

iXs Research Corp.

Cool Robotics

iMCs01Software Manual

Ver. 1.0e



iXs Research Corporation

Contents

1. Introduction	3
2. I/O functions	3
2.1 Motor Outputs (CN101 ~ CN104)	3
2.2 Sensor Inputs (CN105 ~ CN108)	3
2.3 Dipswitch	3
3. Calculation of Control Signal	4
3.1 Control Equations	4
3.2 End Points	4
3.2.1 End Point 1	4
3.2.2 End Point 2	5
3.2.3 End Point 5	6
3.2.4 End Point 6	6
3.3 USB Driver	11
3.3.1 Opening/Closing Devices	11
3.3.2 Reading Data from iMCs01s	11
3.3.3 Initialization of iMCs01s	12
3.3.4 Sending Control Signals to iMCs01s	13
3.3.5 LITTLE_ENDIAN and BIG_ENDIAN	13
3.4 Firmware of iMCs01s	14
3.4.1 Software Counters	14
3.4.2 Dipswitch Setting	15
3.4.3 Functions and Usage of H8	15
4. Execution of Program	16
4.1 Registering a USB Device	16
4.2 Loading USB Driver	16
4.3 Connecting iMCs01 Board	16
4.4 Closing Procedure	17
4.5 Executing Program	18
4.5.1 Reading Sensory Values	18
4.5.2 Motor Control (Open-Loop Motor Control)	18
4.5.3 Motor Control (Position Control by using Potentiometer Feedback)	19
4.5.4 Motor Control (Position Control by using Encoder Software Counter)	19
4.5.5 Motor Control (Position Control by using Encoder Hardware Counter)	19
4.5.6 Motor Control of Multiple iMCs01s	20
4.5.7 How to Make Makefile	21

1. Introduction

This instruction manual describes the software and settings of iMCs01 USB, a 4 channel motor controller. To understand this manual, the basic knowledge of Linux operating system is required. Refer to introductory books for each Linux commands, if necessary.

This manual is subject to modify without any notice, so please download the latest version from the website of the iXs research corporation.

2. I/O functions

2.1 Motor Outputs (CN101 ~ CN104)

		PWM mode		D/A mode	
		PWM pin	BRK/DA pin	PWM pin	BRK/DA pin
CN101	CH0	11bit	BRK	N/A	8bit true
CN102	CH1	9bit	BRK	N/A	8bit true
CN103	CH2	11bit	BRK	12bit (with LPF)	BRK
CN104	CH3	11bit	BRK	12bit (with LPF)	BRK

2.2 Sensor Inputs (CN105 ~ CN108)

		PWM mode		D/A mode	
		Counter pin	A/D pin	Counter pin	A/D pin
CN105	CH0	N/A	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 Dipswitch

Pin	Function	ON	OFF
1	ID (0x00 ~ 0x1f)	1	0
2		1	0
3		1	0
4		1	0
5		1	0
6	PWM logic polarity	Positive	Negative
7	Mode switch	D/A mode	PWM mode
8	Break switch	High	Low

3. Calculation of Control Signal

3.1 Control Equations

The control output u is calculated by the following equation every 1[ms].

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

where the control gains are specified by K_P , K_D , K_I , 16bit integers (from -32768 to 32767), and K_{Px} , K_{Dx} , K_{Ix} , 16bit positive integers (from 0 to 65535). The offset A and the control target values x_d , and \dot{x}_d are 16bit integers (from -32768 to 32767). The control output u is a 16bit positive integer (from 0 to 65535). Please note that, when A/D converters (10 bits) are used, the value of x should be from 0 to 32736, even though the control target values can generally be specified with 16bits. The velocity value \dot{x} is calculated by the subtraction of x (sampling rate of 1 ms) as follows.

