ＡＤＳＰ３２４-１４１

12bit A/D & 16bit D/Aボード
ソフトウェア・ユーザーズ・マニュアル
ＡＤＳＰ６７４－００用

中部電機株式会社

目次

1. 概要

ADSP324-141 サポートソフトウェアは、ADSP324-141 を使用する為の基本機能を含んだ BIOS プログラム(A141_67BIOS)及び、それを用いたサンプルプログラムから構成されています。

2. 機能一覧

A141_67BIOS には以下の機能があります。

1) ADSP324-141 ボードの初期化
2) ソフトウェア同期の A/D & D/A 変換機能
3) タイマー同期の A/D & D/A 変換機能

3. 供給形態

A141_67bios はソースファイル及び、COFF ファイル形式のオブジェクト、ライブラリ形式で供給されています。オブジェクトリンク又はライブラリリンクのいずれかの方法で利用してください。ライブラリリンクにて利用する場合は、A141_67bios.h と A141_67bios.lib を C6x_C_DIR 環境変数の示すディレクトリにコピーしておけば簡単に利用することができます。

4. 供給ファイル一覧

Readme.txt	A141_67bios の簡単な説明が書かれています。
A141_67bios.c	A141_67bios のソースファイル
A141_67bios.h	A141_67bios を使用する為のヘッダファイル
A141_67bios.obj	A141_67bios のオブジェクトファイル
Timer.c	A141_67bios にて使用されているタイマ関連関数ソースファイル
Timer.h	Timer を使用する為のヘッダファイル
Timer.obj	Timer のオブジェクトファイル
A141_67bios.lib	A141_67bios のライブラリファイル
A141_67.cmd	A141_67bios を用いるためのコマンドファイル
Sample.c	A141_67bios を用いたサンプルプログラム

5. 関数一覧

○ 初期化関数

A67X_141init	ボードの初期化及びライブラリの初期化を行います

○ A/D 変換関数

A67X_141adsingle	指定 A/D チャンネルの A/D 変換を行います
A67X_141admulti	指定 A/D チャンネルの A/D 変換を行います
A67X_141adrepeats	指定 A/D チャンネルの A/D 変換を連続で行います
A67X_141adrepeatm	指定 A/D チャンネルの A/D 変換を連続で行います
A67X_141adstatus	A/D 連続変換終了確認

○ D/A 変換関数

A67X_141dasingle	指定 D/A チャンネルの D/A 変換を行います
A67X_141damulti	指定 D/A チャンネルの D/A 変換を行います
A67X_141darepeats	指定 D/A チャンネルの D/A 変換を連続で行います
A67X_141darepeatm	指定 D/A チャンネルの D/A 変換を連続で行います
A67X_141dastatus	D/A 連続変換終了確認

○ A/D & D/A 同期変換

A67X_141adasync	複数チャンネルの A/D & D/A 連続変換を行います
A67X_141adastatus	A/D & D/A 連続変換終了確認

6. 実装方法

１）COFF ファイル形式のオブジェクトリンク
カレント・ディレクトリに A141_67bios.obj、Timer.obj をコピーして、ユーザ・プログラムとリンクしてください。

A141_67bios.c、Timer.c をプロジェクトに追加します。
「Project」-「Add Files to Project」
A141_67bios.c、Timer.c を選択 → 開く

２）ライブラリリンク
環境変数 C6x_C_DIR の示すディレクトリに、A141_67bios.h と A141_67bios.lib をコピーすることにより、リンカの -l オプションでライブラリを指定してリンクして下さい。

環境変数 C6x_C_DIR の内容を、確認します。
C6x_C_DIR = C:¥ti¥c6000¥cgtools¥include ： C:¥ti¥c6000¥cgtools¥lib...
通常は、DSP の C コンパイラのライブラリが格納されているパスがセットされています。
上記の場合は
A03_67bios.h を　C:¥ti¥c6000¥cgtools¥include
A03_67bios.lib を C:¥ti¥c6000¥cgtools¥lib
へコピーします。
プログラムのリンク時に A141_67bios.lib をリンクして下さい。
なお、C6x_C_DIR 環境変数の設定については、TI の C コンパイラの取り扱い説明書を参照して下さい。

