MultiQ-3 ™

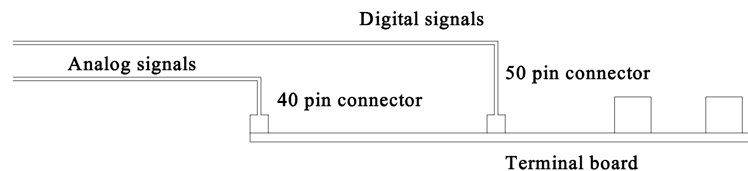
8 Analog to Digital Converters
8 Digital to Analog Converters
Up to 8 Quadrature input decoders/counters
8 Digital inputs
8 Digital outputs
3 Realtime clocks

PROGRAMMING MANUAL

Quanser Consulting

CAUTION

The flat ribbon cables must be inserted into the MultiQ-3 board and the TERMINAL board with the correct *orientation*. Do not force the connectors in.



CAUTION

The board is static electricity sensitive. Be very careful with electrical discharge. Always touch a ground plane BEFORE you handle the board.

CAUTION

Some of the wiring you perform to the terminal board carries POWER. Be sure your wiring is correct as applying power incorrectly may either damage the device you are connecting to, the board or your computer!

NEED HELP ?

CALL US AT (905) 527 5208 or FAX your question to (905) 570 1906

OR EMAIL TO : help@quanser.com

MultiQ-3^{TM} I/O BOARD
Quanser Consulting

1.0 General description

The MultiQ-3 is a general purpose data acquisition and control board which has 8 single ended analog inputs, 8 analog outputs, 16 bits of digital input , 16 bits of digital output, 3 programmable timers and up to 8 encoder inputs decoded in quadrature (option 2E to 8E). Interrupts can be generated by either of the three clocks, one digital input line and the end of conversion from the A/D.

The system is accessed through the PC bus and is adressable via 16 consecutive memory mapped locations which are selected through a DIP switch located on the board.

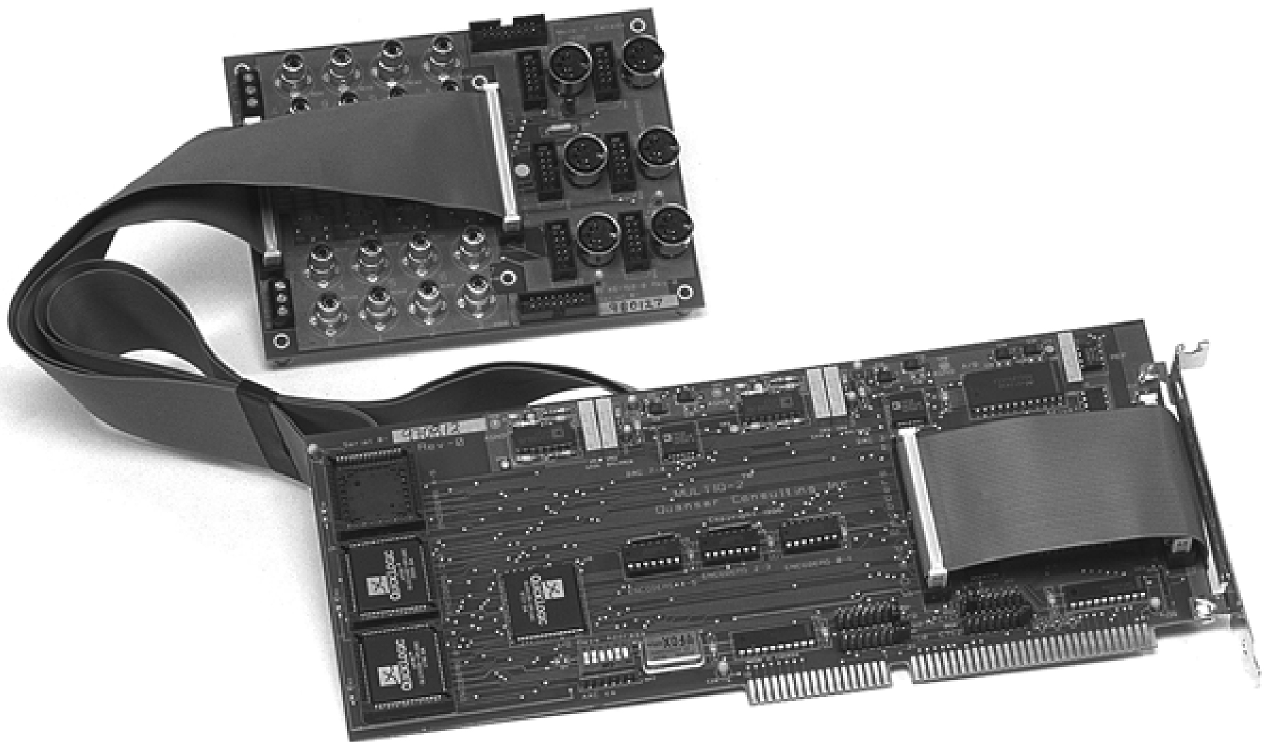
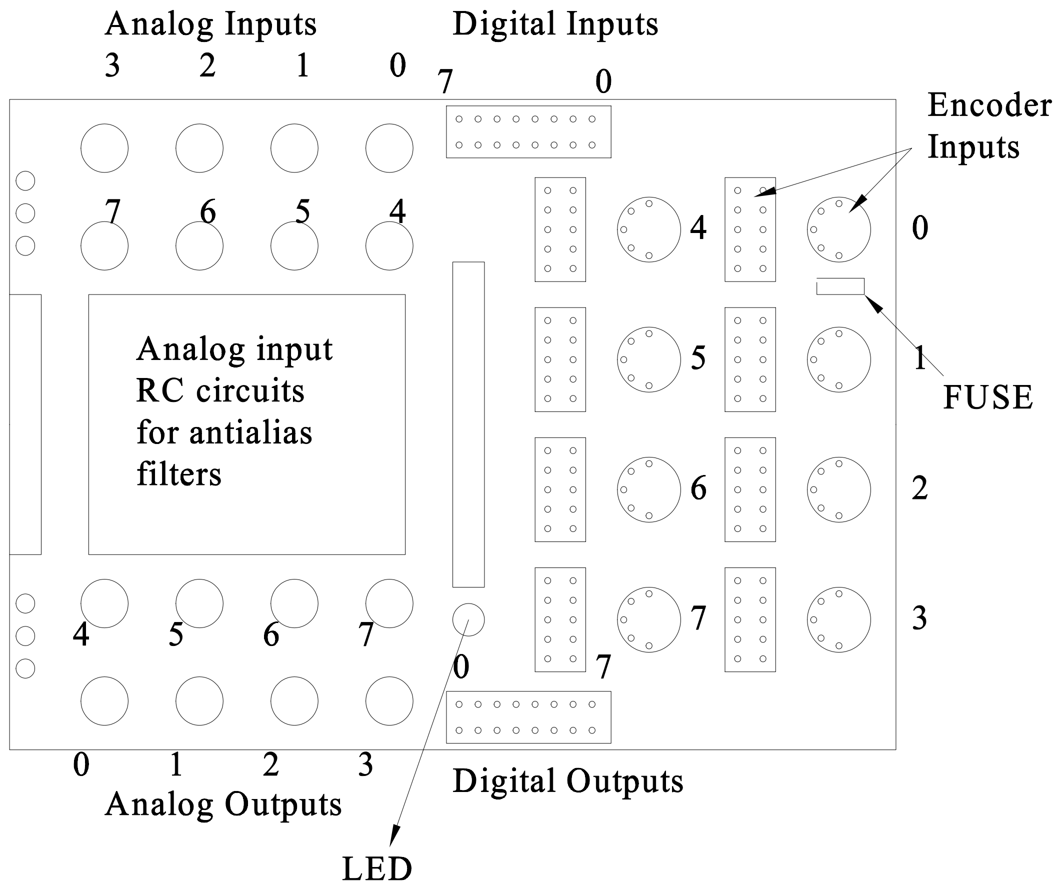


