

# Engineering 6806

## Project Design Labs in Electrical/Computer Engineering

### Lab #3 – Motor Control

#### 1. Introduction

This lab will introduce you to basic concepts of controlling motors using the PIC16F877.

#### 2. A Closed Loop Motor Controller

A closed-loop motor controller is a common means of maintaining a desired motor speed under varying load conditions. In the closed-loop motor controller (Figure 1) a signal proportional to the motor speed is fed into the input where the feedback is subtracted to produce an error signal. This error signal is then input to the controller where a signal is generated and sent to the pulse width modulator (PWM). The PWM signal is then sent directly to the motor.

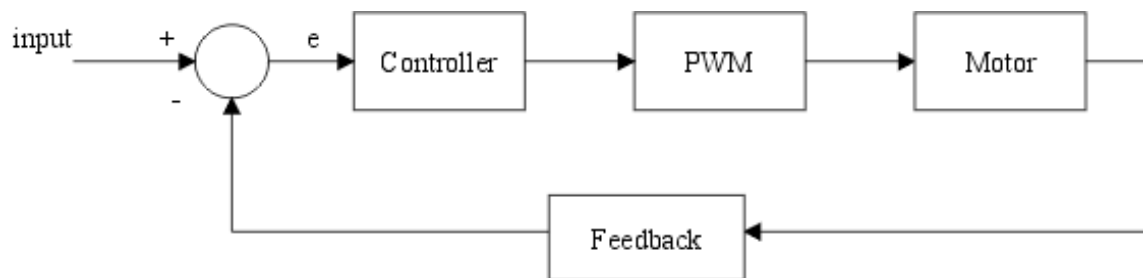


Figure 1: A closed-loop motor controller

#### *Why a PWM signal?*

The speed of a DC motor can be varied by changing the average voltage applied to the input. A pulse width modulated signal is created by switching the output on and off at some duty cycle. For example in Figure 2(a) the output is switched on for a short time duration (lower duty cycle) during each cycle. The result is a low average voltage seen by the motor resulting in a slower speed. In Figure 2(b) the output is switched on for a longer duration (higher duty cycle) during each cycle resulting in a high average voltage and higher motor speed.

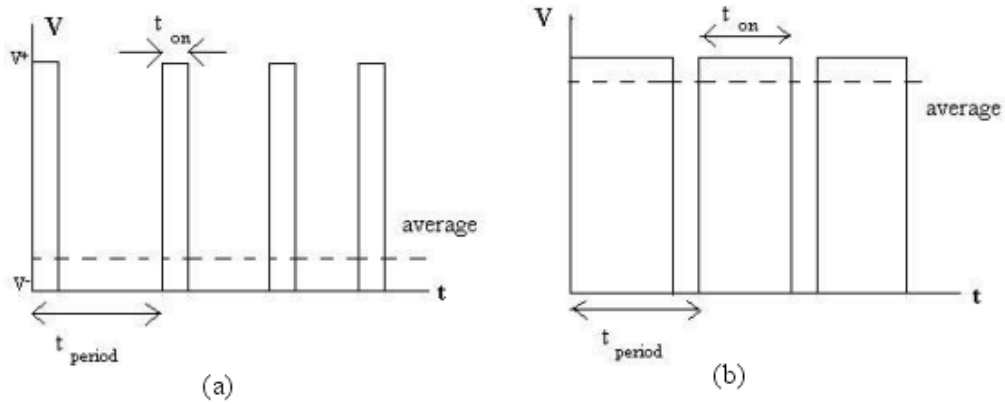


Figure 2: PWM signals.

In the next sections we you look at the controller and the feedback independently and develop programs to:

1. Control the motor in open-loop mode.
2. Decode the motor encoders in the feedback loop.
3. Design and implement a closed loop motor controller.

### 3. Open Loop Motor Controller

Using the MAD card and the LMD18200 H-Bridge connect the circuit as shown in Figure 3. The connection diagram for the motor is shown in Appendix A.