7. 関数詳細

関数名　　ボードの初期化及びライブラリの初期化

記述　　int　　A67X_141init(max, base);

引数　　int　　max;　　// 141 ボードの実装枚数
unsigned int　　base;　　// 141 ボードの初期化パラメータ

戻り値　　_ERR　　パラメータ異常
_NER　　正常に初期化されました

説明　　ADSP32X_141 の初期化(D/A の出力を 0[V]に設定)します。また、ライブラリの諸設定を行います。
ボード実装枚数の指定は 1～8 が指定可能です。
ボードのベースアドレスは、1 枚目のボードから 20h ステップで各ボードを設定し、最初のベースアドレスを指定してください。
初期化構造体の説明は、第 7 章・構造体の説明を参照して下さい。

使用例

```
#include        <A141_67bios.h>

#define         BD_BASE         0x3000000
#define         BD_MAX          8

void  main(void)
{
    A67X_141init(BD_MAX, BD_BASE);
}
```

関数名　　指定チャンネルの A/D 変換

記述　　int　　A67X_141adsingle(ch, data);

引数　　int　　ch;　　// チャンネル番号
　　float　　*data;　　// A/D データ格納ポインタ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定されたチャンネルを A/D 変換します。

使用例

```
#include <A141_67bios.h>

float           AD_DATA[16];

void  main(void)
{
    for( ;; ){
        A67X_141adsingle(0, AD_DATA);
        printf("CH0:AD_DATA = %2.4f[V]¥n", AD_DATA[0]);
    }
}
```

関数名　　指定チャンネルの D/A 変換

記述　　int　　A67X_141dasingle(ch, *data);

引数　　int　　ch;　　// チャンネル番号
　　float　　data;　　// D/A 変換データ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定されたチャンネルを D/A 変換します。

使用例

```
#include <A141_67bios.h>


void  main(void)
{
    A67X_141dasingle(0, 5.0);
}
```

関数名　　複数チャンネルの A/D 変換

記述　　int　　A67X_141admulti(ch, data);

引数　　int　　ch;　　// チャンネル番号
　　float　　*data;　　// A/D データ格納ポインタ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　複数チャンネルを A/D 変換します。

使用例

```
#include <A141_67bios.h>

int             i;
float           AD_DATA[16];

void  main(void)
{
    for( ;; ){
        A67X_141admulti(16, AD_DATA);
        for( i = 0; i < 16; i++ ){
            printf("CH%2d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
        }
    }
}
```

関数名　　複数チャンネルの D/A 変換

記述　　int　　A67X_141damulti(ch, data);

引数　　int　　ch;　　// チャンネル番号
　　float　　*data;　　// D/A データ格納ポインタ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　複数チャンネルを D/A 変換します。

使用例

```
#include <A141_67bios.h>


float           DA_DATA[12] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,
                               10.0, 10.0, 10.0};

void  main(void)
{
    for( ;; )
        A67X_141damulti(12, DA_DATA);
}
```

関数名　　指定チャンネルの A/D 連続変換

記述　　int　　A67X_141adrepeats(prod, ch, size, buff);

引数

```
float           prod;                   // 変換周期
int             ch;                     // 変換チャンネル番号
unsigned int    size;                   // 変換データ数
float           *buff;                  // A/D 変換データ格納ポインタ
```

戻り値

```
_ERR    異常終了
_NER    正常終了
```

説明　　指定されたチャンネルを A/D 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。

使用例

```
#include <A141_67bios.h>

int             i;
float           AD_DATA[32];

void  main(void)
{
    for( ;; ){
        A67X_141adrepeats(100.0, 0, 16, AD_DATA);
        while( A67X_141adstatus() ){}
        for( i = 0; i < 16; i++ ){
            printf("CH%d:%2.4f[V]¥n", i, AD_DATA[i]);
        }
    }
}
```