Photo of MultiQ-2. The MultiQ-3 has 8 encoder connectors on the terminal board.

2.0 Principles of operation

2.0.1 Terminal board
Turn off the PC. Guide the flat ribbon cables carefully out through the backplane and then insert the MultiQ board into the ISA slot. Tighten the screw. Insert the two flat ribbon cables into the terminal board. Make sure you insert them in the correct orientations.

Turn on the PC. **The LED on the terminal board should light up. If not, then the FUSE on the terminal board may be blown.** Check your connections first. If they seem fine, then turn off the computer and check the fuse. It is a 1 ampere field replaceable fuse similar to those used in a PC keyboard. Any computer repair shop should have them.



2.1 Analog to digital conversion

The A/D of the MultiQ is a single ended bipolar signed 13 bit binary (12 bit plus sign) A/D. You can perform a conversion on one of 8 channels by selecting the channel and starting a conversion. the EOC_I (end of conversion interrupt) bit in the STATUS REGISTER indicates that the data is ready and can be read. The data is read by issuing 2 consecutive 8 bit reads from the AD_DATA register.

The data returned is two 8 bit words which must be combined to result in a 16 bit signed word. 5 volts input maps to 0xFFF while 0 volts maps to 0x0 and -5 Volts maps to 0xFFFF000.

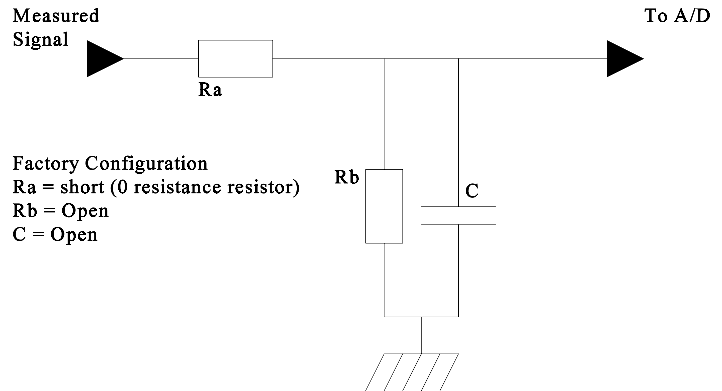
2.1.1 Wiring to the A/D

All inputs to the A/D multiplexer are single ended in the range +/- 5 Volts and should be wired to the RCA jacks labelld Analg inputs.

2.1.2 Antialiasing filters

If you wish to low pass filter the anlog signals before they are applied to the A/D converter, you may do so by populating the section of the terminal board indicated in the above figure**. It is highly recommended that you attach a capacitor to the input of each A/D. This is simply done by soldering a capacitor at the appropriate location.** The factory configuration is Ra = short, Rb = open and C = open. The choice for the component values depends on the impedence of the sensor and the sampling frequency of the control software. Typically you select a cutoff frequency less than half the sampling frequency.

A/D	Ra	Rb	C
0	R1	R5	C2
1	R2	R6	C3
2	R3	R7	C4
3	R4	R8	C5
4	R12	R16	C9
5	R11	R15	C8
6	R10	R14	C7
7	R9	R13	C6

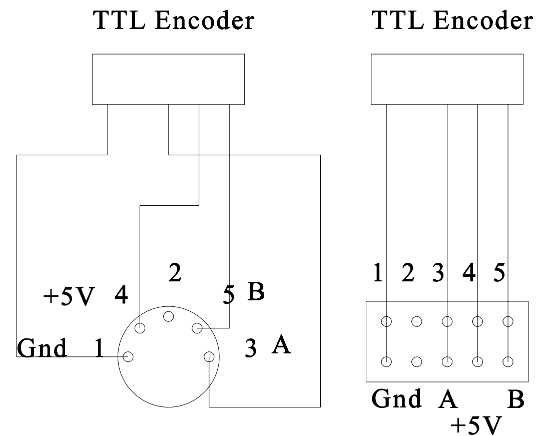


2.2 Analog output

The digital to analog (D/A) converters are 12 bit unsigned binary. An input of -5 volts maps to 0x000, 0 volts to 0x3FF and 5 volts maps to 0xFFF. Your program should write a 12 bit number (0 to 4095) to the appropriate register and latches the data. The analog outputs change when the data is latched.

