iXs Research Corp.
Cool Robotics

iMCs01 ソフトウェアマニュアル

Ver. 1.3

株式会社イクシスリサーチ

目　次

１．はじめに

本マニュアルは USB 接続 4ch モータコントローラ iMCs01 のソフトウェアに関する説明，および諸設定に関して記述してあります．なお，本マニュアルは Linux の基礎知識があるものとして説明がされております．各 Linux コマンド等に関しては，Linux の入門書等を参照してください．

本マニュアルは不定期に改訂されます．最新版は Web よりダウンロードしてください．

２．CH の機能

２．１　モータ出力ピン（CN101 ～ CN104）

		PWM モード		D/A モード	
		PWM ピン	BRK/DA ピン	PWM ピン	BRK/DA ピン
CN101	CH0	11bit	BRK	なし	8bit（※１）
CN102	CH1	9bit	BRK	なし	8bit（※１）
CN103	CH2	11bit	BRK	12bit（※２）	BRK
CN104	CH3	11bit	BRK	12bit（※２）	BRK

（※１）　LPF(Low-pass Filter)不要

（※２）　LPF(Low-pass Filter)を付けること

２．２　センサ入力ピン（CN105 ～ CN108）

		PWM モード		D/A モード	
		カウンタピン	A/D ピン	カウンタピン	A/D ピン
CN105	CH0	なし	10bit	16bit hard	10bit
CN106	CH1	16bit soft	10bit	16bit soft	10bit
CN107	CH2	16bit soft	10bit	16bit soft/hard	10bit
CN108	CH3	16bit soft	10bit	16bit soft	10bit

２．３　ディップスイッチ

Pin	機能	ON	OFF
1	ID　(0x00 ～ 0x1f)	1	0
2		1	0
3		1	0
4		1	0
5		1	0
6	PWM 極性	正論理	負論理
7	モード切替	D/A モード	PWM モード
8	ブレーキ切替	High	Low

３．制御出力の計算

３．１　制御式

制御出力 u は次式から 1[ms]毎に計算されます.

$$u = A - \frac{K_P}{K_{Px}}(x_d - x) - \frac{K_D}{K_{Dx}}(\dot{x}_d - \dot{x}) - \frac{K_I}{K_{Ix}}\sum_0^t (x_d - x) \quad (1)$$

ゲインの分子K_P, K_D, K_I は 16bit整数（-32768 から 32767）が，分母K_{Px}, K_{DX}, K_{IX} は 16bit の正整数（0 から 65535）が指定できます．目標値x_d, $\dot{x}_d$は 16bit整数（-32768 から 32767）が指定できます．オフセットAと制御出力uは 16bitの正整数（0 から 65535）です．目標位置指令値は 16bitで指定可能ですが，A/Dコンバータ（10bit）を使用する場合xの値は 0 から 32736 で，下位 5bitは常に 0 であることに注意してください．速度値$\dot{x}$は制御周期間（1[ms]）でのxの差分により計算されます．制御出力の計算は以下のように行います．

1. （ x_d - x ），（ $\dot{x}_d$ - $\dot{x}$ ），Σ_0^t（ x_d - x ）を計算
2. 誤差の積分値 Σ_0^t（ x_d - x ）が 16bitで表現できる範囲を超えている場合には 32767 あるいは-32768 とする.
3. ERx（32bit レジスタ）に A を代入.
4. K_Pを（ x_d - x ）に掛け，ERy（32bitレジスタ）に入れる.
5. ERyをK_{Px}で割り，ERyに入れる.
6. ERx から ERy を引く.
7. K_Dを（ $\dot{x}_d$ - $\dot{x}$ ）に掛け，ERyに入れる.
8. ERyをK_{DX}で割り，ERyに入れる.
9. ERx から ERy を引く.
10. K_Iを Σ_0^t（ x_d - x ）に掛け，ERyに入れる.
11. ERyをK_{IX}で割り，ERyに入れる.
12. ERx から ERy を引く.
13. ERx が 16bit の範囲を超えている場合には，65535 あるいは 0 とする.

14. 16bit のうち出力ポートのサポートする上位ビットのみを使って出力を行う.

内部では 32bit レジスタを使用しているので各項の計算では桁溢れを起こすことはありませんが, 項の差の計算（ERx - ERy）の際に桁溢れが起こらないよう注意して下さい.（ERx - ERy）の計算の際には桁溢れの検出を行っていません. ただし通常に指定するゲインでは, 最終出力が飽和しないようにすれば桁溢れは起こらないはずです.