関数名　　指定チャンネルの D/A 連続変換

記述　　int　　A67X_141darepeats(prod, ch, size, buff);

引数

```
float           prod;                   // 変換周期
int             ch;                     // 変換チャンネル番号
unsigned int    size;                   // 変換データ数
float           *buff;                  // A/D 変換データ格納ポインタ
```

戻り値

```
_ERR    異常終了
_NER    正常終了
```

説明　　指定されたチャンネルを D/A 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。

使用例

```
#include <A141_67bios.h>

int             i;
float           a;
float           DA_DATA2[12 * 256];

void  main(void)
{
    a = PI2 / 64.0;
    for( i = 0; i < 256; i++ ){
        DA_DATA2[i] = 5 * sin(a * i);
    }
    for( ;; ){
        A67X_141darepeats(100, 0, 256, DA_DATA2);
        while( A67X_141dastatus() );
    }
}
```

関数名　　複数チャンネルの A/D 連続変換

記述　　int　　A67X_141adrepeatm(prod, ch, size, buff);

引数　　float　　prod;　　// 変換周期
int　　ch;　　// 変換チャンネル番号
unsigned int　　size;　　// 変換データ数
float　　*buff;　　// A/D 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　複数チャンネルを A/D 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。
データの配列を下記に示します。
buff + 0　　: 先頭チャンネルのデータ
buff + 15　　: 最終チャンネルのデータ

使用例

```
#include <A141_67bios.h>

int             i, j:
float           AD_DATA[16 * 32];

void  main(void)
{
    for( ;; ){
        A67X_141adrepeatm(100, 16, 4, AD_DATA);
        while( A67X_141adstatus() ){}
        for( i = 0; i < 4; i++ ){
            printf("%4d[usec]:¥n", i * 100);
            for( j = 0; j < 16; j++ ){
                printf("CH%2d: %2.4f[V]", j, AD_DATA[i * 16 + j]);
                if( j == 15 )
                    printf("¥n");
            }
        }
    }
}
```

関数名　　複数チャンネルの D/A 連続変換

記述　　int　　A67X_141darepeatm(prod, ch, size, buff);

引数　　float　　prod;　　// 変換周期
int　　ch;　　// 変換チャンネル番号
unsigned int　　size;　　// 変換データ数
float　　*buff;　　// D/A 変換データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　複数チャンネルを D/A 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。
データの配列を下記に示します。
buff + 0　　: 先頭チャンネルのデータ
buff + 15　　: 最終チャンネルのデータ

使用例

```
#include <A141_67bios.h>

int             i, j, k;
float           DA_DATA[16 * 32];

void  main(void)
{
    a = PI2 / 64.0;
    for( i = 0; i < 256; i++ ){
        j = i % 64;
        for( k = 0; k < 12; k++ ){
            DA_DATA2[i * 12 + k] = (j + k) * sin(a * j);
        }
    }
    for( i = 0; i < 210; i++ ){
        A67X_141darepeatm(100, 12, 256, DA_DATA2);
        while( A67X_141dastatus() ){}
    }
}
```

関数名　　A/D 連続変換終了確認

記述　　int　　A67X_141adstatus(void);

引数　　なし

戻り値　　1　　変換中
0　　変換終了

説明　　A/D 連続変換の終了を確認します。
この関数は、A67X_141adrepeats, A67X_141adrepeatm と共に使用します。

使用例

```
#include <A141_67bios.h>

float           AD_DATA[16 * 32];

void  main(void)
{
    A67X_141adrepeats(100, 16, 32, AD_DATA);
    while( A67X_141adstatus() );
}
```

関数名　　D/A 連続変換終了確認

記述　　int　　A67X_141dastatus(void);

引数　　なし

戻り値　　1　　変換中
0　　変換終了

説明　　A/D 連続変換の終了を確認します。
この関数は、A67X_141darepeats, A67X_141darepeatm と共に使用します。

使用例

```
#include <A141_67bios.h>

float           DA_DATA[16 * 32];

void  main(void)
{
    A67X_141darepeats(100, 16, 32, DA_DATA);
    while( A67X_141dastatus() );
}
```

