I2S オーディオインタフェース サンプルプログラム(RA)

-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。

製品の故障や、データの消失の原因となります。

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「I2S オーディオインタフェース」向けのサンプルプログラムに関しての説明資料です。

ターゲットマイコンボード

本サンプルプログラムは、以下のマイコンボードを対象としています。

マイコンボード型名	対応ブリッジボード
HSBRA6M4F144	ADA-IF_BRIDGE_RA1
HSBRA6M3F176	ADA-IF_BRIDGE_RA2
HSBRA6M2F144	ADA-IF_BRIDGE_RA1
HSBRA6M1F100	ADA-IF_BRIDGE_RA1
HSBRA4M3F144	ADA-IF_BRIDGE_RA1
HSBRA4M2F100	ADA-IF_BRIDGE_RA1

開発環境

ルネサスエレクトロニクス製 e2studio + FSP（3.0.0 以降）

※開発環境は、ルネサスエレクトロニクス Web よりダウンロードしてください。

※RA 向けの FSP 付きの e2studio をダウンロード、インストールしてください。

1. RAxxx_ADA-IF_TEMPLATE1

1.1. 概要

RA 向けの基本的なテンプレートです。

・入力信号をそのまま出力する（スルー）
・入力信号を一定時間遅らせて出力する（ディレイ）

の機能を持たせています。

新規にプロジェクトを作成する場合には、このテンプレートをベースにしてください。

マイコン種毎にプロジェクトを用意しています。RA6M2 向けは RA6M2_ADA_IF_TEMPLATE1（xxx にマイコン種が入ります）です。ファイルは、e2studio で作成してエクスポートしたプロジェクト形式ですので、e2studio のワークスペースにインポートして使用してください。

1.2. プログラムの動作

プログラムを実行すると、入力された信号がそのまま出力される（スルー）の動作となります。マイコンボード上の TEST スイッチを押すと、ディレイ動作に切り替わります（TEST スイッチを押す度に、「スルー」と「ディレイ」が切り替わります）。

SCI9（「マイコンボード 20P コネクタに、USB-ADAPTER を接続」、もしくは「14P コネクタに USB-ADAPTER-RX14 を接続」、もしくは「5P コネクタに市販の USB-Serial 変換ボードを接続」）と PC を接続して、ターミナルソフト（Teraterm 等）を接続すると、

・情報の出力
・キーボードからの指示（動作モードの変更、ディレイ量の変更）

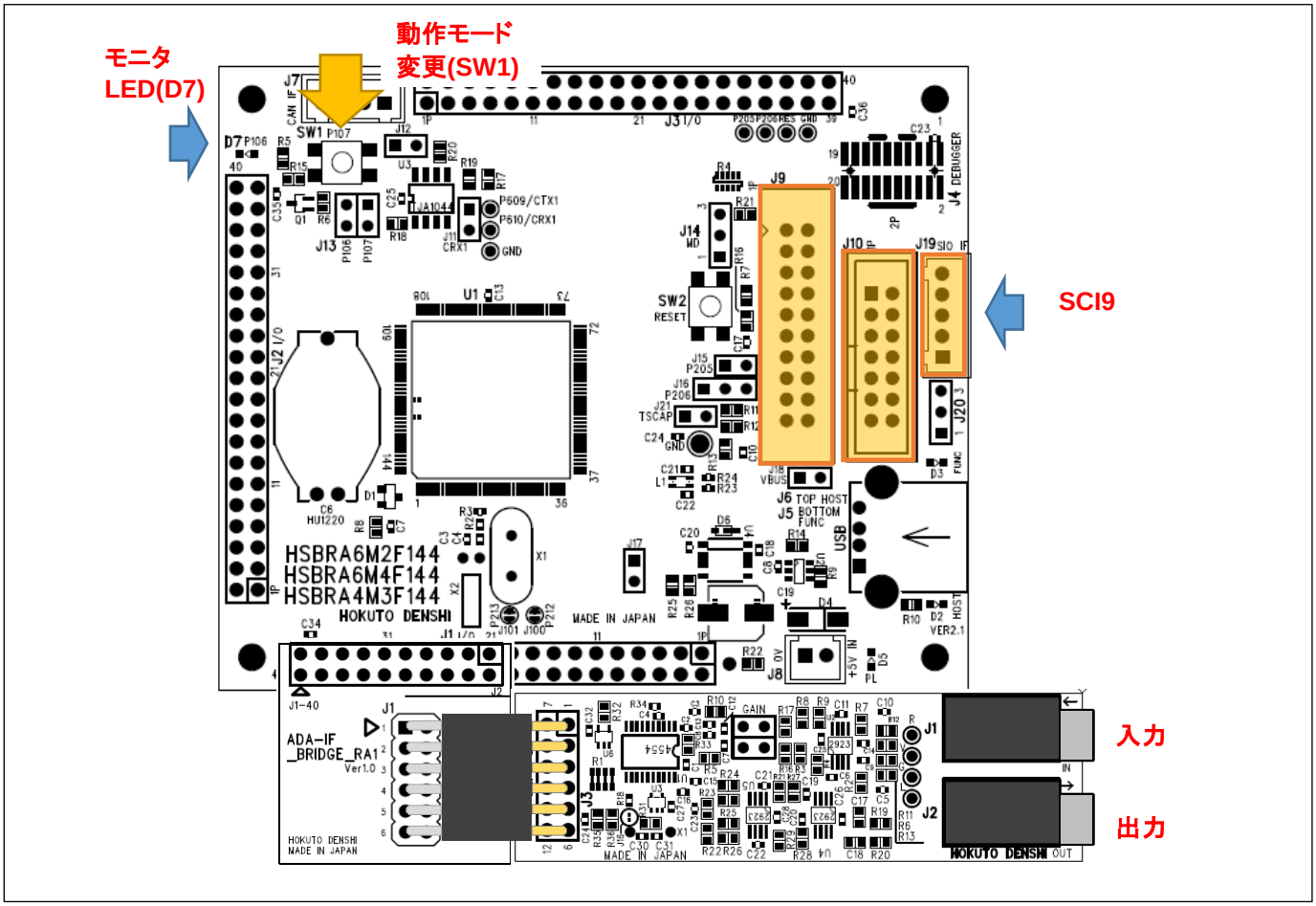
0:スルーモードに切り替え
1:ディレイモードに切り替え
q:ディレイ量 100 減少
a: ディレイ量 10 減少
z: ディレイ量 1 減少
w ディレイ量 100 増加
s: ディレイ量 10 増加
x: ディレイ量 1 増加
p:情報表示
が行えます。

マイコンボード上の LED は、

・スルー時：消灯
・ディレイ時：点滅

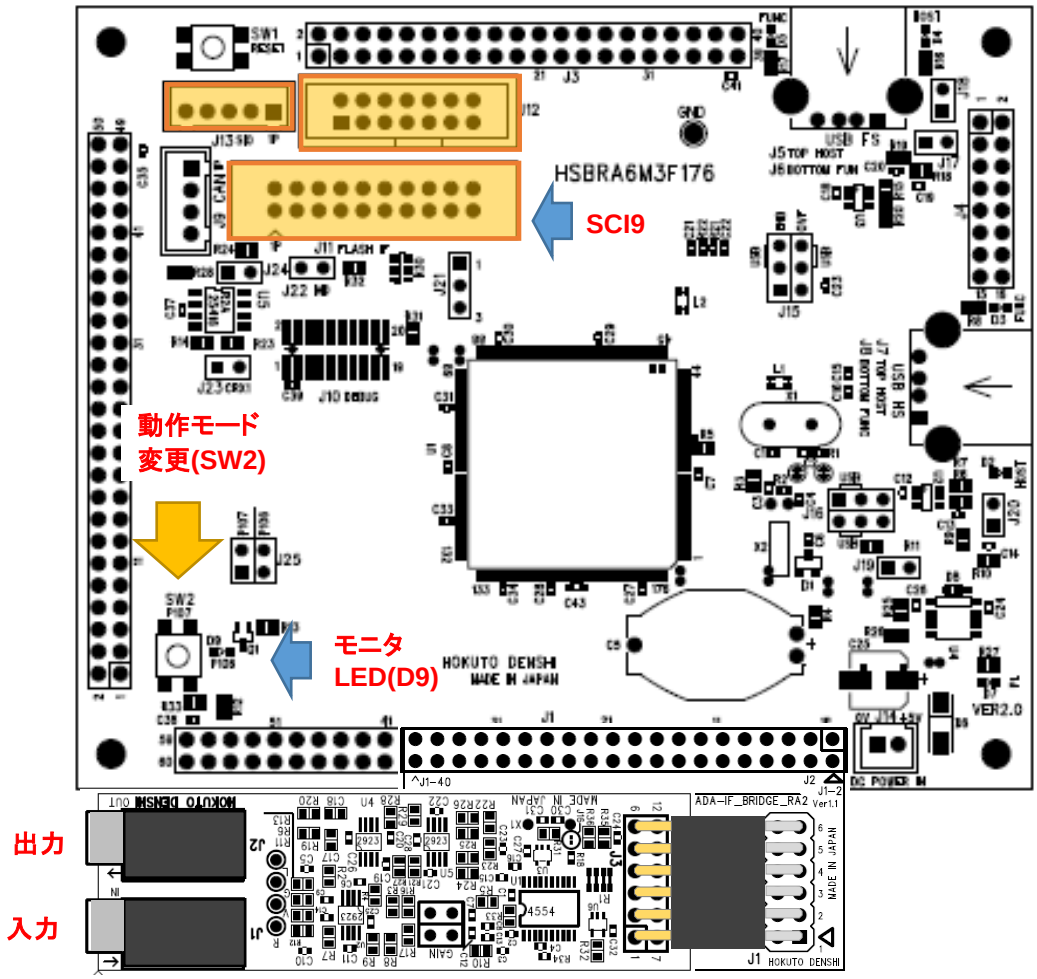
となります。

〇HSBRA6M4F144, HSBRA6M2F144, HSBRA6M1F100, HSBRA4M3F144, HSBRA4M2F100 ボードでのスイッチ LED 位置等



