ＡＤＳＰシリーズ

ＡＤＳＰ３２４－１３

ソフトウェア・ユーザーズ・マニュアル

中部電機株式会社

目　次

１．概要

ＡＤＳＰ３２４－１３サポートソフトウェアは、ＡＤＳＰ３２４－１３を使用するための基本機能を含んだ
ＢＩＯＳプログラム　（Ａ１３ＢＩＯＳ）　および、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。
Ａ１３ＢＩＯＳはアセンブラで記述されており、制御用に使用する上で大きな手掛かりとなると思われます。

２．機能一覧

Ａ１３ＢＩＯＳ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）マルチプレクサ切り替え機能

３．供給形態

Ａ１３ＢＩＯＳはソースファイルおよび、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。ユーザプログラムとリンクして使用してください。
A13BIOS. H と A13BIOS. LIB を 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．供給ファイルの一覧

README.DOC	Ａ１３ＢＩＯＳの簡単な説明が書かれています
A13BIOS.INC	Ａ１３ＢＩＯＳを使用するためのヘッダファイル（アセンブラ用）
A13BIOS. ASM	Ａ１３ＢＩＯＳのソースファイル
A13BIOS. OBJ	Ａ１３ＢＩＯＳのオブジェクトファイル
A13BIOSC. H	Ａ１３ＢＩＯＳへのヘッダファイル（Ｃ用）
A13BIOSC.ASM	Ａ１３ＢＩＯＳをＣから利用するためのソースファイル
A13BIOSC.OBJ	Ａ１３ＢＩＯＳをＣから利用するためのオブジェクトファイル
A13BIOS.LIB	Ａ１３ＢＩＯＳを利用するためのライブラリファイル
DSP_SMP.C	Ａ１３ＢＩＯＳを使用したサンプル　（ＤＳＰ側）
DSP_SMP.OBJ	DSP_SMP.C のオブジェクトファイル
DSP_SMP.LNK	ＤＳＰサンプル用のリンク制御ファイル
DSP_SMP.CMD	ＤＳＰサンプル用のリンクコマンドファイル
DSP_SMP.MAP	ＤＳＰサンプルのマップファイル
DSP_SMP.OUT	ＤＳＰサンプルの実行ファイル
SAMPLE.C	DSP_SMP.OUT を使用したサンプルソース　（ホスト側）
SAMPLE.OBJ	SAMPLE.C のオブジェクトファイル
SAMPLE.EXE	SAMPLE.C の実行ファイル
SMPL.BAT	サンプルプログラムを実行形式にするための、バッチファイル

５．関数の一覧

○初期化関数

A13bios_init	ボードの初期化および、ライブラリーの初期化をおこなう

○Ａ／Ｄ変換関数

A13bios_adinput	指定Ａ／Ｄチャンネルの Ａ／Ｄ変換をおこなう	
A13bios_adperiod１	指定Ａ／Ｄチャンネルの Ａ／Ｄ変換をおこなう	(定周期)
A13bios_adperiod２	指定Ａ／Ｄチャンネルの Ａ／Ｄ変換をおこなう	(外部同期)

○Ｄ／Ａ変換関数

A13bios_daoutput	指定Ｄ／Ａチャンネルの Ｄ／Ａ変換をおこなう	
A13bios_daperiod１	指定Ｄ／Ａチャンネルの Ｄ／Ａ変換をおこなう	(定周期)

○トリガー関数

A13bios_triger	トリガー入力を待つ

○データの正規化

A13bios_adnorm	Ａ／Ｄデータを浮動小数点形式に変換する
A13bios_danorm	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り替え

A13bios_multi	マルチプレクサのチャンネル切り替え

６．関数詳細

関数名　ボードの初期化およびライブラリーの初期化

記述　int A13bios_init (max, adrs, param) ;

引き数
```
int              max ;      /*  １３ボードの実装枚数  */
A13bios_port     *adrs;     /*  １３ボードのベースアドレス  */
A13bios_param    *param;    /*  １３ボードの初期化パラメータ  */
```

戻り値

A13_ERR　初期化異常終了
A13_NER　初期化正常終了

説明　ＡＤＳＰ３２４－１３を初期化（Ｄ／Ａの出力を０Ｖに設定）します。また、ライブラリーの諸設定をおこないます。
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与えてください。
初期化構造体の説明は、７．構造体の説明（P14）を参照してください。

使用例

```
#include       <a13biosc. h>

#define        BD_MAX     4

A13bios_param  init_prm[BD_MAX] = {
   {0,0},
   {0,0},
   {0,0},
   {0,0}
} ;
A13bios_port   *port  =  ( A13bios_port *) 0x900180 ;

void  main ( )
}
A13bios_init (BD_MAX, port, init_prm) ;
} ;
```

関数名　Ａ／Ｄデータの正規化

記述　int A13bios_adnorm (top, chc, dtc, src, dst) ;