３．２　エンドポイント

３．２．１　エンドポイント１

エンドポイント１はバルク転送で１パケットの大きさは 64 バイト, 方向は IN（H8→PC）です.パケットはエンドポイント 6 からパケットを送るよう指示された時のみ送られます. パケットの中身は Fig.1 の構造体で表されます. Short のバイトオーダはリトルエンディアン（インテルなど）です. 従ってインテル系の CPU では, そのまま扱うことが出来ます. H8 内部はビッグエンディアンですが, 転送前にリトルエンディアンに変換しています. A/D. D/A コンバータの値は左詰めです. 例えば A/D の値（10bit）を 0～1023 の値で扱いたい場合には, 6bit 右シフトしてください.

内部カウンタの値は 1[ms]毎に 1 上がり, 65535 の次は 0 になります. intmax, interval はソフトウェア 2 相カウンタルーチンの呼ばれる周期を示します.（現在の単位は 1/64 クロックですが, 周期が 255（255x64/20[MHz]=0.82[ms]）を超えている場合には正しく測定されません).

```
struct uin {
        unsigned short time;    /* 内部カウンタ（1[ms]周期）の値  */
        unsigned short ad[4];   /* A/D コンバータの値（10bit）*/
        short ct[4];            /* 2 相カウンタの値（16bit）*/
        unsigned short da[4];   /* D/A 出力の値（上位 8/12bit が有効）*/
        unsigned char din;      /* (未使用)*/
        unsigned char dout;     /* (未使用）*/
        unsigned short intmax;  /* ソフトウェアカウンタの最大周期*/
        unsigned short interval;/* ソフトウェアカウンタの最近の周期*/
        unsigned short magicno;

        char dmy[30];
};
```

Fig.1 エンドポイント 1, 5 の uin 構造体

３．２．２　エンドポイント２

エンドポイント 2 はバルク転送で 1 パケットの大きさは 64 バイト，方向は OUT（PC→H8）です．パケットの中身は次の構造体で表されます．

```
struct uout {
        struct scmd ch[4];
};
```

scmd 構造体は Fig.2 のようになっています．x，d は 16bit の指令値です．kp，kpx，kd，kdx，ki，kix は PID ゲインです．各制御ゲインの分母 kpx，kdx，kix に 0 は入れないでください．目標位置指令値は 16bit で指定可能ですが，A/D コンバータ（10bit）の値は 0 から 32736 で，下位 5bit は常に 0 であることに注意してください．

```
struct scmd {
        short x;                /* 目標位置*/
        short d;                /* 目標速度*/
        signed short kp;        /* 位置誤差ゲイン（分子）*/
        unsigned short kpx;     /* 位置誤差ゲイン（分母）*/
        signed short kd;        /* 速度誤差ゲイン（分子）*/
        unsigned short kdx;     /* 速度誤差ゲイン（分母）*/
        signed short ki;        /* 積分誤差ゲイン（分子）*/
        unsigned short kix;     /* 積分誤差ゲイン（分母）*/
};
```

Fig.2 scmd 構造体

設定例１：(全チャンネルに目標軌道が Sin 波の位置制御(PD 制御)を行う場合)

```
/*************************************************************
 struct uout ubof;
 int i,j=0;
 unsigned short a = 100.0 * sin( j * 3.14 / 655.35 ) + 512.0;
 a <<= 5;
 for( i=0; i<4; i++) {
        obuf.ch[i].x  = a;
        obuf.ch[i].d  = 0;
        obuf.ch[i].kp = 10;     /* P ゲイン 10.0 */
        obuf.ch[i].kpx= 1;
        obuf.ch[i].kd = 1;      /* D ゲイン 0.2 */
        obuf.ch[i].kdx= 5;
```

```
        obuf.ch[i].ki = 0;      /* I ゲイン 0 */
        obuf.ch[i].kix= 1;
 }
 j++;
**********************************************************/
```

ここで obuf はユーザー定義のエンドポイント 2 の uout 型構造体です.

３．２．３　エンドポイント５

エンドポイント 5 はバルク転送で 1 パケットの大きさは 64 バイト，方向は IN（H8→PC）です．パケットの内容はエンドポイント 1 と同じ（Fig.1）です．パケットは 1 パケット転送される度に次のパケットが用意されます．そのため，連続してデータを読み取ることが可能ですが，しばらくデータを転送しなかった場合（デバイスを open した直後など）には古い時刻のデータが読み出されることがあります．open 直後のパケットは破棄するなどの処理をしてください．理論上の最小転送間隔は 1[ms]ですが，FreeBSD 4.2-STABLE では転送間隔は 2[ms]となりました.