※上図は 144pin のボードとなっていますが、100pin のボードでも概ね同様です

○HSBRA6M3F176 ボードでのスイッチ LED 位置等



1.3. テンプレートをベースにユーザ処理を追加する場合

RX_ADA-IF_TEMPLATE1¥RX_ADA-IF_TEMPLATE1.c

```
while(1)
{

    //データ処理
    if(ada_if_input_index_diff(offset, g_input_index) > data_handling_size)//未処理のデータが
data_handling_size 以上であればデータを処理
    {
        switch(local_sw_state)
        {
        case 0:
            //スルー
            for(i=0; i<data_handling_size; i++)
            {
                g_pcm1[offset] = g_pcm0[offset];//入力データをそのまま出力にコピー★

            offset++;//インデックスを進める
            if(offset >= DATA_BUFFER_SIZE) offset = 0;
            }
            break;
```

★の行が、入力バッファ→演算→出力バッファの部分です。★の行を変える事により、様々な処理を行わせる事ができます。

一例ですが、★の行を変更する例を示します。

・出力の Lch/Rch に入力の Lch と Rch を合成して出力（ステレオ→モノラル化）

```
g_pcm1[offset].ch0 = (g_pcm0[offset].ch0 + g_pcm0[offset].ch1) / 2.0;
g_pcm1[offset].ch1 = g_pcm1[offset].ch0;
```

・ボーカルキャンセル（入力の Lch と Rch 両方に含まれる信号を消して出力）

```
g_pcm1[offset].ch0 = g_pcm0[offset].ch0 - g_pcm0[offset].ch1;
g_pcm1[offset].ch1 = g_pcm0[offset].ch1 - g_pcm0[offset].ch0;
```

1.4. プログラムの説明

ada_if_init();
ada_if_start();

で、I2S オーディオインタフェースボードとマイコンボードが通信を開始し、

・I2S オーディオインタフェースボード→マイコンボードに対し入力音声の A/D 変換結果を送信
・マイコンボードからデジタルデータを送信→I2S オーディオインタフェースボードが D/A 変換を行い音声出力

を行います。

入力音声は、-1.0~+1.0 の範囲に正規化され、g_pcm0 構造体（配列）に格納されます。

g_pcm0[g_input_index].ch0　Lch データ, float 型(-1.0~+1.0)
g_pcm0[g_input_index].ch1　Rch データ, float 型(-1.0~+1.0)

サンプリング周波数は、48kHz ですので 1/48,000=20.83us 毎にデータが格納され、g_input_index が進みます。

g_pcm0 は、n 個（マイコン内蔵メモリサイズによって異なる）の配列となっており、約 s 秒のデータを保持します。配列のインデックスが n まで進むと、次は 0 に戻り、古いデータは、新しいデータで上書きされます。
（配列のサイズ n と保持できる時間 s は後述）

出力は、g_pcm1 構造体（配列）に格納したデータが音声出力されます。g_pcm1 構造体は、g_pcm0 と同様の構成

g_pcm0[g_output_index].ch0　Lch データ, float 型(-1.0~+1.0)
g_pcm0[g_output_index].ch1　Rch データ, float 型(-1.0~+1.0)

を取っています。n 個の配列で、約 s 秒のバッファとなっています。

（-1.0~1.0 を超えるデータを格納した際は、出力時に-1.0~1.0 を越える部分はクリッピングして出力します）

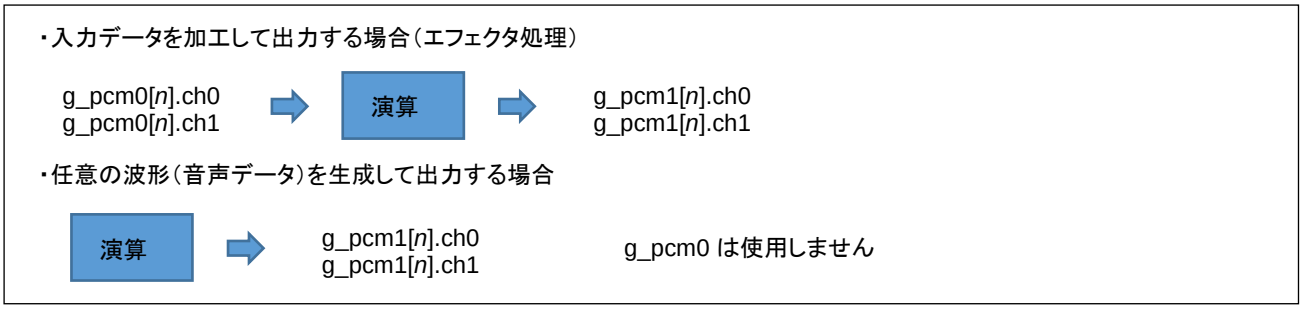


図 1-1 基本的な使い方

1.4.1. 入力バッファの構成

データは、1/48kHz のタイミング（20.83us 毎）で、0→1→2…のバッファに格納されていきます。用意されているテンプレートのプログラムでは、入力バッファは、n 個確保されていますので、約 s 秒で、n-1 個目のバッファまでデータが埋まります。その後は、0 からデータを上書きしていきますので、約 s 秒以上前のデータは残りません。逆に言えば、約 s 秒前までのデータであれば、任意のデータにアクセス可能です。

現在入力バッファのデータがどこまで格納されたかを示す変数(g_input_index)を読み出せば、現在どのデータが最新かを判断できます。例えば、g_inbuf_index=256 の時、g_pcm0[256]が最新のデータで、g_pcm0[257]が約 s 秒前のデータです。入力バッファは、ある時点では、約 s 秒前から現在（インデックス=g_input_index）までの時系列のデータが直線的に配置されている構成です。

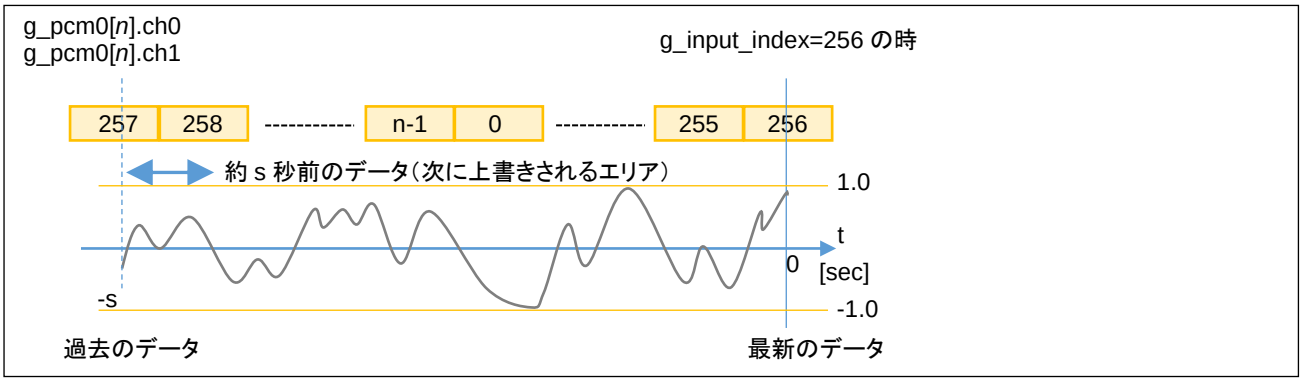


図 1-2 入力バッファ構成

g_input_index は、バックグラウンドで平均で 20.83us(1/48kHz)毎(*1)にインクリメントされる変数です。

(*1)設定する FIFO 段数（デフォルト 4）により、複数データがまとめて処理されます（FIFO 4 段設定の場合は 83.32us 毎に 4 が加算されます）

プログラム実行前(ada_if_start()関数実行前）に 0 を代入しておけば、ada_if_start()後、約 s 秒後に n に達し、次の更新のタイミングで 0 に戻り、インクリメントを続けます。

プログラムの実行中に値を書き換えても問題はありません。

※g_input_index=256 に変化した際は、g_pcm0[257]が一番古いデータとなりますが、20.83us(=1/48kHz)×FIFO 段数後には、g_pcm0[257]に最新のデータが格納されますので、約 s 秒前のデータのデータを使用する際には、データが上書きされる前に読み出せているか注意してください。(g_input_index よりも微妙に大きなインデックスの領域は参照しないのが推奨です)

入力データが格納される、g_pcm0 の配列は構造体で、

```
typedef struct{
   float ch0;
   float ch1;
} PCM_float;

PCM_float g_pcm0[INPUT_BUFFER_SIZE]
```

で定義されています。INPUT_BUFFER_SIZE はマイコン内蔵メモリサイズによって異なります。この値は変更可能です。(詳細は後述「設定値のカスタマイズ」の項を参照してください。)

入力信号は、24bit データ(-8,388,608~8,388,607)を float 型に変換して、-1.0~1.0 の範囲に正規化したものです。

g_pcm0[*n*].ch0, g_pcm0[*n*].ch1 には、-1.0~1.0 の範囲の値が格納されます。.ch0 は Lch 側。.ch1 は Rch 側のデータが入ります。(Lch 側のみ使用して、入力バッファを 2 倍の時間確保する事もできます。詳細は後述「設定値のカスタマイズ」の項を参照してください。)

