TK-850/JH3U-SP ボード

オーディオサンプルプログラム

説明書

（第 1.0 版）

テセラ･テクノロジー株式会社

目次

図の目次

1 概　　説

このマニュアルでは，V850E/V850ES 向けの音声圧縮／伸長ソフトウェア・パッケージ ADPCM-SP を用いた音声再生方法と音声録音方法について，サンプル・プログラムを例に説明します。

1.1　開発環境

このサンプル・プログラムの開発環境および実行環境は次のとおりです。

種類		名称	バージョン
開発ツール（ソフト）	デバイス・ファイル	DF703771[注1]	V1.00
	統合開発環境	PM+	V6.31
	C コンパイラ	CA850	W3.30
	デバッガ	ID850QB	V3.50
ボード	評価ボード	TK-850/JH3U-SP （テセラ・テクノロジー株式会社製）	－
ライブラリ	音声圧縮／伸長ソフトウェア・パッケージ	ADPCM-SP	V1.00

注1.　DF703771 の中の，DF3769.800 を使用します。

1.2　説明項目

本アプリケーション･ノートで説明する項目は以下のとおりです。

- 音声ガイダンス「D/A による音声出力方法」
- 音声メモ「A/D による音声入力方法」

2　アプリケーションの実装例（音声ガイダンス）

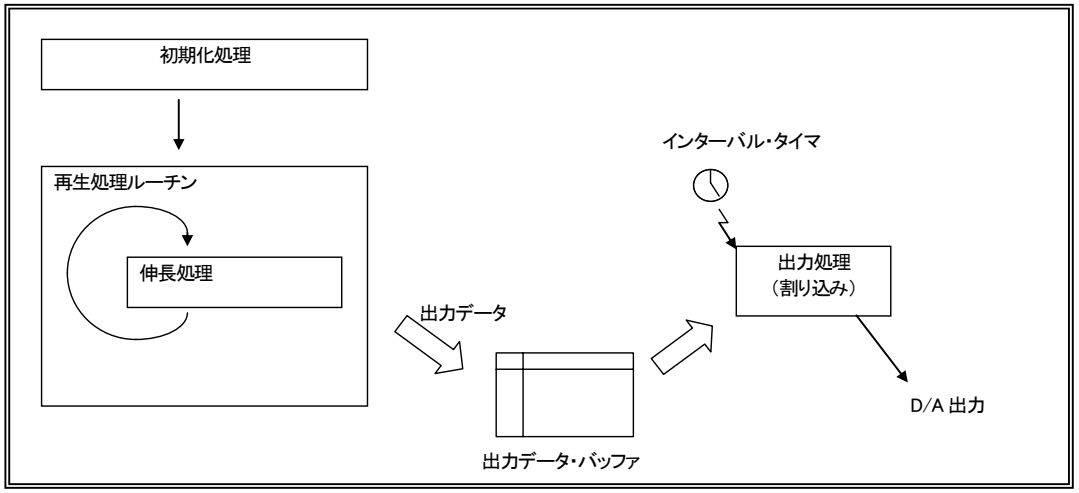
2.1　概要

音声ガイダンスサンプルは，ADPCM ライブラリの伸長機能を使用して，リアルタイムに伸長しながら音声出力を行う，音声ガイダンスプログラムです。
2 種類の音声が SW3，SW4 を押下すると再生されます。
また，DIPSW でサンプリングレートの変更が行えます。

DIPSW 状態		サンプリングレート
SW1-5	SW1-6	
OFF	ON	16kHz
ON	OFF	12kHz
OFF	OFF	8kHz
ON	ON	

次の図にアプリケーション処理のブロック図を示します。

図 2-1 音声再生アプリケーション処理のブロック図



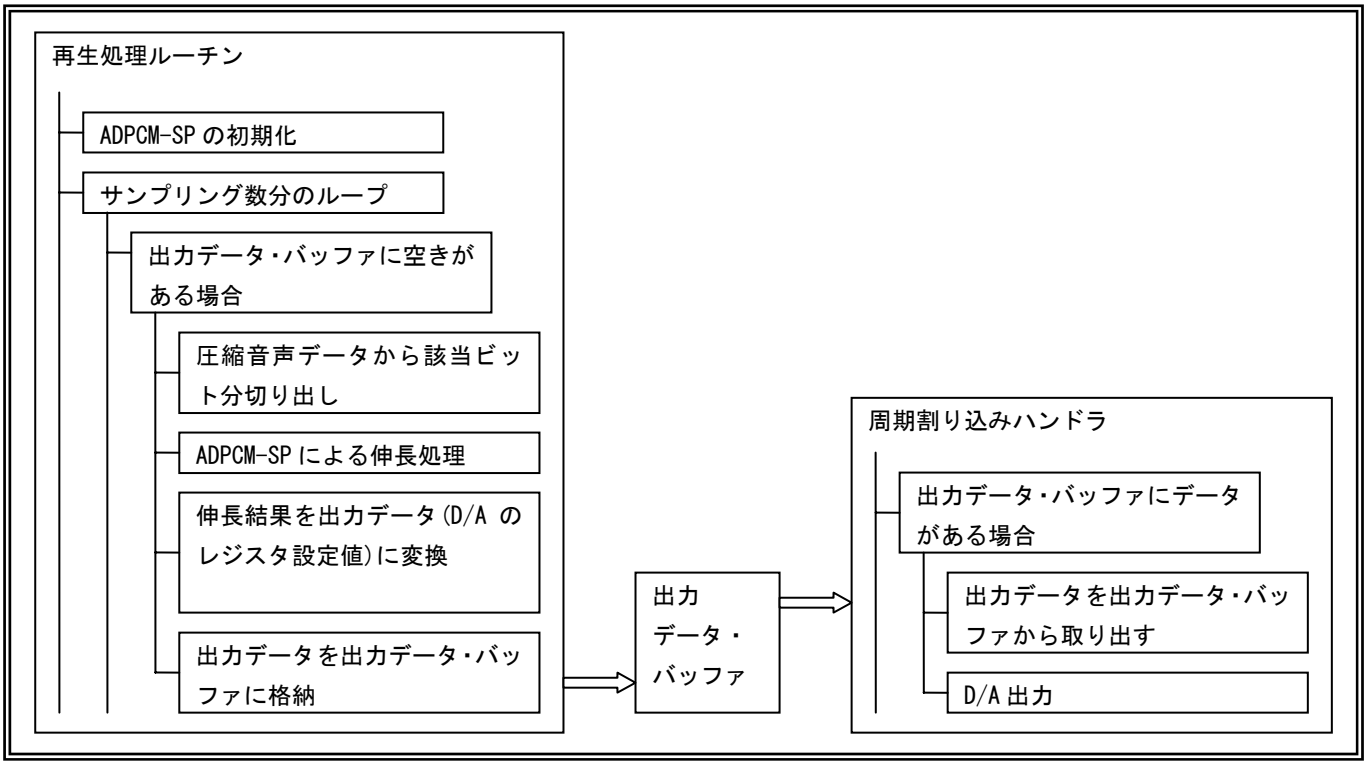
圧縮音声データから該当ビット（4 ビット圧縮の場合は 4 ビット，3 ビット圧縮の場合は 3 ビット，2 ビット圧縮の場合は 2 ビット）を切り出し，伸長処理（ライブラリ呼び出し）を行います。伸長した結果（PCM データ）を出力データ（D/A のレジスタ設定値）に変換して，出力データ・バッファに格納します。出力データ・バッファから，周期割り込みを利用して一定の周期で出力します。

