Inverted Pendulum Microprocessor and FPGA Manual

Sheldon Logan

July 3, 2006

1 Table of Contents

2 Table of Figures

3 Introduction

The following manual contains information on all the PIC microprocessor files and FPGA Verilog Code associated with the inverted pendulum project. The Manual also contains information on how to program the aforementioned microprocessor and FPGA. Instructions on how to adjust settings in the encoder counter and the inverted pendulum C code can be found in the Appendix of the manual.

4 PIC Programming Instructions

The first task in programming the PIC microcontroller is opening a project which is accomplished by clicking on project->open and browsing for the desired project. When the project is opened a project window should appear in the top left hand corner of the screen. The project window should contain a tab called source files with the files associated with the project being displayed under the aforementioned tab. If the user wants to make changes to the file, double click on the file to open it in a editor. After changes have been made to the file, rebuild the project by pressing ctrl and f10. The user should now connect the MPLAB ICD programmer to the board as shown in Figure 1.

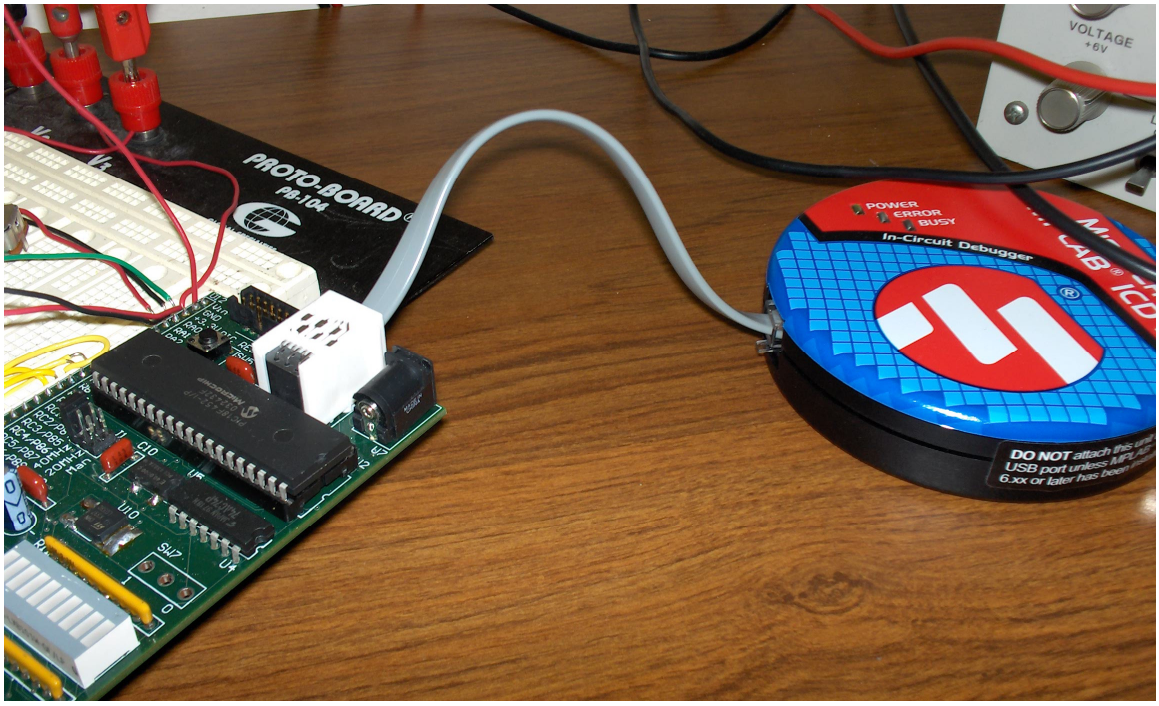

Figure 1: MPLAB ICD Programmer connected to Breadboard

The user should then power on the board by turning on the 5V power supply.

After the project has been successfully built the user should now click on programmer->select programmer->MPLAB ICD 2. The user should then click programmer-> connect

if the MPLAB programmer failed to connect to the device. The next step to programming the PIC is to click programmer->program. After the program is downloaded the user should click programmer-> release from reset.

5 FPGA Programming Instructions

The first task in programming the FPGA is opening a project which is accomplished by clicking on file->open project and browsing for the desired project. When the project is opened several windows appear. The sources window in the top left hand corner of the screen contains the files associated with the project. These files may be edited by double clicking on them and making the necessary adjustments when the open up in the window on the right of the screen. After all the changes have been made the user should click the top level file (ecount.v) in the sources window then double click Generate Programming file in the Processes window. If no errors occur during Synthesize or Implentation the user can proceed to PROM file generation. If they are errors the user can access the error reports by double clicking View synthesis report under the Synthesis-Synplify Pro tab. The user can also access the implantation error reports by double clicking the translation, map, or place and route report under the implement design tab. After all error have been resolved the user should now double click on Generate PROM, ACE or JTAG file under the Generate Programming File tab. When the IMPACT window opens, the user should click select prepare a PROM file then click next. The user should enter a name for the PROM file in the PROM File Name box at the bottom of the screen and then click next. The user should now select Auto Select Prom at the top of the screen and then click next. Then the user should select finish and then click ok when the Add Device window pops up. The user should now select to add the .bit file that is displayed in the directory and then click no when asked to add another device. Next, the user should click Generate File in the Impact Processes window. After the PROM file has been generated the user should close the impact window and click no when asked if they want the project to be saved.

Now the user should select Configure Device(IMPACT) in the Generate Programming File tab. The user should now connect the JTAG programmer to the breadboard as shown in Figure 2 and then turn on the 5V power supply to the board.