1.4.2. 出力バッファの構成

出力バッファも、入力バッファと同様の構成(float 型変数をメンバに持つ、構造体配列)です。

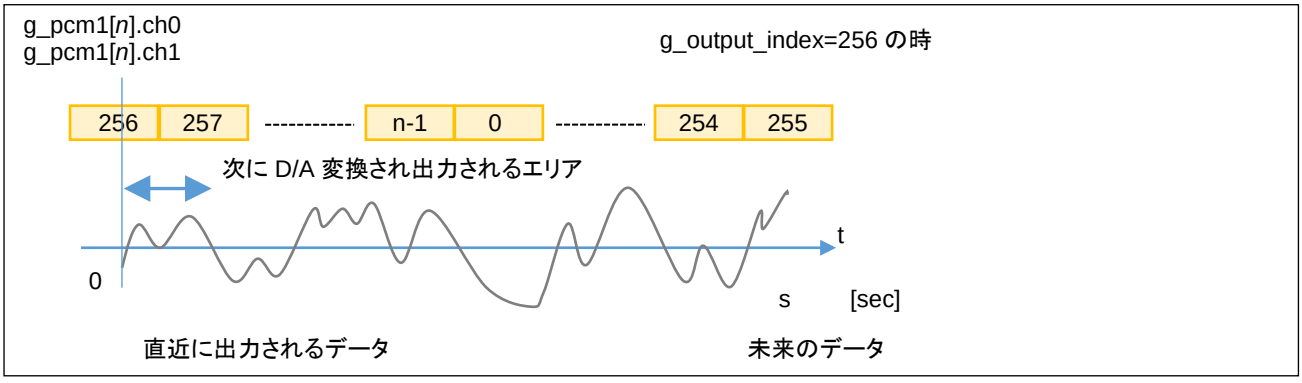


図 1-3 出力バッファ構成

g_output_index は、g_input_index 同様バックグラウンドで 20.83us(1/48kHz)×FIFO サイズ(デフォルト 4)毎にインクリメントされる変数です。

1/48kHz のタイミング（20.83us 毎）で、g_pcm1[g_output_index]のデータを音声出力します。用意されているテンプレートのプログラムでは、出力バッファは、n 個確保されていますので、約 s 秒先のデータまで格納しておく事が可能です。

データ格納後（音声出力前に）データを書き換えたり、別なデータを加算したりしても問題ありません。

（エコーやリバーブの処理等では、いくつかのデータを時間をずらして加算する演算を行いますが、その様な使い方も可能です。）

入力バッファが、過去のデータを保持しているのに対し、出力バッファは未来のデータを格納する使い方です。

現在の出力対象を示す変数(g_output_index)を読み出せば、現在どのデータを出力しているのかをを判断できます。出力バッファは、ある時点では、現在（インデックス=g_input_index）から約 s 秒先までの時系列のデータが直線的に配置されている構成です。

・g_output_index の設定方法

(1)入力バッファとインデックスを揃える

プログラム実行前に

g_input_index = 0;
g_output_index = 0;

を設定する。この様に設定した場合、g_input_index と g_output_index は（多少のラグはありますが、基本的には）同じ値を刻む事となります。同じインデックス値のとき、（ほぼ）同じタイミングという事で判りやすいかと思います。

(2)入力バッファと出力バッファのインデックスにオフセットを付ける

プログラム実行前に

g_input_index = 0;　　（もしくは、g_input_index = 256;
g_output_index = *n* - 256;　　g_output_index = 0;）

を設定する。（n:マイコン内蔵メモリによって異なる）

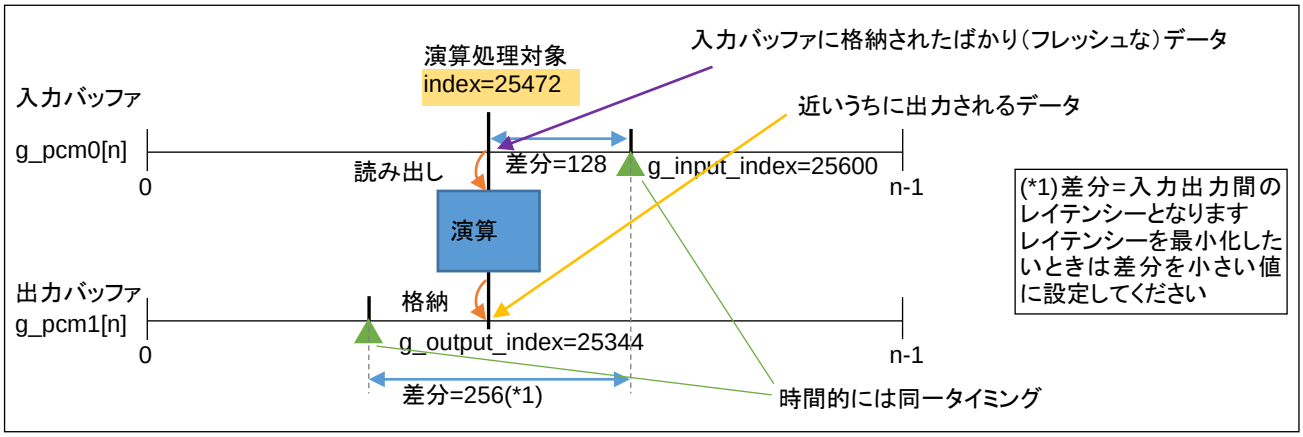


図 1-4 入力バッファと出力バッファのインデックスに差を付けた場合

入力バッファと出力バッファのインデックスに差を付けた場合(例えば 256)、入力バッファと出力バッファのインデックスの差分の範囲内で入力バッファの読み出し、演算、出力バッファの格納を行うと同じインデックス値で処理できるというところがポイントです。

g_pcm1[n].ch0 = g_pcm0[n].ch0 * 0.5;　//入力の半分の音量で出力する

n は、g_output_index < n < g_input_index の範囲内で設定

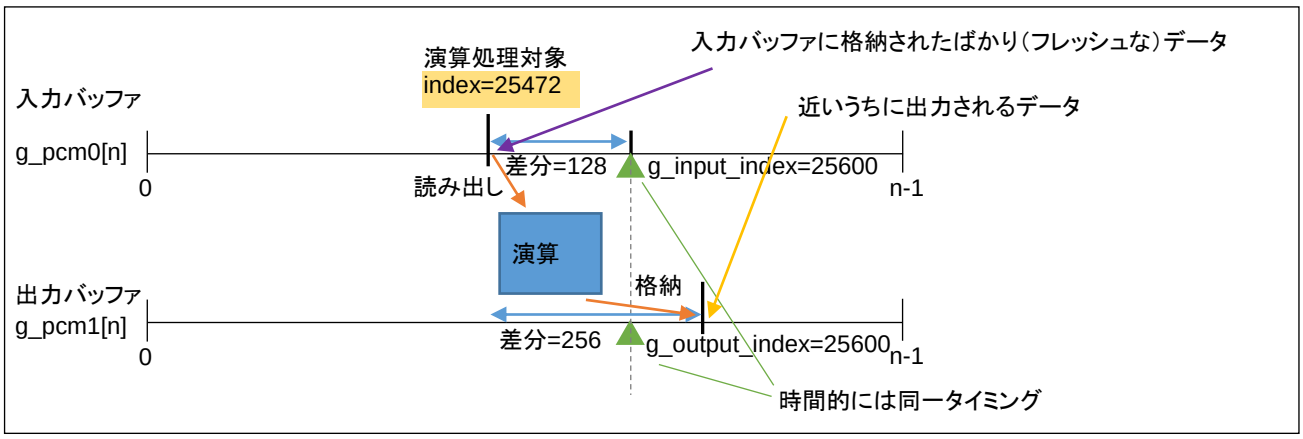


図 1-5 入力バッファと出力バッファのインデックスに差を付けない場合

上記と同じ処理(入力データの 1/2 を出力に格納)を行う場合、

g_pcm1[n+256].ch0 = g_pcm0[n].ch0 * 0.5;

となります。g_input_index, g_output_index の側に差分を持たせるか、g_pcm1[n]の n のところに差分を持たせるかの違いで、本質的な違いはではありません。

※但し、こちらの方式の場合、n-1 と 0 をまたぐ部分の[n+256]与え方にはケアが必要です

0 またぎを考えると、

```
m= n + 256;
if(m >= INPUT_BUFFER_SIZE) m -= INPUT_BUFFER_SIZE;
g_pcm1[m].ch0 = g_pcm0[n].ch0 * 0.5;
```