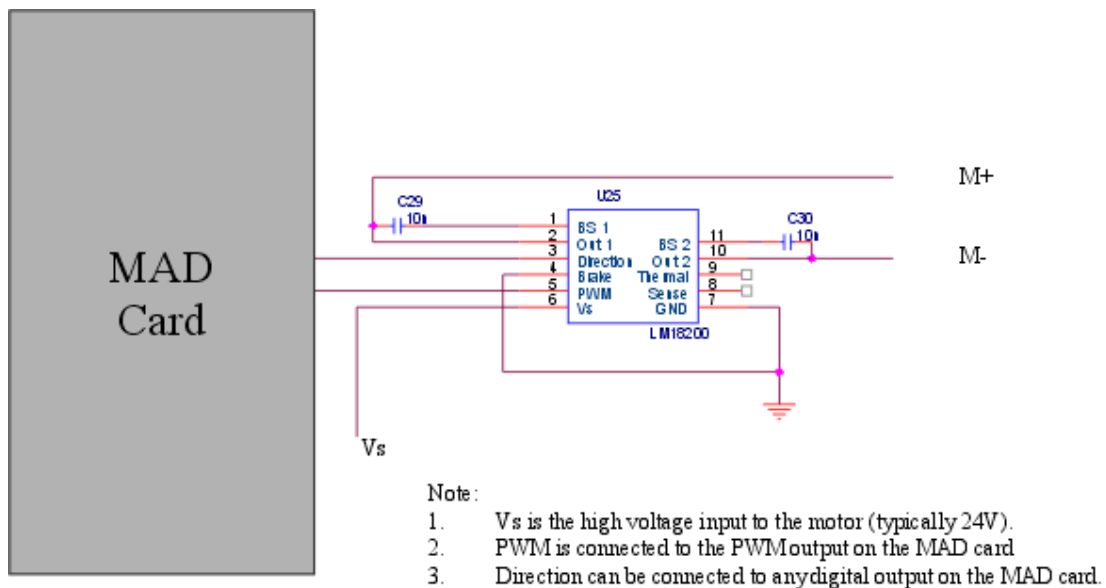
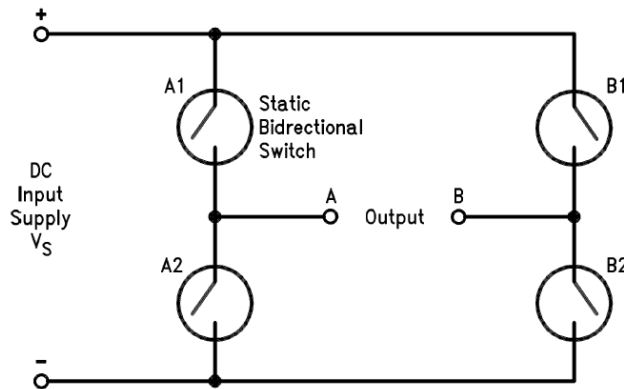


Figure 3: Schematic for connecting the LM18200 to the MAD card.

Write a program that accepts input from the keyboard to vary the duty cycle of the motor. See sample code in appendix B for PWM and serial port examples.