2.1.1　再生処理と周期割り込みの関係

音声は，一定周期（サンプリング周波数が 8kHz の場合，周期は 125μs）での再生（出力データの更新）が必要になります。出力は,CPU 内蔵のインターバル・タイマの割り込みハンドラで D/A 出力するようにします。

再生処理と周期割り込み処理の処理イメージを図 2-2に示します。

図 2-2 再生処理と周期割り込みの処理イメージ



2.1.2　　出力データ・バッファ（リング・バッファ）

再生処理ルーチンで作成した出力データを，周期割り込みのタイミングでD/A出力するまで一時保管するために，バッファを使用します。

このバッファは，アプリケーションの処理量が一時的に多くなった場合でも，一定周期でD/A出力ができるように，複数個のデータを保持するFIFO型のバッファ（リング・バッファ）にします。

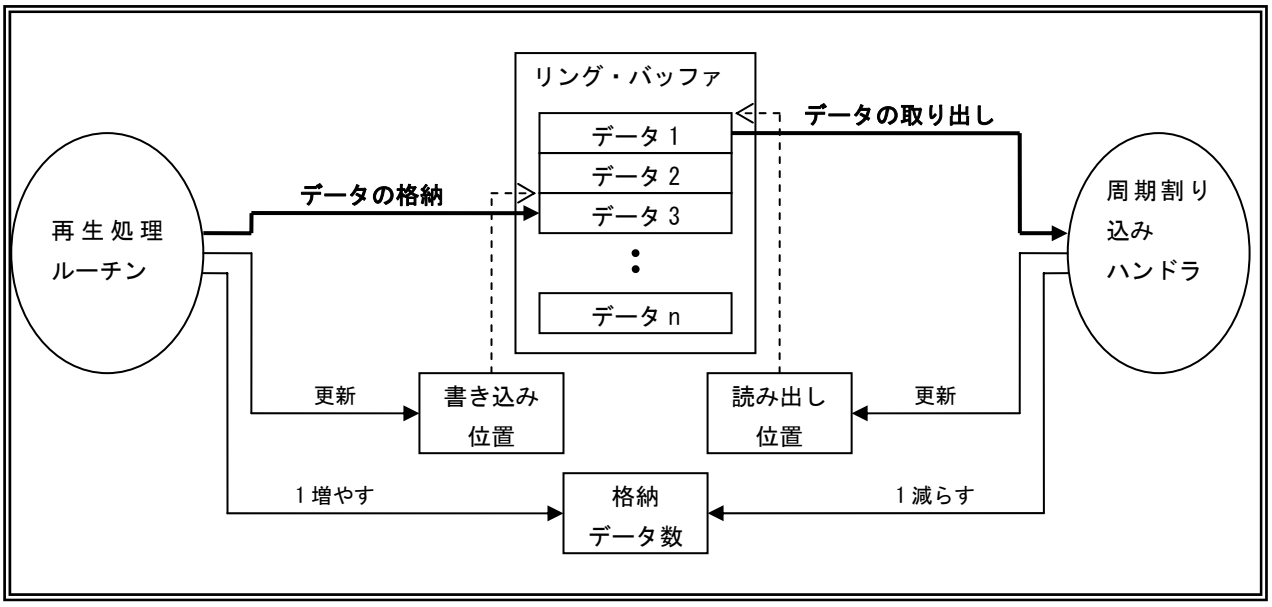
このリング・バッファは，書き込み位置，読み出し位置，格納データ数の変数で制御されます。それぞれの役割を以下に示します。

変数	説明
書き込み位置	再生処理ルーチンがデータを格納する位置（配列のインデクス）を指します。 格納後，次の格納位置を指すように更新します。 （更新方法：1増やします。配列の最後の場合は，0に戻します。）
読み出し位置	周期割り込みハンドラがデータを取り出す位置（配列のインデクス）を指します。 取り出し後，次の取り出し位置を指すように更新します。 （更新方法：1増やします。配列の最後の場合は，0に戻します。）
格納データ数	リング・バッファ内に格納されているデータ数を保持します。 再生処理ルーチンでデータを格納した時に1増やし，周期割り込みハンドラでデータを取り出した時に1減らします。

このリング・バッファの操作方法を図 2-3に示します。

なお，このリング・バッファ操作は，共用変数を複数アクセスするため，割り込み禁止状態（DI）で処理する必要があります。

図 2-3 リング・バッファの操作方法

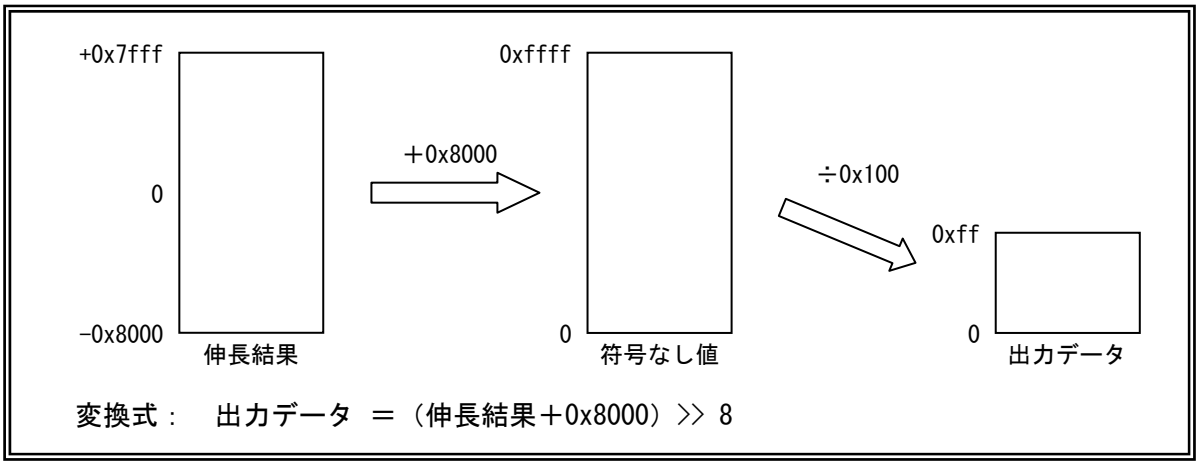


2.1.3　伸長結果から出力データへの変換

ADPCM-SP の伸長によって得られる伸長結果（符号付 16 ビットの PCM コード）は，出力データ（D/A のレジスタ設定値）に変換して，D/A 出力する必要があります。

