ＡＤＳＰ３２４－１１

ＰＩＯ ＆ カウンターボード
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

ADSP32X-11 サポートソフトウェアは、ADSP32X-11 を使用する為の基本機能を含んだ BIOS プログラム(A11_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。

A11_67bios は C で書かれている為実際の使用には速度的に問題がありますが、ADSP32X-11 を動作させる上で大きなヒントになると思われます。

2. 機能一覧

A11_67bios には次の機能があります。

1) ADSP32X-11 ボードの初期化
2) カウンターのデータ入力機能
3) カウンターのデータプリセット機能
4) 絶縁出力ポートへの出力機能
3) 絶縁入力ポートからの入力機能
4) 割り込み機能

3. 供給形態

A11_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。A11_67bios.h と A11_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	A11_67bios の簡単な説明が書かれています
A11_67bios.c	A11_67bios のソースファイル
A11_67bios.h	A11_67bios を使用する為のヘッダファイル
A11_67bios.obj	A11_67bios のオブジェクトファイル
A11_67bios.lib	A11_67bios のライブラリファイル
A11_67.cmd	A11_67bios を用いるためのコマンドファイル
Sample.c	A11_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_11init	ボードの初期化及びライブラリの初期化を行います

○ カウンター入出力関数

A67X_11cntin	指定ボードの指定カウンターからデータを入力します
A67X_11cntout	指定ボードの指定カウンターへデータをプリセットします

○ 絶縁入出力関数

A67X_11out	指定ボードの絶縁出力ポートへデータを出力します
A67X_11set	指定ボードの絶縁出力ポートの指定ビットをセットします
A67X_11reset	指定ボードの絶縁出力ポートの指定ビットをリセットします
A67X_11or	指定ボードの絶縁出力ポートのデータを OR します
A67X_11and	指定ボードの絶縁出力ポートのデータを AND します
A67X_11in	指定ボードの絶縁出力ポートのデータを取得します
A67X_11outm	指定ボードの絶縁出力ポートの状態をモニターします

○ 割り込み関数

A67X_11intentry	割り込み処理関数の登録／解除を行います

6. 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_11init(max, base);

引数　　int　　max;　　// 11 ボードの実装枚数
unsigned int　　base;　　// 11 ボードのベースアドレス

戻り値　　_ERR　　初期化異常終了
_NER　　初期化正常終了

説明　　ADSP32X_11 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 10h ステップで各ボードを設定し、最初のボードのアドレスを指定してください。

使用例

```
#include <A11_67bios.h>

#define BD_BASE          0x3000000        // ボードベース
#define BD_MAX           4

void  main(void)
{
    A67X_11init(BD_MAX, BD_BASE);
}
```

関数名　　指定ボードの指定カウンターデータの入力

記述　　int　　A67X_11cntin(bd, ch, data);

引数　　int　　bd;　　// ボード番号
　　int　　ch;　　// カウンターチャンネル番号
　　unsigned int　　*data;　　// 入力データ格納ポインタ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定ボードの指定カウンターチャンネルからカウンターデータを入力します。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    A67X_11cntin(0, 0, &data);
}
```

関数名　　指定ボードの指定カウンターデータのプリセット

記述　　int　　A67X_11cntout(bd, ch, data);

引数　　int　　bd;　　// ボード番号
int　　ch;　　// カウンターチャンネル番号
int　　data;　　// 出力データ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定ボードの指定カウンターへデータをプリセットします。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    A67X_11cntout(0, 0, 0x12345678L);
}
```

関数名　　指定ボードへの絶縁出力 A67X_11out()

記述　　int　　A67X_11out(bd,data);

引数　　int　　bd;　　// ボード番号(0～3)
　　int　　data;　　// 出力データ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定ボードの絶縁出力ポートへデータを出力します。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    A67X_11out(0, 0x1234);
}
```

関数名　　指定ボードの絶縁出力のビットセット

記述　　int　　A67X_11set(bd, bit);

引数　　int　　bd;　　// ボード番号
　　unsigned int　　bit;　　// ON するビット位置

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定ボードの絶縁出力ポートの指定位置ビットを ON にします。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    A67X_11set(0 ,0);
}
```

関数名　　指定ボードの絶縁出力のビットリセット

記述　　int　　A67X_11reset(bd, bit);

引数　　int　　bd;　　// ボード番号
　　unsigned int　　bit;　　// OFF するビット位置

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定ボードの絶縁出力ポートの指定位置ビットを OFF にします。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    A67X_11reset(0, 0);
}
```

関数名　　指定データの絶縁出力の OR

記述　　int　　A67X_11or(bd, data);

引数　　int　　bd;　　// ボード番号
　　int　　data;　　// OR するデータ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定データの絶縁出力ポートにデータを OR します。

使用例

```
#include        <A11_67bios.h>

void  main(void)
{
    A67X_11or(0, 0x5555);
}
```

関数名　　指定データの絶縁出力の反転 AND

記述　　int　　A67X_11and (bd, data);

引数　　int　　bd;　　// ボード番号
　　int　　data;　　// 反転 AND するデータ

戻り値　　_ERR　　異常終了
　　_NER　　正常終了

説明　　指定データの絶縁出力ポートに反転したデータを AND します。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    A67X_11and(0, 0x5555);
}
```

関数名　　指定ボードの絶縁入力データの入力

記述　　int　　A67X_11in(bd, data);

引数　　int　　bd;　　// ボード番号
unsigned int　　data;　　// 入力データ格納ポインタ

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定ボードの絶縁入力ポートからデータを入力します。

使用例

```
#include <A11_67bios.h>

void  main(void)
{
    unsigned int        data;

    A67X_11in(0, &data);
}
```

関数名　　割り込み関数

記述　　int　　A67X_11intentry(bd,intf,intslp,(*func)());

引数　　int　　bd;　　// ボード番号
int　　intf;　　// 割り込み無／有(0,1)
int　　intslp;　　// 割り込み立ち下がり／上がり(0,1)
void　　(*func)();　　// 割り込み関数

戻り値　　_ERR　　異常終了
_NER　　正常終了

説明　　指定ボードへの割り込み処理関数の設定を行います。
ボードへの割り込みが発生することで、設定された関数が自動的に実行されます。設定する割り込み処理関数へは、引数は渡せません。
尚、この関数は c_intnn ではなく、通常の関数名として下さい。

使用例

```
#include <A11_67bios.h>

void  intr_00()
{
    printf("割り込み処理関数です。プログラムを記述して下さい。¥n");
    return
}

void  main(void)
{
    A67X_11intentry(0, 1, 1, intr_00);
}
```

7. 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。

・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. 8　第２版発行