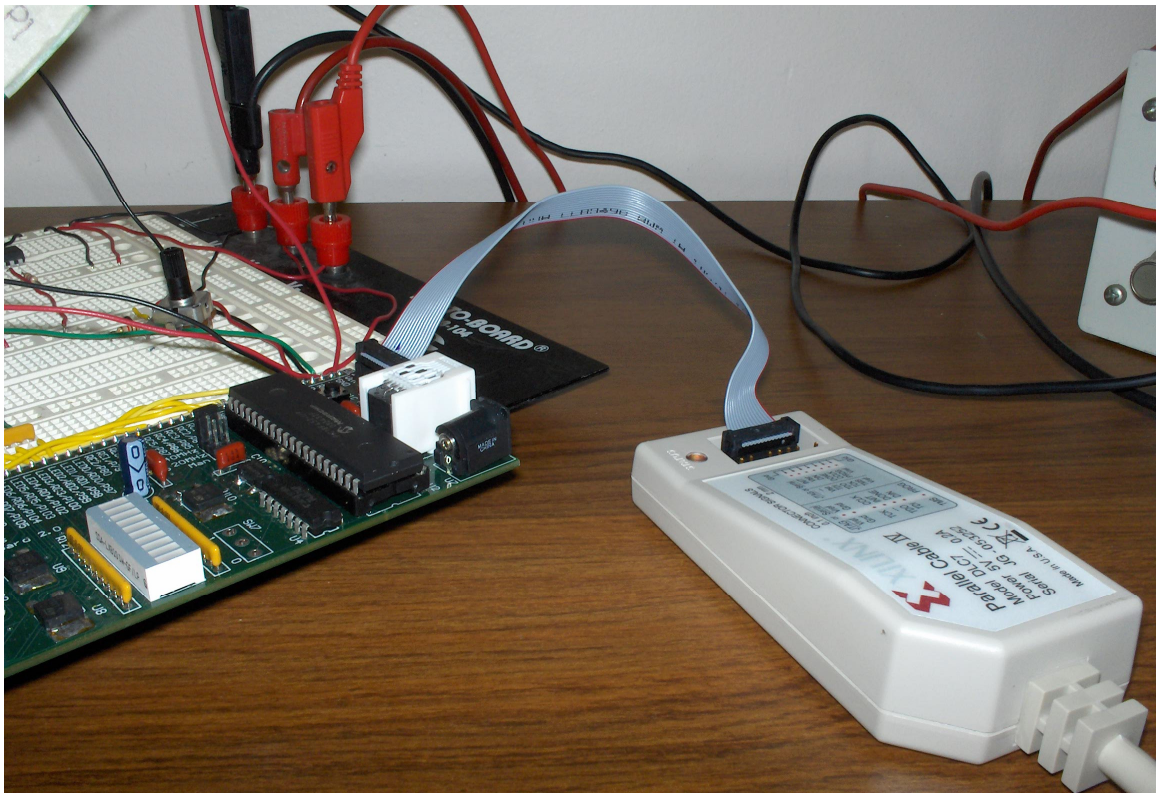

Figure 2: JTAG Programmer connected to Breadboard

The user should now select Configure device using Boundary-Scan(JTAG) then click finish. The user should now select the .mcs file that was created. The user should now click on the XILINX FPGA icon in the IMPACT window. Next the user should click Program in the iMPACT processes window. The user should then click ok. Programming the FPGA sometimes produces verifying errors hence the user may have to attempt to program the FPGA several times before the FPGA is programmed successfully. After the FPGA has been programmed the user should hit the XILINX reset button on the PCB as shown in Figure 3 below.

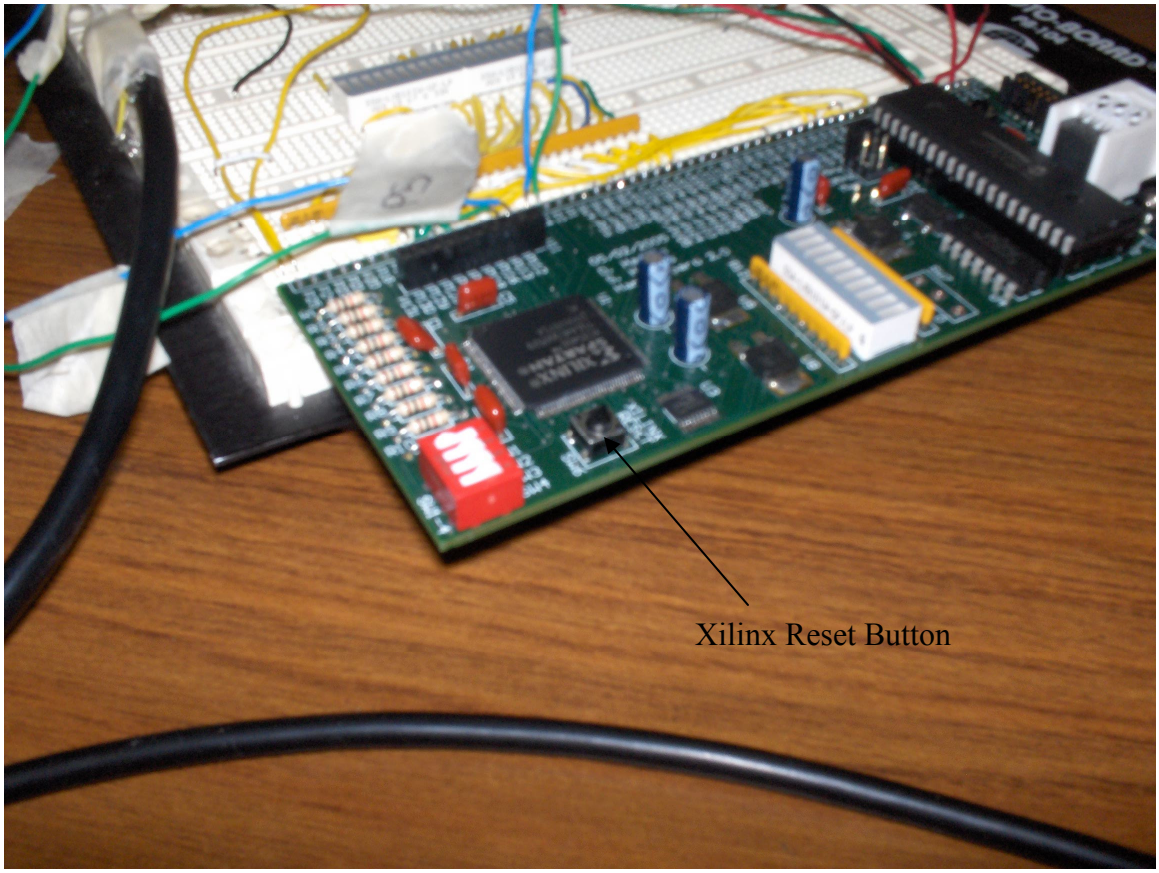

Figure 3: Xilinx Reset Button

6 Appendix A: Verilog Files and Schematics

The two breadboards used in the Inverted Pendulum Controller have two separate FPGA programs on them. The board with the D/A converters has a special module that outputs the pendulum and motor position data in a format that is readable by the converters. The board with the motor drive circuit has a special module that is used to drive the H-Bridge. This bread-board is also wired to transfer the data from the FPGA to the PIC microprocessor.

6.1 Base Verilog Modules

Even though the Breadboards have separate FPGA programs they share many modules. The modules that are included in both programs are shown below. Information on how to adjust these modules are also included.

6.1.1 pendquad

The pendquad verilog file is used to count the pendulum encoder. This encoder produces 2048 pulses per revolution. The resolution of the encoder was increased to 8192 by using quadrature encoder counting. The pendquad module has the ability to count up to 8191 (2^{13} - 1) before it overflows back to zero. If the resolution of the pendulum encoder changes the user should change specific values in the pendquad module. These values have been highlighted in red and blue. If the new encoder does not count to a value of 2^n the user should refer to the motorquad module. If the motor does count to a value of 2^n, then the user should change the numbers highlighted in red to n-1, and the number highlighted in blue to n.

```
module pendquad(clk, quadA, quadB, count,reset);
        input clk, quadA, quadB, reset;
        output [12:0] count;

//pendulum position counter

        reg [2:0] quadA_delayed, quadB_delayed;
        always @(posedge clk) quadA_delayed <= {quadA_delayed[1:0], quadA};
        always @(posedge clk) quadB_delayed <= {quadB_delayed[1:0], quadB};

        wire count_enable = quadA_delayed[1] ^ quadA_delayed[2] ^ quadB_delayed[1]
^ quadB_delayed[2];
        wire count_direction = quadA_delayed[1] ^ quadB_delayed[2];

        reg [12:0] countemp;
        always @(posedge clk or posedge reset)
        if(reset) countemp <= 13'b0;
        else
        begin
                if(count_enable)
                        begin
                        if(count_direction) countemp<=countemp+1;
                                else countemp<=countemp-1;
                                end
                end

  assign count = countemp;


endmodule
```

