

ADSP324-13

16bit A/D & 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.....	2
2. 機能一覧.....	2
3. 供給形態.....	2
4. 供給ファイル一覧.....	2
5. 関数一覧.....	3
5. 実装方法.....	3
6. 関数詳細.....	4
A67X_13init()	4
A67X_13adnom()	5
A67X_13danorm()	6
A67X_13adinput()	7
A67X_13dainput()	8
A67X_13adperiod1()	9
A67X_13daperiod1()	10
A67X_13trigger()	11
A67X_13adperiod2()	12
A67X_13multi()	13
7. 構造体の説明.....	14
8. データ配列.....	15
9. ボード制御ソフトを書く上での注意.....	15
1) ベクタの使用.....	15
2) アセンブラで記述する場合.....	17
3) 割り込み使用時.....	17

1. 概要

ADSP324-13 サポートソフトウェアは、ADSP324-13 を使用する為の基本機能を含んだ BIOS プログラム (A13_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。

A13_67bios は C で書かれています。ADSP324-13 を動作させる上で、大きなヒントになると思われる。

2. 機能一覧

A13_67bios には、次の機能があります。

- 1) ADSP324-13 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
- 4) トリガー待機機能
- 5) マルチプレクサ切替機能

3. 供給形態

A13_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、A13_67bios.h と A13_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	A13_67bios の簡単な説明が書かれています。
A13_67bios.c	A13_67bios のソースファイル
A13_67bios.h	A13_67bios を使用する為のヘッダファイル
A13_67bios.obj	A13_67bios のオブジェクトファイル
Timer.c	A13_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
A13_67bios.lib	A13_67bios のライブラリファイル
A13_67.cmd	A13_67bios を用いるためのコマンドファイル
Sample.c	A13_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_13init ボードの初期化及び、ライブラリーの初期化を行う。

- A/D 変換関数
A67X_13adinput 指定 A/D チャンネルの A/D 変換を行う。
A67X_13adperiod1 指定 A/D チャンネルの A/D 変換を行う。(タイマー同期)
A67X_13adperiod2 指定 A/D チャンネルの A/D 変換を行う。(外部同期)

- D/A 変換関数
A67X_13daoutput 指定 D/A チャンネルの D/A 変換を行う。
A67X_13daperiod1 指定 D/A チャンネルの D/A 変換を行う。(タイマー同期)

- トリガー関数
A67X_13trigger トリガー入力を待つ。

- データの正規化
A67X_13adnorm A/D データを浮動小数点形式に変換する。
A67X_13danorm 浮動小数点データを D/A データに変換する。

- マルチプレクサの切替
A67X_13multi マルチプレクサのチャンネル切替を行う。

5. 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 A13_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。

A13_67bios.c、Timer.c をプロジェクトに追加します。
「Project」 - 「Add Files to Project」
A13_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、A13_67bios.h と A13_67bios.lib をコピーすることにより、リンクの -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

A13_67bios.h を C:\ti\c6000\cgtools\include
A13_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 A13_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。

6. 関数詳細

関数名 ボードの初期化及び、ライブラリーの初期化

記述 `int A67X_13init(max, adrs, param);`

引数 `int max; // 13 ボードの実装枚数`
`A13_67BD_PORT *adrs; // 13 ボードのベースアドレス`
`A13_67PARAM *param; // 13 ボードの初期化パラメータ`

戻り値 `_ERR` 初期化異常終了
 `_NER` 初期化正常終了

説明 ADSP324-13 を初期化 (D/A 出力を 0[V] に設定) します。また、ライブラリーの諸設定を行います。
 ボード実装枚数の指定は、1~4 が設定可能です。
 ボードのベースアドレスは、1 枚目のボードから 10h ステップで連続して設定し、最初のボードの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 <A13_67bios.h>

#define BD_BASE            0x3000000
#define BD_MAX            4

*Port = (A13_67BD_PORT *)BD_BASE;
A13_67PARAM    Param[BD_MAX] =
{
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0}
},

void main(void)
{
    A67X_13init(BD_MAX, Port, &Param);
}
```

関数名 A/D データの正規化

記述 `int A67X_13adnorm(top, chc, dtc, src, dst);`

引数

<code>int</code>	<code>top;</code>	<code>//</code>	<code>先頭チャンネル番号</code>
<code>int</code>	<code>chc;</code>	<code>//</code>	<code>変換チャンネル数</code>
<code>int</code>	<code>dtc;</code>	<code>//</code>	<code>変換データ数</code>
<code>int</code>	<code>*src;</code>	<code>//</code>	<code>A/D 変換データ格納ポインタ</code>
<code>float</code>	<code>*dst;</code>	<code>//</code>	<code>正規化データ格納ポインタ</code>

戻り値

<code>_ERR</code>	異常終了
<code>_NER</code>	正常終了

説明

A/D 変換されたデータを、A/D の入力ゲインで設定されたデータを基に、浮動小数点形式の値に変換します。
 入力ゲインは、初期化時の値が使用されます。
 データの配列を下記に示します。データ配列については第 8 章・データ配列の説明を参照して下さい。

<code>src + 0</code>	:	先頭チャンネルのデータ
<code>src + chc - 1</code>	:	最終チャンネルのデータ
<code>src + chc</code>	:	次のデータ

使用例