の様なコードとなります。

g_inbuf_index と g_outbuf_index の初期値をどう設定するかは、ユーザ次第です。判りやすい方を選択して頂ければ良いと考えます。

1.4.3. リングバッファの構成

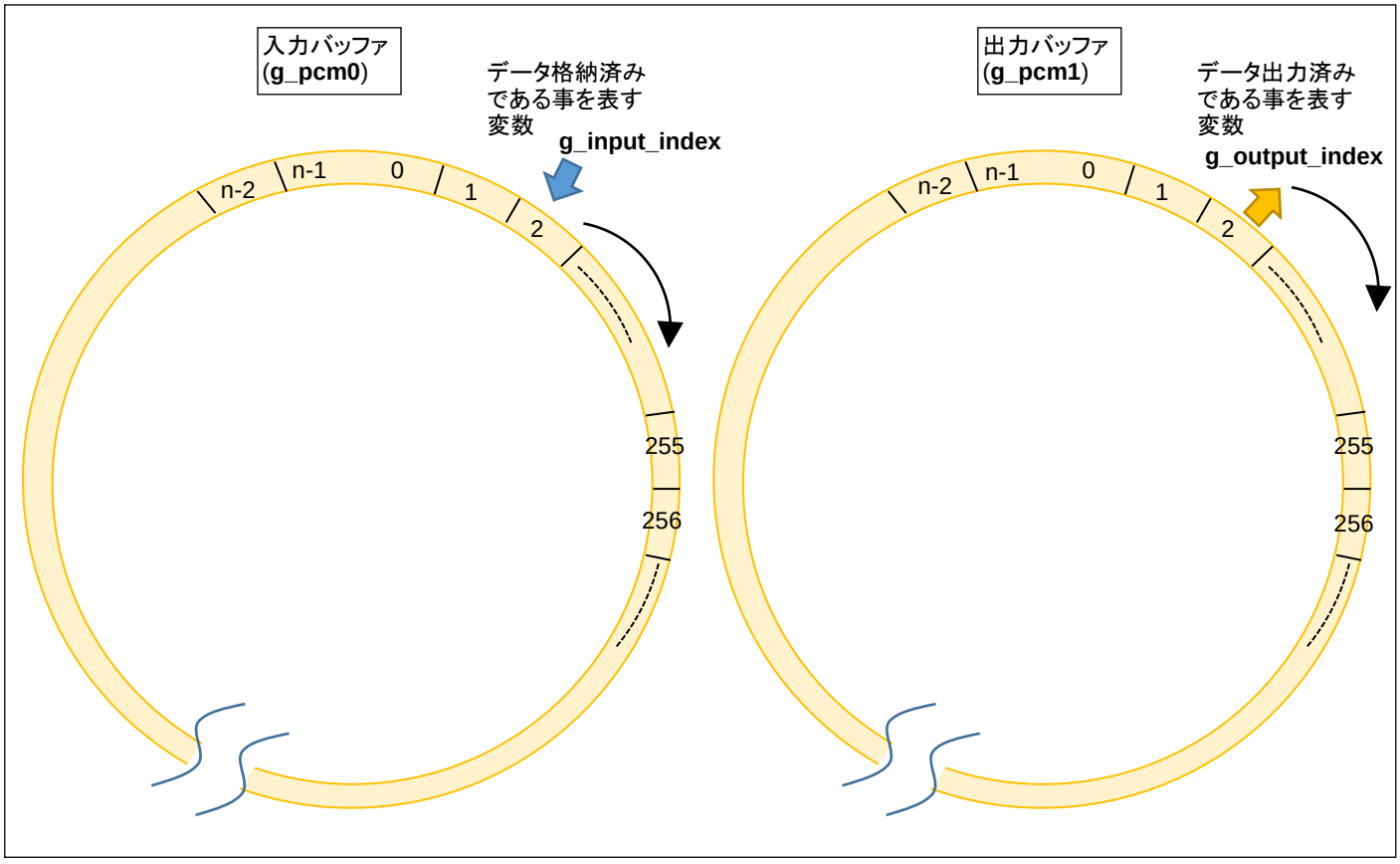


図 1-6 リングバッファ構成

入力バッファと出力バッファは、常に終わりが来ない（インデックスの最大値まで来ると、0 に戻る）リングバッファの構成です。

※出力バッファは 2 面（出力処理を行うバッファとデータ格納を行うバッファを別にするために、出力バッファを 2 つ持たせて、交互に切り替える）にするという手もありますが、このテンプレートでは 1 つです。
（g_pcm1 からは常に読み出しデータ出力が行われていますが、同時に g_pcm1 を書き換えていく手法です）

図 1-2, 1-3 では直線状のイメージでしたが、実動作としてはリングバッファの構成だと考えた方が判りやすいと思います。

テンプレートのプログラムは、g_pcm0 の入力バッファと、g_pcm1 の出力バッファを使いこなす事が鍵になってくると思います。

1.4.4. PCM データのビット深度

I2S オーディオインタフェースに搭載されている、A/D, D/A は 24bit のデータの入出力を行います。(量子化ビット数=24bit です)

最大値 = 2^{23}-1 (=8,388,607)
最小値 = -2^{23} (=8,388,608)

です。この値を、演算上扱い易い様に、+1.0~-1.0 に正規化(オリジナルデータを 2^{24} で割った値)して、g_pcm0 に格納しています。

g_pcm0[n].ch0, g_pcm0[n].ch1 の値は、float 型としています。RA の処理系(GCC コンパイラを使った場合)、float は 4 バイトの浮動小数点数です。

GCC 処理系における、float のε(機械イプシロン)の値は、

1.1920928955078125e-007

です。(表現可能な最小の分解能に相当する数値)

一方、1/2^{24} = 5.9e-8 です。(機械イプシロンの値より小さい)

float で扱う限り、24bit の分解能は生かせませんが、アナログ系のノイズにより D/A, A/D とも、24bit の精度を維持する事は難しい事や、ハンドリングするデータサイズや、演算速度を考えて、float で十分であると考えています。

24bit 精度に拘る場合は、

・double 型の変数で PCM データを取り扱う(テンプレートの float の部分を double に変更する)

様に変更してください。(現状 4 バイトの float で入出力バッファを確保しているので、8 バイトの double とした場合(時間的な)バッファ容量は半分になります。)

1.4.1. ボリュームの設定

本テンプレートでは、当社製品「サウンドエフェクタ」のプログラムを実行できる様に作っています。

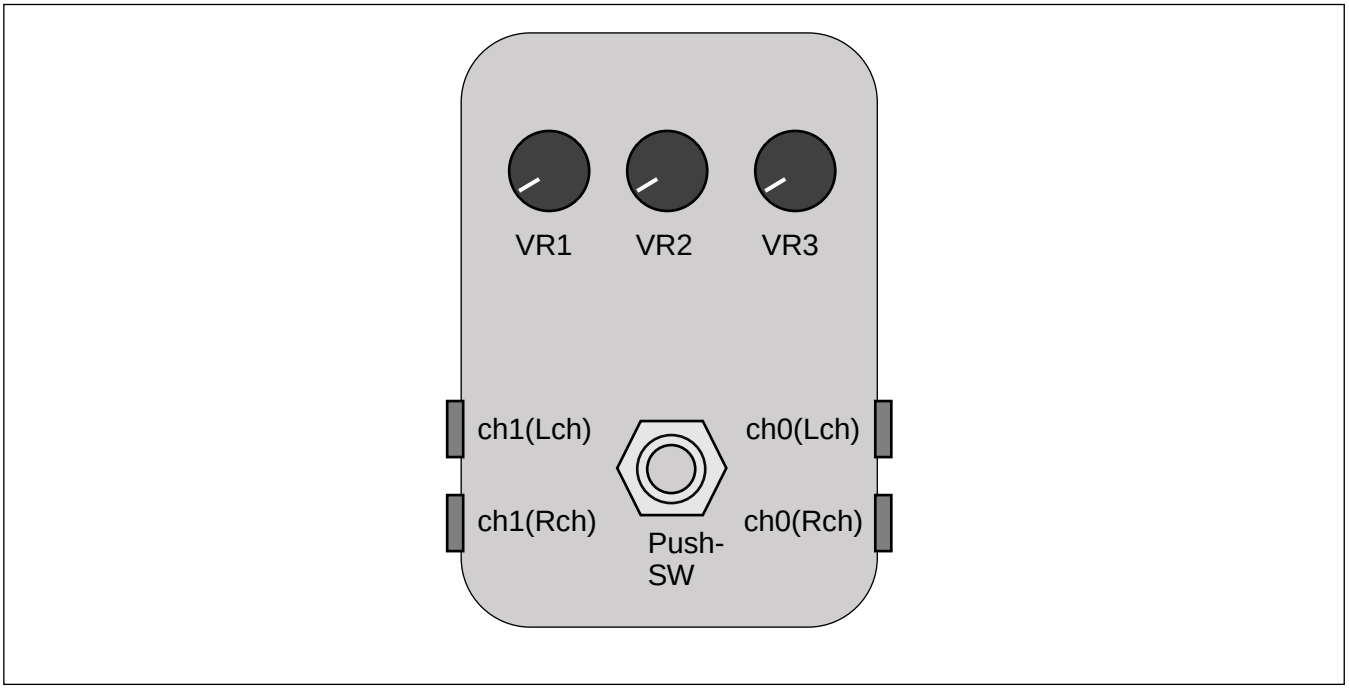


図 1-6 サウンドエフェクタ外観イメージ

サウンドエフェクタは、エレキギター等で使用可能な、プログラマブルなエフェクターで、VR1~VR3 の 3 つのボリュームでエフェクトの掛け方を調整できる様になっています。本テンプレートでも、ボリュームを使うことができる様になっており、

(1)マイコンのアナログポート(AN000~AN002)に実際のボリューム（可変抵抗）をつないで制御する
(2)PC のキーボードで、ボリュームのツマミを回す動作を模擬する

のどちらか（もしくはボリュームを使わない）を選択できる様になっています。(1)の場合は、マイコンボードの拡張 I/O 端子にボリュームを接続する必要があります。(2)の場合は、マイコンボードのシリアル(SCI9)を PC に接続し、PC 上で仮想端末（Teraterm 等）を使って、マイコンボードと接続する必要があります。

※ボリュームをつないだときは、マイコンの 12bit A/D コンバータでボリュームの値を読み取ります（0~4095 の値となります）。PC のキーボードでボリュームを模擬した場合も、値は 0~4095 の範囲で変化するようにしています。