2.3 Encoder Inputs

The board can be equipped with up to eight encoder decoders. (Models -2E, 4E , 6E and 8E). The encoders data is decoded in quadrature and used to increment or decrement a 24 bit counter. With 24 bits, you can obtain 16,777,215 counts. With a 2000 line encoder in quadrature, this results in 8000 counts per revolution and 2097 revolutions can be measured without overflowing the counters. Higher counts can be handled by software.



2.4 Inserting the encoder chips

If you are upgrading the board and inserting the encoder chips yourself, please ensure that they are inserted in the orientation shown on the last page of this manual. Also make sure the legs are not bent.
Each encoder chip accommodates two quadrature encoders.

2.3.1 Wiring to the Encoder inputs.

Each encoder connector is equipped with one 5 pin DIN socket and one 10 pin header on the terminal board. **You may use *either* of these to attach an encoder to the desired channel.** The connector supplies a +5V, and GND to bias the encoders and receives an 'A' channel and a 'B' channel from the encoder. **You must use 5 Volt output encoders only.** The figures above show the pin definition for the 5 pin DIN connector and the 10 pin header and the way to wire them. **Note that this is the view from the insertion direction.**

2.4 Digital inputs

The board can read 16 digital input lines mapped to one I/O address. The digital input is normally high ('1') and results in a low ('0') when the line is pulled to GND. Digital input line #0 can be tied to an interrupt using the jumpers supplied.

2.5 Digital outputs

The board can control 16 individual digital outputs mapped to one I/O address. Writing a '0' to the appropriate bit results in zero volts (TTL LOW) at the output while writing a '1' results in 5 Volts (TTL HIGH).

2.6 Realtime clocks

The board is equipped with three independent programmable clock timers. Each timer can be programmed to run at a frequency between 2 MHz Hz and 30.52 Hz. The principle of operation is to write a divisor (N) to the desired clock and the output frequency will be 2.0/N MHz. (N) is a 16 bit integer value between 2 and 65535 (0xFFFF). The output of any of the three clock can be tied to an interrupt line using a jumper on the board.

3.0 PROGRAMMING

3.1 Base address selection

The base address is selected by using the dip switches(SW2) on the board. Factory configuration is 0x320. make sure there are no other devices on that address and up to Base+0xF (0x32F factory configuration).

Base Address											
B	A	9	8	7	6	5	4	3	2	1	0
0	0	SW2-F	SW2-E	SW2-D	SW2-C	SW2-B	SW2-A	Decoded on board			

3.1.1 Factory configuration

Base Address								
0	0	SW2-F	SW2-E	SW2-D	SW2-C	SW2-B	SW2-A	Decoded on board
0	0	1(OFF)	1(OFF)	0(ON)	0(ON)	1(OFF)	0(ON)	0
3				**2**				**0**

3.2 Board Registers The table below shows the registers on the MULTIQ. Each register is described in its appropriate section.

Base +	Read	Write	Size
0	DIGIN_PORT	DIGOUT_PORT	16 bit
1			
2		DAC_DATA	16 bit
3			
4	AD_DATA	AD_CS	16 bit write 8 bit read
5			
6	STATUS	CONTROL	16 bit
7			

8	N/A	CLK_DATA	8 bit
9	N/A	N/A	N/A
A	N/A	N/A	N/A
B	N/A	N/A	N/A
C	ENC_DATA	ENC_DATA	8 bit
D	N/A	N/A	N/A
E	ENC_CONTROL	ENC_CONTROL	8 bit
D	N/A	N/A	N/A

3.2.1 Digital port : Base + 0 (DIGIN_PORT & DIGOUT_PORT)

3.2.1.1 Write (DIGOUT_PORT)

This is the Digital outport port. A 16 bit write to this port outputs the 16 bit data to the digital output header on the terminal board. Eg. writing a 0x0F40 results in bits 11,10,9,8 and 4 to go high.

DIGIN_PORT Write															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DO 15	DO 14	DO 13	DO 12	DO 11	DO 10	DO 9	DO 8	DO 7	DO 6	DO 5	DO 4	DO 3	DO 2	DO 1	DO 0

3.2.1.2 Read (DIGIN_PORT)

This is the Digital input port. A 16 bit read from this board returns the digital levels at the header labelled Digital input on the terminal board. The inputs are tied high and a read with nothing connected to the header results in reading a 0xFFFF. A returned value of 0xF5FF means that bits 11 and 9 have been pulled low by an external device (for example a switch).

DIGIN_PORT Read															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DI 15	DI 14	DI 13	DI 12	DI 11	DI 10	DI 9	DI 8	DI 7	DI 6	DI 5	DI 4	DI 3	DI 2	DI 1	DI 0

3.2.2 D/A data Port : Base + 2 (DAC_DATA)

3.2.2.1 Write (DAC_DATA)

A write to this port sets up the Data for the D/A output. The data should be written to bits(DO11 to DO0). A value of 0 puts out -5 Volts, a value of 0x1FF puts out 0 Volts and 0xFFF puts out +5 Volts. The output channel is selected by writing to bits (DA2 DA1 DA0) of the CONTROL REGISTER and the output is latched when a (11) is written to bits (LD0 LD1) of the CONTROL REGISTER.

DAC_DATA Write															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NA	NA	NA	NA	AO 11	AO 10	AO 9	AO 8	AO 7	AO 6	AO 5	AO 4	AO 3	AO 2	AO 1	AO 0

3.2.2.2 Read (NOT APPLICABLE)

3.2.3 A/D Register : Base + 4 (AD_CS and AD_DATA)

3.2.3.1 Write (AD_CS)

A write to this register (any data) initiates a conversion after the A/D has been properly set up using the CONTROL REGISTER.The program must wait for (EOC) to go high in the STATUS REGISTER *before* initiating a conversion.