３．２．４　エンドポイント６

エンドポイント 6 はバルク転送で 1 パケットの大きさは 64 バイト，方向は OUT（PC→H8）です．パケットの中身は Fig.3 の構造体で表されます.

```
struct ccmd {
        unsigned char retval;
                        /* 次に内部カウンタが上がったときの値を EP1 から返す*/
        unsigned char setoffset;/* オフセットをセットするチャンネルの指定*/
        unsigned char setcounter;/* カウンタの値をセットするチャンネルの指定*/
        unsigned char resetint; /* 積分地をリセットするチャンネルの指定*/
        unsigned char selin;    /* counter(0) / ADC(1) の選択*/
        unsigned char dout;     /* 未使用*/
        unsigned short offset[4];/* オフセットの値*/
        short counter[4];       /* カウンタの値*/
        unsigned char selout;   /* PWM モード(0) / DA モード(1) の選択 */
        unsigned char wrrom;
        unsigned short magicno;
        unsigned char posneg;    /* PWM パルスの正/負 */
        unsigned char breaks;    /* break 出力 */
        char dummy[36];
};
```

Fig.3 エンドポイント 6 の ccmd 構造体

また以下が定義されています.

```
        /* ビットとチャンネルの対応 */
#define CH0     1
#define CH1     2
#define CH2     4
#define CH3     8

#define RETURN_VAL      1       /* retval */
#define SET_SELECT      0x80    /* 値変更フラッグ */
#define SET_POSNEG      0x80    /* 値変更フラッグ(PWM の正論理/負論理指定用) */
#define SET_BREAKS      0x80    /* 値変更フラッグ(ブレーキの Hi/Low 指定用) */
#define SET_CH2_HIN     0x40    /* ハードウェアカウンタ使用時の CH2 の指定*/
```

各設定値は，チャンネル 0，1，2，3 に対してビット 0，1，2，3（CH0，..,3）が相当します.

retval には次に内部カウンタが上がった時にエンドポイント 1 からパケットを転送する場合には，RETURN_VAL を，そうでない場合には 0 を指定します.

オフセットの値は制御式(1)の A の値で，チャンネルごとに 0～65535 の範囲で指定でき，中央値は 32767(0x7fff)となります.

設定例：(全チャンネルに 0x7fff を設定する場合)

```
/*****************************************************************
    cmd.offset[0] = cmd.offset[1] = cmd.offset[2] = cmd.offset[3] = 0x7fff;
*****************************************************************/
```

ここで cmd はユーザー定義のエンドポイント 6 の ccmd 型構造体です.

カウンタの値をチャンネルごとに 0～65535 の範囲で指定できます.iMCs01 は,setcounter がセットされた値からエンコーダの値をカウントします.

設定例：(全チャンネルに 0 を設定する場合)

```
/*****************************************************************
    cmd.counter[0] = cmd.counter[1] = cmd.counter[2] = cmd.counter[3] = 0;
*****************************************************************/
```

setoffset で，指定したチャンネルのオフセットの値を cmd.offset[i]（ i=0,1,2,3 ）の値にセットします.

設定例 1：(全チャンネルのオフセット値をセットする場合)

```
/*************************************************************
    cmd.setoffset = CH0 | CH1 | CH2 | CH3;
*************************************************************/
```

設定例 2：(全チャンネルのオフセット値をセットしない場合)

```
/*************************************************************
    cmd.setoffset = 0;
*************************************************************/
```

setcounter で，指定したチャンネルのカウンタの値を cmd.counter[i] (i=0,1,2,3)の値にセットします.

設定例 1：(全チャンネルのカウンタ値をセットする場合)

```
/*************************************************************
    cmd.setcounter = CH0 | CH1 | CH2 | CH3;
*************************************************************/
```

設定例 2：(全チャンネルのカウンタ値をセットしない場合)

```
/*************************************************************
    cmd.setcounter = 0;
*************************************************************/
```

resetint は，積分値のリセットを行うチャンネルの指定をビット(指定する場合には 1，そうでない場合には 0) で指定します.

設定例：(全チャンネルをリセットする場合)

```
/*************************************************************
    cmd.resetint   = CH0 | CH1 | CH2 | CH3;
*************************************************************/
```

selin で，使用するセンサ入力をチャンネルごとに指定できます．エンコーダ入力を使用する場合は 0 を，アナログ入力を使用する場合は 1 を指定します.