1. Calculate $(x_d - x)$, $(\dot{x}_d - \dot{x})$, $\sum_0^t (x_d - x)$.
2. When the integration of control errors $\sum_0^t (x_d - x)$ is out of 16bit range, it is set at either 32767 or -32768.
3. Substitute A to ERx (32bit register).
4. Multiply K_P and $(x_d - x)$, then set it to ERy (32bit register).
5. Divide ERy by K_{Px} , then set it to ERy.
6. Subtract ERy from ERx.
7. Multiply K_D and $(\dot{x}_d - \dot{x})$, then set it to ERy.
8. Divide ERy by K_{Dx} , then set it to ERy.
9. Subtract ERy from ERx.
10. Multiply K_I and $\sum_0^t (x_d - x)$, then set it to ERy.
11. Divide ERy by K_{Ix} , then set it to ERy.
12. Subtract ERy from ERx.
13. When the ERx is out of 16 bit range, set it to either 65535 or 0.
14. Output the signal according to the supported significant bits of the 16bit value calculated above.

Note that the subtraction of terms (ERx - ERy) could be overflowed, whereas the terms themselves should not be overflowed because of 32bit register. However, if the gains are specified properly so that the output value is not saturated, the overflow should not be occurred.

3.2 End Points

3.2.1 End Point 1

The End Point 1 is bulk transfer, the size of each packet is 64 bytes, and the direction is "IN"(H8 PC). The packet is sent only when the End Point 6 specifies to do so. The structure of the packet is shown in Fig. 1. The order of *short* bytes is little endian. Therefore it does not need to be modified when the Intel type processors are used. Although the internal calculation of H8 is big endian, the

values are converted before each transmission. The values of A/D and D/A converters are shifted to left. For example, when a user wants to use the value of 10bit A/D converter in the range of 0 to 1023, it should be shifted 6bit to right.

The internal counter increment every 1[ms], and it is set to 0 after 65535. The intmax and interval indicate the period of software two-phase counter routine. (At the moment, the values might not be able to measure correctly when the period is more than 255 (i.e. $255 \times 64 / 20 [\text{MHz}] = 0.82 [\text{ms}]$), because the unit of these values is 1/64 clock.)

```
struct uin {
    unsigned short time;          /* internal counter (the period of 1[ms]) */
    unsigned short ad[4];        /* A/D converter (10bit) */
    short ct[4];                 /* 2 phase counter (16bit) */
    unsigned short da[4];        /* D/A output (only 8/12bit is used) */
    unsigned char din; /* Not used */
    unsigned char dout; /* Not used */
    unsigned short intmax;       /* Maximum period of software counter */
    unsigned short interval; /* Latest period of software counter */
    unsigned short magicno;
    char dmy[30];
};
```

Fig.1 uin structure of End Point 1 and 5

3.2.2 End Point 2

The End Point 2 is also a bulk transfer, the size of packet is 64 bytes, and the direction is “OUT” (PC H8) . The structure of packed is as follows.

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

The scmd structure is shown in Fig.2 . The values of x and d are the 16bit control targets .kp ,kpx , kd , kdx , ki , kix are PID gains . Note that kpx , kdx , kix should not be 0 . It is possible to specify the control target value with 16bit, nevertheless the value of A/D converter (10bit) is between 0 and 32736, namely the subordinate 5 bits are always 0.

```
struct scmd {
    short x;          /* target position */
    short d;          /* target velocity */
    signed short kp;  /* gain for position error (numerator) */
};
```

```
    unsigned short kpx; /* gain for position error (denominator) */
    signed short kd;    /* gain for velocity error (numerator) */
    unsigned short kdx; /* gain for velocity error (denominator) */
    signed short ki;    /* gain for integration error (numerator) */
    unsigned short kix; /* gain for integration error (denominator) */
};
```

Fig.2 scmd structure

Example 1: (4 channel position control (PD control) of sinusoidal target trajectory)

```
/******
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 gain = 10.0 */
    obuf.ch[i].kpx= 1;
    obuf.ch[i].kd = 1; /* D gain = 0.2 */
    obuf.ch[i].kdx= 5;
    obuf.ch[i].ki = 0; /* I gain = 0 */
    obuf.ch[i].kix= 1;
}
j++;
*****/
```

Here the variable “obuf” is a user-defined “uout” structure.

3.2.3 End Point 5

The End Point 5 is a bulk transfer, the size of packet is 64 bytes, and the direction is “IN” (H8 PC). The content of packet is the same as the End Point 1 (Fig. 1). The next packet is set as soon as the previous packet is transmitted. Therefore it is possible to read the successive data, although the data could be older one in the case that there is no transmission (such as right after opening device). For this reason, the data after opening device should be ignored. For instance, the transmission interval of FreeBSD 4.2-STABLE was 2[ms], although the theoretical transmission interval should be 1 [ms].