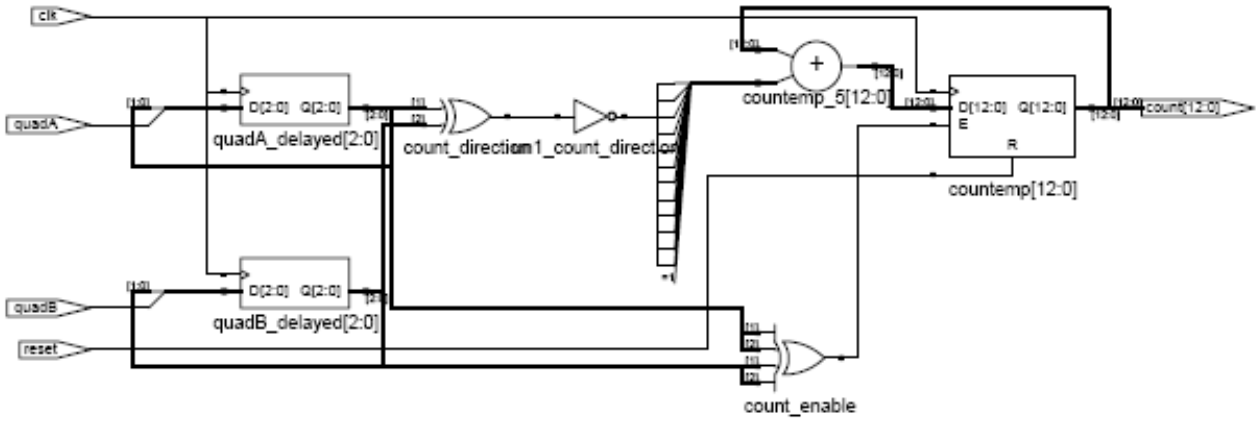

Figure 4: Pendquad RTL Schematic

6.1.2 motorquad

The motorquad verilog file is used to count the motor encoder. This encoder produces 1000 pulses per revolution. The resolution of the encoder was increased to 4000 by using quadrature encoder counting. The motorquad module has the ability to count up to 4000 before it overflows back to zero. If the resolution of the motor encoder changes the user should change specific values in the motorquad module. These values have been highlighted in red, blue and green. If the new encoder counts to a value of 2^n, the pendquad module can be used to count the motor position. If the new encoder does not count to a value of 2^n the user should make the following changes to the motorquad module. The numbers highlighted in red should be changed to n, where n is the smallest 2^n number larger than the maximum counts per revolution. The blue number should be changed to n. The green numbers should be changed to cpr -1 where cpr is the maximum counts per revolution. It should be noted that the red numbers and blue numbers for pendquad and motorquad have to be the same. Consequently the red numbers and blues numbers should be the largest n-1 and n respectively between motorquad and pendquad.

```
module motorquad(clk, quadA, quadB, count,reset);
        input clk, quadA, quadB, reset;
        output [12:0] count;

        reg [2:0] quadA_delayed, quadB_delayed;
        always @(posedge clk) quadA_delayed <= {quadA_delayed[1:0], quadA};
        always @(posedge clk) quadB_delayed <= {quadB_delayed[1:0], quadB};

        wire count_enable = quadA_delayed[1] ^ quadA_delayed[2] ^ quadB_delayed[1]
^ quadB_delayed[2];
        wire count_direction = quadA_delayed[1] ^ quadB_delayed[2];

        reg [12:0] countemp;
        always @(posedge clk or posedge reset)
```

```
        if(reset) countemp <= 13'b0;
        else
                begin
                        if(count_enable)
                                begin
                                        if(count_direction)
                                                begin
                                                if(countemp == 3999)
                                                countemp <=0;
                                                else
                                                countemp<=countemp+1;
                                                end
                                        else
                                                begin
                                                if(countemp == 0)
                                                countemp <= 3999;
                                                else
                                                countemp<=countemp-1;

                                                end
                                        end

                                end


  assign count = countemp;

endmodule
```

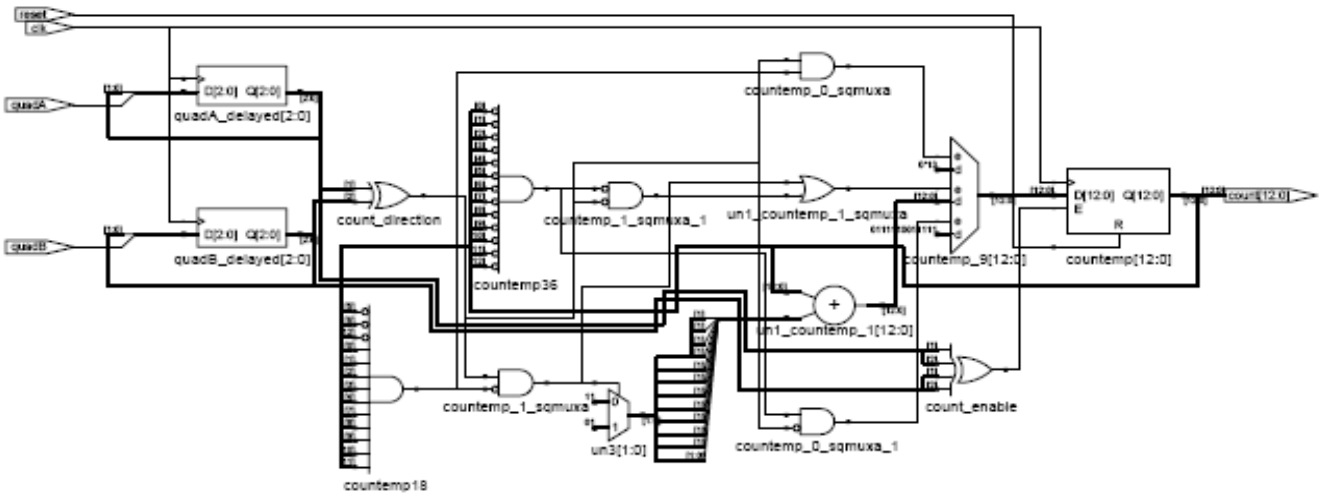

Figure 5: Motorquad RTL Schematic

6.1.3 enflopr

The enflopr verilog module is used to store the values of the pendquad and motorquad module in an enable flip-flop. The highlighted red numbers should be changed to the red number in motorquad or pendquad if they change. The same applies to the highlighted blue number.

```
module enflopr(clk, reset, en, d, out);

        input clk;
        input reset;
        input en;
        input [12:0] d;
        output [12:0] out;

        reg [12:0] q;
        always @(posedge clk or posedge reset)
                if (reset) q <= 13'b0;
                else if (en) q <= d;
        assign out = q;
endmodule
```

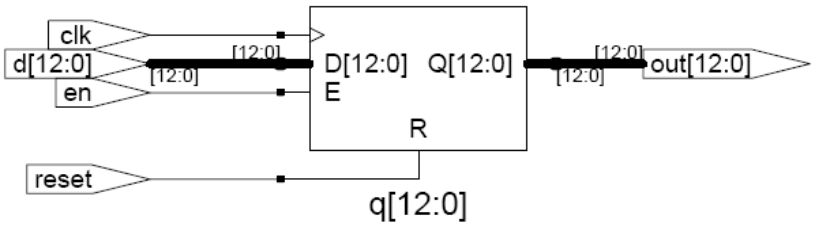

Figure 6: Resetable Enable Flip Flop RTL Schematic

6.1.4 mux2

The mux2 verilog module is used to create a 4 input mux. The highlighted red numbers should be changed to the red number in motorquad or pendquad if they change.

```
module mux2(d0, d1, s, y);

        input [12:0] d0, d1;
```

```
        input s;
        output [12:0] y;
        assign y = s ? d1 : d0; // if s=1, y=d1, else y=d0

endmodule
```

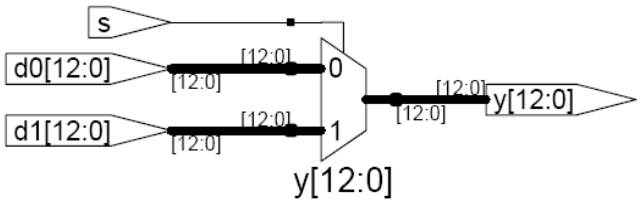

Figure 7: 2 Input 13 Bit Mux RTL Schematic

6.1.5 mux4

The mux4 verilog module was created to allow the microprocessor to switch between the pendquad and motorquad outputs by using a 4 input mux. The highlighted red numbers should be changed to the red number in motorquad or pendquad if they are change.

```
module mux4(d0, d1, d2, d3, s, y);

        input [12:0] d0, d1, d2, d3;
        input [1:0] s;
        output [12:0] y;
        wire [12:0] low, high;

        mux2 lowmux(d0, d1, s[0], low);
        mux2 highmux(d2, d3, s[0], high);
        mux2 finalmux(low, high, s[1], y);

endmodule
```