設定例 1：(全チャンネルをアナログ入力に設定する場合)

```
/************************************************************
    cmd.selin = SET_SELECT | CH0 | CH1 | CH2 | CH3;
************************************************************/
```

設定例 2：(全チャンネルをエンコーダ入力に設定する場合)

```
/************************************************************
    cmd.selin = SET_SELECT;
************************************************************/
```

設定例 3：(CH0, CH1 をエンコーダ入力に, CH2, CH3 をアナログ入力に設定する場合)

```
/************************************************************
    cmd.selin = SET_SELECT | CH2 | CH3;
************************************************************/
```

selout で. 出力信号の種類をチャンネルごとに指定できます. PWM 出力の場合は, 1 を, アナログ出力の場合は, 0 を指定します.

設定例 1：(全チャンネルを PWM 出力に設定する場合)

```
/************************************************************
    cmd.selout = SET_SELECT | CH0 | CH1 | CH2 | CH3;
************************************************************/
```

設定例 2：(全チャンネルをアナログ出力に設定する場合)

```
/************************************************************
    cmd.selout = SET_SELECT;
************************************************************/
```

posneg で, PWM 信号の極性を逆転することが出来ます. 正論理出力をする場合は, 1 を, 負論理出力をする場合は 0 を指定します.

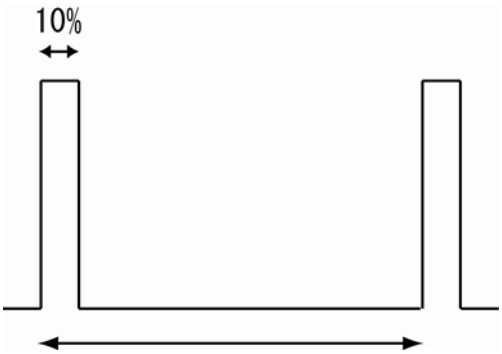
設定例 1：(全チャンネルを正論理出力に設定する場合)

```
/************************************************************
    cmd.posneg = SET_POSNEG | CH0 | CH1 | CH2 | CH3;
************************************************************/
```

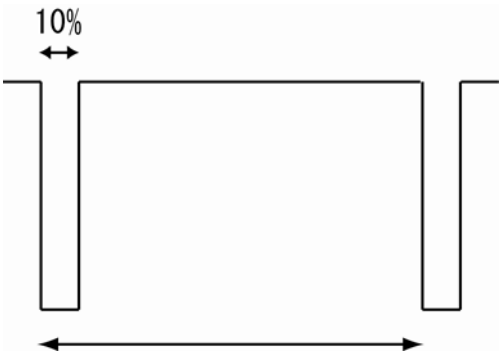

設定例 2：(全チャンネルを負論理出力に設定する場合)

```
/************************************************************
    cmd.posneg = SET_POSNEG;
************************************************************/
```

例えば 10%デューティーの PWM 信号を出力する場合，正論理(1)の場合，



のようになり，負論理(0)の場合，



のようになります.

Break で，ブレーキ信号の Hi，Low を指定することが出来ます．Hi を出力する場合は，1 を，Low を出力する場合は 0 を指定します.

設定例 1：(全チャンネルに Hi を出力する場合)

```
/************************************************************
    cmd.breaks = SET_BREAKS | CH0 | CH1 | CH2 | CH3;
************************************************************/
```

設定例 2：(全チャンネルに Low を出力する場合)

```
/************************************************************
    cmd.breaks = SET_BREAKS;
************************************************************/
```

ブレーキの ON/OFF とブレーキ信号の Hi/Low は，モータドライバに依存しますので，モータドライバのマニュアルを参照してください.

３．３　USB ドライバについて

３．３．１　デバイスのオープン，クローズ

USB デバイスをオープンするためには，プログラム内で

```
int fd;
fd = open( “/dev/urbtc0” ,O_RDWR);
```

と記述します．デバイスのオープンに失敗したときは，戻り値-1 が返ります．クローズするためには

```
close(fd)
```

とします.

プログラムは以下のようになります.

```
/*****************************************************************
  char *dev = "/dev/urbtc0";
  if (argc>1) dev = argv[1];

  if ((fd = open(dev, O_RDWR)) == -1) {
    fprintf(stderr, "%s: Open error¥n", dev);
    exit(1);
  }
*****************************************************************/
```

３．３．２　iMCs01 からデータを取り込む

iMCs01 からのデータを連続して取り込むには，まず

```
ioctl(fd,URBTC_CONTINUOUS_READ);
```

を実行しておく必要があります．（一度実行すれば，変更があるまで有効）その後，

```
read(fd, &buf, sizeof(buf));
```

で値を取得します.ここで buf はユーザー定義のエンドポイント 1,5 の uin 型構造体です.