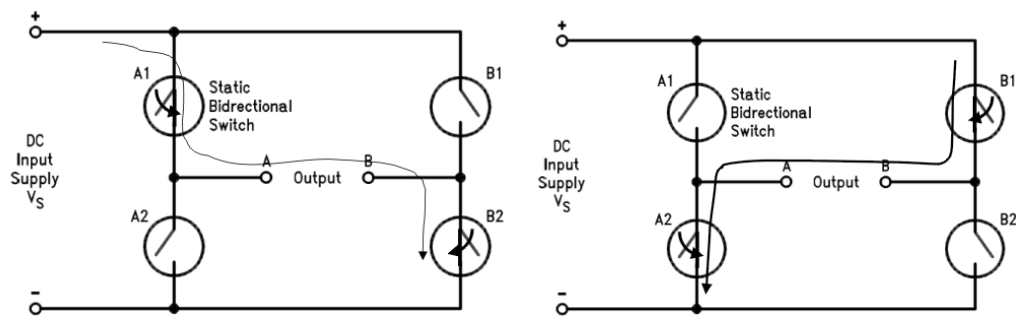
### ***How does a H-Bridge work?***

Controlling the speed of the motors will be very important for this project. The PIC micro-controller has two built in pulse width modulators that can be used to drive an h-bridge allowing you to vary the speed of the motor. By varying the duty-cycle of the pulse width modulated signal the average voltage seen by the motor can be varied thus varying the speed.



**Figure 4: H-Bridge circuit**

The H-Bridge takes a DC supply voltage and provides 4-quadrant control to a load connected between two pairs of power switching transistors. Because the switches allow current to flow bi-directionally, the voltages across the load (A-B) can be of either polarity. For example consider the circuit in Figure 4. When switches A1 and B2 are closed current will flow through the output from A to B. Similarly when switches B1 and A2 are closed current will flow from B to A.



**Figure 5: Controlling the switches on the h-bridge can reverse Motor.**

Fortunately H-Bridges are very popular and several pre-packaged chips are available that simplify motor control. The H-Bridge used on this project is the LMD18200. It simply requires two inputs, a digital input for setting the direction and a pulse width modulated input for controlling speed as shown in Figure 9.

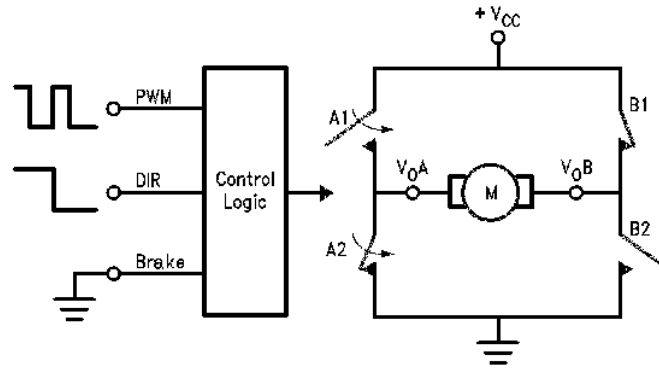


Figure 6: LMD18200 inputs.

#### 4. Decoding the Encoder for Feedback

In this section you will be required to write a program to decode the encoder and print the number of motor revolutions back to the hyper-terminal display. To simplify this task a fixed voltage can be applied to the motor input from a lab power supply. The encoder lines will should be connected to any digital inputs on the MAD card as shown in figure 5.