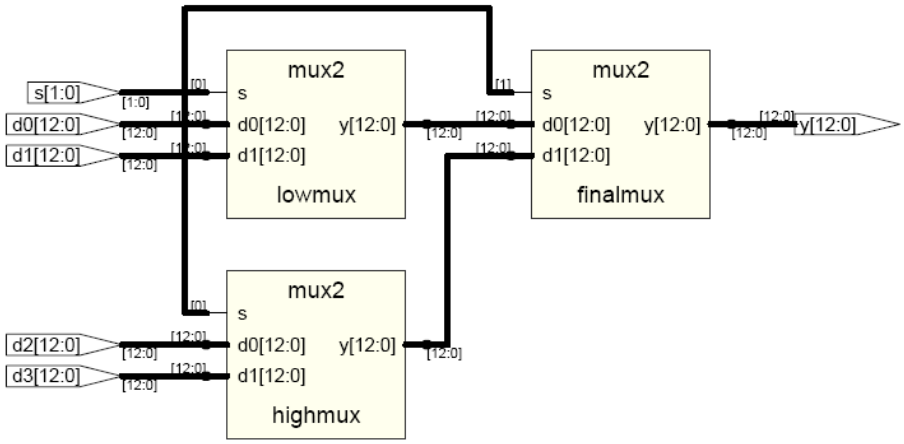

Figure 8: 4 Input 13 Bit Mux RTL Schematic

6.2 Motor Drive Verilog Modules

6.2.1 ecount

The ecount verilog module is the top-level module for the encoder counter. This module integrates all the base modules into the encoder counter utilized to control the inverted pendulum system. The highlighted red numbers should be changed to the red number in motorquad or pendquad if they are changed. Also the highlighted blue number should be changed to the red number in motorquad or pendquad if they are changed.

```
module ecount(clk,A1,reset,B1, A2, B2, enable, sel, vreset1, vreset2,
evalue,pwm,dir,y1,y2,trien);

      input clk;
      input reset;
      input A1;
      input B1;
      input A2;
      input B2;
      input enable;
      input [1:0] sel;
      input vreset1;
      input vreset2;
      input pwm,dir,trien;
      output y1,y2;
      output [12:0] evalue;

      assign y1 = pwm&dir;
      assign y2 = pwm&~dir;
```

```
        wire [12:0] count1temp;
        wire [12:0] count2temp;
        wire [12:0] count3temp;
        wire [12:0] count4temp;

        pendquad pendpos(clk, A1, B1, count1temp,reset);
        motorquad motorpos(clk, A2, B2, count2temp,reset);
        pendquad pendvel(clk, A1, B1, count3temp,vreset1);
        motorquad motorvel(clk, A2, B2, count4temp,vreset2);

        wire [12:0] count1t;
        wire [12:0] count2t;
        wire [12:0] count3t;
        wire [12:0] count4t;

        enflopr enflopr1(clk, reset, enable, count1temp, count1t);
        enflopr enflopr2(clk, reset, enable, count2temp, count2t);
        enflopr enflopr3(clk, reset, enable, count3temp, count3t);
        enflopr enflopr4(clk, reset, enable, count4temp, count4t);

        wire [12:0] evaluet;

        mux4 encoutmux(count1t, count2t, count3t, count4t, sel, evaluet);
        assign evalue = trien ? evaluet: 13'bz;

    endmodule
```

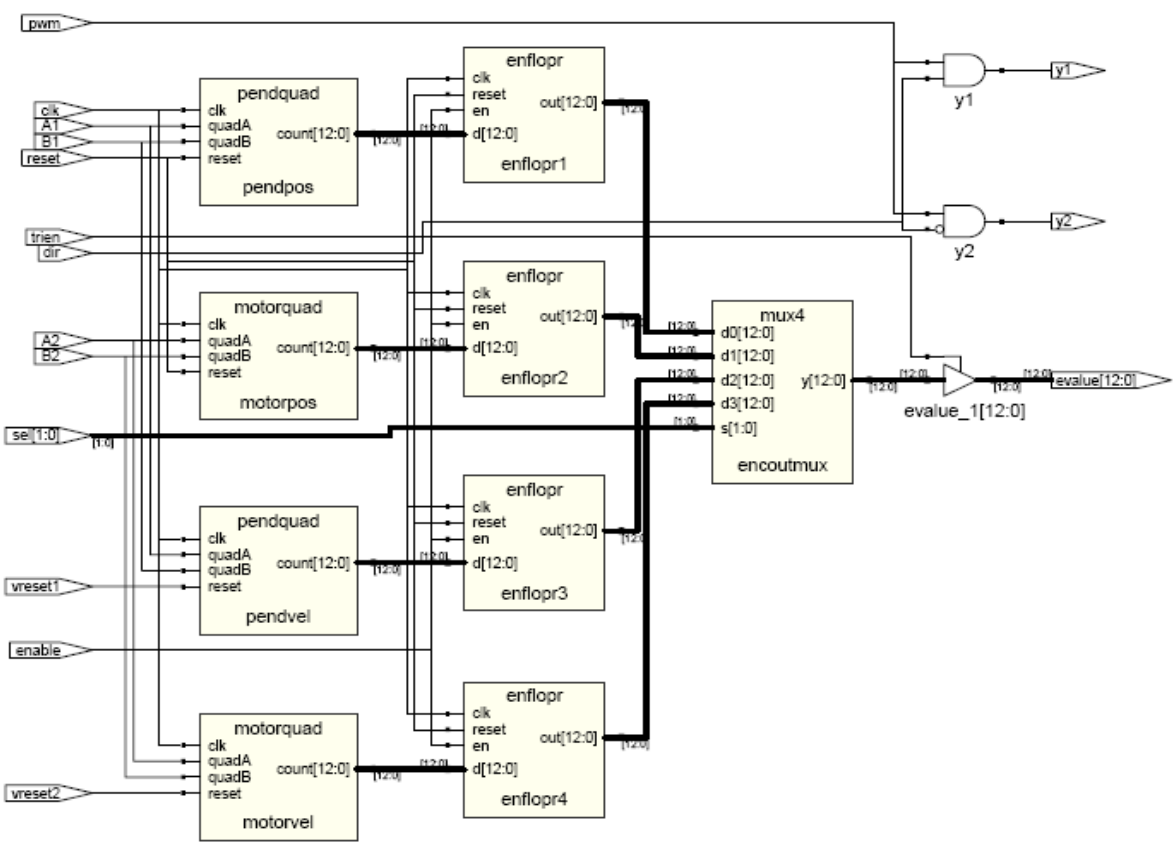

Figure 9: Encoder Counter (Motor Drive Board) RTL Schematic

6.3 D/A Verilog Modules

6.3.1 pulse16

The pulse16 verilog module is used to create the SYNC input signal for the D/A converters.

```
module pulse16(clk,reset,rpuls);

      input clk;
      input reset;
      output rpuls;

      reg[4:0] q;

      always @(posedge clk, posedge reset)
                        if (reset) q <=5'b0;
                              else q <= q+1;

            assign rpuls = q[4];

endmodule
```

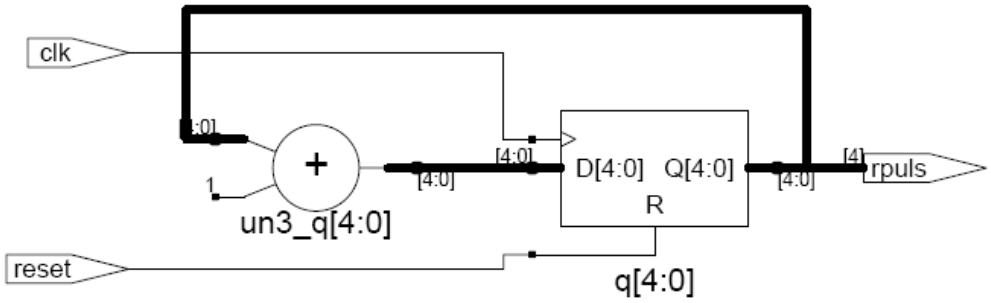

Figure 10: Pulse16 RTL Schematic

6.3.2 clkreduce

The clkreduce verilog module is used create the SCLK signal for the D/A converters. It reduces the 40 MHz clock used to run the FPGA to an 8 MHz clock.

```
module clkreduce(clk,reset,rclk);
      input clk;
      input reset;
      output rclk;
```

```
      reg[3:0] q;

            always @(posedge clk, posedge reset)
                        if (reset) q <=4'b0;
                              else q <= q+1;

            assign rclk = q[3];

endmodule
```