関数名　　A/D & D/A 同期変換

記述　　int　　A67X_141adasync(prod, ad_size, ad_buff, ad_ch,
　　da_size, da_buff, da_ch);

引数

```
float          prod;          // 変換周期
unsigned int   ad_size;       // A/D 変換データ数
float          *ad_buff;      // A/D 変換データ格納ポインタ
int            ad_ch;         // A/D 変換チャンネル番号
unsigned int   da_size;       // D/A 変換データ数
float          *da_buff;      // D/A 変換データ格納ポインタ
int            da_ch;         // D/A 変換チャンネル番号
```

戻り値　　_ERR　異常終了
　　_NER　正常終了

説明　　複数チャンネルを A/D & D/A 変換します。
変換データ数は変換するデータサイズを指定します。
変換周期には、周期を[μSec]単位で指定します。
データの配列を下記に示します。

```
ad_buff + 0          : 先頭チャンネルのデータ(A/D)
ad_buff + 15         : 最終チャンネルのデータ(A/D)
da_buff + 0          : 先頭チャンネルのデータ(D/A)
da_buff + 15         : 最終チャンネルのデータ(D/A)
```

使用例

```
#include <A141_67bios.h>

int             i, j, k;
float           AD_BUFF[16 * 32], DA_BUFF[12 * 32];

void  main(void)
{
    for( i = 0; i < 256; i++ ){
        for( j = 0; j < 12; j++ )
            DA_DATA2[i * 12 + j] = (float)(j + 1) * i / 10;
    }

    for( ;; ){
        A67X_141adasync(100, 15, AD_DATA2, 16, 15, DA_DATA2, 12);
        while( A67X_141adastatus() );
        for( j = 0; j < 4; j++ ){
            printf("%4d[usec]:¥n", j * 100);
            for( k = 0; k < 16; k++ ){
                printf("CH%2d: %2.4f[V]", k, AD_DATA2[j * 16 + k]);
                if( k == 15 )
                    printf("¥n");
            }
        }
    }
}
```

関数名　　A/D & D/A 同期変換終了確認

記述　　int　　A67X_141adastatus(void);

引数　　なし

戻り値　　1　　変換中
0　　変換終了

説明　　A/D & D/A 連続変換の終了を確認します。
この関数は、A67X_141adsync と共に使用します。

使用例

```
#include <A141_67bios.h>

float   AD_BUFF[16 * 32];
float   DA_BUFF[12 * 32];

void  main(void)
{
    A67X_141adasync(100, 32, AD_BUFF, 16, 32, DA_BUFF, 12);
    while( A67X_141adastatus() );
}
```

8. 構造体の説明

構造体は、typedef を用いて<A13_67bios.h>の中で定義されています。

1) A/D & D/A ポートの定義

```
  typedef  struct
{

        unsigned  int       AD[16],          // A/D 入力ポート
                            DA[12],          // D/A 出力ポート
                            AD_BUSY,         // 変換中ポート
                            CTRL,            // 制御ポート
                            DMAD[3];         // 空き
                            INT_RST,         // 割り込みリセットポート
   }AO3_67BD_PORT;
```

9. ボード制御ソフトを書く上での注意

ボード制御ソフトをユーザーサイドで独自に作る場合においての注意点を説明します。

1) ベクタの使用

割り込みを複数のボードで使用する上で、ベクタ番号は重要な役割を持ちます。ベクタ番号の設定は、DSW104 で行うことができボード間で重複しないように設定します。

例)
1 枚目のボード DSW104-1 を ON、他は OFF
2 枚目のボード DSW104-2 を ON、他は OFF

このように設定しておくことにより、どのボードから割り込み要求が来たかを知ることができるようになります。

方法は、ベースアドレスの下位 20 ビットが 3fffc のアドレス(nnn3fffc)番地を読むことによって行います。()内の nnn は、ボードのベースアドレスの上位 12 ビットの設定です。
この事からわかる様に、割り込みを使用するボード全てのベースアドレスの上位 12 ビットは同一の設定である必要があります。