プログラムは以下のようになります.

```
/*****************************************************************
  struct uin buf;

  if (ioctl(fd, URBTC_CONTINUOUS_READ) < 0){
    fprintf(stderr, "ioctl: URBTC_CONTINUOUS_READ error¥n");
    exit(1);
  }
```

```
  if (read(fd, &buf, sizeof(buf)) != sizeof(buf)) {
    fprintf(stderr, "Warning: read size mismatch");
    continue;
  }
*************************************************************/
```

３．３．３　iMCs01 に初期化データ書き込む

iMCs01 に初期化データ（エンドポイント 6 の ccmd）を書き込むには，まず

```
ioctl(fd,URBTC_COUNTER_SET);
```

を実行しておく必要があります．（一度実行すれば，変更があるまで有効）その後，

```
write(fd, &cmd, sizeof(cmd));
```

で値を書き込みます．ここで cmd はユーザー定義のエンドポイント 6 の ccmd 型構造体です．
プログラムは以下のようになります．

```
/*************************************************************
  struct ccmd cmd;

  if (ioctl(fd, URBTC_COUNTER_SET) < 0){
    fprintf(stderr, "ioctl: URBTC_COUNTER_SET error¥n");
    exit(1);
  }
  if (write(fd, &cmd, sizeof(cmd)) < 0) {
    fprintf(stderr, "write error¥n");
    exit(1);
  }
*************************************************************/
```

３．３．４　iMCs01 に制御データを書き込む

iMCs01 に制御データ（エンドポイント 2 の scmd）を書き込むには，まず

```
ioctl(fd,URBTC_DESIRE_SET);
```

を実行しておく必要があります．（一度実行すれば，変更があるまで有効）その後，

```
write(fd, &obuf, sizeof(obuf));
```

で値を書き込みます．ここで obuf はユーザー定義のエンドポイント 2 の uout 型構造体です．
プログラムは以下のようになります．

```
/*************************************************************
  struct uout obuf;
  if (ioctl(fd, URBTC_DESIRE_SET) < 0){
```



```
    fprintf(stderr, "ioctl: URBTC_DESIRE_SET error¥n");
    exit(1);
  }
  if (write(fd, &obuf, sizeof(obuf)) < 0) {
    printf("write err¥n");
    break;
  }
*************************************************************/
```

３．３．５　LITTLE_ENDIAN, BIG_ENDIAN について

コンピュータは 2 バイト以上のデータを扱う際に 1 バイトごとに分割して処理しますが，これを最下位のバイトから順番に記録/送信する方式をリトルエンディアン(LITTLE ENDIAN)と呼び，最上位のバイトから順番に記録/送信する方式をビッグエンディアン(BIG ENDIAN)と呼びます．Intel 系のプロセッサはリトルエンディアン，Motorola 系のプロセッサはビッグエンディアンのため，PC から送信する際に，データの上位バイトと下位バイトの入れ替えの必要が生じる場合があります．

iMCs01 は LITTLE ENDIAN 形式でデータを扱っているため，Motorola 系のプロセッサを持つコンピュータと接続する場合，データの入れ替えが必要です．例えば，初期化データ(ccmd 型構造体)の offset に 0x7fff を代入する場合，以下のような記述になります．

```
/*************************************************************
  unsigned char offset = 0x7fff;

#if __BYTE_ORDER == __LITTLE_ENDIAN
  cmd.offset[0] = offset;
#else
  cmd.offset[0] = (0xff & offset)<<8 | (0xff00 & offset)>>8;
#endif
*************************************************************/
```

ただし，使用する PC が，どちらか一方に決まっている場合は，どちらか一方を記述するだけで正しく処理されます．

３．４　iMCs01 側のプログラムについて

３．４．１　ソフトウェアカウンタの実装

4 逓倍の 2 相カウンタの，カウントのアップダウンの条件は Table 3，4 のようになります．これをエッジ入力がない場合にソフトウェアでエッジを検出するときの入力とカウントの対応は，Table 5 のようになります．ブール代数を用いても簡単化できないので，x を 0 とした表をプログラム内に持ち，1 時刻前と現在のエンコーダの値から表を読み，0/+1/-1 を加算することでソフトウェアカウンタを実現しています．なお．ソフトウェアカウンタ

には，カウントできる速度に上限があります．4000pps 以上の速度でパルスが入るシステムの場合は，ソフトウェアカウンタではなく，ハードウェアカウンタを用いてください．

