000; x++) nop();//★[意図的にウェイトを挿入]

                P26 = 1; //このコードを追加[時間観測開始]
            }
        }
```



信号処理の部分のコードに意図的にウェイトを挿入した場合は、上図の様な波形となります。(P26 は、H に張り付いた様に見えますが、L となる瞬間もあります)

入力は、シグナルジェネレータから、サイン波を入れています。スルーの処理ですので、出力からは一定の遅延の後、入力波形が出てくるはずですが、無音の部分が出てしまっています。これは、出力バッファが空になる時間帯があるためです。

テンプレートのプロジェクトでは、ポートデバッグを有効にした場合、P26, P27, P30 の端子は、汎用出力に設定されますが、これらの端子を入力にして、外部から何らかのトリガーを与える事も可能です。

P26, P27, P30 は、ボード上でプルアップされていますので、入力モードに設定した場合は、端子のレベルは H(1)となります。

・P30 を入力とする例

```
PORT3.PDR.BIT.B0 = 0;  //P30 を入力に設定
```

上記コード実行後、

```
PORT3.PIDR.BIT.B0
```

の（レジスタ）値を観測すれば、P30 に L/H どちらの信号が入力されているか判りますので、外部からの信号をデバッグに使用する事ができます。

プログラムでは、レジスタ名の別名を sound_effector.h 内で定義していますので、どちらの名称でアクセスしても等価です。

・ポートレジスタの別名定義

別名	レジスタ名	用途
P26	PORT2.PODR.BIT.B6	P26 出力データの設定
P27	PORT2.PODR.BIT.B7	P27 出力データの設定
P30	PORT3.PODR.BIT.B0	P30 出力データの設定

9.10.シリアルポートデバッグ

ポートデバッグで使用可能な端子のうち、P26, P30 端子は、

P26→TXD1
P30→RXD1

の機能を持つ端子となっており、これらの端子は、SCI(UART)として使用する事ができます。

P26, P30, GND に、「USB-1S」「USB-Adapter」「USB-Adapter-RX14」（左記 3 点、北斗電子製）または、市販の USB-Serial 変換機器を接続する事により、PC 上の仮想端末に情報の出力を行ったり、PC のキーボードから本機器に指示を与える事ができます。

	本機器デバッグコネクタ(14P)	USB-1S(5P)	USB-ADAPTER (20P)	USB-Adapter-RX14(14P)
P26/TXD1	5	3(TXD) (*1)	15(TXD) (*1)	コネクタ直挿し（SW2 は RUN 側）
P30/RXD1	11	4(RXD) (*2)	17(RXD) (*2)	
GND	2,12,14	5(GND)	2,4,6,8,10,12,14,16 (GND)	

(*1)USB-1S, USB-ADAPTER 側は入力端子です（TXD は接続相手基準の表記です）
(*2)USB-1S, USB-ADAPTER 側は出力端子です（RXD は接続相手基準の表記です）

本機器と、USB-Serial 変換機器を接続する場合、上記の端子同士を接続してください。市販の USB-Serial 変換機器を接続する場合は、P26/TXD1 に RXD（受信端子）、P30/RXD1 に TXD（送信端子）を接続してください。

P26/TXD1, P30/RXD1 は、0-VCC(3.3V)系の CMOS 入出力です。USB-VBUS(5V)と、これらの端子間にプルアップ抵抗が挿入されている機器は接続しないでください。

USB-Adapter-RX14 は、本機器に直挿しで接続できますが、USB-Adapter-RX14 の USB コネクタが下向きとなりますので、本機器を適当な台の上に置いた形で、USB-Adapter-RX14 を挿してください。
（USB-Adapter-RX14 は、デバッガ接続用の 14P コネクタを持っていますので、デバッガ接続と併用可能です）

テンプレート

TEMPLATE¥RX65_SOUND_EFFECTOR_TEMPLATE_SCI

は、シリアルポートデバッグのサンプルとなっています。

上記テンプレートは、Push-SW により、スルーとボリュームを切り替える（RX65_SOUND_EFFECTOR_TEMPLATE と同じ）動作となります。

115,200bps, 8bit, ストップビット=1, パリティなしの設定で、仮想端末（仮想 COM ポート）を開き、プログラムを実行すると、端末には下記の表示が出ます。