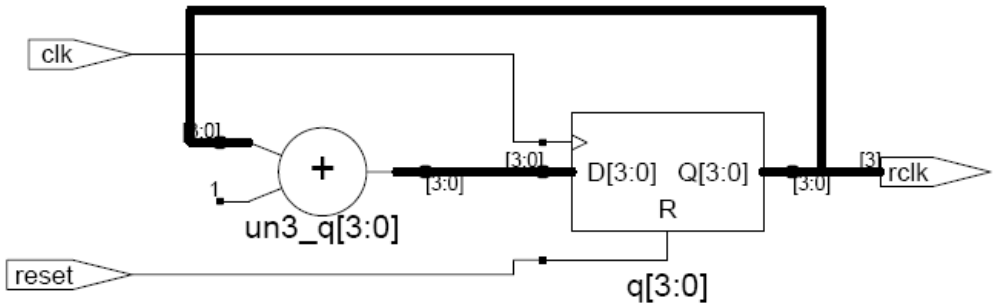

Figure 11: clk reduce RTL schematic

6.3.3 ShiftReg

The ShiftReg verilog module is used to create the SDIN input for the D/A converters. It essentials uses a shift register to send the encoder counter data to the converters.

```
module ShiftReg(clk,idata,odata,sclk,reset,dsync);

      input clk,reset;
      input [11:0] idata;
      output odata;
      output sclk;
      output dsync;

      wire rclk;
      wire sen;

      clkreduce clkreduce1(clk,reset,rclk);
      pulse16 pulse161(rclk,reset,sen);

      assign sclk = rclk;

      reg[15:0] q;

      always@(posedge rclk)
      begin
            if(sen)
                  q = {4'b0,idata[11:0]};
```

```
            else
                  begin
                        q = {q[14:0],1'b0};
                  end
            end

      wire odatatemp;
      assign odatatemp = q[15];

      reg otemp1;
            always @(posedge rclk)
                        otemp1 <= odatatemp;

      reg otemp2;
            always @(posedge rclk)
                        otemp2 <= otemp1;

            assign odata  = otemp2;

      reg syn1;
            always @(posedge rclk)
                        syn1 <= sen;

      reg syn2;
            always @(posedge rclk)
                        syn2 <= syn1;

            assign dsync = syn2;

endmodule
```

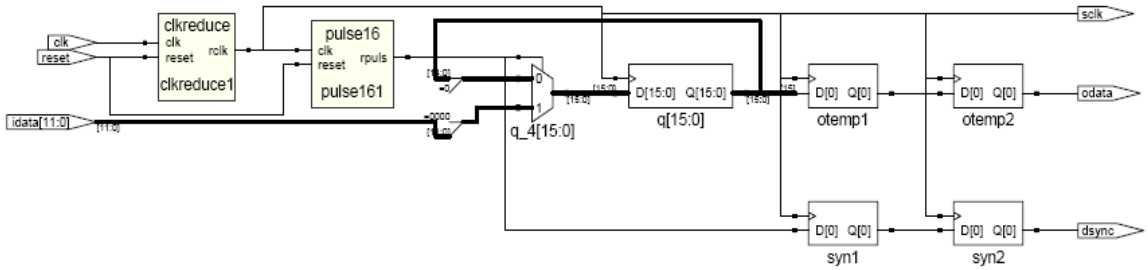

Figure 12: Shift Register for D/A RTL Schematic

6.3.4 ecount

This ecount verilog module is similar to the one used in the motor drive circuit with the major difference being that this module has the capability to send information to the D/A converters. The highlighted red numbers should be changed to the red number in motorquad or pendquad if they are changed.

```
module ecount(clk,A1,reset,B1, A2, B2, enable, sel, vreset1, vreset2,
evalue,odata,dsync,sclk);

      input clk;
      input reset;
      input A1;
      input B1;
      input A2;
      input B2;
      input enable;
      input [1:0] sel;
      input vreset1;
      input vreset2;
      output [12:0] evalue;
      output [1:0] odata;
      output [1:0] sclk;
      output [1:0]dsync;

      wire [12:0] count1temp;
      wire [12:0] count2temp;
      wire [12:0] count3temp;
      wire [12:0] count4temp;

      pendquadp pendpos(clk, A1, B1, count1temp,reset);
      motorquad motorpos(clk, A2, B2, count2temp,reset);
      pendquad pendvel(clk, A1, B1, count3temp,vreset1);
      motorquad motorvel(clk, A2, B2, count4temp,vreset2);

      ShiftReg
ShiftReg1(clk,count1temp[12:1],odata[0],sclk[0],reset,dsync[0]);
      ShiftReg
ShiftReg2(clk,count2temp[12:1],odata[1],sclk[1],reset,dsync[1]);

      wire [12:0] count1t;
      wire [12:0] count2t;
      wire [12:0] count3t;
      wire [12:0] count4t;

      enflopr enflopr1(clk, reset, enable, count1temp, count1t);
      enflopr enflopr2(clk, reset, enable, count2temp, count2t);
      enflopr enflopr3(clk, reset, enable, count3temp, count3t);
      enflopr enflopr4(clk, reset, enable, count4temp, count4t);

      mux4 encoutmux(count1t, count2t, count3t, count4t, sel, evalue);
endmodule
```