```
#include <A13_67bios.h>

int      i;
int      AD_BUFF[2];
float    AD_DATA[2];

void main(void)
{
    for(;;){
        A67X_13adinput(0, 2, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 2; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

関数名 D/A データの正規化

記述 `int A67X_13danorm(top, chc, dtc, dst, src);`

引数

<code>int</code>	<code>top;</code>	// 先頭チャンネル番号
<code>int</code>	<code>chc;</code>	// 変換チャンネル数
<code>int</code>	<code>dtc;</code>	// 変換データ数
<code>float</code>	<code>*src;</code>	// D/A 変換データ格納ポインタ
<code>int</code>	<code>*dst;</code>	// 正規化データ格納ポインタ

戻り値

<code>_ERR</code>	異常終了
<code>_NER</code>	正常終了

説明

D/A 変換する浮動小数点形式のデータを、D/A 変換器の出力形式に変換します。
データの配列を下記に示します。

<code>src + 0</code>	: 先頭チャンネルのデータ
<code>src + chc - 1</code>	: 最終チャンネルのデータ
<code>src + chc</code>	: 次のデータ

使用例

```
#include <A13_67bios.h>

int          DA_BUFF[512]
float        DA_DATA[512]

void main(void)
{
    for(;;){
        A67X_13adinput(0, 2, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        A67X_13danorm(0, 2, 1, AD_DATA, DA_BUFF);
        A67X_13daoutput(0, 2, DA_BUFF);
    }
}
```

関数名 指定チャンネルの A/D 変換(ソフトウェア同期)

記述 `int A67X_13adinput(top, chc, buf);`

引数 `int top;` // 先頭チャンネル番号
`int chc;` // 変換チャンネル数
`int buf;` // 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範囲を A/D 変換します。
データの配列を下記に示します。
`src + 0` : 先頭チャンネルのデータ
`src + chc` : 次のデータ
`src + chc - 1` : 最終チャンネルのデータ

使用例

```
#include <A13_67bios.h>

int i;
int AD_BUFF[2];
float AD_DATA[2];

void main(void)
{
    for(;;){
        A67X_13adinput(0, 2, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 2; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```


関数名 指定チャンネルの D/A 変換(ソフトウェア同期)

記述 `int A67X_13daoutputt(top, chc, buf);`

引数 `int top;` // 先頭チャンネル番号
`int chc;` // 変換チャンネル数
`int buf;` // 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 指定されたチャンネル範囲を D/A 変換します。
データの配列を下記に示します。
`src + 0` : 先頭チャンネルのデータ
`src + chc` : 次のデータ
`src + chc - 1` : 最終チャンネルのデータ

使用例

```
#include <A13_67bios.h>

int DA_BUFF[512]
float DA_DATA[512]

void main(void)
{
    for(;;){
        A67X_13adinput(0, 2, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        A67X_13danorm(0, 2, 1, AD_DATA, DA_BUFF);
        A67X_13daoutput(0, 2, DA_BUFF);
    }
}
```

関数名 指定チャンネルの A/D 変換(タイマー同期)

記述 `int A67X_13adperiod1(top, chc, dtc, prod, buf);`

引数

<code>int</code>	<code>top;</code>	<code>//</code>	先頭チャンネル番号
<code>int</code>	<code>chc;</code>	<code>//</code>	変換チャンネル数
<code>int</code>	<code>dtc;</code>	<code>//</code>	変換データ数
<code>int</code>	<code>prod;</code>	<code>//</code>	変換周期
<code>int</code>	<code>*buf;</code>	<code>//</code>	A/D 変換データ格納ポインタ

戻り値

<code>_ERR</code>	異常終了
<code>_NER</code>	正常終了

説明

指定されたチャンネル範囲を A/D 変換します。
 変換データ数には、変換するデータサイズを指定します。
 変換周期には、変換間隔を μ 秒単位で指定します。
 データの配列を下記に示します。

<code>src + 0</code>	: 先頭チャンネルのデータ
<code>src + chc - 1</code>	: 最終チャンネルのデータ
<code>src + chc</code>	: 次のデータ(変換周期毎)

使用例

```
#include <A13_67bios.h>

int      i;
int      AD_BUFF[2*256]

void main(void)
{
    for(;;){
        A67X_13adperiod1(0, 2, 1, 100, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 2; i++)
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

関数名 指定チャンネルの D/A 変換(タイマー同期)

記述 `int A67X_13daperiod1(top, chc, dtc, prod, buf);`

引数 `int top;` // 先頭チャンネル番号
`int chc;` // 変換チャンネル数
`int dtc;` // 変換データ数
`int prod;` // 変換周期
`int *buf;` // 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 指定されたチャンネル範囲を A/D 変換します。
 変換データ数には、変換するデータサイズを指定します。
 変換周期には、変換間隔を μ 秒単位で指定します。
 データの配列を下記に示します。