TCLKA	L	↑	H	↓
TCLKB	↑	H	↓	L

Table 3 カウントアップ条件

TCLKA	↓	L	↑	H
TCLKB	H	↓	L	↑

Table 4 カウントダウン条件

A(t-1)	B(t-1)	A(t)	B(t)	カウント
L	L	L	L	0
L	L	L	H	+1
L	L	H	L	-1
L	L	H	H	x(あり得ない)
L	H	L	L	-1
L	H	L	H	0
L	H	H	L	x(あり得ない)
L	H	H	H	+1
H	L	L	L	+1
H	L	L	H	x(あり得ない)
H	L	H	L	0
H	L	H	H	-1
H	H	L	L	x(あり得ない)
H	H	L	H	-1
H	H	H	L	+1
H	H	H	H	0

Table 5 ソフトウェアエッジ検出による 2 相カウンタのカウント条件

３．４．２　ディップスイッチの設定

iMCs01 上のディップスイッチの Pin1～5 の操作により iMCs01 に固有の ID 番号を振ることが出来ます．全てのピンを 0 とすることで ID は 0x00（0）になり，全てのピンを 1 にすることで ID は 0x1f（31）になります．

ID	Pin				
	1	2	3	4	5
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	1	0	0	0
4	0	0	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮
30	0	1	1	1	1
31	1	1	1	1	1

Pin6 の操作により PWM 信号の極性を逆転することが出来ます．このピンの状態は，その後のプログラムで変更することが可能です．

Pin7 の操作により出力形態のモードを切り替えることが出来ます．D/A モード(Pin7 = 1)の場合，ch0 および ch1 のモータ出力コネクタの 3 ピンが DA ポートとなり，256 段階の分解能(8bit)でアナログ出力することが出来ます．また，ch2，ch3 に関しては，モータ出力コネクタの 2 ピンに Low Pass Filter を付けることで，アナログ信号になります．PWM モード(Pin7 = 0)の場合，ch0，ch1，ch2 および ch3 のモータ出力コネクタの 2 ピンは全て PWM 信号が出力され，モータ出力コネクタの 3 ピンは全てブレーキ信号になります．また，このピンの状態は，その後のプログラムで変更することが可能です．

Pin8 の操作により電源投入時でのブレーキの状態を指定することが出来ます．High(Pin8 = 1）の場合，電源投入時に BRK ピンに 1(High)が出力されます．Low(Pin8 = 0)の場合，電源投入時に BRK ピンに 0(Low)が出力されます．また，このピンの状態は，その後のプログラムで変更することが可能です．

３．４．３　H8 について

H8 のタイマ機能は次のように使っています．

- 16bit timer 0：PWM 出力 0
- 16bit timer 1：PWM 出力 1
- 16bit timer 2：位相係数カウンタ
- 8bit timer 0, 1：1[ms]タイマ（カスケード接続）
- 8bit timer 2：1000 クロックタイマ（ソフトウェアカウンタ呼び出し）
- 8bit timer 3：1/64 クロックカウンタ（ソフトウェアカウンタ呼び出しのタイミング測定）

WDT 機能は使っていません．

４．プログラムの実行

本章では，LINUX（Kernel 2.4 以上）で iMCs01 を操作する方法を説明します．以下の操作は全て root 権限で行ってください．なお，以下の説明では，コマンドラインからの入力は黒背景にしてあり，ユーザーモードでの入力は「$」で，ルートモードでの入力は「#」で記述されています．

```
$ su -l
```

４．１　USB デバイスの登録

USB デバイスを登録します．本操作は，各 PC において最初の 1 回だけ行います．
コマンドライン上で以下のように入力してください．

```
# mknod /dev/urbtc0 c 180 100
# chmod 666 /dev/urbtc0
```

４．２　USB ドライバのロード

iMCs01 は，USB マウス等，他の USB 機器との併用は出来ません．既に他の USB 機器を接続されている場合は，それらの USB 機器を外してください．また，hid のドライバが入っている場合は，以下のようにしてドライバを削除してください．

```
# rmmod hid
```

また，USB ドライバ urbtc.o をロードするに当たり，UHCI(Universal Host Controller Interface)がロードされている必要があります．ロードされていない場合は，以下のようにしてドライバをロードしてください．

```
# insmod uhci
```

urbtc のファイルがあるディレクトリに移動し，USB のモジュールをロードします． まだ iMCs01 を USB ポートに接続しないでください．

```
# cd /home/user1/urbtc/
# make
# insmod urbtc.o
```