3.2.3.1 Read (AD_DATA)

This is an **8 bit read register** and contains the high byte on the first read and the low byte on the second read. The structure of the two 8 bit reads is shown below:

AD_DATA Read (First time)							
7	6	5	4	3	2	1	0
SIGN	SIGN	SIGN	SIGN	AI 11	AI 10	AI 9	AI 8

AD_DATA Read (Second time)							
7	6	5	4	3	2	1	0
AI 15	AI 14	AI 13	AI 12	AI 11	AI 10	AI 9	AI 8

To convert the data to a voltage the two bytes should be combined into a 16 bit word as follows:

```
integer_data = (high_byte<<8)|(low_byte&0xFF);
volts = integer_data * 5.0/4096;
```

where (<<) is the shift left operator, (|) is bitwise 'OR' and (&) is bitwise 'AND'. This means mask off the left 8 bits of the low byte just in case there is extraneous noise, and then merge it with the high byte shifted left 8 bits. This results in a number between (-4096) and (4095). These are mapped to -5V and +5V.

3.2.4 CONTROL REGISTER: Base + 6 (STATUS & CONTROL)

3.2.4.1 Write (CONTROL)

CONTROL REGISTER Writre															
X	X	X	LD1	LD0	CLK	S/H	CAL	AZ	EN	A2 E2	A1 E1	A0 E0	DA2	DA1 RC1	DA0 RC0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits 0-2 (DA0 DA1 DA2) : Select the D/A channel number. eg writing a 0x03 selects channel D/A ch3
Bits 0-1 (RC0 RC1) : Select the realtime clock register
Bits 3-5 (A0 A1 A2) : Select the analog input channel you want to Multiplex to the A/D
Bits 3-5 (E0 E1 E2) : Select the encoder channel on which you want to perform operations
Bit 6: (MX) Enable the 8 channel multiplexer
Bit 7: (AZ) Enable Auto Zero on the A/D
Bit 8: (CAL)Enable Autocalibration on the A/D
Bit 9: (S/H) Disable Sample and Hold on the A/D **Keep this bit high all the time**
Bit 10 (CLK): Select base clock frequency for the A/D. 1 is 4 MHz, 0 is 2 MHz. (Always use 4 Mhz). **Keep this bit high all the time**
Bit 11-12(LD0,LD1): Latch data to the selected D/A channel when both bits are set high.

3.2.4.2 Read (STATUS)

STATUS REGISTER Read															
											EOC_I	EOC	CT1	CT2	CT0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits 0-2(CT0,CT1 CT2): Counter timeout states for the three timers.
Bit 3:(EOC) End of conversions on the A/D. Goes high when A/D is ready for another conversion.
Bit 4:(EOC_I) End of conversion Interrupt. Goes high when A/D conversion is complete.
3.2.5 Clock data register: base + 8 CLK_DATA

3.2.5.1 Write (CLK_DATA)

This an **8 bit** write only register that accesses any of the four registers on the realtime clock chip. The register (CLK_0 to CLK_4, see section on programming) is selected by writing to bits (RC1 RC0) of the CONTROL REGISTER and then writing the data to CLK_DATA.

CLK_DATA Write							
7	6	5	4	3	2	1	0
CD 7	CD 6	CD 5	CD 4	CD 3	CD 2	CD 1	CD 0

3.2.5.2 Read (NOT APPLICABLE)

3.2.6 ENCODER REGISTERS Base + 0xC (ENC_DATA) and Base + 0xD (ENC_CONTROL)

In order to read or write to the appropriate channel, you must first select the channel by writing to bits (E2 E1 E0) of the CONTROL REGISTER (Base + 0x06). Writing to this register selects which of the 8 encoder counters you want to operate on. The encoder number is specified in bits (E2 E1 E0)

3.2.6.1.1 Read ENC _DATA

You should always reset the **byte counter** before you read from this register. You then read three consecutive bytes from this register. The first read is the low byte of the data, the second read is the mid byte and the third read is the high byte. You construct the 24 bit counter by merging the three bytes as follows:

```
data = low_byte | (mid_byte << 8) | (high_byte <<16)
```

ENC_DATA Read (first time)							
7	6	5	4	3	2	1	0
E7	E6	E5	E4	E3	E2	E1	E0

ENC_DATA Read (second time)							
7	6	5	4	3	2	1	0
E15	E14	E13	E12	E11	E10	E9	E8

ENC_DATA Read (third time)							
7	6	5	4	3	2	1	0
E23	E22	E21	E20	E19	E18	E17	E16

3.2.6.1.2 Write ENC _DATA

This is an 8 bit write register to which you can write preload values to the counters as well as the clock filtering frequency. The clock filter frequency determines the frequency of the digital filter which is used to filter the A & B signals from the encoders. The base frequency is 4 MHz. Normally you keep the frequency at 4 MHz.

3.2.6.2 Write ENC_CONTROL

Writing to this register allows for various functions to be performed. These are described below:

ENC_CONTROL Write							
7	6	5	4	3	2	1	0
EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0

EC7: Keep always low

EC6	EC5	
0	0	Select Reset and load decoder register (RLD)
0	1	Select Counter mode register (CMR)
1	0	Select I/O control register (IOR)
1	1	Select index control register (IDR)

Once you have determined which of the above 4 registers you want to write to, you then select the lower 5 bits according to the function you want to achieve.

3.2.6.2.1 Writing to reset and load registers (RLD)

EC6	EC5	
0	0	Select Reset and load decoder register (RLD)
EC4	**EC3**	
0	0	NOP
0	1	Transfer preload to counter. Used to preload the counters with a value
1	0	Transfer counter to output latch. Must be performed before a read.
1	1	Transfer prescale factor to prescaler. Used to set digital filter frequency.
EC2	**EC1**	
0	0	NOP
0	1	Reset 24 bit counter to 0
1	0	Reset Borrow, Carry and Compare toggle flip flops(NOT USED)
1	1	Reset Error

EC0: Reset Byte pointer. Writing a '1' to this bit resets the byte pointer for the following read operations. Resetting the byte pointer ensures that the read cycle from the EN_DATA regsiter remain in synch with the expected sequence of lowbyte, mid byte and high byte.