`src + 0` : 先頭チャンネルのデータ
`src + chc - 1` : 最終チャンネルのデータ
`src + chc` : 次のデータ(変換周期毎)

使用例

```
#include <A13_67bios.h>

int DA_BUFF[2*256]

void main(void)
{
  for(;;){
    A67X_13adinput(0, 2, AD_BUFF);
    A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
    A67X_13danorm(0, 2, 1, AD_DATA, DA_BUFF);
    A67X_13daperiod1(0, 2, 2, 100, AD_BUFF);
  }
}
```

関数名 トリガー待ち

記述 `int A67X_13trigger (bd, lvl, slope);`

引数 `int bd; // ボード番号`
`double lvl; // トリガーレベル`
`int slope; // トリガースロープ`

戻り値 `_ERR` 異常終了
`_NER` 正常終了

説明 指定したボードで、トリガー監視をします。
トリガーレベルには、浮動小数点形式で±10[V]を指定します。
トリガースロープは、0なら立ち上がりスロープ、0以外で立ち下がりスロープです。
トリガーを検出するまで戻りません。

使用例

```
#include <A13?67bios.h>

void main(void)
{
    A67X_13trigger (0, 5.0, 0);
}
```

関数名 指定チャンネルの A/D 変換(外部同期)

記述 `int A67X_13adperiod2(top, chc, dtc, prod, buf);`

引数

<code>int</code>	<code>top;</code>	<code>//</code>	先頭チャンネル番号
<code>int</code>	<code>chc;</code>	<code>//</code>	変換チャンネル数
<code>int</code>	<code>dtc;</code>	<code>//</code>	変換データ数
<code>int</code>	<code>prod;</code>	<code>//</code>	変換周期
<code>int</code>	<code>*buf;</code>	<code>//</code>	A/D 変換データ格納ポインタ

戻り値

<code>_ERR</code>	異常終了
<code>_NER</code>	正常終了

説明

指定されたチャンネル範囲を A/D 変換します。
 変換データ数には、変換するデータサイズを指定します。
 変換周期には、変換間隔を μ 秒単位で指定します。
 通常は、TCLK0 の設定も行うので、ディップスイッチの設定で変換クロックの選択が TCLK0 なら、指定した変換周期で変換されます。外部クロックを選択した場合は、外部クロック入力端子に特定のクロックを入力しておいて下さい。
 データの配列を下記に示します。

<code>src + 0</code>	:	先頭チャンネルのデータ
<code>src + chc - 1</code>	:	最終チャンネルのデータ
<code>src + chc</code>	:	次のデータ(変換周期毎)

使用例

```
#include <A13_67bios.h>

int          i;
int          AD_BUFF[2]

void main(void)
{
    for(;;){
        A67X_13adperiod2(0, 2, 1, 100, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 2; i++)
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

関数名 マルチプレクサのチャンネル切替

記述 `int A67X_13multi (bd, ch);`

引数 `int bd;` // ボード番号
`int ch;` // マルチプレクサのチャンネル番号

戻り値 `_ERR` 異常終了
`_NER` 正常終了

説明 指定されたボード番号のマルチプレクサの入力チャンネルを、指定したチャンネルに切り替えます。ボード番号には、0~3, チャンネル番号には、0~7 を指定します。

注意 この関数を使用するには、マルチプレクサ(オプション)が実装されている必要があります。また、それに伴い DSW103-4 を ON(16ch 入力)に設定する必要があります。

使用例

```
#include <A13_67bios.h>

int i;
int AD_BUFF[2];
float AD_DATA[2];

void main(void)
{
    for(;;){
        A67X_13adinput(0, 2, AD_BUFF);
        A67X_13adnorm(0, 2, 1, AD_BUFF, AD_DATA);
        for( i = 0; i < 2; i++ )
            printf("CH%d:AD_DATA = %2.4f[V]¥n", i, AD_DATA[i]);
    }
}
```

7. 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[2],           // A/D 入力ポート
                   DMAD1[2],       // 空き
                   DA[2],          // D/A 出力ポート
                   DMAD2[2],       // 空き
                   AD_BUSY,        // 変換中ポート
                   CTRL,           // 制御ポート
                   TRIG_LVL,       // トリガーレベルポート
                   AD_GAIN,        // A/D 入力ゲインポート
                   INT_RST,        // 割り込みリセットポート
                   MULTI_CH,       // マルチプレクサ
                   DMAD[2];        // 空き
} A13_67BD_PORT;
```

2) 初期化構造体の定義

```
typedef struct
{
    unsigned int    AD_Gain[2];     // A/D 入力ゲイン
                                     (0: 1倍, 1: 2倍, 2: 4倍, 3: 8倍)
} A13_67PARAM;
```

8. データ配列

chc を 4 にした場合。

	データ 1	データ 2	
Ch0	src+0	src+chc	
Ch1	src+1	src+chc+1	
Ch2	src+2	src+chc+2	...
Ch3	src+chc-1	src+chc+3	

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ける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) 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