ここで，正常にモジュールがロードされているかを確認します．

```
# lsmod
```

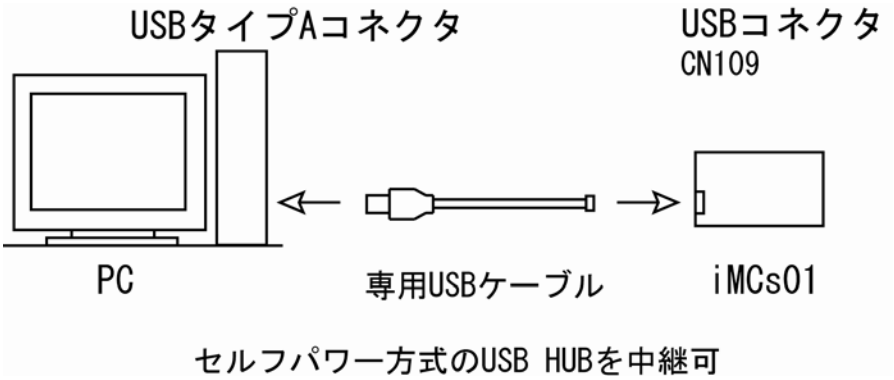
と入力し，

```
Module                  Size  Used by
urbtc                   7360   0  (unused)
```

と表示されることを確認してください．

４．３　ボードの接続

ボードを接続します．



接続後,

```
# dmesg
```

と入力し,

```
usb.c: registered new driver urbtc
urbtc.c: H8 based USB motor controller driver v0.1
hub.c: USB new device connect on bus1/2, assigned device number 2
urbtc.c: USB robot controller now attached to urbtc0
```

と表示されることを確認してください. また, HUB を中継する場合は, HUB の接続後 (HUB には iMCs01 をまだ接続しないでください),

```
# dmesg
```

と入力し,

```
ub.c: USB new device connect on bus1/2, assigned device number 3
hub.c: USB hub found
hub.c: 4 ports detected
```

と表示されることを確認してください.

次に iMCs01 が正しく認識されているかを確認するために,

```
# make
# ./h8test
```

と入力し,

```
Vendor xxxxxxxx
Product xxxxxxxx
read status 0
read status 1
write status 3
write status 3
```

と表示されることを確認してください.

４．４　終了処理

最後に, iMCs01 を USB ポートから取り外す際, dmesg で,


```
usb.c: USB disconnect on device 2
urbtc.c: urbtc0 now disconnected
```

と表示されることを確認してください．その後，

```
# rmmod urbtc
```

と入力し，USB ドライバを解放します．

４．５　プログラムの実行

各サンプルプログラムを実行します．添付の CD-ROM の内容を適当な場所にコピーして使用してください．全てのサンプルプログラムは iMCs01 と弊社モータドライバ iMDs03 とを接続した場合について書かれています．なお，iMCs01 に付属するドライバ，サンプルソースの Makefile の INCLUDE は

```
/*************************************************************
INCLUDE= /usr/src/linux/include
*************************************************************/
```

となっています．RedHat7.3，RedHat8.0 等を使用して，コンパイルエラーが出る場合は，

```
/*************************************************************
INCLUDE= /usr/src/linux-2.4/include
*************************************************************/
```

のように，Path を変更してみてください．

４．５．１　センサ値の取得

サンプルプログラムを実行します．サンプルで添付されているセンサ読込みプログラム uread を実行する場合は，

```
# make uread
# ./uread
```

と入力します．

```
32479 511 511 511 511 0 0 0 0 127 127 65407 65407 30 0 21 6 0
32480 511 511 511 511 0 0 0 0 127 127 65407 65407 30 0 21 6 0
32481 511 511 511 511 0 0 0 0 127 127 65407 65407 30 0 21 6 0
```

のような値が表示されます．

左から，time，ad[0]，ad[1]，ad[2]，ad[3]，ct[0]，ct[1]，ct[2]，ct[3]，da[0]，da[1]，da[2]，da[3]，din，dout，intmax，interval，magicno を表しています．

４．５．２　モータの制御(オープンループでモータを回す)

サンプルプログラムを実行します．サンプルで添付されている無制御プログラム urobot_open_loop を実行する場合は，

```
# make urobot_open_loop
# ./urobot_open_loop
```

と入力します．

４．５．３　モータの制御(ポテンショメータを用いた位置制御)

サンプルプログラムを実行します．サンプルで添付されている AD フィードバック位置制御プログラム urobot_ad を実行する場合は，

```
# make urobot_ad
# ./urobot_ad
```