－PC のキーボードを使用したボリュームの調整－

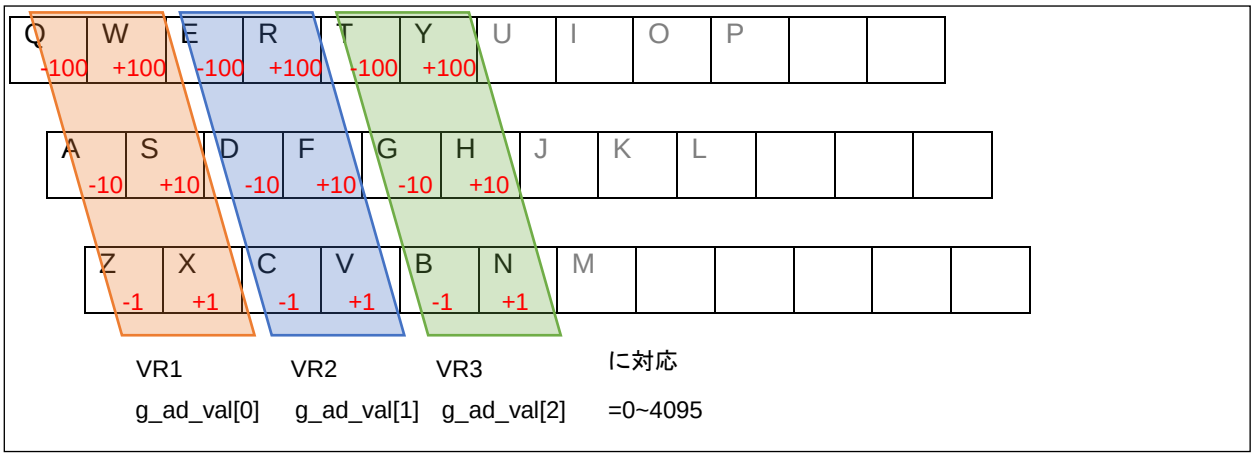


図 1-7 キーボードでのボリューム調整

ボリュームは、g_ad_val[0]~g_ad_val[2]に値が保持されており、値としては 0~4095（12bit）を取ります。起動後は中点(2048)となり、キーボードの q（小文字 Q）を押すと g_ad_val[0]値が 100 減ります（2048→1948）。

本テンプレートのプログラムでは、g_ad_val[0]（VR1 に対応するボリューム値）のみ使用しており、数値により遅延時間が変化します。（遅延時間は、入力バッファと出力バッファのバッファ時間の合計となります。g_ad_val[0]=0 の時は、ほぼ遅延なし。g_ad_val[0]=2048（初期値）の時は最大値の半分。g_ad_val[0]=4095（最大値）の時は最大の遅延となります。）

キーボードによるボリューム調整ではなく、実際のボリュームを使用したい場合は、マイコンの A/D 変換入力ポートにボリュームをつないでください。（なお、「設定値のカスタマイズ」の項（後述）の定義値も変更が必要です。）

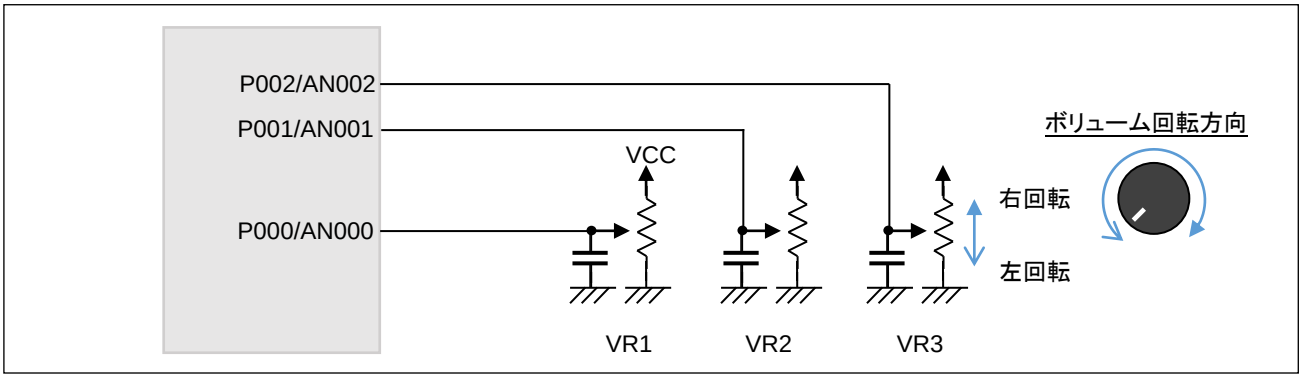


図 1-8 マイコンースイッチ，ボリューム接続

・接続端子

	HSBRA6M5F176 HSBRA6M3F176	HSBRA6M4F144, HSBRA6M2F144, HSBRA4M3F144 HSBRA6M1F100, HSBRA4M2F100
VCC(3.3V)	J1-39	J1-37
P40/AN000	J1-25	J1-33
P41/AN001	J1-26	J1-34
P42/AN002	J1-27	J1-35

※ブリッジボード端子からの引き出しとなります

1.4.2. 動作モードを切り替えるスイッチ

マイコンボード上の SW(TEST, P107)は、モード切替のスイッチに使用可能です。このスイッチを押す度に

g_sw_state

変数がインクリメントされますので、この変数を見て動作を変更することが出来ます。本テンプレートでは、

g_sw_state = 0 : スルー
g_sw_state = 1 : ディレイ

動作となります。(動作の切り替えは、PC のキーボードの 0, 1 のキーを押す事でも変えられます。)

1.4.3. LED

マイコンボード上の LED(P106)を点灯、点滅、消灯させることが出来ます。unsigned long(4 バイト、32bit)の変数

g_led_pattern

に、点灯のパターンを代入してください。b0 から b31 まで 50ms 毎に読み込んで LED の ON/OFF を制御します。

g_led_pattern = 0; //消灯
g_led_pattern = 0xffffffff; //点灯
g_led_pattern = 0x1; //間欠点滅(50ms ON, 50x31=1.55s OFF の繰り返し)
g_led_pattern = 0x5; //2 回点滅(50ms ON, 50ms OFF, 50ms ON, 50x31=1.45s OFF の繰り返し)

b31	b30	…	b3	b2	b1	b0
0(消灯)	0(消灯)	0(消灯)	0(消灯)	1(点灯)	0(消灯)	1(点灯)
50ms 消灯	50ms 消灯	…	50ms 消灯	50ms 点灯	50ms 消灯	50ms 点灯

g_led_pattern = 0xffff0000; //点滅(0.8 秒 OFF, 0.8 秒 ON の繰り返し)

50ms×32(1.6 秒)の周期で、50ms 毎に ON/OFF を制御できます。

1.5. ファイル構成

本テンプレートは、CS+のプロジェクトです。プロジェクトフォルダの、src 以下がソース格納フォルダです。

RA*xxx*_ADA-IF_TEMPLATE1¥				プロジェクトフォルダ
	src¥			ソース格納フォルダ
				ユーザソース
		ada_if¥	**ada_if_userdef.h** ada_if.c ada_if.h	ユーザ定義値 I2S 制御プログラム I2S ヘッダ
		board¥	board.h	ボード定義ヘッダ
		callback¥	callback.c callback.h	割り込みコールバック関数
		sci¥	sci.c sci.h	SCI(UART)プログラム
		hal_entry.c		メイン関数相当の処理

xxx:マイコン名が入ります、RA6M2 向けですと xxx=6M2 です。

基本的に書き換える必要があるのは、太字のファイルです。
ada_if_userdef.h は各種動作を定義します

1.6. 設定値のカスタマイズ

1.6.1. 定義ファイルの変更

RAxxx_ADA-IF_TEMPLATE1¥src¥ada_if¥ada_if_userdef.h に、設定可能項目が列挙されていますので、必要に応じて書き換えてください。

・使用 ch 数

```
//出力ch
#define USE_CH (2)//"1"（ch0(Lch)のみ扱う，ch1(Rch)は0出力）または"2"（ch0(Lch),
ch1(Rch)両方を扱う）を定義
```

デフォルトでは、2ch 入力、2ch 出力です。楽器のエフェクト等、1ch のみ使用する場合は(1)を定義してください。(1)を定義した場合は、Lch のみ有効で、Rch 入力は使用せず、Rch の出力は無音となります。また、(1)を定義した場合は、入力、出力バッファのサイズが倍になります。

・データバッファサイズ

```
//データバッファサイズ
#define DATA_BUFFER_SIZE_SAME (1)//"0"入力バッファと出力バッファのサイズを変える、
"1"入力バッファサイズと出力バッファサイズを同じとする
```

データのバッファサイズは、ada_if.h 内で定義されています。

デフォルトでは、入力バッファサイズ＝出力バッファサイズとしています（どちらかに多くのサイズを割り振る事も可能です）。マイコン種により搭載メモリが異なりますので、マイコン種毎にデフォルトのバッファサイズが異なります。

g_pcm0（入力バッファ）と g_pcm1（出力バッファ）のサイズは、デフォルトでは以下です。

	RA6M4	RA6M3	RA6M2	RA6M1	RA4M3	RA4M2
マイコン搭載 RAM サイズ[kB]	256	640	384	256	128	128
入出力バッファ割り当てサイズ[kB]	192	512	256	128	64	64
USE_CH = (2)	12288 （0.26 秒）	32768 （0.68 秒）	16384 （0.34 秒）	8192 （0.17 秒）	4096 （0.09 秒）	4096 （0.09 秒）
USE_CH = (1)	24576 （0.51 秒）	65536 （1.37 秒）	32768 （0.68 秒）	16384 （0.34 秒）	8192 （0.17 秒）	8192 （0.17 秒）