3.2.4 End Point 6

The End Point 6 is a bulk transfer, the size of packet is 64 bytes, and the direction is “OUT” (PC

H8). The content of packet is shown in Fig.3.

```
struct ccmd {
    unsigned char retval;
        /* return the value of EP1 when the internal counter increments */
    unsigned char setoffset; /* specify a channel to set the offset */
    unsigned char setcounter; /* specify a channel to set the counter value */
    unsigned char resetint; /* specify a channel to set the integration value */
    unsigned char selin; /* select counter(0) / ADC(1) */
    unsigned char dout; /* not used */
    unsigned short offset[4]; /* offset value */
    short counter[4]; /* counter value */
    unsigned char sellout; /* select PWM mode(0) / DAmode(1) */
    unsigned char wrrom;
    unsigned short magicno;
    unsigned char posneg; /* porality of PWM pulse */
    unsigned char breaks; /* break output */
    char dummy[36];
};
```

Fig.3 ccmd structure of End Point 6

Moreover, please note the following definition.

```
/* correspondence between bit and channel */
#define CH0 1
#define CH1 2
#define CH2 4
#define CH3 8

#define RETURN_VAL 1 /* retval */
#define SET_SELECT 0x80 /* flag to change value */
#define SET_POSNEG 0x80 /* flag to change value (to specify polarity) */
#define SET_BREAKS 0x80 /* flag to change value (to specify break HI/LOW) */
#define SET_CH2_HIN 0x40 /* set CH2 hardware counter */
```

Channel 0, 1, 2, 3 correspond to bit 0, 1, 2, 3 (CH0 , ...,3), respectively.

Set RETURN_CAL to “retval” in case that a packet is transmitted when the internal counter is raised.

Otherwise set it to 0.

“offset” is the value A in equation (1), which can be the range between 0 and 65535 for each channel.

Example: (Set 0x7fff to all channels)

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

where “cmd” is a “ccmd” structure defined by a user. .

The value of counter can be set between 0 and 65535 for each channel individually. An iMCs01 starts counting encoder outputs from this value.

Example: (Set 0 to all channels)

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

By using “setoffset”, the offset value of the specified channel can be set to cmd.offset[i] (i=0,1,2,3).

Example 1: (Set the offset value of all channels)

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

Example 2: (Not set the offset value)

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

By using “setcounter”, the counter value of the specified channel can be set to cmd.counter[i] (i=0,1,2,3).

Example 1: (Set the counter value of all channels)

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

Example 2: (Not set the counter value)

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

“resetint” specifies the channel to reset the integration value (set 1 to reset).

Example: (Reset all channels)

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

“selin” specifies the sensory input of each channel. In case of encoder, this value should be 0, otherwise the input should be analog.

Example 1: (Set all channels for analog inputs)

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

Example 2: (Set all channels for encoder inputs)

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

Example 3: (Set CH0 and CH1 to encoder inputs, and CH2 and CH3 to analog inputs)

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

“selout” specifies the type of output signal. When it is 0, the output should be PWM signal, otherwise analog signal.

Example 1: (Set all channels to PWM outputs)

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

Example 2: (Set all channels to analog outputs)

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

By using “posneg”, the polarity of PWM signal can be inverted. When it is 1, the output should be positive logic, otherwise negative.

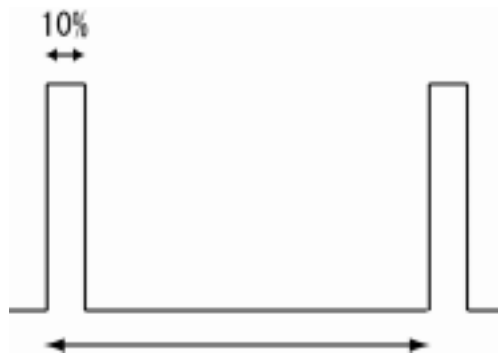
Example 1: (Set all channels to positive logic)

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

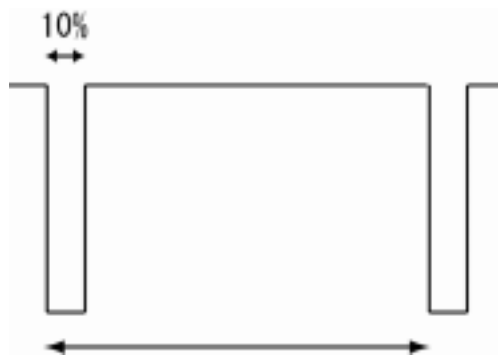
Example 2: (Set all channels to negative logic)

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

For instance, the duty ratio 10% of PWM signal output during the positive logic should be as follows.