```
Copyright (C) 2019 HokutoDenshi. All Rights Reserved.
-Sound Effector debug console-
debug MENU:
  i: Input device value print
  p: Max-Min input print(every 5sec) toggle
>
```

キーボードから i を入力すると、Push-SW とボリュームの情報が表示されます。

```
input: Push-SW->01, Volume(0,417,627)
```

Push-SW は、01（押していない）, 00（押している）。ボリュームはカッコ内の 3 値(0-4095)での表示となります。

キーボードから p を入力すると、入力の信号のレベルの max-min 値が表示されます。（5 秒毎に更新、もう一度 p を入力すると、表示 OFF となります）

```
print-max-min: toggle ON
max-min:
  ch0: (24902 - -24907)
  ch1: (4 - -5)
max-min:
  ch0: (24902 - -24907)
  ch1: (4 - -5)
```

デバッグ情報を、仮想端末に表示される様にすると、より細かなデバッグができますので、このようなデバッグ手法も使用できるという事を認識ください。

※テンプレートでは、以下の関数が使用できます

sciInit()　**初期化関数**
sciRead()　**キーボードからの文字の読み取り**
sciWrite(unsigned char c) / sciWriteBuf(unsigned char c)　**c（1 文字）を表示**
sciPrint(char *str) / sciPrintBuf(char *str)　***str（文字列）を表示**
sciPrintByte(unsigned char a) / sciPrintByteBuf(unsigned char a)　**a(Byte)を 16 進数 2 桁で表示**
sciPrintWord(unsigned short a) / sciPrintWordBuf(unsigned short a)　**a(Word)を 16 進数 4 桁で表示**
sciPrintDword(unsigned long a) / sciPrintDwordBuf(unsigned long a)　**a(Dword)を 16 進数 8 桁で表示**
sciPrintByteInt(unsigned char a) / sciPrintByteIntBuf(unsigned char a)　**a(Byte)を 10 進数で表示**
sciPrintShortInt(unsigned short a) / sciPrintShortIntBuf(unsigned short a)　**a(Word)を 10 進数で表示**
sciPrintShortSignedInt(short a) / sciPrintShortSignedIntBuf(short a)　**a(Word)を符号付き 10 進数で表示**
sciPrintLongtInt(unsigned long a) / sciPrintLongIntBuf(unsigned long a)　**a(Dword)を 10 進数で表示**

表示関数は、Buf が付いているものと、Buf なしがあります。Buf なしは、表示が完了するまで関数から戻りません。Buf 付きは、バッファへのコピー後関数からリターンします。

SCI(UART)の動作は、非常に低速ですので、Buf なしの動作は非常に時間がかかります。通常は Buf 付きを使用してください。

文字表示は、1 文字の表示に 100us 程度掛かりますので、オーバフローしないレートで文字を表示させてください。

標準では、バッファは 2047 文字分としています。短時間に大量に情報を出力する際は、バッファを増やすか、Buf なし関数を使用してください（Buf なし関数の出力処理中は、データ通信等の処理はバックグラウンドで実行されますが、main 関数内の処理は止まります）。

浮動小数点の数値を表示させる場合は、標準関数である sprintf を使い、文字列に変換した後で、sciPrintBuf を使用してください。

※デバッグ向けの表示を行うと、少なからず処理時間を消費します。表示させる情報が多くなると、デバッグ表示のために、メインの処理に支障を来たす事も考えられますので、表示させる情報量や頻度には留意ください。

取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2019.11.30	—	初版発行

お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

URL:http://www.hokutodenshi.co.jp

サウンドエフェクタ　ソフトウェア編(4)
取扱説明書

株式会社 北斗電子

 2019 年 11 月 30 日改訂 REV.1.0.0.0 (191130)