・FIFO 段数

```
//FIFOバッファサイズ
#defineFIFO_BUFFER_SIZE(4)
//値を小さくすると入出力のレイテンシーは最小となります
//値が大きい方が処理のオーバヘッドは小さくなります
```

I2S データのアクセス頻度を定義します。(1)とした場合、1/48kHz(20.83us)毎に入力バッファ g_pcm0 が更新され、出力バッファ g_pcm1 を読み出して出力します。(2)以上の数値を指定すると、入出力バッファのアクセス頻度が減ります（その分まとめてデータを処理します）。入力バッファの更新、出力バッファのアクセスは割り込みで処理されています。複数データをまとめて処理した方が CPU 負荷は減ります（ユーザプログラムを実行する時間的リソースが増えます）。値を小さくした方が、入力バッファの更新のリアルタイム性が高くなります。基本的には変更しなくても良いと思います自由に変更頂いて構いません。16×FIFO_BUFFER_SIZE（デフォルトでは 64Bytes）のメモリを消費します。

・音声出力後の出力バッファの値

```
//出力バッファデータ送信後の挙動
#define OUTPUT_BUFFER_AFTER_SEND (0)//"0"データを消す、"1"データを消さない
```

音声出力が終わった後の、出力バッファ(g_pcm1)のデータの取り扱いです。出力後に参照したい（出力バッファの値を見てディレイやリバーブの処理を行う）場合は、(1)を定義してください。

入力信号を加工して出力する場合は、どちらでも良いと思います。

入力信号を使わずにマイコン内部で生成したデータを出力する場合、(プログラムの作り方によっては)一度出力バッファにコピーしたデータが何度も再生される(バッファ容量のサイズ毎に同じ音声が出力される)事がありますので、その様な状態を抑止したい場合は(0)を定義してください。

・スイッチの状態

```
//スイッチの状態を何種類設けるか
#define SW_STATE_NUM (2)//ボード上のスイッチを押す度に g_sw_state が
0-(SW_STATE_NUM-1)に遷移 (0はスイッチを使用しない)
```

マイコンボード上のスイッチを押す度に、g_sw_state が 0→1→2→…SW_STATE_NUM-1→0 の様に変化します。状態(動作モード)を 2 種類設けたいときは(2)を定義してください。(その場合、g_sw_state は 0,1 のトグル動作となります。)スイッチを使わないときは、(0)としてください。

・LED の使用

```
//ボード上のLEDを使用
#defineUSE_LED (1)//"0"使用しない, "1"LEDを使用する
```

g_led_pattern 変数で LED を制御したい場合(1)。不要な場合(0)としてください。

・A/D 変換(ボリュームの読み取り)

```
//ADCを使用する
#define USE_ADC_PORT (0)//"0"使用しない, "1"AN000-AN002をvolume0-volume2の読み取
り値に使用
```

マイコンボードにボリュームを接続して、ボリュームで g_ad_val[0]~[2]の値を変化させるときは、(1)。外部接続のボリュームをつながないときは(0)としてください。

・SCI の使用

```
//SCIを使用する
#defineUSE_SCI (1)//"0"使用しない, "1"SCI を使用する
```

SCI(UART)を使ってボリュームの調整や、情報の出力(メッセージの表示)を行いたいときは(1)。SCI を使用しないときは、(0)としてください。

1.6.2. メモリの設定

マイコン種毎に搭載メモリが異なりますので、バッファサイズとスタックサイズもマイコン種に応じて変えています。

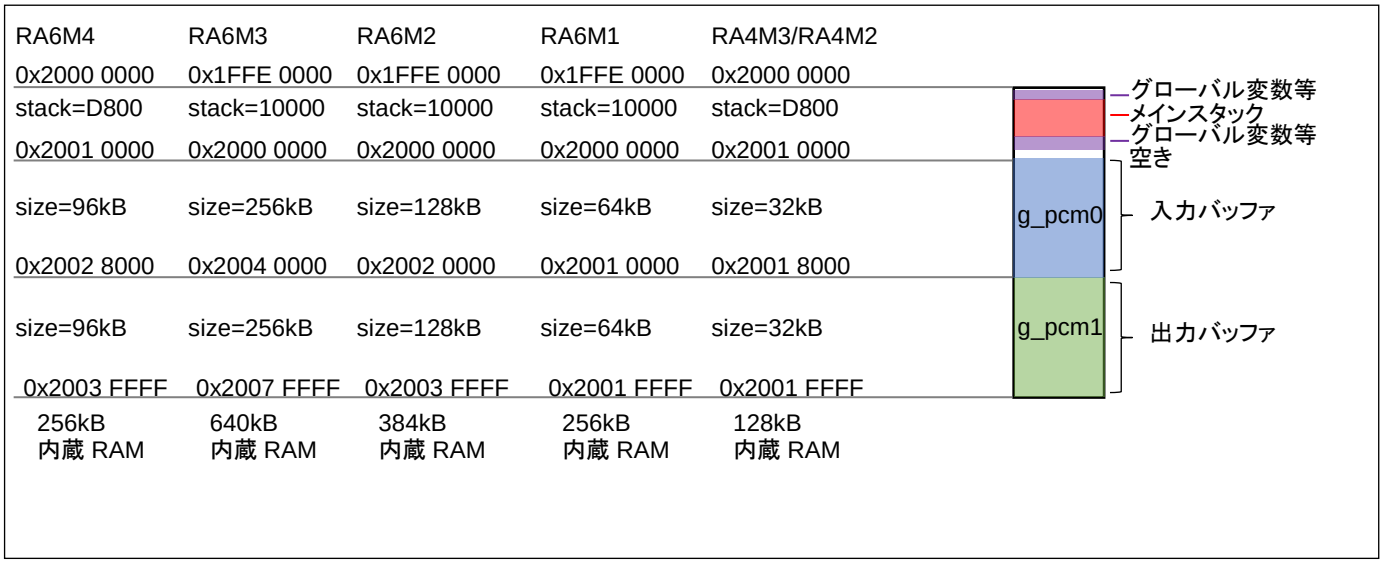


図 1-9 デフォルトのメモリマップ

自動変数が割り当てられるメインスタックは、RA6M4/RA4M3/RA4M2 では 54kB(0xD800)としています。RA6M3/RA6M2/RA6M1 では、64kB(0x10000)です。関数内での変数定義等は、メインスタックに割り振られます。グローバル変数や、static 指定で確保した変数等は、メインスタックとは別な領域に割り振られます。

I2S オーディオの FIFO バッファとして、64B（デフォルト）、SCI の TX バッファとして 2kB 等のグローバル変数を定義済みです。

ユーザが使える変数のサイズとしては、以下となります。

〇RA6M4/RA4M3/RA4M2(Coretex-M33 の RAM が 0x2000 0000~に割り振られているマイコン）

・入出力バッファ以外で 64kB
　自動変数（メインスタック） 52kB(0xD800)以内
　残り使用可能メモリ（static 変数等） 約 5.5kB 以内

0x2001 0000~は、入力バッファが割り当てられていますので、ユーザ変数が 0x2000 FFFF までに収まるようにしてください。（収まらない時は、入出力バッファを削ったり、メインスタックと使用可能メモリのサイズを調整してください。）

〇RA6M3/RA6M2/RA6M1(Coretex-M4 の RAM が 0x1FFE 0000~に割り振られているマイコン）

・入出力バッファ以外で 128kB
　自動変数（メインスタック） 64kB(0x10000)以内
　残り使用可能メモリ（static 変数等） 約 60kB 以内

　0x2000 0000~は、入力バッファが割り当てられていますので、ユーザ変数が 0x1FFF FFFF までに収まるようにしてください。（収まらない時は、入出力バッファを削ったり、メインスタックと使用可能メモリのサイズを調整してください。

・バッファサイズの変更（RA6M2の例）

RA6M2_ADA-IF_TEMPLATE1¥src¥ada_if¥ada_if.h

```
if (USE_CH == 1)

typedef struct{
  float ch0;
} PCM_float;

#if (DATA_BUFFER_SIZE_SAME == 1)

#define DATA_BUFFER_SIZE (32768)//128+128=256kB, 入力バッファと出力バッファのサイズを同じとする
#define INPUT_BUFFER_SIZEDATA_BUFFER_SIZE
#define OUTPUT_BUFFER_SIZEDATA_BUFFER_SIZE

#else

#define INPUT_BUFFER_SIZE(32768)//128kB
#define OUTPUT_BUFFER_SIZE(32768)//128kB

#endif

#elif (USE_CH == 2)

typedef struct{
  float ch0;
  float ch1;
} PCM_float;

#if (DATA_BUFFER_SIZE_SAME == 1)

#define DATA_BUFFER_SIZE (16384)//128+128=256kB, 入力バッファと出力バッファのサイズを同じとする
#define INPUT_BUFFER_SIZEDATA_BUFFER_SIZE
#define OUTPUT_BUFFER_SIZEDATA_BUFFER_SIZE

#else

#define INPUT_BUFFER_SIZE(16384)//128kB
#define OUTPUT_BUFFER_SIZE(16384)//128kB

#endif

#endif
```

1ch 使用の場合

2ch 使用の場合

バッファサイズは、ada_if.h 内で定義されています（マイコン種毎にサイズは異なります）。赤字の部分が配列のサイズとなりますので、メモリ容量の範囲内で調整してください。