引き数
```
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             *src ;         /*  Ａ／Ｄ変換データポインタ  */
float           *dst ;         /*  正規化データポインタ  */
```

戻り値

A13_ERR	異常終了
A13_NER	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、初期化時のものが使用されます。
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
          以下省略
```

使用例

```
#include            <a13biosc.h>

int                 AD_BUF [512] ;
float               AD_DATA[512];

void  main ( )
 {
A13bios_adnorm (0,2,256,AD_BUF, AD_DATA) ;
};
```

関数名　Ｄ／Ａデータの正規化

記述　int A13bios_danorm (top, chc, dtc, src, dst) ;

引き数
```
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  */
float           *src ;          /*  Ｄ／Ａ変換データポインタ  */
int             *dst ;          /*  正規化データポインタ  */
```

戻り値

A13_ERR　異常終了
A13_NER　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。

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
以下省略

使用例

```
#include        <a13biosc.h>

int             DA_BUF [512] ;
float           DA_DATA [512] ;

void  main ( )
 {
A13bios_danorm (0,2,256,DA_DATA, DA_BUF) ;
} ;
```

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 A13bios_adinput (top, chc, buf);

引き数
```
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変換データポインタ  */
```

戻り値

A13_ERR	異常終了
A13_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。

```
buf + 0          :  先頭チャンネルのデータ
buf + chc － 1 :  最終チャンネルのデータ
          以下省略
```

使用例

```
#include          <a13biosc.h>

int               AD_BUF [2];

void  main ( )
 {
A13bios_adinput (0,2,AD_BUF);
};
```

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 A13bios_daoutput (top, chc, buf);

引き数
```
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変換データポインタ  */
```

戻り値

A13_ERR	異常終了
A13_NER	正常終了

説明　指定されたチャンネル範囲にD／A変換します。
データの配列を下記に示します。

buf + 0	:	先頭チャンネルのデータ
buf + chc − 1	:	最終チャンネルのデータ

以下省略

使用例

```
#include            <a13biosc.h>

int                 DA_BUF [2];

void  main ( )
 {
A13bios_daoutput (0,2,DA_BUF);
};
```

関数名　指定チャンネルのＡ／Ｄ変換　（タイマー同期）

記述　int A13bios_adperiod1 (top, chc, dtc, prod, buf) ;

引き数
```
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 ;         /*  Ａ／Ｄ変換データポインタ  */
```

戻り値

A13_ERR	異常終了
A13_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。

```
buf + 0          :  先頭チャンネルのデータ
buf + chc － 1   :  最終チャンネルのデータ
buf + chc        :  次のデータ（変換周期ごと）
          以下省略
```

使用例

```
#include          <a13biosc.h>

int               AD_BUF [2*256] ;

void  main ( )
 {
A13bios_adperiod1(0,2,256,100,AD_BUF) ;
};
```

関数名　指定チャンネルのD／A変換　(タイマー同期)

記述　int　A13bios__daperiod1 (top, chc, dtc, prod, buf) ;

引き数
```
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変換データポインタ  */
```

戻り値

A13_ERR	異常終了
A13_NER	正常終了

説明　指定されたチャンネル範囲へ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、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。

```
buf + 0          :  先頭チャンネルのデータ
buf + chc － 1   :  最終チャンネルのデータ
buf + chc        :  次のデータ (変換周期ごと)
          以下省略
```

使用例

```
#include        <a13biosc.h>

int             DA_BUF [2*256];

void  main (  )
 {
A13bios_daperiod1 (0,2,256,100,DA_BUF) ;
};
```

関数名　トリガー待ち

記述　int A13bios_triger (bd, lvl, slope) ;

引き数

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

戻り値

A13_ERR	異常終了
A13_NER	正常終了

説明　指定されたボードで、トリガー監視をします。
レベルの指定は浮動小数点形式で、±１０Ｖの指定が可能です。
ストロープの指定は、０なら立ち上がりストロープ、０以外なら立ち下がりストロープです。
トリガーを検出するまで、戻りません。

使用例

```
#include        <a13biosc.h>

void  main (  )
 {
A13bios_triger(0,2.5,0) ;
} ;
```

関数名　指定チャンネルのＡ／Ｄ変換　(外部同期)

記述　int　A13bios_adperiod2 (top, chc, dtc, prod, buf) ;

引き数
```
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 ;          /*  Ａ／Ｄ変換データポインタ  */
```

戻り値

A13_ERR	異常終了
A13_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ならば、変換周期で変換されます。
外部クロックを選択している場合、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ください。
データの配列を下記に示します。

```
buf  +  0        :  先頭チャンネルのデータ
buf  +  chc  －  １     :  最終チャンネルのデータ
buf  +  chc      :  次のデータ  (変換周期ごと)
            以下省略
```

使用例