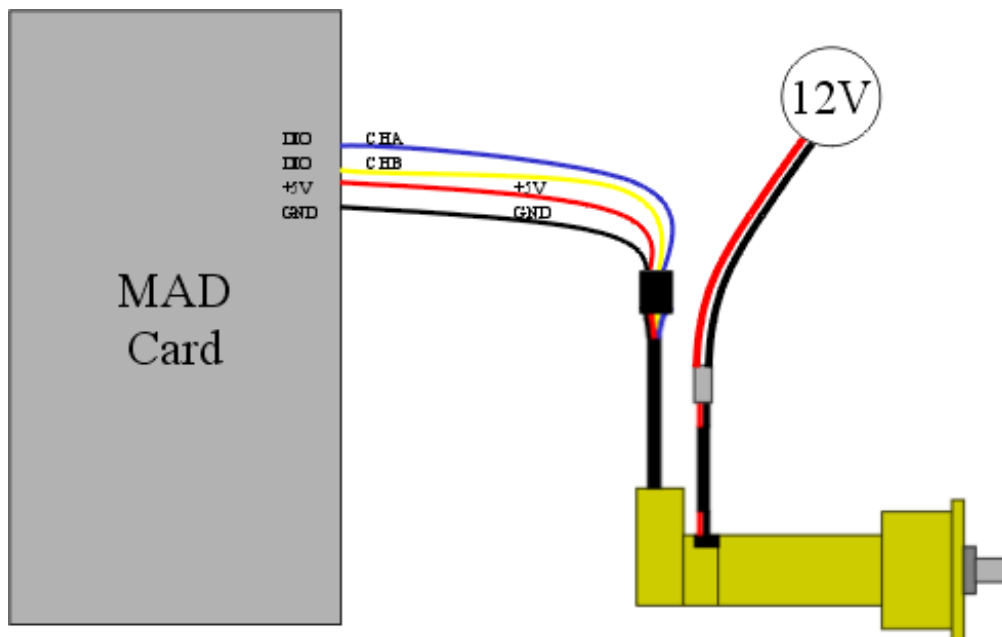


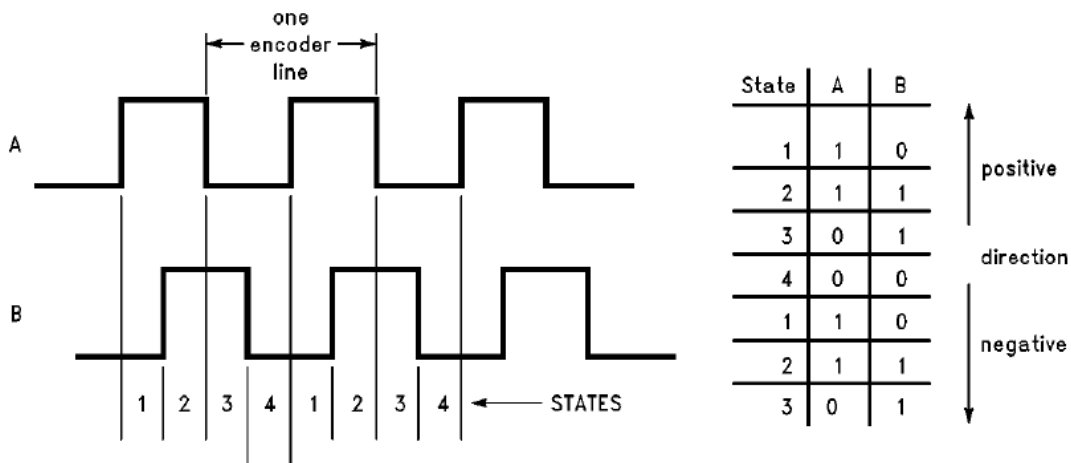
Figure 7: Connecting the quadrature encoder to the MAD card.

### *Decoding a quadrature encoder*

The output of a quadrature encoder consists of two square waves separated by 90 degrees. This results in four unique states providing enough information to determine both the number of revolutions of the motor shaft as well as the direction of travel. The basic algorithm for decoding a quadrature encoder is as follows:

1. Initialize by sampling CHA and CHB and set old\_state.
2. Sample CHA and CHB and set new\_state.
3. Compare new\_state to old\_state and update count (see Figure 4)

In order to ensure adequate sampling rate you should determine the minimum required sampling rate and use the *rtcc* timer to create an interrupt at a rate equal to or greater than this sampling rate. When decoding the encoders it should be noted that the encoders are on the input shaft of the motor. The maximum speed of the output shaft is 800 rpm at 30VDC. Since there is a 5.9:1 gear ratio this means the input shaft has a maximum speed of 4720 rpm. Also note that the encoders have 512 increments, however because the above method of decoding results in 4 counts per encoder increment the total number of counts per revolution is  $4 \times 512 = 2048$  counts/rev. Also since you will be using 24V supply the maximum speed of the motor will be reduced to approximately  $4720 \times 24/30 = 3775$ .