And the negative logic should be as follows.



“break” specifies the break signal. When it is 1, the output should be Hi, otherwise Low.

Example 1: (Set all channels to output Hi)

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

Example 2: (Set all channels to output Low)

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

The correspondence between break signal and break itself depends on the motor driver. Please refer to the instruction of motor driver connected.

3.3 USB Driver

3.3.1 Opening/Closing Devices

Open a USB device.

```
int fd;  
fd = open(“/dev/urbt0”,O_RDWR);
```

When it is failed to open the device, it returns -1. To close the device, execute the following line.

```
close(fd)
```

Here is a sample program.

```
/******  
char *dev = "/dev/urbt0";  
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 Reading Data from iMCs01s

To read continuous data from iMCs01, execute the following line once. It is active until it will be modified later again.

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

Then you can read the data as follows.

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

where “buf” is the user defined “uin” type structure.

Here is a sample program.

```
/******  
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 Initialization of iMCs01s

Execute the following line to initialize the data in an iMCs01 (“cmd” data at End Point 6). It is active until it will be modified later again.

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

Then you can write the data as follows.

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

where the “cmd” is a user defined “cmd” type structure.

Here is a sample program.

```
/******  
struct cmd 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 Sending Control Signals to iMCs01s

To write control data (“scmd” at End Point 2) in an iMCs01, execute the following line once. It is active until it will be modified later again.

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

Then you can write the data as follows.

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

where “obuf” is a user defined “uout” type structure.

Here is a sample program.

```
/******
```

```
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 and BIG_ENDIAN

When a computer handles more than two byte data, it deals with one byte to another. Therefore there are two different methods to do it, namely Little Endian and Big Endian. When a computer register/transmit data from the least significant bit, it is called Little Endian, and when it does from the most significant bit, it is called Big Endian. In general, Intel type processors use Little Endian, and Motorola type use Big Endian, it may sometimes need to switch the order of data.

Since an iMCs01 handles data with Little Endian, you have to convert the data when you use a Motorola type processor. For example, when you write a “ccmd” type structure, 0x7fff, it should be as follows.

```
/******
```

```
unsigned char offset = 0x7fff;

#if __BYTE_ORDER == __LITTLE_ENDIAN
    cmd.offset[0] = offset;
#else
```

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

Note that, if you know which processor type you use, you need only one of them.

3.4 Firmware of iMCs01s

3.4.1 Software Counters

The condition of up/down for a double phase counter is shown in Table 3 and 4. When there is no edge input, the software counts according to the condition shown in Table 5. The software interprets “x” as 0. By following these rules, the software counter executes 0/+1/-1 every one computational step depending on the state of encoder inputs. Note that there is an upper limit for the software counter (approximately 4000pps). In this case, you should use a hardware counter instead.

TCLKA	L		H	
TCLKB		H		L

Table 3 Condition for a count up

TCLKA		L		H
TCLKB	H		L	

Table 4 Condition for a count down

A(t-1)	B(t-1)	A(t)	B(t)	Count
L	L	L	L	0
L	L	L	H	+1
L	L	H	L	-1
L	L	H	H	x(N/A)
L	H	L	L	-1
L	H	L	H	0
L	H	H	L	x(N/A)
L	H	H	H	+1
H	L	L	L	+1
H	L	L	H	x(N/A)
H	L	H	L	0
H	L	H	H	-1
H	H	L	L	x(N/A)
H	H	L	H	-1
H	H	H	L	+1
H	H	H	H	0

Table 5 Condition for a double phase counter by using software edge detection**3.4.2 Dipswitch Setting**

You can specify a user defined ID number by using the pins 1 to 5 of the dipswitch on iMCs01. When all pins of the switch are 0, the ID is 0x00 (0), and when all pins are 1, the ID is 0x1f (31).