と入力します．本サンプルプログラムは，アナログ入力値(32767)を中心に sin 波を出力しています．

本サンプルプログラムの仕様は以下のとおりです．

チャンネル	全チャンネル共通
出力	PWM 出力
入力	アナログ入力
オフセット	0x7fff
ブレーキ	解除
PWM 波形の論理	正論理

４．５．４　モータの制御(エンコーダを用いた位置制御(ソフトウェアカウンタ使用))

サンプルプログラムを実行します．サンプルで添付されているエンコーダフィードバック位置制御プログラム urobot_enc を実行する場合は，

```
# make urobot_enc
# ./urobot_enc
```

と入力します．本サンプルプログラムは，初期状態を中心に sin 波を出力しています．

本サンプルプログラムの仕様は以下のとおりです．

チャンネル	CH0, CH1,CH2
出力	PWM 出力
入力	エンコーダ入力
オフセット	0x7fff
ブレーキ	解除
PWM 波形の論理	正論理

４．５．５　モータの制御(エンコーダを用いた位置制御(ハードウェアカウンタ使用))

サンプルプログラムを実行します．サンプルで添付されている 1ch エンコーダフィードバック位置制御プログラム urobot_enc を実行する場合は，

```
# make urobot_1ch_enc
# ./urobot_1ch_enc
```

と入力します．本サンプルプログラムは，初期状態を中心に sin 波を出力しています．

本サンプルプログラムの仕様は以下のとおりです．

チャンネル	-
入力チャンネル	CH0(CN105)
出力チャンネル	CH2(CN103)
制御式	CH2 の制御式を使用
出力	PWM 出力
入力	エンコーダ入力
オフセット	0x7fff
ブレーキ	解除
PWM 波形の論理	正論理

ハードウェアカウンタを使用した 1ch エンコーダフィードバック位置制御では，結線が特殊ですので，注意して接続してください．また，selin，selout の指定の方法は，下記のように記述してください．これにより自動的にハードウェアカウンタが選択されます．

```
/*************************************************************
    cmd.selin = SET_SELECT | SET_CH2_HIN;
    cmd.selout = SETSELECT | CH2:
*************************************************************/
```

なお，各種制御ゲイン，目標値は，CH2 のパラメータに設定します．

４．５．６　モータの制御(iMCs01 の複数台同時接続)

2 台目以降のコントローラを接続する場合は，以下のように接続する数だけ USB ドライバを追加します．本操作は，各 PC において最初の 1 回だけ行います．

```
# mknod /dev/urbtc1 c 180 101
# chmod 666 /dev/urbtc1
```

ここで，180 は USB ドライバのメジャーNo，101 はマイナーNo です．

3 台同時接続の場合は以下のようになります．

```
# mknod /dev/urbtc1 c 180 101
# chmod 666 /dev/urbtc1
```

```
# mknod /dev/urbtc2 c 180 102
# chmod 666 /dev/urbtc2
```

続いてサンプルプログラム（2 台接続時）を実行します．サンプルで添付されている 2 台同時接続無制御プログラム urobotm を実行する場合は，

```
# make urobotm
# ./urobotm
```

と入力します．本サンプルプログラムは，2 枚の iMCs01 の全チャンネルに初期状態を中心に sin 波を出力しています．

本サンプルプログラムの仕様は以下のとおりです．

ID	全 ID 共通
チャンネル	全チャンネル共通
出力	PWM 出力
入力	アナログ入力
オフセット	0x7fff
ブレーキ	解除
PWM 波形の論理	正論理

４．５．７　Makefile の作り方

サンプルプログラム以外のプログラムを構築した場合，そのコンパイルは Makefile を書き換えることで，簡単になります．下記の例を参考に Makefile を書き足してください．

例）robot_ctrl.c というプログラムを作成した場合

```
/*****************************************************************
robot_ctrl: robot_ctrl.c
        $(CC) $(CFLAGS) -o $@ $< -lm
*****************************************************************/
```

Makefile を書き換えた後，robot_ctrl を実行する場合は，

```
# make robot_ctrl
# ./robot_ctrl
```

と入力します．

改訂履歴

2002 年 5 月　　初版(仮)

2002 年 6 月　　初版

　USB ドライバについて，追加

　ディップスイッチの設定，追加

　モータの制御，追加

2003 年 3 月　　Ver.1.2

お問合せ(お問い合わせはメールにてお願いいたします)

株式会社イクシスリサーチ

E-mail ： info@ixs.co.jp

本社所在地

　〒212-0055

　神奈川県川崎市幸区南加瀬 4-17-14

横浜工場

　〒230-0071

　神奈川県横浜市鶴見区駒岡 5-14-10

この取扱説明書は，再生紙を使用しています.