出力データのMAX値が 2^n-1（例えば 0xff）の場合の変換方法を図 2-4に示します。

図 2-4 伸長結果から出力データへの変換方法



この場合，変換式は下式となり，高速に変換できます。

出力データ　＝（伸長結果＋0x8000）>> 8

一方，出力データのMAX値が 2^n-1 でない場合，変換式は下式となり，変換にやや時間がかかります。

出力データ　＝（(伸長結果＋0x8000）×（MAX＋1)）>> 16

D/A のレジスタ設定値の最大は，0xff であるため高速に変換できます。

また，シフト処理の前に補正値を加えることによって，シフト処理による「切捨て」を「四捨五入」に変更することができます。これにより，音質が向上する場合があります。この場合，変換式は以下の通りになります。

出力データ　＝（伸長結果＋0x8080）>> 8　　（出力データの MAX 値が 0xff の場合） 出力データ　＝（伸長結果＋0x8040）>> 7　　（出力データの MAX 値が 0x1ff の場合）

2.2　ソース・コードとその解説

(1) インクルード/定義/宣言部

インクルード/定義/宣言部

```
#pragma ioreg

#include "adpcmsp.h"
#include "Chime_8kHz.c"
#include "Chime_12kHz.c"
#include "Chime_16kHz.c"
#include "NAct_8kHz.c"
#include "NAct_12kHz.c"
#include "NAct_16kHz.c"

/* ADPCM */
unsigned short g_adpcmWork[16];                 /* ADPCM work area           */

/* Output data buffer (Ring buffer) */
#define OUTBUF_SIZE     8                       /* Output buffer size        */
unsigned short g_outbufData[OUTBUF_SIZE];       /* Output buffer data        */
volatile unsigned char g_outbufCnt;             /* Output buffer data count  */
unsigned char g_outbufIdxW;                     /* Index for writing         */
unsigned char g_outbufIdxR;                     /* Index for reading         */

/* Switch flag */
volatile unsigned char sw3;                     /* SW3 Push flag             */
volatile unsigned char sw4;                     /* SW4 Push flag             */

/* Local procedures */
void InitCpu(void);
void StartOutput(void);
void StopOutput(void);
void PlaySound(unsigned long sampleCount, const unsigned char *adpcmData);

/* Local define */
#define SPK_VOL             0x79                /* Speaker volume            */
#define SW_PUSH             0x01                /* Push switch               */
#define SW_CLR              0x00                /* Push switch clear         */
```

------ ①

------ ②

------ ③

① インクルード部

ADPCM-SP ライブラリが提供する，“adpcmsp.h”および，CvADPCM より生成された圧縮音声データ・ファイルをインクルードします。

② 変数定義部

ADPCM-SP ライブラリの作業用領域として，“g_adpcmWork[16]”を用意します。この領域は，ADPCM-SP ライブラリを通してアクセスされるものであり，アプリケーションが直接アクセスすることはありません。

また，出力データ・バッファ（リング・バッファ）として“g_outbufData[]”，リング・バッファ内に格納されているデータ数として“g_outbufCnt”,再生処理ルーチンがデータを格納する位置として“g_outbufIdxW”,周期割り込みハンドラがデータを取り出す位置として“g_outbufIdxR”を用意します。このうち,“g_outbufCnt”は,再生処理ルーチンおよび周期割り込みハンドラ双方において,処理のトリガとして使用するため,volatile 宣言する必要があります。

最後に，スイッチ押下検出に用いるフラグ変数を用意します。

③ マクロ定義部

SPK_VOL は任意の値に変更可能です。0x00～0xFF の範囲で、値が大きいほどスピーカーのボリュームが大きくなります。

(2) メイン処理

メイン処理

```
void main(void)
{
    unsigned long sampleCount;
    const unsigned char *adpcmData;

    /* Initialize CPU */
    InitCpu();                                                    ---------- ①

    /* Initialize output data buffer  */
    g_outbufCnt = 0;
    g_outbufIdxW = 0;
    g_outbufIdxR = 0;                                             ---------- ②

    /* Initialize switch flag */
    sw3 = 0;
    sw4 = 0;

    /* Start output interface */
    StartOutput();                                                ---------- ③

    while(1){
        if(sw3){
            if( (P7H&0x03) == 0x01 ) {
                /* Play sound (16kHz) */
                sampleCount = (unsigned long)sizeof(Chime16k)*2;
                adpcmData = Chime16k;
                TAA1CCR0 = 0x05DB;              /* Interval=62.5us          */
            } else if( (P7H&0x03) == 0x02 ) {
                /* Play sound (12kHz) */
                sampleCount = (unsigned long)sizeof(Chime12k)*2;
                adpcmData = Chime12k;
                TAA1CCR0 = 0x07CE;              /* Interval=83.3us          */
            } else {                                              ---------- ④
                /* Play sound (8kHz) */
                sampleCount = (unsigned long)sizeof(Chime8k)*2;
                adpcmData = Chime8k;
                TAA1CCR0 = 0x0BB5;              /* Interval=125us           */
            }
            /* Play one sound */
            PlaySound(sampleCount, adpcmData);
            /* Switch flag clear */
            sw3 = SW_CLR;
            sw4 = SW_CLR;
        }
```

```
        else if(sw4){
            if( (P7H&0x03) == 0x01 ) {
                /* Play sound (16kHz) */
                sampleCount = (unsigned long)sizeof(NAct16k)*2;
                adpcmData = NAct16k;
                TAA1CCR0 = 0x05DB;              /* Interval=62.5us          */
            } else if( (P7H&0x03) == 0x02 ) {
                /* Play sound (12kHz) */
                sampleCount = (unsigned long)sizeof(NAct12k)*2;
                adpcmData = NAct12k;
                TAA1CCR0 = 0x07CE;              /* Interval=83.3us          */
            } else {
                /* Play sound (8kHz) */
                sampleCount = (unsigned long)sizeof(NAct8k)*2;
                adpcmData = NAct8k;
                TAA1CCR0 = 0x0BB5;              /* Interval=125us           */
            }
            /* Play one sound */
            PlaySound(sampleCount, adpcmData);
            /* Switch flag clear */
            sw3 = SW_CLR;
            sw4 = SW_CLR;
        }
    }
}
```

④

① 最初に，“InitCpu()”を呼び出して，CPU の初期化を行います。
② 出力データ・バッファ（リング･バッファ），SW の制御変数を初期化します。
③ “StartOutput()”を呼び出して，出力機構を開始します。
④ SW3，SW4 を押下すると，DIPSW の状態によってサンプリングレート 16kHz，12kHz，8kHz の音声を再生します。
DIPSW の状態とサンプリングレート，タイマの設定値の対応は以下のようになっています。

DIPSW 状態		サンプリングレート	タイマコンペア値(インターバル時間)
DIPSW5	DIPSW6		
OFF	ON	16kHz	0x05DB(62.5μs)
ON	OFF	12kHz	0x07CE(83.3μs)
OFF	OFF	8kHz	0x0BB5(125μs)
ON	ON		

(3) 初期化処理

CPU 初期化処理を示します。
CPU 初期化処理は，クロック，バス，汎用 I/O，内蔵機能などの設定を行います。
ここでは音声再生に伴う，内蔵機能の初期化の例を示します。