3.2.6.2.2 Counter mode register (CMR)

EC6	**EC5**	
0	1	Select Counter mode register (CMR)
EC4	**EC3**	
0	0	NOT USED
0	1	NOT USED
1	0	NOT USED
1	1	Quadrature count
EC2	**EC1**	
0	0	Normal count
0	1	NOT USED
1	0	NOT USED
1	1	NOT USED

EC0: '0' = Binary count, '1' = BCD. Normally use '0'.

3.2.6.2.3 Counter mode register (IOR)

EC6	**EC5**	
1	0	Select I/O control register (IOR)
EC4	**EC3**	
0	0	NOT USED
0	1	NOT USED
1	0	NOT USED
1	1	NOT USED
EC2	**EC1**	
0	0	NOT USED
0	1	NOT USED
1	0	NOT USED
1	1	NOT USED

EC0: '0' = Disable counter , '1' = Enable counter. Normally '1'

3.2.6.2.4 Index control register (IDR)

EC6	**EC5**	
1	1	Select index control register (IDR)
EC4	**EC3**	
0	0	NOT USED
0	1	NOT USED
1	0	NOT USED
1	1	NOT USED
EC2	**EC1**	
0	0	NOT USED
0	1	NOT USED
1	0	NOT USED
1	1	NOT USED

EC0: '0' = Index disable. Normally '0'.

3.2.6.3 READ ENC_CONTROL

An eight bit read from the encoder control register will read the following byte:

ENC_CONTROL READ							
7	6	5	4	3	2	1	0
EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0

EC4 indicates that noise was picked up on lines A or B of the inputs. Reset EC4 by writing to (EC2 EC1) of the RLD.

4.0 SAMPLE PROGRAMS

The following are sample programs written in Turbo C and can be used by your main program. These functions are included in the file **mq3.drv**. The file also contains the definitions of the register locations based on a base address 0x320 as shown below:

```
#define base_port 0x320

#define digin_port      base_port + 0x00
#define digout_port     base_port + 0x00
#define dac_cs          base_port + 0x02
#define ad_cs           base_port + 0x04
#define status_reg      base_port + 0x06
#define control_reg     base_port + 0x06
#define clk_reg         base_port + 0x08
#define enc_reg1        base_port + 0x0c
#define enc_reg2        base_port + 0x0e

#define AD_SH           0x200 /* active low  */
#define AD_AUTOCAL      0x100 /* active high */
#define AD_AUTOZ        0x80  /* active high */
#define AD_MUX_EN       0x40  /* active high */
#define AD_CLOCK_4M 0x400  /* high = 4 MHz */
```

```
/* IMPORTANT */
/* sample and hold disabled to prevent exrtaneous sampling */
/* and fix the clock speed to 4 MHz */

#define CONTROL_MUST (AD_SH | AD_CLOCK_4M)

unsigned int control_word = CONTROL_MUST;

/* ENCODER CHIP COMMANDS */

#define CLOCK_DATA       0      // FCK frequency divider
#define CLOCK_SETUP      0X18   // transfer PR0 to PSC
#define INPUT_SETUP      0X41   // enable inputs A and B
#define QUAD_X4          0X38   // quadrature
#define BP_RESET         0X01   // reset byte pointer
#define CNTR_RESET       0X02   // reset counter
#define TRSFRPR_CNTR     0X08   // transfer preset register to counter
#define TRSFRCNTR_OL     0X10   // transfer CNTR to OL (x and y)
#define EFLAG_RESET      0X06   // reset E bit of flag register
```

4.1 D/A operation

- Write the analog output channel number to bits (DA2 DA1 DA0) of CONTROL along with a (11) to LD1 and LD0 of the CONTROL REGISTER
- Write the actual data to DAC_DATA (16 bit write lowest 12 bits carry the data).
- release the latch by writing a (00) to both bits (LD1 LD0) of the CONTROL REGISTER.

4.1.1 Sample C function

```
void daout(int ch, int ivalue)
{
outport(control_reg, 0x1800 | ch | CONTROL_MUST);
outport(dac_cs,ivalue);
outport(control_reg, CONTROL_MUST);
}
```

4.1.2 Reset the D/A outputs

```
void reset_da(void)
{
int zero_v;
zero_v = vtoi(0.0); /* see this function below */
daout(0,zero_v);
daout(1,zero_v);
daout(2,zero_v);
daout(3,zero_v);
daout(4,zero_v);
daout(5,zero_v);
daout(6,zero_v);
daout(7,zero_v);
}
```

4.1.3 Voltage to integer conversion

This function is used to convert a desired voltage (in volts) to the appropriate integer value for the D/A.

```
int vtoi(float v)
{
return(ceil( v*2048/5.+2047));
}
```

4.2 A/D OPERATIONS

4.2.1 Calibrating the A/D

This can be performed **once only** at the start of a program. Once calibrated, the offset and gain are used for all subsequent measurements. To calibrate:

1) write a '1' to bit 'CAL' and to 'S/H' of CONTROL REGISTER
2) write a '1' to 'S/H' of CONTROL REGISTER
3) wait for EOC to go high in STATUS REGISTER

4.2.1.1 Sample C function

```
void reset_ad(void)
{
/* start calibration */
outport(control_reg, AD_AUTOCAL | CONTROL_MUST);
outport(control_reg, CONTROL_MUST);
while( (inport(status_reg)&0x08) == 0x00 );
}
```

4.2.2 Acquiring a sample

1) select the channel and write it to control register bits (A2 A1 A0) along with a '1' to (EN) , a '1' to (S/H) and a '1' to (CLK) bits of the CONTROL REGISTER. Also write a '1' to bit (AZ) if you want auto zero before the sample. Note that autozero takes longer and is not normally necessary.
2) wait until (EOC) in STATUS REGISTER goes high.
3) Initiate a conversion by a write to AD_CS (any value)
4) wait until EOC_I in STATUS REGISTER goes high
5) read high byte from AD_DATA
6) read low_byte from AD_DATA