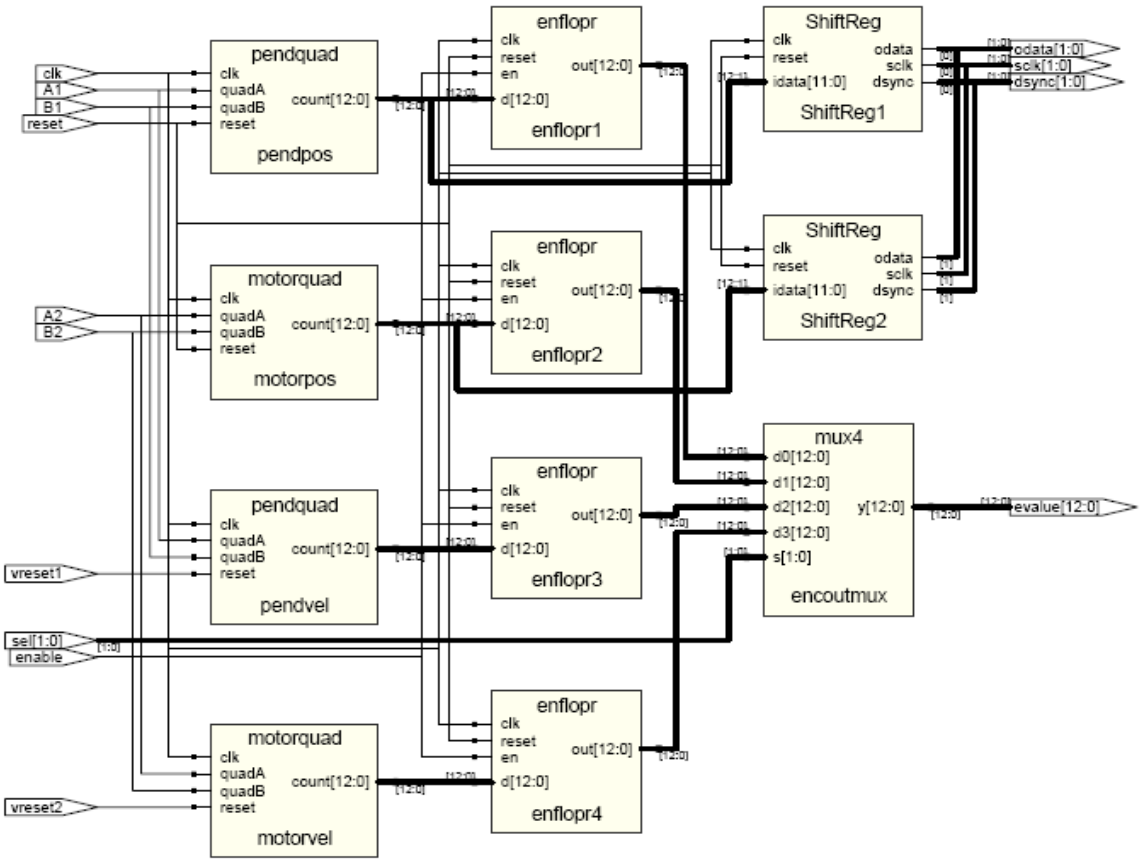

Figure 13: Encoder Counter (D/A Board) RTL schematic

7 Appendix B: Microprocessor Code

The PIC programs that are used in the Inverted Pendulum control system are written in both assembly language and C. In fact all the C programs are intertwined with some assembly language since assembly was the language of choice in retrieving encoder data from the FPGA. There are four main programs used in the inverted pendulum control system. They are invpen.c, invpentrack.c, Observer5.c and pwmadj.asm. A detailed explanation of each program will be given below

7.1 C Code

7.1.1 invpen.c

The invpen.c file contains the C code to implement the state space control of the inverted pendulum system. If the user desires to change the gains of the system (highlighted in red and blue) he should run the MATLAB function gaincalc and copy and paste the values into the code. The red numbers represent the feedback gains while the blue numbers represent the energy control gains. If the user desires to change the linear range he should change the numbers highlighted in green. The user should keep in mind that 2π radians are represented by a count of 8192.

```
/*Inverted Pendulum Controller
* Author: Sheldon Logan
*/


#include <p18f452.h>
#include <timers.h>

/* Function Prototypes */
void main(void);
```

```
void main(void) {

//Configure PORTE as digital I/O ports

int pendpos,pendvel,motorpos,motorvel;
unsigned int mtemp,prodtemp;
float gaintemp,gtemp;
char signe;


      ADCON1 = 7;

//SET PORTA,PORTB, PORTD, PORTE, as inputs PORTC as outputs

      TRISA = 255;
      TRISB = 255;
      TRISC = 0;
      TRISD = 255;
      TRISE = 1;

//Clear PORTS B,C,D and E

      PORTA = 0;
      PORTB = 0;
      PORTC = 0;
      PORTD = 0;
      PORTE = 0;

//Reset and Enable Encoder Counter

      PORTC = 193;
      PORTC = 2;

      T0CON = 0x87;                             //turn on timer0,
prescale 256;

//Get Pendulum position
while(1)
      {

            TMR0L = 121;
            TMR0H = 254;                        //Timer0 offset

            PORTC = 0;                          //turn off mux enable
      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 2;                          //turn on mux enable
            prodtemp = PROD;
            pendpos = prodtemp - 4096;          /*shift by pi since we
want the upright
                                                      //upright
position to be 0*/
```

```
            PORTC = 18;                          //select motor position

      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm

            PORTC = 18;                                     //turn on
mux enable
            mtemp = PROD;                                   //Place
link 1 position as a
            if(mtemp > 2000)                          //unsigned int.
If greater than
                  {
      //2000 then subtract 4000
                  motorpos = mtemp-4000;
                  }
            else
                  {
                  motorpos = mtemp;
                  }

            PORTC = 32;
      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 98;
            PORTC = 34;
            prodtemp = PROD;

            if(PROD>4096)
                  {
                  pendvel = prodtemp - 8192;
                  }
            else
                  {
                  pendvel = prodtemp;
                  }

      PORTC = 48;

      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 178;
            PORTC = 2;

            mtemp = PROD;                                   //Place
link 1 velocity as a
```

```
            if(mtemp > 2000)                          //usigned int. If
greater than
                  {
      //2000 then subtract 4000
                  motorvel = mtemp-4000;
                  }
            else
                  {
                  motorvel = mtemp;
                  }


      //set up PWM
      T2CON = 6;
      CCP1CONbits.CCP1M3 = 1;
      CCP1CONbits.CCP1M2 = 1;
      PR2 = 255;

      //if motor is in linear region use state space Control
      //try out 400
            if(pendpos<1200&&pendpos>-1200)
                  {
                  gaintemp = -(5.1282*pendpos +  86.6299*pendvel +
0.3940*motorpos + 56.2886*motorvel);

                  if(gaintemp < 0)                          //spinning
negatively
                        {
                        PORTCbits.RC3 = 1;
                        if(gaintemp<-255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              gtemp = -gaintemp;
                              CCPR1L = (char)(gtemp);
                              }
                        }
                  else
      //spinning postively
                        {
                        PORTCbits.RC3 = 0;
                        if(gaintemp>255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              CCPR1L = (char)(gaintemp);
                              }
                        }
                  }

      //if motor is out of linear range use swing-up control.

            else
```

```
                  {
                  if(pendpos < 2049 && pendpos > -2049)
      //cos(pendpos) is positive
                        {
                        if(pendvel<0)
                              {
                              signe = -1;
                              }
                        else
                              {
                              signe = 1;
                              }
                        }
                  else
                        {
                        if(pendvel<0)
            //cos(pendpos) is negative
                              {
                              signe = 1;
                              }
                        else
                              {
                              signe = -1;
                              }
                        }

                  if(pendpos<2500&&pendpos>-2500)
                        {
                        gaintemp = .05*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
                  else if(pendpos<3800&&pendpos>-3800)
                        {
                        gaintemp = .06*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
                  else
                        {
                        gaintemp = .1*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }


//                  gaintemp = 12*(-signe*pendpos*pendpos*.000010918+
signe*pendvel*pendvel*0.0012 + signe*motorvel*motorvel*0.0021);

                  if(gaintemp < 0)                          //spinning
negatively
                        {
                        PORTCbits.RC3 = 1;
                        if(gaintemp<-255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              gtemp = -gaintemp;
```

```
                        CCPR1L = (char)(gtemp);
                        }
                }
            else
//spinning postively
                {
                PORTCbits.RC3 = 0;
                if(gaintemp>255)
                        {
                        CCPR1L = 255;
                        }
                else
                        {
                        CCPR1L = (char)(gaintemp);
                        }
                }
            }
INTCONbits.TMR0IF = 0;
while(!INTCONbits.TMR0IF);

}
}
```