1.7. 関数の説明

ada_if_init

概要：I2S オーディオインタフェースボードの初期化関数
宣言：
　　void ada_if_init(void)
説明：
・I2S オーディオインタフェースボードの初期化
を行います。
引数： なし
戻り値： なし
使用例：
　マイコンのクロック設定等の初期化後に本関数を呼び出してください。

ada_if_start

概要：I2S オーディオインタフェースボードの動作開始関数
宣言：
　　int ada_if_start(void)
説明：
・I2S オーディオインタフェースボードの動作開始（音声信号の送受信）
を行います。
引数： なし
戻り値：
　　0: 正常終了
　　1: I2S オーディオインタフェースボードと通信が出来ない
使用例：
　ada_if_init 後に本関数を呼び出してください。

ada_if_input_index, ada_if_output_index

概要:入力,出力バッファのインデックスの有効値算出関数
宣言:
　　int ada_if_input_index(int n)
　　int ada_if_output_index(int n)
説明:
・入力,出力バッファのインデックスの有効値の算出
を行います。
引数:
　　int n　インデックス値
戻り値:
　　インデックス値（リングバッファで考え有効範囲内の数値）
使用例:
　入力バッファのインデックス値の有効値が 0~65535 の時

```
offset = 65408;
x = ada_if_input_index(offset + 256);　　//x は、128（インデックスの有効範囲内）となります
a = g_pcm0[n];

x = ada_if_input_index(-256);　　//x は、65280（インデックスの有効範囲内）となります
```

　指定された引数(n)が、インデックスの有効範囲から外れている場合、0 と 65536 が重なっていると考え有効なインデックス値を算出します。

ada_if_input_index_diff, ada_if_output_index_diff

概要:入力,出力バッファのインデックスの差分算出関数
宣言:
　　int ada_if_input_index_diff(int n, int m)
　　int ada_if_output_index_diff(int n, int m)
説明:
・入力,出力バッファのインデックスの差分値の算出
を行います。
引数:
　　int n　インデックス値（基準）
　　int m　インデックス値
戻り値:
　　n に対する m のインデックス値の差分

使用例:

x = ada_if_index_diff(5000, 5250);　//x=250

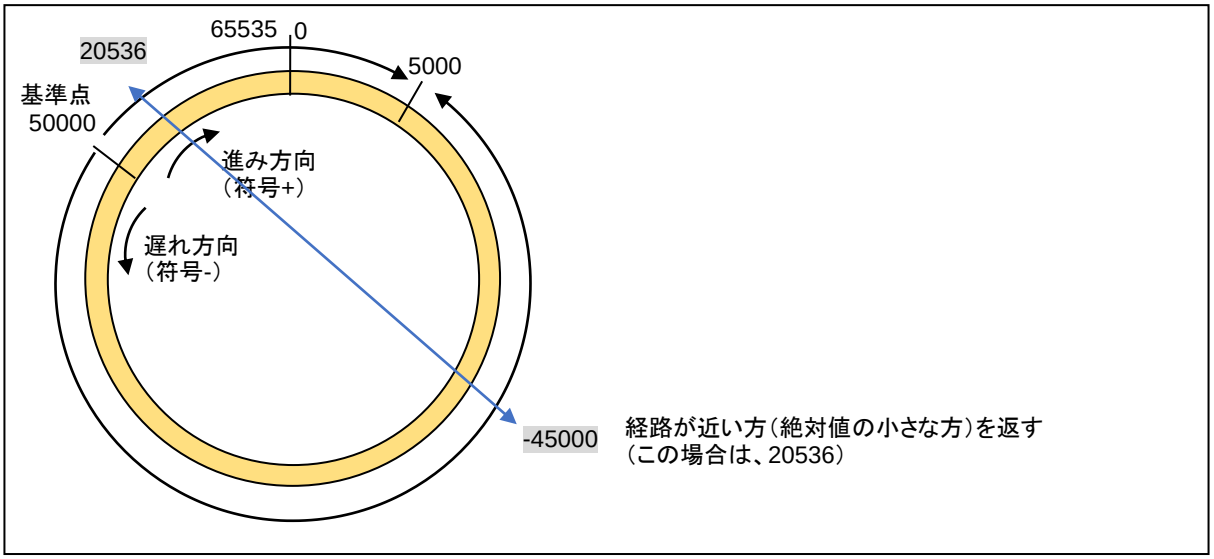
x = ada_if_index_diff(5250, 5000);　//x=-250

2 点間の差分を求めます。

入力バッファのインデックス値の有効値が 0~65536 の時

x = ada_if_index_diff(50000, 5000);　//x=22343

x は、-65536/2 ~ 65536/2 の範囲の値となります。リングバッファで考え、経路が近いほうの差分を返します。



1.8. プログラム作成上の注意点

1.8.1. 浮動小数点値の取り扱い

float（32bit 浮動小数点）のオーディオデータを取り使う際、計算の途中で double に変換されると、極端に計算速度が落ちるケースがあります。

－NG のケース－

```
static void IIR_resonator(float fc, float Q, float a[], float b[])
{
  fc = tanf((float)M_PI * fc) / (2.0f * (float)M_PI);

  a[0] = 1.0f + 2.0f * (float)M_PI * fc / Q + 4.0f * (float)M_PI * (float)M_PI * fc * fc;
  a[1] = (8.0f * (float)M_PI * (float)M_PI * fc * fc - 2.0f) / a[0];
  a[2] = (1.0f - 2.0f * (float)M_PI * fc / Q + 4.0 * (float)M_PI * (float)M_PI * fc * fc) / a[0];
  b[0] = 2.0f * (float)M_PI * fc / Q / a[0];
  b[1] = 0.0f;
  b[2] = -2.0f * (float)M_PI * fc / Q / a[0];

  a[0] = 1.0f;
}
```

上記のフィルタ処理では、マイコンによってはリアルタイム処理という観点で、計算が間に合いませんでした。

－OK のケース－

```
static void IIR_resonator(float fc, float Q, float a[], float b[])
{
  fc = tanf((float)M_PI * fc) / (2.0f * (float)M_PI);

  a[0] = 1.0f + 2.0f * (float)M_PI * fc / Q + 4.0f * (float)M_PI * (float)M_PI * fc * fc;
  a[1] = (8.0f * (float)M_PI * (float)M_PI * fc * fc - 2.0f) / a[0];
  a[2] = (1.0f - 2.0f * (float)M_PI * fc / Q + 4.0f * (float)M_PI * (float)M_PI * fc * fc) / a[0];
  b[0] = 2.0f * (float)M_PI * fc / Q / a[0];
  b[1] = 0.0f;
  b[2] = -2.0f * (float)M_PI * fc / Q / a[0];

  a[0] = 1.0f;
}
```

こちらは問題ありませんでした。

2 者の違い判りますでしょうか。本当に些細な違いですが、処理が間に合わない方には、

－NG のケース－

```
  a[2] = (1.0f - 2.0f * (float)M_PI * fc / Q + 4.0 * (float)M_PI * (float)M_PI * fc * fc) / a[0];
```

4.0 の後ろに f がありません。そのため、計算が double で行われ、実行時間が余計に掛かってしまいます。
（double で計算された後、a[2]に float 型に変換されて代入）

同様に、π^2 を計算している部分で

－NG のケース－

```
a[1] = (8.0f * (float)M_PI * M_PI * fc * fc - 2.0f) / a[0];
```

－OK のケース－

```
a[1] = (8.0f * (float)M_PI * (float)M_PI * fc * fc - 2.0f) / a[0];
```

円周率定数 M_PI の前に(float)のキャストがないケースでも、double で計算されて処理時間が長くなります。

```
 9          a[0] = 1.0f + 2.0f * (float)M_PI * fc / Q + 4.0f * (float)M_PI * (float)M_PI * fc * fc;
            conversion from 'double' to 'float' may change value [-Wfloat-conversion]
10          a[1] = (8.0f * (float)M_PI * M_PI * fc * fc - 2.0f) / a[0];
11          a[2] = (1.0f - 2.0f * (float)M_PI * fc / Q + 4.0f * (float)M_PI * (float)M_PI * fc * fc) / a[0];
12          b[0] = 2.0f * (float)M_PI * fc / Q / a[0];
```

計算結果が double で計算された結果が、float の変数に代入された場合、デフォルトでは

conversion from 'double' to 'float' may change value [-Wfloat-conversion]

というワーニングが出ます。このワーニング自体は、問題ではないケースもありますが、無駄に double で計算されて時間を食っている事が考えられますので、double の精度が必要ない場面では計算が float 型で行われているかを気にしてください。

RA（本サンプルプログラム対象マイコンの Cortex-M4,Cortex-M33）は、浮動小数点ユニットを持っていますので、float（32bit 浮動小数点数）の四則演算であれば、1 命令でハードウェアで処理できますが、double での計算はライブラリを使ったソフトウェアでの処理となるので、四則演算でも複数の命令で構成されたかなり長い処理となります。

RA（+GCC 処理系）では、

2.0　double 型（ソフトウェアライブラリを使ったソフトウェア処理での演算）
2.0f　float 型（浮動小数点ユニットを使ってハードウェアで演算処理可能）

の様に"f"が付く、付かないで大きく変わりますので、double 型の精度が必要のない場面では、数値定数に f のサフィックスを必ず付けるようにしてください。

RX マイコンでは、

・倍精度浮動小数点ユニットを持たないマイコン(RX651 等）

CC-RX コンパイラのデフォルトオプションでは double=float（32bit 浮動小数点数）なので、2.0 の様な数値を計算に用いても計算が遅くなるという事はありません。