4.2.2.1 Sample C Function

```
int adin(int ch)
{
unsigned int hb,lb;
int toolong,maxcnt;
maxcnt = 30;
nosound();
control_word = CONTROL_MUST | AD_MUX_EN | (ch<<3); /* select channel and enable mux  start S/H*/

/* use the next line instead of above line if you want to auto zero before every sample */
/*control_word = CONTROL_MUST | AD_AUTOZ | AD_MUX_EN | (ch<<3);*/
/* NOTE THAT IT IS SLOWER WITH AUTO ZERO */

outport(control_reg,control_word);
toolong = 0;
while( ((inport(status_reg)&0x8) == 0x00 ) && (toolong <maxcnt) ) toolong++;
if(toolong>=maxcnt) sound(400);
outportb(ad_cs,0);
while( (inport(status_reg)&0x10) == 0x00 );
```

```
hb = inport(ad_cs) & 0xff;
lb = inport(ad_cs) & 0xff;
outport(control_reg,CONTROL_MUST);
return ( (hb<<8) | lb);
}
```

Note the limited wait loop:

```
toolong = 0;
while( ((inport(status_reg)&0x8) == 0x00 ) && (toolong <maxcnt) ) toolong++;
if(toolong>=maxcnt) sound(400);
```

which ensures that the wait for EOC is left if it takes too long . If this happens, an error has occurred on the A/D Chip (National Semiconductor ADC1251) during a conversion and the chip is not ready for a conversion after sufficient waiting. *This is not usually necessary but is good practice.* If this error occurs, the computer will generate a sound until issuing 'nosound()' from the calling C program. If you do not want the sound to go on just delete the 'sound(400)' statement.

4.2.3 Integer to voltage conversion

The following function converts from an integer value read by the A/D to a floating point value in volts.

```
float itov(int iv)
{
return(iv*5/4095.);
}
```

4.3 Digital input operation:

- Read a 16 bit word from DIGIN_PORT

4.3.1 Sample C Function

```
int digin(void)
{
return inport(digin_port);
}
```

4.4 Digital output operation

- Write a 16 bit word to DIGOUT_PORT

4.4.1 Sample C Function

```
void digout(int dig_value)
{
outport(digout_port,dig_value);
}
```

4.5 Encoder operations

4.5.1 Encoder reset

-Write to CONTROL register (E2 E1 E0) the channel you want to reset.
-Write a reset Error to the RLD (ie 0x06 to ENC_CONTROL)
-Reset the byte pointer (ie 0x01 to ENC_CONTROL)
-Write a clock frequency divider to ENC_DATA (usually 0)
-Transfer the divider to the the prescaler (ie 0x18 to ENC_CONTROL)

-Enable the counter inputs (0x41 to ENC_CONTROL)
-Setup to quadrature mode(0x38 to ENC_CONTROL)
-Reset the counter to zero(0x2 to ENC_CONTROL)
-Reset byte pointer (ie 0x01 to ENC_CONTROL)

4.5.1.1 Sample C function

```
void enc_reset(int ch)
{

control_word = CONTROL_MUST | (ch<<3); // select channel and enable mux  start S and H

outportb(control_reg,control_word); /* select the encoder channel */

        /*initialize the ENCODER CHIP*/
        outportb(ENC_CONTROL, EFLAG_RESET);         // reset E bit of flag register
        outportb(ENC_CONTROL, BP_RESET);            // reset byte pointer (x and y)
        outportb(ENC_DATA, CLOCK_DATA);             // FCK frequency divider
        outportb(ENC_CONTROL, CLOCK_SETUP);         // transfer PR0 to PSC (x and y)
        outportb(ENC_CONTROL, INPUT_SETUP);         // enable inputs A and B (x and y)
        outportb(ENC_CONTROL, QUAD_X4);             // quadrature multiplier to 4 (x and y)
        outportb(ENC_CONTROL, CNTR_RESET);          // reset counter (x and y)
        }
```

4.5.2 Encoder read

-Write to CONTROL register (E2 E1 E0) the channel you want to read.
-Latch the data to the output registers (0x10 to ENC_CONTROL)
-Reset byte pointer (ie 0x01 to ENC_CONTROL)
- Read low byte from ENC_DATA
- Read mid byte from ENC_DATA
- Read high byte from ENC_DATA

4.5.2.1 Sample C function

```
long int enc_in(int ch)
{
unsigned int low_byte, mid_byte, high_byte;
unsigned int high_word,low_word;
unsigned long result;
control_word = CONTROL_MUST | AD_MUX_EN| (ch<<3); // select channel

outportb(control_reg,control_word); /*SELECT THE ENCODER CHANNEL USING THE MUX */

outportb(ENC_CONTROL, BP_RESET);          // reset byte pointer

outportb(ENC_CONTROL, TRSFRCNTR_OL);      // latch the data

low_byte  = inportb(ENC_DATA)&0xff;   // least significant byte

mid_byte  = inportb(ENC_DATA)&0xff;
low_word = low_byte | (mid_byte <<8)&0xffff;

high_byte = inportb(ENC_DATA)&0xff;          // most significant byte
high_word = high_byte&0xffff;

if(high_word & 0x80) high_word = high_word | 0xff00; /*convert to signed 32 bit*/

result = ((long unsigned)high_word << 16) | low_word;
return ((long) result);

}
```

4.6 Clock operations

The three clocks are imbedded in a single integrated circuit (INTEL 82C54). Four registers located on the clock chip are addressed via bits (RC1 RC0) of the MULTIQ's CONTROL REGISTER. In order to write a desired value to a specific clock register, first write to (RC1 RC0) of the CONTROL REGISTER the value for the register you want to access and then write the desired value to the CLOCK DATA REGISTER (Base + 8).