7.1.2 invpentrack.c

The invpentrack.c file contains the C code to implement the state space control of the inverted pendulum system with horizontal link tracking abilities. If the user desires to change the gains of the system (highlighted in red and blue) he should run the MATLAB function gaincalc and copy and paste the values into the code. The red numbers represent the feedback gains while the blue numbers represent the energy control gains. If the user desires to change the linear range he should change the numbers highlighted in green. The user should keep in mind that 2π radians are represented by a count of 8192. If the user wants to increase the magnitude of oscillations they should increase the numbers highlighted in orange. The user should keep in mind that 2π radians are represented by a count of 4000. The speed of the oscillations can be increased by increasing the number highlighted in purple. Unfortunately due to the time constraints of the project, a m-file was not developed that would convert to orange and purple numbers to familiar units such as radians and radians/sec.

```
/*Inverted Pendulum Controller
* Author: Sheldon Logan
*/


#include <p18f452.h>
#include <timers.h>

/* Function Prototypes */
void main(void);

void main(void) {

//Configure PORTE as digital I/O ports

int pendpos,pendvel,motorpos,motorvel,motorpost,mcount;
unsigned int mtemp,prodtemp;
float gaintemp,gtemp;
char signe;

mcount = 0;

      ADCON1 = 7;

//SET PORTA,PORTB, PORTD, PORTE, as inputs PORTC as outputs

      TRISA = 255;
      TRISB  = 255;
      TRISC = 0;
      TRISD = 255;
      TRISE = 1;

//Clear PORTS B,C,D and E

      PORTA = 0;
      PORTB = 0;
      PORTC = 0;
      PORTD = 0;
      PORTE = 0;

//Reset and Enable Encoder Counter

      PORTC = 193;
      PORTC = 2;

      T0CON = 0x87;                              //turn on timer0,
prescale 256;

//Get Pendulum position
while(1)
      {

            TMR0L = 121;
            TMR0H = 254;                         //Timer0 offset
```

```
            PORTC = 0;                          //turn off mux enable
      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 2;                          //turn on mux enable
            prodtemp = PROD;
            pendpos = prodtemp - 4096;          /*shift by pi since we
want the upright
                                                      //upright
position to be 0*/

            PORTC = 18;                         //select motor position

      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm

            PORTC = 18;                                   //turn on
mux enable
            mtemp = PROD;                                 //Place
link 1 position as a
            if(mtemp > 2000)                      //unsigned int.
If greater than
                  {
      //2000 then subtract 4000
                  motorpos = mtemp-4000;
                  }
            else
                  {
                  motorpos = mtemp;
                  }

            PORTC = 32;
      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 98;
            PORTC = 34;
            prodtemp = PROD;

            if(PROD>4096)
                  {
                  pendvel = prodtemp - 8192;
                  }
            else
                  {
                  pendvel = prodtemp;
```

```
                }

      PORTC = 48;

      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 178;
            PORTC = 2;

            mtemp = PROD;                                   //Place
link 1 velocity as a
            if(mtemp > 2000)                          //usigned int. If
greater than
                  {
      //2000 then subtract 4000
                  motorvel = mtemp-4000;
                  }
            else
                  {
                  motorvel = mtemp;
                  }


      //set up PWM
      T2CON = 6;
      CCP1CONbits.CCP1M3 = 1;
      CCP1CONbits.CCP1M2 = 1;
      PR2 = 255;

      //if motor is in linear region use state space Control
      //try out 400
            if(pendpos<1200&&pendpos>-1200)
                  {
                  if(mcount < 200)
                        {
                        mcount = mcount + 1;
                        motorpost = (mcount*0.1*100)+motorpos;
                        }
                  else if(mcount == 200)
                        {
                        mcount = 400;
                        }
                  else if(mcount == 201)
                        {
                        mcount = 0;
                        }
                  else if(mcount > 200)
                        {
                        mcount = mcount - 1;
                        motorpost = ((mcount-200)*0.1*100)+motorpos;
                        }
```

```
                  gaintemp = -(5.1282*pendpos +  86.6299*pendvel +
0.3940*motorpost + 56.2886*motorvel);

                  if(gaintemp < 0)                          //spinning
negatively
                        {
                        PORTCbits.RC3 = 1;
                        if(gaintemp<-255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              gtemp = -gaintemp;
                              CCPR1L = (char)(gtemp);
                              }
                        }
                  else
      //spinning postively
                        {
                        PORTCbits.RC3 = 0;
                        if(gaintemp>255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              CCPR1L = (char)(gaintemp);
                              }
                        }
                  }

      //if motor is out of linear range use swing-up control.

            else
                  {
                  if(pendpos < 2049 && pendpos > -2049)
      //cos(pendpos) is positive
                        {
                        if(pendvel<0)
                              {
                              signe = -1;
                              }
                        else
                              {
                              signe = 1;
                              }
                        }
                  else
                        {
                        if(pendvel<0)
            //cos(pendpos) is negative
                              {
                              signe = 1;
                              }
                        else
                              {
```

```
                                signe = -1;
                                }
                        }

                if(pendpos<2500&&pendpos>-2500)
                        {
                        gaintemp = .05*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
                else if(pendpos<3800&&pendpos>-3800)
                        {
                        gaintemp = .06*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
                else
                        {
                        gaintemp = .1*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }


//              gaintemp = 12*(-signe*pendpos*pendpos*.000010918+
signe*pendvel*pendvel*0.0012 + signe*motorvel*motorvel*0.0021);

                if(gaintemp < 0)                        //spinning
negatively
                        {
                        PORTCbits.RC3 = 1;
                        if(gaintemp<-255)
                                {
                                CCPR1L = 255;
                                }
                        else
                                {
                                gtemp = -gaintemp;
                                CCPR1L = (char)(gtemp);
                                }
                        }
                else
        //spinning postively
                        {
                        PORTCbits.RC3 = 0;
                        if(gaintemp>255)
                                {
                                CCPR1L = 255;
                                }
                        else
                                {
                                CCPR1L = (char)(gaintemp);
                                }
                        }
                }
        INTCONbits.TMR0IF = 0;
        while(!INTCONbits.TMR0IF);

        }
}
```