初期化処理

```
void InitCpu(void)
{
    __DI();                             /* maskable interrupt disable        */

    /* Initialize D/A output */                                                  ①
    P1.0 = 1;
    PM1.0 = 1;                          /* P10: ANO0 pin output              */
    P1.1 = 1;
    PM1.1 = 0;                          /* P11: ANO1 pin output              */
    DAOM = 0x00;                        /* Normal mode                       */
    DAOCS1 = SPK_VOL;
    DAOCE1 = 1;

    /* Initialize Interval Timer (TAA1) */                                       ②
    TAA1CTL0 = 0x01;                    /* Count clock=fxx/2 (T=41.7ns)      */
    TAA1CTL1 = 0x00;                    /* Interval timer mode               */
    TAA1IOC0 = 0x00;                    /* Output disabled                   */
    TAA1CCR0 = 0x05DB;                  /* Interval=62.5us                   */
    TAA1CCMK0 = 0;                      /* Enable interrupt                  */

    /* Push SW setting */                                                        ③
    PMC9H = 0x03;                       /* INTP12/INTP13 input               */
    PFC9H = 0x03;                       /* PFC98&PFC99   -> 1                */
    PMK12 = 0;                          /* Enable interrupt                  */
    PMK13 = 0;                          /* Enable interrupt                  */

    __EI();                             /* maskable interrupt enable         */
}
```

① D/A 出力のための初期化

アナログ出力機能を用いてアナログ出力端子に任意の電圧を出力します。

このときの CPU 機能初期化内容は次の表のようになります。

レジスタ名	設定値	意味
PM1.0	1	P10 を ANO0 端子出力として使用する
PM1.1	1	P11 を ANO1 端子出力として使用する
DAOM	0x00	通常モード
DAOCS1	SPK_VOL	音声の出力レベルを設定する。
DAOCE1	1	D/A コンバータの動作を許可する。

② インターバル・タイマの初期化

一定周期ごとに再生データを出力するため，インターバル・タイマによる割り込みを使用します。この割り込みハンドラで，D/A 出力を行うことになります。

ここでは，タイマ TAA1 を使用した周期割り込みを使用しています。

タイマ名	動作モード	ｲﾝﾀｰﾊﾞﾙ間隔	ー	割り込み
TAA1	ｲﾝﾀｰﾊﾞﾙ･ﾀｲﾏ	62.5μs	ー	許可

このときの CPU 機能初期化内容は次の表のようになります。

レジスタ名	設定値	意味
TAA1CTL0	0x01	カウント・クロック＝fxx/2＝24MHz（T=41.7ns）
TAA1CTL1	0x00	インターバル・タイマ・モード
TAA1IOC0	0x00	出力禁止
TAA1CCR0	0x05DB	インターバル間隔＝1,499×41.7ns＝62.5μs
TAA1CCMK0	0	割り込み許可

③ スイッチのための初期化

SW3，SW4 を使用します。

レジスタ名	設定値	意味
PMC9H	0x03	P98 を INTP12 入力端子として使用する。
PFC9H	0x03	P99 を INTP13 入力端子として使用する。
PMK12	0	割り込み許可
PMK13	0	割り込み許可

(4) 出力開始／終了処理

出力開始および終了時の周辺機能レジスタ設定の実装を示します。

出力開始／終了処理

```
void StartOutput(void)
{
    /* Start D/A output (DAC0) */
    DAOCS0 = 0x80;                      /* Central value                */  ------ ①
    DAOCE0 = 1;                         /* Start operation              */

    /* Start Interval Timer (TMP1) */
    TAA1CE = 1;                         /* Start operation              */  --------- ②
}
void StopOutput(void)
{
    /* Stop Interval Timer (TMP1) */
    TAA1CE = 0;                         /* Stop operation               */  ------ ③
    TAA1CCIF0 = 0;                      /* Clear interrupt request flag */

    /* Stop D/A output (DAC0) */
    DAOCE0 = 0;                         /* Stop operation               */  --------- ④
}
```

備考　“StartOutput()”は再生処理の開始時に呼び出します。
“StopOutput()”は再生処理の終了時に呼び出します。

① D/A 出力開始

再生処理の開始時に設定します。

レジスタ名	設定値	意味
DA0CS0	0x80	中央値（無音出力）
DA0CE0	1	動作許可（動作開始）

② インターバル・タイマの開始

再生処理の開始時に設定します。

レジスタ名	設定値	意味
TAA1CE	1	動作許可（動作開始）

③ インターバル・タイマの停止

再生処理の終了時に設定します。

レジスタ名	設定値	意味
TAA1CE	0	動作禁止（動作停止）
TAA1CCIF0	0	割り込み要求フラグのクリア

④ D/A 出力停止

再生処理の終了時に設定します。

レジスタ名	設定値	意味
DA0CE0	0	動作禁止（動作停止）

(5) 再生処理

圧縮音声データの再生処理を示します。

再生処理

```
void PlaySound(unsigned long sampleCount, const unsigned char *adpcmData)
{
    unsigned long scnt;                     /* counter                  */
    unsigned char byte;                     /* cache data               */
    unsigned char adpcmCode;                /* ADPCM code               */
    short pcmCode;                          /* PCM code                 */
    unsigned short outputData;              /* output data              */

    /* Initialize ADPCM */
    adpcm_init(g_adpcmWork);                                    ---------- ①

    scnt = 0;
    while(scnt < sampleCount){              /* Loop of sample count     */
        if(g_outbufCnt < OUTBUF_SIZE){      /* If space exists in buffer */  ---------- ②

            /* Get ADPCM code */
            if(!(scnt & 0x01)){             /* Even number              */
                byte  = *adpcmData;
                adpcmCode = byte&0x0f;                                  ------ ③
            }else{                          /* Odd number               */
                adpcmCode = byte>>4;
                adpcmData++;
            }

            /* Decode ADPCM code to PCM code */
            pcmCode = adpcm_l32_dec(adpcmCode, g_adpcmWork);        ---------- ④
```

```
            /* Convert PCM code to output data */
            if(pcmCode >= 0x7f80){              /* Check overflow            */  --------- ⑤
                outputData = 0xff;
            }else{
                outputData = (unsigned short)(pcmCode+0x8080)>>8;     ------------------- ⑥
            }

            /* Set output data to the output buffer */                                  ┐
            __DI();                                                                     │
            g_outbufData[g_outbufIdxW] = outputData;                                    ├------ ⑦
            g_outbufIdxW = (g_outbufIdxW+1)&(OUTBUF_SIZE-1);                            │
            g_outbufCnt++;                                                              │
            __EI();                                                                     ┘

            scnt++;
        }else{
            /* Other jobs under playing sound */                        ------------------- ⑧
        }
    }
    while(g_outbufCnt != 0) ;                   /* Wait for the final output  */  --------- ⑨
}
```