RC1	RC0	Clock IC register
0	0	CLOCK 0 DATA REGISTER
0	1	CLOCK 1 DATA REGISTER
1	0	CLOCK 2 DATA REGISTER
1	1	CLOCK COMMAND REGISTER

The CLOCK COMMAND REGISTER has the following bits:

CLOCK COMMAND REGISTER							
7	6	5	4	3	2	1	0
SC1	SC0	RW1	RW2	M2	M1	M0	BC

The clock you want to perform an operation on is selected using bits (SC1 SC0) and the mode of operation is selected using bits (M2 M1 M0).

In order to program a specific clock to run at a given frequency, you must first select a divider for the clock. For example if you want to run at a frequency 'F' the divisor is obtained by using the equation:

DIV = CEIL(2e6/Freq)

where 2e6 Hz is the base frequency for all three clocks. This will be a 16 bit integer value (0 to 65535). Note that if Freq is smaller than (2e6/65535) the clock will actually run much faster than you expect!

Next you need to write to the CLOCK COMMAND REGISTER the clock number into (SC1 SC0) and select mode2 into (M2 M1 M0)(Baud rate Generator). You do this by **first writing to the CONTROL REGISTER** bits (RC1 RC0) a (1 1) indicating you will be writing to the CLOCK COMMAND REGISTER next and then you write the desired data .

After you select the clock number and the mode of operation into the CLOCK CONTROL REGISTER, write to the CLOCK DATA REGISTER (bits RC1 RC0 in the CONTROL REGISTER) the low byte and then the high byte of the divisor DIV. At this point, the clock you selected will start running at the frequency you specified.

4.6.1 Sample C functions

```
void clockdiv(int clk_num,int div_value)
{
unsigned int lb,hb;
lb = div_value & 0xff;
hb = (div_value & 0xff00)>>8;
control_word = 3 | CONTROL_MUST;
outport(control_reg,control_word);/*select register 3 of RTC */
outportb(clk_reg,((clk_num<<6)|0x34));
control_word = clk_num | CONTROL_MUST;
outport(control_reg,control_word);
outportb(clk_reg,lb);
outportb(clk_reg,hb);
}
```

/* this function sets the clock frequency of the desired clock. It calls clockdiv() */

```
void set_clk_freq(int clk_num,float clk_freq)
{
float base_freq = 2000000;
int divider;
divider = ceil(base_freq/clk_freq);
clockdiv(clk_num,divider);
}
```

The states of the three clock can be monitored through bits (CT2 CT1 and CT0) of the STATUS REGISTER.

5.0 Tying to interrupts

You may wire the following lines to some of the interrupt lines on the PC bus.

Bit	Status
CT0	Clock Timer 0 overflow
CT1	Clock Timer 1 overflow
CT2	Clock timer 3 overflow
EOC_I	End of conversion interrupt
DI0	Digital input bit 0

The interrupts lines that can be tied to are

Interrupt#	**Normally used by (on standard PC)**	**Vector address**
3	COM1: Serial port	0xB
5	**PC Fixed disk controller/ NOT NORMALLY USED** ***USED BY MultiQ factory Configuration***	**0xD**
7	LPT1: Printer	0xF
9	RESERVED	RESERVED
10	UNUSED	0x72
11	UNUSED	0x73
12	UNUSED	0x74
15	UNUSED	0x77

The interrupt lines should be physically connected on the board using the jumpers provided. You should be certain that no other device is tied to the interrupt line you want to use.

5.1 Writing interrupt service routines

The following is a short explanation on how to write interrupt service routines on an IBM PC compatible system. More detailed information can be obtained from any good book on PC architecture and the C compiler you are using.

An interrupt service routine (ISR) is initiated every time a specified interrupt line associated with the ISR goes high. The interrupt mask register on the PC is located at address 0x21. It is an eight bit port and in order to activate a certain interrupt line, you must write a '0' to the associated bit in the interrupt mask register. For example if you want to allow interrupts only from lines 2 and 7 you must write a (01111011) or (0x7b) to memory location 0x21.

Each interrupt lines causes a jump to the address located in the vector table shown above. For example if you want a function:

```
extern void interrupt far newtimer(void);
```

to be executed every time interrupt #5 occurs, you set it up in the following manner:

```
disable(); /* disable interrupts */
oldtimer = getvect(0xd); /*save old isr address, see vecctor in column 3 of table above */
setvect(0xd,newtimer); /* setup the new isr */
int_mask = inportb(0x21); /* get the present mask  */
outportb(0x21,int_mask&0xdf); /* write out a new mask that sets bit 5 to 0) */
enable(); /* enable interrupts, at this point newtimer is active and will be executed every time line 3 goes high */
```

Now what happens inside the isr is also very important.
The ISR should have the following structure:

```
extern void interrupt far newtimer(void)
{
asm fsave data87 /* assembly code to save floating point processor status */
_clear87(); /* clear the floating point processor */
disable(); /* disable other interrupts */

/* here is where you write your code */
/* this will be executed every time the interrupt occurs */

asm frstor data87 /* restore floating point processor status */
outportb(0x20,0x65); /* acknowledges interrupt to 8259 */
enable();
}
```

data87 must be declared globally as

```
char *data87[94];
```

6.0 Testing your board

Turn off the computer and install the MultiQ into an ISA slot in your PC. Make sure it is well seated and tighten the screw. Start the computer.

The programs 'test_mq3.c', ‘test_sw3.c' and the driver file 'mq3.drv' are supplied with the board. In order to compile these programs you need Turbo C and Turbo assembler. Executable versions are also supplied.

-make a directory MQ3 on your hard drive
- copy all files to that directory
- Run the program **test_mq3** or **test_sw3**

The program **test_sw3** uses the IBM PC SOFTWARE INTERRUPT and changes the speed of the realtime clock used for time of day. Run this program to test all the functions on the board except the realtime clocks. Reset the time of day from DOS if necessary.

The program **test_mq3** uses the MultiQ CLOCK #1 for hardware interrupts on interrupt line #5. In order to run this program you should install the jumper labelled CTC1 to pin interrupt 5.(FACTORY CONFIGURATION)

In order to test the encoder preload capability, the programs preload the encoder counters #0 to #7 with the values 0 to 70000 in increments of 10000 consecutively. It also outputs preset values to the D/A channels. You may loopback from a D/A output to an A/D input to ensure that the value being put out is being measured correctly.