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

Pin 6 specifies the polarity of PWM signal. It can be modified by a program later.

Pin 7 specifies the output mode. When Pin 7 = 1, the motor outputs of ch0 and ch1 are D/A mode, 8 bit resolution analog outputs. Moreover, by adding low pass filters, the ch2 and 3 can also be analog outputs. When Pin 7 = 0, the PWM mode, ch 0, 1, 2 and 3 output PWM signal and the pin 3 of motor output connectors is a break signal. This mode can also be modified by a program later.

Pin 8 specifies the state of break when the controller is turned on. When Pin 8 = 1, BRK pin is low (0). This mode can also be modified by a program later.

3.4.3 Functions and Usage of H8

The timer functions of a H8 are as follows.

- 16bit timer 0 : PWM output 0
- 16bit timer 1 : PWM output 1
- 16bit timer 2 : a counter coefficient of phase
- 8bit timer 0 , 1 : 1[ms] timer (cascade connection)
- 8bit timer 2 : 1000 clock timer (call for software counters)
- 8bit timer 3 : 1/64 clock counter (measuring timing to call for software counters)

The WDT function is not used.

4. Execution of Program

This section describes how to operate iMCS01 by using Linux (Kernel 2.4 or later). Please execute all of the command with root permission. Note that inputs from command line have black background. “\$” denotes the input by user mode, and “#” denotes input by root mode.

```
$ su -l
```

4.1 Registering a USB Device

To register a USB device, execute the following command once in a computer.

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

```
# chmod 666 /dev/urbtc0
```

4.2 Loading USB Driver

You can not use iMCs01s together with the other USB devices such as USB mouse. Please disconnect all other USB devices. In addition, please delete hid driver as follows, when it is installed.

```
# rmmmod hid
```

UHCI(Universal Host Controller Interface) is required to load the USB driver, urbtc.o. When it is not loaded yet, execute the following command.

```
# insmod uhci
```

Change directory to one where there is urbtc files, and load the USB module. Please do not connect any iMCs01 to USB ports before this installation.

```
# cd /home/user1/urbtc/
```

```
# make
```

```
# insmod urbtc.o
```

Here we check if the module is loaded properly.

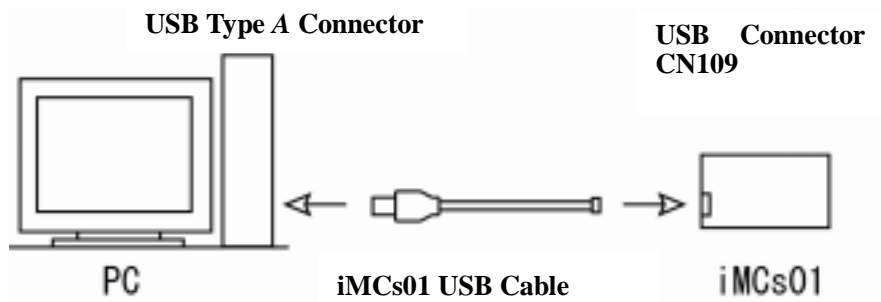
```
# lsmod
```

Then check the following is displayed.

Module	Size	Used by
urbtc	7360	0 (unused)

4.3 Connecting iMCs01 Board

Connect an iMCs01 to a PC.



Note: It is also possible to go through self powered USB HUBs.

After it is connected, input the following line.

```
# dmesg
```

And check if the following lines are displayed.

```
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
```

When you connect iMCs01s via a USB hub, execute the following line before connect iMCs01.

```
# dmesg
```

And check the following lines are displayed.

```
usb.c: USB new device connect on bus1/2, assigned device number 3
```

```
hub.c: USB hub found
```

```
hub.c: 4 ports detected
```

To test if the iMCs01 is recognized properly, execute following commands.

```
# make
```

```
# ./h8test
```

And check if the following lines are displayed.

```
Vendor xxxxxxxx
```

```
Product xxxxxxxx
```

```
read status 0
```

```
read status 1
```

```
write status 3
```

```
write status 3
```

4.4 Closing Procedure

Finally, when you disconnect iMCs01s, check the following lines are displayed by dmesg command.

```
usb.c: USB disconnect on device 2
```

```
urbtc.c: urbtc0 now disconnected
```

Then you can remove the USB driver as follows.

```
# rmmod urbtc
```

4.5 Executing Program

Here we test the sample programs. Copy the files in the enclosed CD-ROM to a preferred directory. All of the programs are assumed that the motor drivers iMDs03s are connected to the iMCs01. Note that “INCLUDE” of the Makefiles in the drivers and sample source codes is as follows.