① 一連の伸長処理の前に，“adpcm_init()”を呼び出して，ADPCM-SP ライブラリの初期化を行います。
② リング・バッファに空きがない場合は，伸長処理を行いません。
③ ADPCM データから ADPCM コードを切り出します。4 ビット圧縮の場合，4 ビットずつ切り出します。
（3 ビット圧縮の場合は，3 ビットずつ，2 ビット圧縮の場合は，2 ビットずつ切り出します。）
④ ADPCM コードを伸長して PCM コードを取得します。4 ビット圧縮の場合，“adpcm_l32_dec()”を呼び出します。
（3 ビット圧縮の場合は，“adpcm_l24_dec()”を，2 ビット圧縮の場合は，“adpcm_l16_dec()”を呼び出します。）
⑤ ⑥の加算処理でオーバフローしないように，ここでオーバフローのチェックを行います。
⑥ PCM コードから出力データ（D/A のレジスタ設定値）に変換します。
⑦ リング・バッファ“g_outbufData[]”へ出力データを書き込みます。
同時に，リング・バッファの書き込み位置“g_outbufIdxW”と格納データ数“g_outbufCnt”を更新します。
また，このリング・バッファ操作は，割り込みハンドラとの共用変数を複数アクセスするため，割り込み禁止状態（DI）で処理する必要があります。
⑧ この条件成立時は，再生期間中に行う他のアプリケーション処理を行うことができます。
⑨ 最終データの出力完了まで待ち合わせます。

(6) 周期割り込みハンドラ

周期割り込みハンドラを示します。
周期ハンドラはサンプリング周波数が 8kHz の場合，周期は 125μs で実行されます。

周期割り込みハンドラ

```
#pragma interrupt INTTAA1CC0     inttaa1cc0
__interrupt
void inttaa1cc0(void)
{
    if(g_outbufCnt > 0){                        /* If data exists in buffer   */  --------- ①
        DAOCS0 = (unsigned char)g_outbufData[g_outbufIdxR];
        g_outbufIdxR = (g_outbufIdxR+1)&(OUTBUF_SIZE-1);                          ------- ②
        g_outbufCnt--;
    }
}
```

① 再生処理のループ内に他の処理を追加した場合は,再生周期に伸長処理が間に合わなくなることもあります。その場合には“g_outbufCnt”が 0 になるため，ここでそれを判断することができます。
② リング・バッファ“g_outbufData[]”から出力データを読み出し，D/A のレジスタに設定します。
同時に，リング・バッファの読み出し位置“g_outbufIdxR”と格納データ数“g_outbufCnt”を更新します。

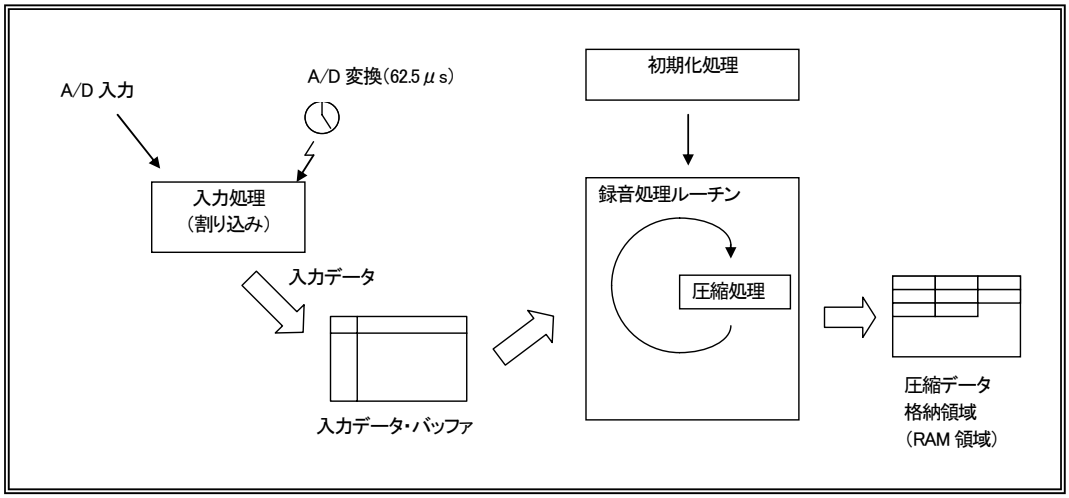
3　アプリケーションの実装例（音声メモ）

3.1　概要

音声メモサンプルは，ADPCM ライブラリの圧縮機能を使用して，ほぼリアルタイムに入力音声を出力しながら録音を行う，音声メモプログラムです。
SW3 を押下すると一定時間録音を行い，SW4 押下で録音した音声を再生します。

次の図にアプリケーション処理のブロック図を示します。

図 3-1 音声録音アプリケーション処理のブロック図



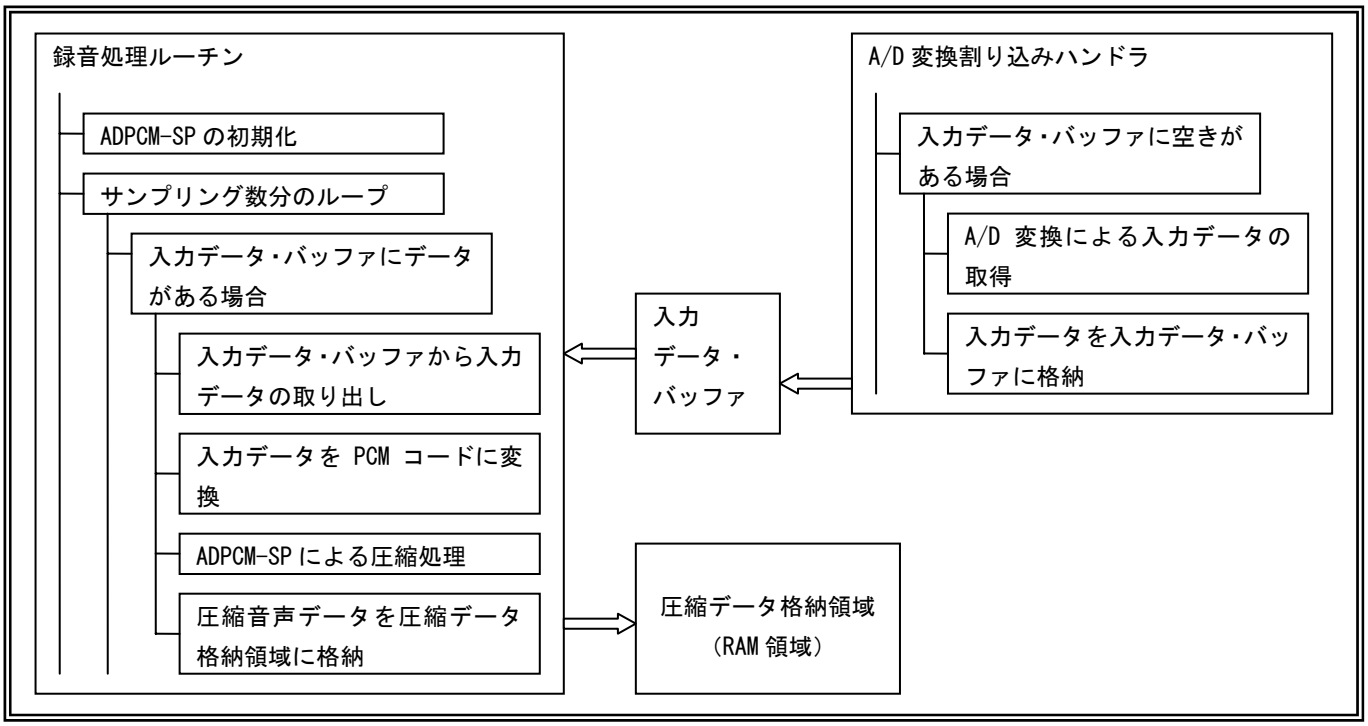
A/D 変換割り込みを利用して A/D 入力データを取得し，入力データ・バッファに格納します。入力データ・バッファから，該当ビット（4 ビット圧縮の場合は 4 ビット，3 ビット圧縮の場合は 3 ビット，2 ビット圧縮の場合は 2 ビット）で圧縮処理（ライブラリ呼び出し）を行い，圧縮データ格納領域（RAM 領域）にデータを格納します。