```
#include        <a13biosc.h>

int             AD_BUF [2*256];

void  main (  )
 {
A13bios_adperiod2 (0,2,256,100,AD_BUF) ;
} ;
```

関数名　マルチプレクサのチャンネル切り替え

記述　int　A13bios_multi (int bd, int ch);

引き数
```
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

A13_ERR	異常終了
A13_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されたチャンネルへ切り替えます。ボード番号の指定範囲は０～３、 チャンネル番号の指定範囲は０～７です。

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

使用例

```
#include        <a13biosc.h>

int             AD_BUF [2];

main (  )
 {
  A13bios_multi (0,1);
  A13bios_adinput (0,2,AD_BUF);
}
```

７．構造体の説明

構造体定義は typedef を用いて、<a13biosc.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 long       AD [2] ,        /*  Ａ／Ｄポート  */
                                    RSV1 [2] ,      /*  予約１  */
                                    DA [2] ,        /*  Ｄ／Ａポート  */
                                    RSV2 [2] ,      /*  予約２  */
                                    AD_BUSY,        /*  Ａ／Ｄ変換中ポート  */
                                    CTRL,           /*  ボード制御ポート  */
                                    TRIG_LVL,       /*  トリガーレベル  */
                                    AD_GAIN,        /*  Ａ／Ｄ入力ゲイン  */
                                    INT_RESET,      /*  割り込みリセット  */
                                    MULTI_CH,       /*  マルチチャンネル設定  */
                                    RSV3 [2] ;      /*  予約３  */
                }    A13bios_port ;
```

２）初期化構造体の定義

```
typedef    struct  {
                int                 AD_Gain [2] ;   /*  Ａ／Ｄ入力ゲイン  */
                /*  0=1 倍, 1=2 倍, 2=4 倍, 3=8 倍  */
                }     A13bios_param ;
```

８．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）割り込み使用時の注意

１－１）　割り込みプログラム

ADSP324-13 ボードは、割り込みの発生を 100nSec のワンショット生成しています。
また、TMS320C31 の割り込みはレベル割り込みになっています。
このことにより、ADSP324-13 ボードで割り込みを使用すると、１回の割り込み条件で連続して２回の割り込みが発生してしまいます。
これを回避するためには、割り込み処理プログラムに下記のプログラムを挿入して回避してください。
ボードの割り込みは INT3 です。

```
int_entry :     push    st              ;   必要レジスタの退避
                push    dp
                push    r0
                ldi     if, r0          ; IF レジスタのコピー
                and     0008h, r0       ; INT3 位置のマスク
                bnz     int_exit        ; 多重割り込みなら回避

; 通常の割り込み処理

int_exit :      pop     r0              ;       レジスタの復帰
                pop     dp
                pop     st
                reti
```

１－２）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きたか知ることができるようになります。
知る方法は、ベースアドレスの下位１６ビット全てが１のアドレス（nnffffH）番地を読むことによって行います。
(　)内の nn は、ボードのベースアドレスの上位８ビットの設定です。
このことからも解るとおり、割り込み要求をだしているボードの DSW104 のＯＮ位置のビットが０になります。

下記にベクタを用いたプログラム例を示します。

例)

ボードは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。

```
BD1_VECT      ・set     0001h              ;  ボード１のベクタビット
BD2_VECT      ・set     0002h              ;  ボード２のベクタビット

              ・bss     bd_base, 1

VECT_MASK     ・word    0000ffffh

int_entry :     push    st                 ;  必要レジスタの退避
                push    dp
                push    r0
                push    ar0
                ldi     if,r0              ;  IF レジスターのコピー
                and     0008h,r0           ;  INT3 位置のマスク
                bnz     int_exit           ;  多重割り込みならば回避
                ldp     bd_base
                ldi     @bd_base,ar0       ;  ボードのベースアドレス
                ldp     VECT_MASK
                or      @VECT_MASK,ar0     ;  下位１６ビットのセット
                ldi     *ar0,r0            ;  ベクタの取得
                and     BD1_VECT,r0        ;  ボード１のベクタ番号
                callz   bd1_prog           ;  ボード１の処理を実行
                and     BD2_VECT,r0        ;  ボード２のベクタ番号
```

```
                callz   bd2_prog            ;   ボード２の処理を実行
                sti     r0,*ar0             ;   全てのボードの割り込み解除
int_exit :      pop     ar0                 ;   レジスタの復帰
                pop     r0
                pop     dp
                pop     st
                reti
```

bd_base は割り込み許可をする前に、設定されているものとします。
bd1_prog と bd2_prog は各ボードの割り込み処理プログラムであり、全てのレジスタを破壊しないものとします。

・本マニュアルの内容は製品の改良のため予告無しに
変更されることがありますので、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

ADSP324-13

ｿﾌﾄｳｪｱ・ﾕｰｻﾞｰｽﾞ・ﾏﾆｭｱﾙ

2004.11　　第３版発行