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

When some compilation errors occur under RedHat7.3, RedHat8.0 etc., modify the path as follows.

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

4.5.1 Reading Sensory Values

In order to run the sample program of reading sensory values, which is called “uread”, input following commands.

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

Then the following lines are displayed.

```
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
```

These lines show the values of 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, from left to right.

4.5.2 Motor Control (Open-Loop Motor Control)

To run the sample program of open-loop motor control, which is called “urobot_open_loop”, input the following commands.

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

4.5.3 Motor Control (Position Control by using Potentiometer Feedback)

To run the sample program of A/D feedback motor position control, which is called “urobot_ad”, input the following commands.

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

This program outputs a sinusoidal wave around the analog input value 32767. Here is the specification of this sample program.

Channel	All channels
Output	PWM
Input	Analog
Offset	0x7fff
Break	Released
Polarity of PWM signal	Positive

4.5.4 Motor Control (Position Control by using Encoder Software Counter)

To run the sample program of encoder feedback motor position control, which is called “urobot_enc”, input the following commands.

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

This program outputs a sinusoidal wave around the initial position. Here is the specification of this sample program.

Channel	CH0 , CH1,CH2
Output	PWM
Input	Encoder
Offset	0x7fff
Break	Released
Polarity of PWM signal	Positive

4.5.5 Motor Control (Position Control by using Encoder Hardware Counter)

To run the sample program of motor position control with one channel encoder feedback, which is called “urobot_1ch_enc”, input the following commands.

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

This program outputs a sinusoidal wave around the initial position. Here is the specification of this sample program.

Channel	-
Input channel	CH0(CN105)
Output channel	CH2(CN103)
Control Equation	Equation of CH2
Output	PWM
Input	Encoder
Offset	0x7fff
Break	Released
Polarity of PWM signal	Positive

In case of the position control with 1ch encoder feedback using the hardware counter, the cable connection is different. In addition, by setting the “selin” and “selout” as follows, the hardware counter should be automatically selected.

```

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

```

Note that the gain and target values should be set for the ch2.

4.5.6 Motor Control of Multiple iMCs01s

To connect more than two controllers, the corresponding USB nodes should be made as follows. This procedure need to be done once in a PC.

```

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

```

where “180” is the major number of the USB node and “101” is the minor number.

When three controllers are connected, the procedure is as follows.

```

# mknod /dev/urbtc1 c 180 101
# chmod 666 /dev/urbtc1
# mknod /dev/urbtc2 c 180 102
# chmod 666 /dev/urbtc2

```

To run the sample program of two-motor control, which is called “urobotm”, input the following commands.

```

# make urobotm
# ./urobotm

```

This sample program sends the command to two motor controller boards, where the outputs are the sinusoidal position around the initial condition to all of the channels. Here is the specification of this sample program.

ID	All the same IDs
Channel	All channels
Output	PWM
Input	Analog
Offset	0x7fff
Break	Released
Polarity of PWM signal	Positive

4.5.7 How to Make Makefile

To make a user program, it would be easier to change the Makefile. Refer to the following example.

Example: to make a program called robot_ctrl.c

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

After the Makefile is modified, input the following lines to run the program “robot_ctrl”.

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

Revision History

2003. October Ver.1.0e (First Edition in English)

Contact (Please contact via email)

iXs Research Corporation

E-mail : info@ixs.co.jp

Headquarter

4-17-14 Minamikase Saiwai-ku Kawasaki-shi Kanagawa 212-0055 Japan

Yokohama factory

5-14-10 Komaoka Tsurumi-ku Yokohama-shi Kanagawa 230-0071 Japan

Copying or duplicating this manual in part or in whole is forbidden.

The information and specifications in this manual are subject to change without notice.