The program is interactive. You can plot selected data in realtime. You can select the parameters associated with the letters in brackets [ ]. For example entering the letter [v] you will be prompted to enter a voltage which will be output to D/A channel specified by hitting [o]. The program is interrupt driven and all variables are monitored on the screen in realtime. The A/D channel associated with the letter [i] can be plotted in realtime as well as the encoder input associated with the letter [e]. Entering [z] will reset the encoder selected. You can output a digital word to the digital output port by entering [d] followed by a hexadecimal number. You can select the data you want to plot in realtime by hitting [V]. Hit 'x' to exit realtime plotting. Selecting [T] allows you to alter the duration of the x axis of the realtime plot.

[i]: A/D input channel number
[o]: D/A output channel number
[v]: output volts
[d]: Digital word out
[e]: Encoder channel
[z]: Encoder reset
[F]: Sampling Frequency(Hz)
[T]: X axis time duration(Sec)
[V]: Y axis variable: eg voltage or encoder
[R]: Realtime plot
[Q]: Quit program

6.1 Compiling the source code

If you need to change anything in the source code of the above programs, you will need to recompile them. Use the following lines to obtain new executable code.

Turbo C users

tcc -r -B -f87 -ml %1 graphics.lib

Borland C users

tcc -r -B -f87 -ml %1 graphics.lib

where %1 is the name of the program.

Note Turbo assembler (TASM) should be in the PATH as well as (TC\BIN) or (BC\BIN). The file EGAVGA.BGI must be present in the directory.

6.2 Execution speed

The program **`speed.c`** tests the speed of the various operations on the board using the drivers listed above. The speeds are obtained on a Pentium II 200MHz and may vary with the processor you use. The speeds do not solely depend on the board but also on execution time of the instructions in the drivers. For example, a single A/D conversion is performed in only 8 µseconds but it really takes 19.2 µseconds to actually acquire the data into the program. The extra time is due to bus access (inport, outport wait loops).

The results from 'speed.c' are tabulated below. These are calculated by obtaining an average over one million consecutive operations using the driver functions given above.

Function	Execution speed in μ seconds
Digital input (16 bit)	2
Digital output (16 bit)	2
Encoder read (24 bit)	7
Analog todigital conversion (13 bit)	19.2
Digital to analog conversion (12 bit)	5.0

The above information is important when determining what is the maximum sampling frequency you can set in an ISR. Suppose you would like to sample all 8 analog input channels, output to all 8 channels, read 8 encoders and perform 16 bit digital I/O in a single interrupt service routine. These would take approximately (2+2+7x6+19.2x8+5x8) = 239.6 microseconds. Therefore the ISR should be called at a frequency slower than 4.1 Khz. If you want to perform calculations in an ISR (which you typically would for a controller), then the time for calculations should also be taken into consideration. Assuming the calculations you are performing take another 230 microseconds, then the ISR should execute slower than 2 Khz. You would also like to have some time left over for foreground jobs. Lets's say 50% of processor time left for foreground operations(plotting, user interaction, etc), then the ISR should be set to a maximum of 1 kHz. This is the suggested maximum sampling frequency when the board is being used at full capacity. Of course, the less channels are use and the simpler the controller, the faster you can set the speed of the ISR.

7 WIRING

7.1 Flat ribbon Cable and Terminal board

The flat ribbon cables must be inserted into the MULTIQ board and the TERMINAL board with the correct orientation. Note the small ridges on the flat ribbon cable ends. Do not force the connector in!

7.2 Wiring to an A/D

The inputs to the A/D's are available at the RCA plugs labelled Analog inputs. The outside shield of the connector is the ground. The inside is the signal. Wire the analog signal from a source in the range of of +/- 5 Volts . Wire a common ground to the top part of the header.

7.3 Wiring to a D/A

The outputs from the D/A's are available at the RCA plugs labelled Analog outputs. The outside shield of the connector is the ground. The inside is the signal.

7.4 Wiring to the Encoder Inputs

The encoders are powered from the terminal board using the +5 volt supply from the computer. Use the wiring diagram in section 3.2.1 as a guide.

7.5 Wiring to Digital Inputs

The digital inputs are applied to the terminal block labelled DIGITAL INPUTS.

7.6 Wiring to Digital Outputs

The digital outputs are available at the terminal block labelled DIGITAL OUTPUTS.

7.7 Selecting interrupt sources

Select the interrupt source using the jumpers at the headers labelled **Interrupts**. USE ONE INTERRUPT JUMPER PER HEADER.

1) Placing a jumper at the header labelled A/D will cause an interrupt from EOC_I to occur at the line to which the jumper is attached. ie if the jumper is attached between A/D and the pin labelled '5', an EOC_I will cause an interrupt number 5 to occur.

2) Placing a jumper at the header labelled CTC0 will cause an interrupt from CLOCK 0 to occur at the line to which the jumper is attached. ie if the jumper is attached between CTC0 and the pin labelled '5', an interrupt number 5 will occur at the frequency at which CLOCK 0 is operating.

3) Placing a jumper at the header labelled CTC1 will cause an interrupt from CLOCK 1 to occur at the line to which the jumper is attached. ie if the jumper is attached between CTC1 and the pin labelled '5', an interrupt number 5 will occur at the frequency at which CLOCK 1 is operating. **THIS IS FACTORY CONFIGURATION.**

4) Placing a jumper at the header labelled CTC2 will cause an interrupt from CLOCK 2 **OR FROM** Digital Input #0 to occur at the line to which the jumper is attached. The source depends on the jumper labelled CTC2INT. If the jumper is between pins (12) then the source is clock 2, if the jumper is between (23) then the source is DIN0. With the CTC2INT jumper located at (12) and one jumper at (CTC2,5) an interrupt number 5 is generated at the speed of CLOCK2. With the CTC2INT jumper located at (23) and one jumper at (CTC2,5) an interrupt number 5 is generated every time the digital input number 0 is pulsed from LOW to HIGH to LOW