ベクトポートは割り込みが発生していない場合、下位 8 ビット全てが 1 です。割り込みが発生した場合、割り込み要求を出しているボードの DSW104 の ON 位置のビットが 0 になります。

以下に、ベクタを用いたプログラムを示します。

例) ボードが 2 枚実装されているものとし、1 枚目はベクタ番号 1(DSW104-1 を ON)、2 枚目はベクタ番号 2(DSW104-2 を ON)とします。

```
#define     BD_MAX          2                               // 実装ボード枚数
#define     BD_BASE         0x3000000                       // ボードのベースアドレス
#define     VECT_PORT(a)    (unsigned int *)(((unsigned int)(a) & 0xff00000) + 0x3fffc)

static      A06_67BD_PORT   *FST_BASE;                      // 1 枚目のボードアドレス
static      A06_67BD_PORT   *BD_BASE[BD_MAX];               // 各ボードのベースアドレス
static int  VECT_MASK = 0;                                  // ベクタマスク
int         *int_bd = (int*)0x303fffc;                      // 割り込みボード確認用

interrupt void BD1(void)
{
    printf("割り込みプログラム¥n");
}

interrupt void BD2(void)
{
    printf("割り込みプログラム¥n");
}
```

```
//=============================================
//          割り込み処理
//=============================================
interrupt void c_int90(void)
{
    int             int_no;
    unsigned int    vect, bd, bit;

    asm("           nop     2               ");     // 無効割り込み検査
    asm("           mvc     IFR, b3         ");
    asm("           mvk     0080h, b4       ");
    asm("           and     b4, b3, b0      ");
    asm("   [b0]    b       int_exit        ");
    asm("           nop     5               ");

    vect = ~(*VECT_PORT(FST_BASE) | (~VECT_MASK));   // 割り込み発生状況
    for( bd = 0, bit = 1; bd < BD_MAX; ++bd, bit <<= 1 ){// 各ボードの割り込みスキャン
        if( vect & bit ){                            // 有効割り込み確認
            if( int_bd & 0x0f == 0x0e )
                int_no = 1;
            else if( int_bd & 0x0f == 0x0d )
                int_no = 2;

            BD_BASE[bd]->INTR = 0;                   // 割り込みのリセット
            if( int_no == 1 )
                BD1();                               // 1 枚目のボードの処理
            else if( int_no == 2 )
                BD2();                               // 2 枚目のボードの処理
            }
    }
    asm(" int_exit:                         ");
}
```

2) ユーザソフトをアセンブラで記述する場合

拡張バスのメモリにデータを書き込む場合は “STB”“STH”命令は使用しないで下さい。以下に説明を示します。

STB 命令では 8 bit 単位で、STH 命令では 16 bit 単位でメモリの読み書きを行います。
しかし、拡張ボードにデータを書き込む場合は 32 bit(1 word)単位で実効する必要があります。

以上の様に、それぞれ扱うデータサイズが異なるため STB・STH 命令を使用した場合は不完全なデータになります。

3) 割り込み使用時

割り込みを使用する関数を実行するときは、ユーザプログラムにてグローバル割り込みレジスタ(GIE)を有効（Enable）設定してください。

例）

```
// GIE enable
    asm("       mvc         CSR, b0         ");
    asm("       or          1, b0, b0       ");
    asm("       mvc         b0, CSR         “);
```

・本マニュアルの内容は製品の改良のため予告無しに
変更される事がありますので、ご了承下さい。

中部電機株式会社

〒440-0004　愛知県豊橋市忠興３丁目２－８
TEL <0532>61-9566

FAX <0532>63-1081
URL　　: http://www.chubu-el.co.jp
E-mail : csg@chubu-el.co.jp

2002. 11　　第３版発行