3.1.1　　録音処理と A/D 変換割り込みの関係

録音は，一定周期（サンプリング周波数が 16kHz の場合，周期は 62.5μs）での入力データの取得が必要になります。入力データの取得は，一定周期で発生する A/D 変換割り込みのハンドラで行うようにします。

録音処理とA/D変換割り込み処理の処理イメージを図 3-2に示します。

図 3-2 録音処理と A/D 変換割り込みの処理イメージ



3.1.2　　入力データ・バッファ（リング・バッファ）

A/D 変換割り込みのハンドラで取得した入力データを圧縮処理するまで一時保管するために，バッファを使用します。

このバッファは，アプリケーションの処理量が一時的に多くなった場合でも，一定周期で A/D 変換ができるように，複数個のデータを保持する FIFO 型のバッファ（リング･バッファ）にします。

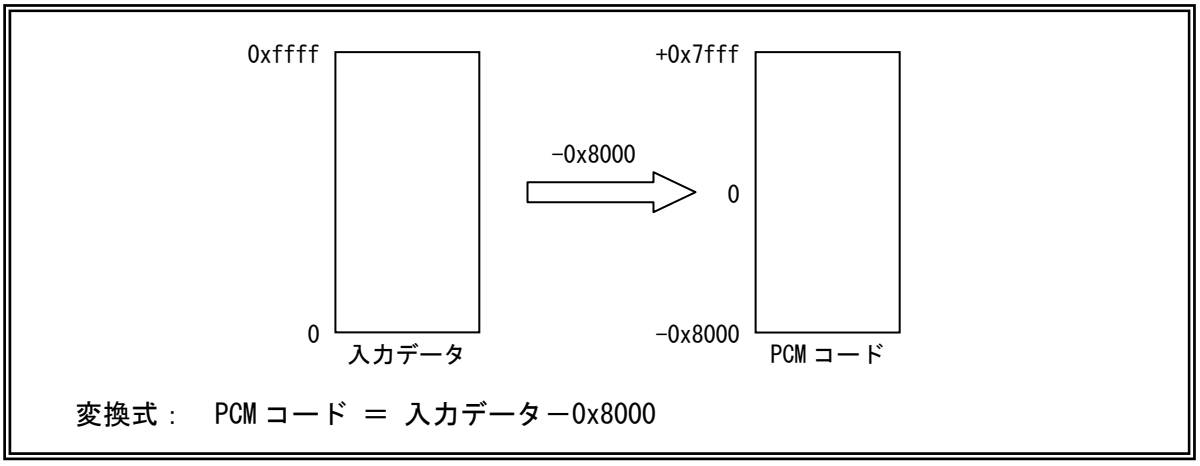
リング・バッファの詳細は2.1.2を参照してください。

3.1.3　　入力データから PCM コードへの変換

入力データ（A/D 変換によって取得された符号なし 16 ビット・データ）は，ADPCM-SP の圧縮関数の入力値である PCM コード（符号付 16 ビット・データ）に変換を行う必要があります。

変換方法を図 3-3に示します。

図 3-3 入力データから PCM コードへの変換方法



3.2　ソース・コードとその解説

(1) インクルード/定義/宣言部

インクルード/定義/宣言部

```
#pragma ioreg

#include "adpcmsp.h"
#include "sound_pipi.c"                                                    ------ ①

/* ADPCM */
unsigned short g_adpcmWork[16];                 /* ADPCM work area           */

/* A/D */
#define SAMPLE_COUNT   80000                    /* Sample count              */
unsigned char g_encodeData[(SAMPLE_COUNT+1)/2]; /* Encode data buffer(in RAM) */

/* Input data buffer (Ring buffer) */
#define INBUF_SIZE    8                         /* Input buffer size         */
unsigned short g_inbufData[INBUF_SIZE];         /* Input buffer data         */
volatile unsigned char g_inbufCnt;              /* Input buffer data count   */
unsigned char g_inbufIdxW;                      /* Index for writing         */
unsigned char g_inbufIdxR;                      /* Index for reading         */      ------ ②

/* Output data buffer (Ring buffer) */
#define OUTBUF_SIZE     8                       /* Output buffer size        */
unsigned short g_outbufData[OUTBUF_SIZE];       /* Output buffer data        */
volatile unsigned char g_outbufCnt;             /* Output buffer data count  */
unsigned char g_outbufIdxW;                     /* Index for writing         */
unsigned char g_outbufIdxR;                     /* Index for reading         */

/* Switch flag */
volatile unsigned char sw3;                     /* SW3 Push flag             */
volatile unsigned char sw4;                     /* SW4 Push flag             */

/* Local procedures */
void InitCpu(void);
void StartOutput(void);
void StopOutput(void);
void StartInput(void);
void StopInput(void);
void PlaySound(unsigned long sampleCount, const unsigned char *adpcmData);
void RecordSound(unsigned long sampleCount, unsigned char *adpcmData);
```

```
/* Local define */
#define SPK_VOL            0x79          /* Speaker volume          */
#define MIC_VOL          0x4005          /* Mic level               */
#define SW_PUSH            0x01          /* Push switch             */
#define SW_CLR             0x00          /* Push switch clear       */
#define AD_MASK_SET        0x01          /* A/D Interrupt mask set  */
#define AD_MASK_CLR        0x00          /* A/D Interrupt mask clear*/
```

① ------ ①

① インクルード部

ADPCM-SP ライブラリが提供する，“adpcmsp.h”および，CvADPCM より生成された圧縮音声データ・ファイルをインクルードします。

② 変数定義部

ADPCM-SP ライブラリの作業用領域として，“g_adpcmWork[16]”を用意します。この領域は，ADPCM-SP ライブラリを通してアクセスされるものであり，アプリケーションが直接アクセスすることはありません。