・倍精度浮動小数点ユニットを持つマイコン(RX72N 等）

double 型（64bit 浮動小数点数）も、FPU で処理されるので、double 型を使っても極端に計算が遅くなる事はありません。

それに対し、RA マイコンでは float と double で大きく計算速度が変わってくるので、RX でのプログラムに慣れている方は注意してください。

1.8.2. 数値計算の速度に関して

本サンプルプログラムで取り扱っているのは、オーディオストリームで、サンプリング周波数は 48kHz なので、20.83us に 1 データの入力、出力があります。

1 データを平均 20.83us 以内で演算処理できなければ、取りこぼしが生じます（一般的には無音期間や、プツプツといったノイズが生じるといった現象に見えてきます）。

そのため、どのぐらいの速度で数値計算が行えるのかというのは、非常に重要な要素となります。
（2 章では、色々な音声処理に掛かる時間のベンチマーク結果を載せていますので参考にしてください。）

各種演算を 10,000 回行った場合の実行時間[us]を示します。（ループ変数の加算や条件分岐も含んだ時間ですので、単純な演算時間ではありませんが、参考にしてください。）

最適化はデフォルト(-O2)で、RX は CC-RX, RA は GCC を使った場合の結果です。

		RX72M, 240MHz (RXv3)[参考]	**RA6M2, 120MHz (Cortex-M4)**	**RA6M4, 200MHz (Cortex-M33)**
+(add)	float	540	**1260**	**1440**
	double	750	**11000**	**6840**
-(sub)	float	540	**1260**	**1440**
	double	750	**11000**	**6840**
*(mul)	float	540	**1260**	**1440**
	double	794	**5080**	**5000**
/(div)	float	1120	**2320**	**2090**
	double	1960	**54400**	**33700**
sin	float	3350(1250)	**18200**	**9360**
	double	8480	**228000**	**136000**
pow	float	3380	**46800**	**29300**
	double	6580	**438000**	**340000**
hypot(*1)	float	12700(1330)	**12500**	**8640**
	double	20100	**152000**	**108000**
$\sqrt{(a^2+b^2)}$(*1)	float	1460	**3080**	**2900**
	double	2490	**100000**	**61600**

単位[us], 10,000 回当たり

RX72M でカッコ内()の値は、三角関数演算器を有効化した（三角関数等をハードウェアで処理する）場合です。sin, cos, atan, hypot を float で計算する際に高速化されます。

(*1)hypot 関数と$\sqrt{(a^2+b^2)}$：sqrt(a*a+b*b)は同じ演算ですが、RX72M で float 型で計算する場合、三角関数演算器を使う hypot が有利だと思います。それ以外の条件では、本試行では sqrt(a*a+b*b)で計算した方が速いという結果となりました。（a=2.0, b=3.0 の場合）引数の値によって速度に差があるのかもしれません。

・double の計算は RA に対し RX72M が圧倒的に速い
→ハードウェアで処理するため、RX では必要に応じて double を使用しても大きなデメリットにはならないと考えます
→RA ではできるだけ double を使用しない方が望ましいと考えられます

・RA の pow（べき乗）が遅い
→サンプルプログラム(RAxxx_ADA_IF_SAMPLE1)では、pow 演算が遅くてリアルタイム処理が間に合わなかったので、powf()を四則演算に置き換えています
→RA では powf()を実行する回数が多い場合は、別な演算に置き換えることを検討してください

2. RAxxx_ADA-IF_SAMPLE1

2.1. 概要

1ch 処理（入力、出力とも Lch しか使用しない）の楽器用エフェクト処理を集めたプログラムです。

(0)スルー（入力信号をそのまま出力する）
(1)ディレイ
(2)リバーブ
(3)ディストーション
(4)ワウ
(5)トレモロ
(6)ビブラート
(7)コーラス
(8)フランジャ
(9)ラジオボイス

の機能を持たせています。

本プログラムは、「C 言語ではじめる 音のプログラミング オーム社(ISBN-978-4-274-20650-4)」「サウンドプログラミング入門 音響合成の基本と C 言語による実装 技術評論社(ISBN-978-4-7741-5522-7)」の著者である青木直史氏に作成頂いたプログラムを 1 つにまとめたものです。

各処理がどのようなプログラムとなっているかは、上記参考文献で詳細に説明されていますので、音声処理のプログラムを学びたい方は参考にして頂きたく。

2.2. プログラムの動作

プログラムを実行すると、入力された信号がそのまま出力される（スルー）の動作となります。マイコンボード上の TEST スイッチを押すと、ディレイ動作に切り替わります。以下、TEST スイッチを押す度に、「リバーブ」「ディストーション」…と動作が切り変わります。

SCI9 と PC を接続して、ターミナルソフト（Teraterm 等）を接続すると、

・情報の出力
・キーボードからの指示（動作モードの変更、エフェクト量の変更）

が行えます。

0-9:動作モード切り替え（0:スルー～9:ラジオボイス）
q,a,z: volume1 エフェクト量減少（q=-100, a=-10, z=-1)
w,s,x: volume1 エフェクト量増加（w=+100, s=+10, x=+1)
e,d,c: volume2 エフェクト量減少（e=-100, d=-10, c=-1)
r,f,v: volume2 エフェクト量増加（r=+100, f=+10, v=+1)
t,g,b: volume3 エフェクト量減少（t=-100, g=-10, b=-1)
y,h,n: volume3 エフェクト量増加（y=+100, h=+10, n=+1)
p:情報表示

が行えます。

エフェクト量は、下記を変更します。

	volume1	volume2	volume3
(0)スルー	未使用	←	←
(1)ディレイ	遅延時間	減衰量	未使用
(2)リバーブ	遅延時間	減衰量	未使用
(3)ディストーション	ゲイン	レベル	未使用
(4)ワウ	depth	rate	未使用
(5)トレモロ	depth	rate	未使用
(6)ビブラート	depth	rate	未使用
(7)コーラス	depth	rate	未使用
(8)フランジャ	depth	rate	未使用
(9)ラジオボイス	未使用	←	←

2.3. 設定値

1.7 の「設定値のカスタマイズ」で説明している設定値としては、以下としています。

RAxxx_ADA-IF_SAMPLE1¥src¥¥ada_if¥ada_if_userdef.h

```
//出力 ch
#define USE_CH (1)    1ch(Lch)のみ使用

//データバッファサイズ
#define DATA_BUFFER_SIZE_SAME (1)    g_pcm0, gpcm1 を同じサイズにする

//FIFO バッファサイズ
#define FIFO_BUFFER_SIZE (4)    デフォルト

//出力バッファデータ送信後の挙動
#define OUTPUT_BUFFER_AFTER_SEND (1)    出力後データを消さない

//スイッチの状態を何種類設けるか
#define SW_STATE_NUM (10)    10 種類の動作モード

//ボード上の LED を使用
#define USE_LED (1)    LED を使う
```

```
//ADC を使用する
#define USE_ADC_PORT (0)     ADC(ボリュームの外部接続)は使わない

//SCI を使用する
#define USE_SCI (1)          SCI を使う
```

マイコンボード上の LED は、選択した動作(1)~(9)に対し、1~9 回の点滅となります。

2.4. ベンチマーク結果[参考]

本サンプルプログラムで切り替えできる 9 種類のエフェクトに関してベンチマークを取った結果を示します。

処理時間は、128 データを処理する時間です。

48kHz で 128 データなので、実時間で 2.67ms のデータを処理するのに掛かる時間です。この時間が 2.67ms に収まらないとリアルタイム処理が出来ない（音が途切れる原因）となります。

MCU	RX72M（RXv3 コア, 240MHz）[参考]			RA6M3(Cortex-M4, 120MHz)		RA6M4(Cortex-M33, 200MHz)		RA4M2(Cortex-M33, 100MHz)	
コンパイラオプション	-O0	-O2	-O2 三角関数ハード使用	-O0	-O2	-O0	-O2	-O0	-O2
(1)ディレイ	28	8	←	80	20	87(*1)	14	82(*1)	18
(2)リバーブ	672	530	←	**698**(*2)	**194**(*2)	**712**(*2)	**134**(*2)	**742**(*2)	**188**(*2)
(3)ディストーション	31	15	←	70	22	66	14	71	22
(4)ワウ	492	314	279	1190	448	992	242	1190	468
(5)トレモロ	68	48	38	190	106	128	53	189	106
(6)ビブラート	111	70	56	764	568	460	307	784	590
(7)コーラス	119	76	59	298	140	229	76	300	146
(8)フランジャ	119	77	61	298	146	229	77	296	150
(9)ラジオボイス	114	39	←	410	55	382	45	438	56

単位[us]　※各処理に掛かる時間が 2,670us 以下である必要がある

(*1)200MHz のマイコンの方が何故か処理が遅い
(*2)RA マイコンは処理系に GCC を使っているが、powf（べき乗を求める関数）の処理が極端に遅く 2.67ms の周期に間に合わなかったので、乗算に置き換えて処理している（RX で同様の置き換えを行うと RA より高速になると考えます）

音声処理では、主に浮動小数点の計算を行っています。

RX マイコン（RX72M 等の RXv3 コア）は、浮動小数点の計算も速く、三角関数の計算もハードウェアで行えるため、音声処理にも適しています。

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	2021.8.3	—	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

I2S サウンドインタフェース
サンプルプログラム(RA)

株式会社 北斗電子
 2021 年 8 月 3 日改訂 REV.1.0.0.0 (210803)