**Figure 8: Quadrature encoder output.**

## 5. Closed Loop Motor Control

In this section you will combine the feedback algorithm with the open loop controller to produce a closed loop speed controller. A block diagram of the system you are to design is shown in Figure 6.

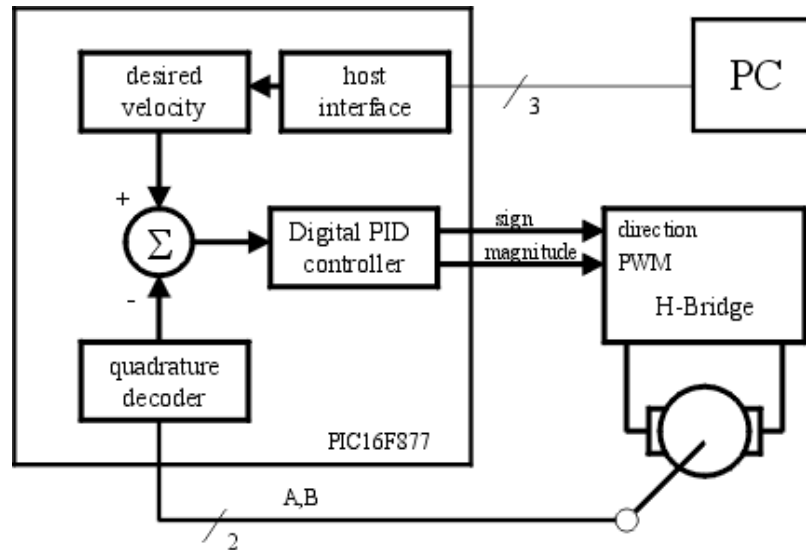


Figure 9: Closed loop motor controller block diagram.

One of the key results from this lab should be an understanding of the requirements for performing real-time procedures on a micro-controller. Depending on the efficiency of your implementation it may not be possible to decode the encoders and perform various other functionality on a single PIC micro-controller. One of the challenges of this project will be to design a method of decoding the encoders external to the MAD board.

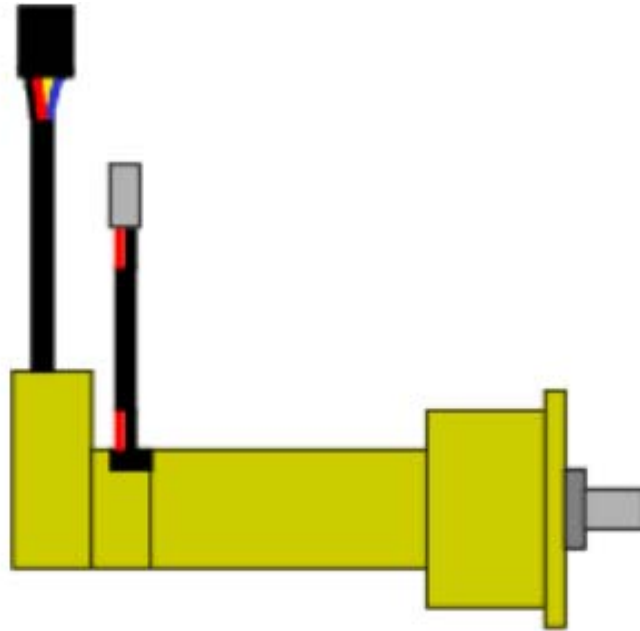
## Appendix A – Motor Connection Diagram