7.1.3 Observer5.c

The Observer5.C program implements the state space control of the inverted pendulum with an observer estimating the pendulum and link velocity. The observer poles are placed to be 5 times faster than the closed loop poles. If the user desires to change the gains of the system (highlighted in red and blue) he should run the m-file function gaincalcobs and copy and paste the values into the code. The red numbers represent the feedback gains while the blue numbers represent the energy control gains. If the user desires to change the linear range he should change the numbers highlighted in green. The section highlighted in orange is where the estimates are updated. If the user desires to change the observer poles he should run the m-file obscoecalc and copy and paste the coefficients produced into the code.

```
/*Inverted Pendulum Controller with Observer
* Author: Sheldon Logan
*/

#include <p18f452.h>
#include <timers.h>

/* Function Prototypes */

void main(void);

void main(void) {

//Configure PORTE as digital I/O ports

int pendpos,pendvel,motorpos,motorvel;
unsigned int mtemp,prodtemp;
float gaintemp,gtemp;
float pendposoldest,pendveloldest,motorposoldest,motorveloldest,torq;
float pendposnewest,motorposnewest,pendvelnewest,motorvelnewest;
float pendposa,motorposa,torque;
char signe;
```

```
      pendposoldest = 0;                              //initalise
estimates
      pendveloldest = 0;
      motorposoldest = 0;
      motorveloldest = 0;

      ADCON1 = 7;

//SET PORTA,PORTB, PORTD, PORTE, as inputs PORTC as outputs

      TRISA = 255;
      TRISB  = 255;
      TRISC = 0;
      TRISD = 255;
      TRISE = 1;

//Clear PORTS B,C,D and E

      PORTA = 0;
      PORTB = 0;
      PORTC = 0;
      PORTD = 0;
      PORTE = 0;

//Reset and Enable Encoder Counter

      PORTC = 193;
      PORTC = 2;

      T0CON = 0x87;                                   //turn on timer0,
prescale 256;

//Get Pendulum position
while(1)
      {
            TMR0L = 121;
            TMR0H = 254;                              //Timer0 offset

            PORTC = 0;                                //turn off mux
enable
      _asm
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm
            PORTC = 2;                                //turn on mux
enable
            prodtemp = PROD;
            pendpos = prodtemp - 4096;          /*shift by pi since we
want the upright
                                                            //upright
position to be 0*/
            PORTC = 18;                               //select motor
position

      _asm
```

```
            movf PORTB,0,0
            andlw 0x1F
            movwf PRODH,0
            movff PORTD,PRODL
      _endasm

            PORTC = 18;                                     //turn on
mux enable
            mtemp = PROD;                                   //Place
link 1 position as a
            if(mtemp > 2000)                          //unsigned int.
If greater than
                  {
      //2000 then subtract 4000
                  motorpos = mtemp-4000;
                  }
            else
                  {
                  motorpos = mtemp;
                  }

            PORTC = 2;


      //set up PWM
      T2CON = 6;
      CCP1CONbits.CCP1M3 = 1;
      CCP1CONbits.CCP1M2 = 1;
      PR2 = 255;

      //if motor is in linear region use  state space Control

            if(pendpos<800&&pendpos>-800)
                  {

                  gaintemp = -(6686.2*pendposoldest +
1129.5*pendveloldest + 250.8559*motorposoldest +
358.3445*motorveloldest);

                  if(gaintemp < 0)                          //spinning
negative
                        {
                        PORTCbits.RC3 = 1;
                        if(gaintemp<-255)
                              {
                              torque = -255;
                              CCPR1L = 255;
                              }
                        else
                              {
                              torque = (char)gaintemp;
                              gtemp = -gaintemp;
                              CCPR1L = (char)(gtemp);
                              }
                        }
                  else
      //spinning postively
```

```
                        {
                        PORTCbits.RC3 = 0;
                        if(gaintemp>255)
                              {
                              torque = 255;
                              CCPR1L = 255;
                              }
                        else
                              {
                              CCPR1L = (char)(gaintemp);
                              torque = CCPR1L;
                              }
                        }
                  }

            //if motor is out of linear range use swing-up control.

            else
                  {
                  if(pendpos < 2049 && pendpos > -2049)
      //cos(pendpos) is positive
                        {
                        if(pendvel<0)
                              {
                              signe = -1;
                              }
                        else
                              {
                              signe = 1;
                              }
                        }
                  else
                        {
                        if(pendvel<0)
            //cos(pendpos) is negative
                              {
                              signe = 1;
                              }
                        else
                              {
                              signe = -1;
                              }
                        }

/*                if(pendpos<2500&&pendpos>-2500)
                        {
                        gaintemp = .05*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
                  else if(pendpos<3800&&pendpos>-3800)
                        {
                        gaintemp = .06*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
                  else
                        {
```

```
                        gaintemp = .1*(signe*pendpos*pendpos*.00039443
+ signe*pendvel*0.0402*pendvel);
                        }
*/

                  gaintemp = 10*(-
signe*pendposoldest*pendposoldest*46.3971+
signe*pendveloldest*pendveloldest*0.5259 +
signe*motorveloldest*motorveloldest*0.2114);

                  if(gaintemp < 0)                          //spinning
negatively
                        {
                        PORTCbits.RC3 = 1;
                        if(gaintemp<-255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              gtemp = -gaintemp;
                              CCPR1L = (char)(gtemp);
                              }
                        }
                  else
      //spinning postively
                        {
                        PORTCbits.RC3 = 0;
                        if(gaintemp>255)
                              {
                              CCPR1L = 255;
                              }
                        else
                              {
                              CCPR1L = (char)(gaintemp);
                              }
                        }
                  }


      torq = torque*0.0014;
      pendposa = .00076699*pendpos;
      motorposa = 0.0016*motorpos;

// Create Observer estimates
      pendposnewest = (-0.6461*pendposoldest)+(0.01*pendveloldest)-
(0.0021*torq)+(1.6487*pendposa)+(0.0551*motorposa)-
(0.0551*motorposoldest);
      motorposnewest= (-0.0171*pendposoldest)-
(0.6469*motorposoldest)+(.01*motorveloldest)+(0.0021*torq)+(0.0167*pend
posa)+(1.6469*motorposa);
      pendvelnewest = (-67.8542*pendposoldest)+(1.0026*pendveloldest)-
(0.4188*torq)+(68.3836*pendposa)+(4.4226*motorposa)-
(4.4226*motorposoldest);
      motorvelnewest = (-1.3427*pendposoldest)-
(.0004*pendveloldest)+(motorveloldest)+(0.4185*torq)+(1.2548*pendposa)+
(67.7620*motorposa)-(67.7620*motorposoldest);
```

```
      pendposoldest = pendposnewest;
      pendveloldest = pendvelnewest;
      motorposoldest = motorposnewest;
      motorveloldest = motorvelnewest;

      INTCONbits.TMR0IF = 0;
      while(!INTCONbits.TMR0IF);
      }
}
```

7.2 Assembly Code

7.2.1 pwmadj.asm

The pwmadj.asm program is used to create a pulse width modulated (PWM) signal from a sampled voltage (read at pin RA1).

```
;Program to provide a .996s delay


;Use the 18F452 PIC microprocessor

      LIST p = 18F452
      include "p18F452.inc"

;allocate variables
VHIG EQU 0x00                             ;input voltage
NUMI EQU 0x01                             ;counter for loop
VLOW EQU 0x02                             ;how long to hold low for

NUMT EQU 0x03                             ;0xff
;begin main program
      org 0
      bra start               ;go to start of program

      org 0x0008
      bra interc              ;branch to interrupt handler

      org 0x0020
start
      movlw 0xff              ;put 0xff in NUMT register
      movwf NUMT
      clrf TRISD
      clrf TRISC
      clrf PORTD              ;set all port d ouputs to zero
      movlw 0xC1;
      movwf PORTC
```

```
      movlw 0x10;
      movwf PORTC

interc
      movlw 0x07
      movwf ADCON1
;     movwf TRISE
      clrf INTCON             ;clear intcon register
      bcf PIR1,6              ;clear ADIF bit
      bsf IPR1,6              ;Put ADIP at high priority
      bsf INTCON,7            ;reenable the interrupt bits
      bsf INTCON,6
      bsf PIE1,6              ;enable A/D interrupt flag

      movf ADRESL,0           ;Only use top 8 bits of 10 bit A/d
Converter
      rrncf WREG              ;rotate ADRESL twice to the right and
place
      rrncf WREG              ;in regiser low bits from ADRESH in top 2
      bcf WREG,7              ;bits of register, Use this value as the
      bcf WREG,6              ;'high' period, and subtract from ff for
      btfsc ADRESH,1          ;'low' period
      bsf WREG,7
      btfsc ADRESH,0
      bsf WREG,6
      movwf VHIG              ;length of time for output to be high
      subwf NUMT,0
      movwf VLOW              ;length of time for output to be low
      movff VHIG,PORTD        ;display sampled signal on LEDS

;     bsf TRISC,0             ;make PortC bit 0 a input
      bcf TRISC,2             ;make Port C bit 2 a output
      bcf PORTC,0
      bsf T2CON,TMR2ON        ;turn on timer 2
      bsf T2CON,T2CKPS1       ;scale PWM period by 16
                              ;PWM frequency is 2.44 KHz
      bsf CCP1CON,CCP1M3      ;set up CCP1CON register for PWM mode
      bsf CCP1CON,CCP1M2
      movlw 0xff
      movwf PR2               ;set PR2 register to 0xff for CCP PWM
module
;     btfss PORTC,0
;     bra negc
      movff VHIG,CCPR1L       ;Place PWM duty cycle in CCPR1L register
;     bra main
;negc
;     movff VLOW,CCPR1L
;     bcf CCP1CON,DC1B1
;     bcf CCP1CON,DC1B0
main
      movlw 0x89              ;configure A/D to use fosc/64
      movwf ADCON0            ;use An1 as an analoog input, use the
vref and
      movlw 0xC2              ;ground as reference voltages and ouput
is
      movwf ADCON1            ;right justitified
```

```
      movlw 0xC8              ;set timer0 to desired configuration
      movwf T0CON             ;timer is used to delay for 25.6 micro
seconds
      movlw 0x00              ;initialise counter
      movwf TMR0H
      movlw 0x00
      movwf TMR0L
      bcf INTCON,2            ;clear timer overflow flag

wait
      btfss INTCON,2          ;check to see if timer is finished
counting
      bra wait                ;if not check again
      bcf INTCON,2            ;clear timer overflow flag
      bsf ADCON0,2            ;acquire data from A/D converter.

      end
```