また，圧縮した音声データを格納する圧縮データ格納領域として“encodeData[]”を用意します。

また，入力データ・バッファ（リング・バッファ）として“g_inbufData[]”，リング・バッファ内に格納されているデータ数として“g_inbufCnt”，A/D 変換割り込みハンドラがデータを格納する位置として“g_inbufIdxW”，録音処理ルーチンがデータを取り出す位置として“g_inbufIdxR”を用意します。このうち，“g_inbufCnt”は，録音処理ルーチンおよび A/D 変換割り込みハンドラ双方において，処理のトリガとして使用するため，volatile 宣言する必要があります。

また，録音した音声を再生する場合，サンプル 1 同様の再生用変数の定義も必要です。

最後に，スイッチ押下検出に用いるフラグ変数を用意します。

③ マクロ定義部

SPK_VOL は任意の値に変更可能です。0x00～0xFF の範囲で、値が大きいほどスピーカーのボリュームが大きくなります。

MIC_VOL は任意の値に変更可能です。0x4000～0x4007 の 8 段階でマイクの録音レベルを調節できます。

(2) メイン処理

メイン処理

```
void main(void)
{
    unsigned long sampleCount;
    const unsigned char *adpcmData;

    /* Initialize CPU */
    InitCpu();                                                    ---------- ①

    /* Initialize output data buffer  */
    g_inbufCnt = 0;
    g_inbufIdxW = 0;
    g_inbufIdxR = 0;

    /* Initialize output data buffer  */
    g_outbufCnt = 0;                                              ---------- ②
    g_outbufIdxW = 0;
    g_outbufIdxR = 0;

    /* Initialize switch flag */
    sw3 = 0;
    sw4 = 0;

    /* Start input interface */
    StartInput();                                                 ---------- ③
    /* Start output interface */
    StartOutput();                                                ---------- ④

    while(1){
        if(sw3){
            /* Play sound (pipi) */
            StopInput();
            sampleCount = (unsigned long)sizeof(sound_pipi)*2;
            adpcmData = sound_pipi;
            PlaySound(sampleCount, adpcmData);

            /* Record sound */
            StartInput();                                         ---------- ⑤
            RecordSound(SAMPLE_COUNT, g_encodeData);

            /* Play sound (pipi) */
            StopInput();
            sampleCount = (unsigned long)sizeof(sound_pipi)*2;
            adpcmData = sound_pipi;
            PlaySound(sampleCount, adpcmData);
```

```
            /* Switch flag clear */
            sw3 = SW_CLR;
            sw4 = SW_CLR;
            StartInput();
        }
        if(sw4){
            /* Play sound (pipi) */
            StopInput();
            sampleCount = (unsigned long)sizeof(sound_pipi)*2;
            adpcmData = sound_pipi;
            PlaySound(sampleCount, adpcmData);

            /* Play recorded sound */
            sampleCount = SAMPLE_COUNT;
            adpcmData = g_encodeData;
            PlaySound(sampleCount, adpcmData);

            /* Play sound (pipi) */
            sampleCount = (unsigned long)sizeof(sound_pipi)*2;
            adpcmData = sound_pipi;
            PlaySound(sampleCount, adpcmData);

            /* Switch flag clear */
            sw3 = SW_CLR;
            sw4 = SW_CLR;
            StartInput();
        }
    }
}
```

⑤

⑥

① 最初に，“InitCpu()”を呼び出して，CPU の初期化を行います。
② 入出力データ・バッファ（リング･バッファ），SW3，SW4 の制御変数を初期化します。
③ “StartInput()”を呼び出して，入力機構を開始します。
④ “StartOutput()”を呼び出して，出力機構を開始します。
⑤ SW3 を押下すると，ピピッ音を再生後に“RecordSound()”を呼び出し，録音を開始します。
パラメータとして，録音サンプル数と圧縮データ格納領域の先頭アドレスを渡します。
最後に再度ピピッ音を再生し，スイッチ用フラグをクリアします。
⑥ SW4 を押下すると，ピピッ音を再生後に録音した音声を再生します。
再生が終了すると，ピピッ音を再度再生し，スイッチ用フラグをクリアします。

(3) 初期化処理

CPU 初期化処理を示します。
CPU 初期化処理は，クロック，バス，汎用 I/O，内蔵機能などの設定を行います。
ここでは音声録音に伴う，内蔵機能の初期化の例を示します。

初期化処理

```
void InitCpu(void)
{
    __DI();                             /* maskable interrupt disable        */

    /* Initialize D/A output */                                                  ┐
    ....................                                                        │
                                                                                │
    /* Initialize Interval Timer (TAA1) */                                      │
    ....................                                                        │
                                                                                ├------ ①
    /* Push SW setting */                                                       │
    ....................                                                        │
                                                                                │
                                                                                │
                                                                                ┘

    /* CSI for Audio device control */                                          ┐
    PMC2  = 0x0C;                                                               │
    PM2   = 0xF3;                                                               │
    P4.0  = 1;                                                                  ├------ ②
    PM4.0 = 0;                                                                  │
    CF2CTL0 = 0xC1;                                                             │
    CF2CTL1 = 0x05;                                                             │
    CF2CTL2 = 0x0F;                                                             ┘

    /* Initialize A/D output (ADC0) */                                          ┐
    ADA0M0 = 0x02;                      /* Continuous, Timer trigger mode    */ │
    ADA0M1 = 0x84;                      /* High-speed conversion mode        */ ├------ ③
    ADA0M2 = 0x01;                      /* Timer trigger mode 0 (INTTAA2CC0) */ │
    ADA0S  = 0x02;                      /* 2ch                               */ │
    ADMK   = 0;                         /* Enable interrupt                  */ ┘

    /* Initialize Interval Timer (TAA2) */                                      ┐
    TAA2CTL0 = 0x00;                    /* Count clock=fxx/2 (T=41.7ns)      */ │
    TAA2CTL1 = 0x00;                    /* Interval timer mode               */ ├------ ④
    TAA2IOC0 = 0x00;                    /* Output disabled                   */ │
    TAA2CCR0 = 0x05DB;                  /* Interval=62.5us                   */ │
    TAA2CCMK0 = 0;                      /* Enable interrupt                  */ ┘
```

```
    /* MIC level Set */
    P4.0  = 0;
    CF2TX = MIC_VOL;
    while( (CF2STR & 0x80) == 0x80);
    P4.0 = 1;

    __EI();                          /* maskable interrupt enable          */
}
```

⑤

① 再生処理のための初期化
録音した音声を再生する場合，再生処理のための初期化も必要です。説明は，2.2(3)を参照してください。

② CSI の初期化
音声デバイスの制御に用います。

レジスタ名	設定値	意味
PMC2	0x0C	P22 を SOF2 出力端子として使用する。 P23 を SCKF2 入出力端子として使用する。
PM2	0xF3	出力モード
P4.0	1	1 を出力
PM4.0	0	出力モード
CF2CTL0	0xC1	CSIF2 動作許可，送信動作許可，通信起動トリガ有効
CF2CTL1	0x05	通信クロック＝Fxx/64，マスタ・モード
CF2CTL2	0x0F	16 ビット転送モード

③ A/D 変換のための初期化
タイマ・トリガ・モードを利用して，一定周期毎に A/D 変換を行います。
ここでは，端子 ANI0 に対して A/D 変換を行っています。