Encoder

CHA	Blue
CHB	Yellow
+5V	Red
GND	Black

Motor

M+	Red
M-	Black



## Appendix B – Code Examples

```
////////////////////////////////////
////                                  EX_PWM.C                                  ////
////                                  ////
//// This program will show how to use the built in PIC PWM.                ////
//// The program takes an analog input and uses the digital                  ////
//// value to set the duty cycle. The frequency is set by                    ////
//// the user over the RS-232.                                              ////
////                                  ////
//// Configure the CCS prototype card as follows:                             ////
////   Connect a scope to pin 3 (C2)                                         ////
////   Connect 9 to 15 (pot)                                                 ////
////   See additional connections below.                                       ////
////                                  ////
////////////////////////////////////

#if defined(__PCM__)
#include <16C74.H>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BRGH1OK) // Jumpers: 8 to 11, 7 to 12

#elif defined(__PCH__)
#include <18C452.H>
#fuses HS,NOPROTECT
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BRGH1OK) // Jumpers: 8 to 11, 7 to 12
#endif

main() {
    char selection;
    byte value;

    printf("\r\nFrequency:\r\n");
    printf("  1) 19.5 khz\r\n");
    printf("  2) 4.9 khz\r\n");
    printf("  3) 1.2 khz\r\n");

    do {
        selection=getc();
    } while((selection<'1')||(selection>'3'));

    setup_ccpl(CCP_PWM); // Configure CCP1 as a PWM

    // The cycle time will be (1/clock)*4*t2div*(period+1)
    // In this program clock=10000000 and period=127 (below)
    // For the three possible selections the cycle time is:
    // (1/10000000)*4*1*128 = 51.2 us or 19.5 khz
    // (1/10000000)*4*4*128 = 204.8 us or 4.9 khz
    // (1/10000000)*4*16*128= 819.2 us or 1.2 khz

    switch(selection) {
        case '1' : setup_timer_2(T2_DIV_BY_1, 127, 1);
                    break;
        case '2' : setup_timer_2(T2_DIV_BY_4, 127, 1);
                    break;
        case '3' : setup_timer_2(T2_DIV_BY_16, 127, 1);
                    break;
    }

    setup_port_a(ALL_ANALOG);
    setup_adc(adc_clock_internal);
    set_adc_channel( 0 );
    printf("%c\r\n",selection);

    while( TRUE ) {
```

```

        value=read_adc();

        printf("%2X\r",value);
        set_pwm1_duty(value);
        // This sets the time the pulse is high each cycle. We use the A/D
        // input to make a easy demo. the high time will be:
        // if value is LONG INT:
        //   value*(1/clock)*t2div
        // if value is INT:
        //   value*4*(1/clock)*t2div
        // for example a value of 30 and t2div
        // of 1 the high time is 12us
        // WARNING: A value to high or low will
        //           prevent the output from
        //           changing.
    }

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                               EX_SISR.C                               ////
////                               ////
//// This program shows how to implement an interrupt service          ////
//// routine to buffer up incoming serial data.                        ////
////                               ////
//// If the PIC does not have an RDA interrupt pin, B0 may be used    ////
//// with the INT_EXT.                                                ////
////                               ////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#if defined(__PCM__)
#include <16C74.H>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Jumpers: 8 to 11, 7 to 12

#define BUFFER_SIZE 32
byte buffer[BUFFER_SIZE];
byte next_in = 0;
byte next_out = 0;

#int_rda
void serial_isr() {
    int t;

    buffer[next_in]=getc();
    t=next_in;
    next_in=(next_in+1) % BUFFER_SIZE;
    if(next_in==next_out)
        next_in=t; // Buffer full !!
}

#define bkbhit (next_in!=next_out)

byte bgetc() {
    byte c;

    while(!bkbhit) ;
    c=buffer[next_out];
    next_out=(next_out+1) % BUFFER_SIZE;
    return(c);
}

main() {

    enable_interrupts(global);

```

```
enable_interrupts(int_rda);

printf("\r\nRunning...\r\n");

        // The program will delay for 10 seconds and then display
        // any data that came in during the 10 second delay

do {
    delay_ms(10000);
    printf("\r\nBuffered data => ");
    while(bkbhit)
        putc( bgetc() );
} while (TRUE);

}
```