このときの CPU 機能初期化内容は次の表のようになります。

レジスタ名	設定値	意味
ADA0M0	0x02	タイマ・トリガ・モード
ADA0M1	0x84	高速変換モード
ADA0M2	0x01	タイマ・トリガ・モード 0 （INTTAA2CC0 割り込み要求発生時）
ADA0S	0x02	2ch
ADMK	0	割り込み許可

④ インターバル・タイマの初期化
A/D 変換開始のタイミングとして，タイマ TMP2 の INTTP2CC0 を使用します。

タイマ名	動作モード	ｲﾝﾀｰﾊﾞﾙ間隔	ー	割り込み
TAA2	ｲﾝﾀｰﾊﾞﾙ･ﾀｲﾏ	62.5μs	ー	許可

このときの CPU 機能初期化内容は次の表のようになります。

レジスタ名	設定値	意味
TAA2CTL0	0x00	カウント・クロック＝fxx＝20MHz（T=50ns）
TAA2CTL1	0x00	インターバル・タイマ・モード
TAA2IOC0	0x00	出力禁止
TAA2CCR0	0x05DB	インターバル間隔＝1,499×41.7ns＝62.5μs
TAA2CCMK0	0	割り込み許可

⑤ マイクアンプの初期化
CSI を使ってマイクアンプの録音レベルを制御します。 CF2STR が通信終了を示すまで待ちます。

(4) 入力開始／終了処理

入力開始および終了時の周辺機能レジスタ設定の実装を示します。

入力開始／終了処理

```
void StartInput(void)
{
    /* Start A/D input (ADC0) */
    ADA0CE = 1;                         /* Start operation                 */   ------ ①

    /* Start Interval Timer (TMP2) */
    TAA2CE = 1;                         /* Start operation                 */   ------ ③
}

void StopInput(void)
{
    /* Stop Interval Timer (TMP2) */
    TAA2CE = 0;                         /* Stop operation                  */   ------ ④
    TAA2CCIF0 = 0;                      /* Clear interrupt request flag    */

    /* Stop A/D input (ADC0) */
    ADA0CE = 0;                         /* Stop operation                  */   ------ ②
}
```

備考　“StartInput()”は録音処理の開始時に呼び出します。
“StopInput()”は録音処理の終了時に呼び出します。

① A/D 入力開始
録音処理の開始時に設定します。

レジスタ名	設定値	意味
ADA0CE	1	A/D 変換動作許可

② A/D 入力停止
録音処理の終了時に設定します。

レジスタ名	設定値	意味
ADA0CE	0	A/D 変換動作停止
ADIF	0	割り込み要求フラグのクリア

③ インターバル・タイマの開始
録音処理の開始時に設定します。

レジスタ名	設定値	意味
TP2CE	1	動作許可（動作開始）

④ インターバル・タイマの停止
録音処理の終了時に設定します。

レジスタ名	設定値	意味
TP2CE	0	動作禁止（動作停止）

(5) 録音処理

圧縮で録音を行う際の処理を示します。

録音処理

```
void RecordSound(unsigned long sampleCount, unsigned char *adpcmData)
{
    unsigned long scnt;                     /* counter                  */
    unsigned short inputData;               /* input data               */
    short pcmCode;                          /* PCM code                 */
    unsigned char adpcmCode;                /* ADPCM code               */
    unsigned char byte;                     /* cache data               */

    /* Initialize ADPCM */
    adpcm_init(g_adpcmWork);                                          ---------- ①

    scnt = 0;
    while(scnt < sampleCount){              /* Loop of sample count     */
        if(g_inbufCnt > 0){                 /* If data exists in buffer */  ---------- ②

            /* Get input data from the input buffer */
            __DI();
            inputData = g_inbufData[g_inbufIdxR];
            g_inbufIdxR = (g_inbufIdxR+1)&(INBUF_SIZE-1);                 ---------- ③
            g_inbufCnt--;
            __EI();

            /* Convert input data to PCM code */
            pcmCode = (short)(inputData-0x8000);                          ---------- ④

            /* Encode PCM code to ADPCM code */
            adpcmCode = adpcm_l32_enc(pcmCode, g_adpcmWork);              ---------- ⑤

            /* Set ADPCM code */
            if(!(scnt & 0x01)){             /* Even number              */
                byte = adpcmCode;                                         ---------- ⑥
            }else{                          /* Odd number               */
                *adpcmData++ = (adpcmCode<<4)|byte;
            }

            scnt++;
        }else{
            /* Other jobs under recording sound */                        ---------- ⑦
        }
    }
}
```

① 一連の圧縮処理の前に，“adpcm_init()”を呼び出して，ADPCM-SP ライブラリの初期化を行います。
② リング・バッファにデータがない場合は，圧縮処理を行いません。
③ リング・バッファ“g_inbufData[]”から入力データを読み出します。
同時に，リング・バッファの読み出し位置“g_inbufIdxR”と格納データ数“g_inbufCnt”を更新します。
また，このリング・バッファ操作は，割り込みハンドラとの共用変数を複数アクセスするため，割り込み禁止状態（DI）で処理する必要があります。
④ 入力データ（A/D 変換結果）から PCM コードに変換します。
⑤ PCM コードを圧縮して ADPCM コードを取得します。4 ビット圧縮の場合，“adpcm_l32_enc()”を呼び出します。
（3 ビット圧縮の場合は，“adpcm_l24_enc()”を，2 ビット圧縮の場合は，“adpcm_l16_enc()”を呼び出します。）
⑥ ADPCM コードを圧縮データ格納領域に格納します。4 ビット圧縮の場合，4 ビットずつ格納します。
（3 ビット圧縮の場合は，3 ビットずつ，2 ビット圧縮の場合は，2 ビットずつ格納します。）
⑦ この条件成立時は，録音期間中に行う他のアプリケーション処理を行うことができます。

(6) A/D 変換割り込みハンドラ

A/D 変換割り込みハンドラを示します。
A/D 変換割り込みハンドラは 62.5μs 周期で実行されます。

A/D 変換割り込みハンドラ

```
#pragma interrupt INTAD     intad
__interrupt void intad(void)
{
    DAOCSO = ADAOCR2H;                                                    --------- ①

    if(g_inbufCnt < INBUF_SIZE){               /* If space exists in buffer  */  --------- ②
        g_inbufData[g_inbufIdxW] = ADAOCR2;
        g_inbufIdxW = (g_inbufIdxW+1)&(INBUF_SIZE-1);                     -------- ③
        g_inbufCnt++;
    }
}
```

① A/D 変換値をそのまま D/A のレジスタ設定値に設定することで，リアルタイムで入力音声を出力しています。
② 録音処理のループ内に他の処理を追加した場合は，録音周期に圧縮処理が間に合わなくなることもあります。その場合には“g_inbufCnt”が INBUF_SIZE になるため，ここでそれを判断することができます。
③ A/D 変換結果を読み出し，リング・バッファ“g_inbufData[]”に書き込みます。
同時に，リング・バッファの書き込み位置“g_inbufIdxW”と格納データ数“g_inbufCnt”を更新します。