1300 Henley Court
Pullman, WA 99163
509.334.6306
www.store.digilent.com

Lab 6b: Closed-loop Process Control

Revised May 23, 2017
This manual applies to Unit 6, Lab 6b.

1 Objectives

1. Generate PWM outputs to implement analog motor supply voltage.
2. Implement a tachometer operation using PIC32 Timers.
3. Develop MPLAB X projects that implement closed-loop motor control.
4. Develop the C program code to implement a PI controller and a moving averaging digital filter.
5. Manage multiple background tasks in an interrupt driven system.
6. Send real-time data to monitoring devices.

2 Basic Knowledge

1. Fundamental knowledge of linear systems.
2. How to configure IO pins on a Microchip® PIC32 PPS microprocessor.
3. How to implement a real-time system using preemptive foreground-background task control.
4. How to generate a PWM output with the PIC32 processor.
5. How to configure the Analog Discovery 2 to display logic traces.
6. How to implement the design process for embedded processor based systems.

3 Equipment List

3.1 Hardware

1. Basys MX3 trainer board
2. Workstation computer running Windows 10 or higher, MAC OS, or Linux
3. 2 Standard USB A to micro-B cables
4. 5 V DC motor with tachometer
5. 5 V, 4A power supply

In addition, we suggest the following instruments:

6. Analog Discovery 2

3.2 Software

The following programs must be installed on your development work station:

1. Microchip MPLAB X® v3.35 or higher
2. PLIB Peripheral Library
3. XC32 Cross Compiler
4. WaveForms 2015 (if using the Analog Discovery 2)
5. PuTTY Terminal Emulation
6. Spreadsheet application (Microsoft Excel)

4 Project Takeaways

1. How to read an analog voltage with a PIC32 processor.
2. How to use the PIC32 Output Compare to implement a PWM analog output.
3. How to use the PIC32 Timer external input to measure frequency to implement a tachometer.
4. How to use the PIC32 Input Capture period measurement to implement a tachometer.
5. Fundamental analog and filtering concepts for data smoothing and closed-loop control.

5 Fundamental Concepts

The purpose of this laboratory exercise is to implement a closed-loop control system to control the speed on a DC electric motor. The processing involves two different types of analog inputs and generating an analog output using pulse-width modulation. We will see how both analog and digital signal conditioning can reduce measurement noise that can degrade system performance. This lab requires that an electronic circuit be constructed to provide an interface between the Basys MX3 board and the DC motor tachometer.

5.1 Feedback Control

Feedback control is a common and powerful tool when designing a control system that can compensate for load variations and perturbations. A feedback loop as portrayed in Fig. 5.1, taking the system output into consideration. This enables the system to adjust its performance to meet a desired output response in spite of variations in motor characteristics, noise, and disturbances that may be introduced anywhere in the system.

The controller action and feedback compensation is implemented using digital filtering and digital control theory. Digital filtering involves discrete time and discrete amplitude signals that are generated when continuous signals are sampled with an analog-to-digital converter at fixed time intervals.[1] The primary objective of the control system is to achieve zero error, designated as "e_n" in Fig. 5.1. The error signal is the difference between the sum of the control inputs, r_n, and the feedback signal, y_n. The variable c_n is the output from the control algorithm used to drive the system machine or process. The subscript "n" on these variable denotes the fact that they are generated at discrete times.

[1] Signal Processing for Communications, Chapter 1.3, http://www.sp4comm.org/webversion/livre.html#x1-100001.3

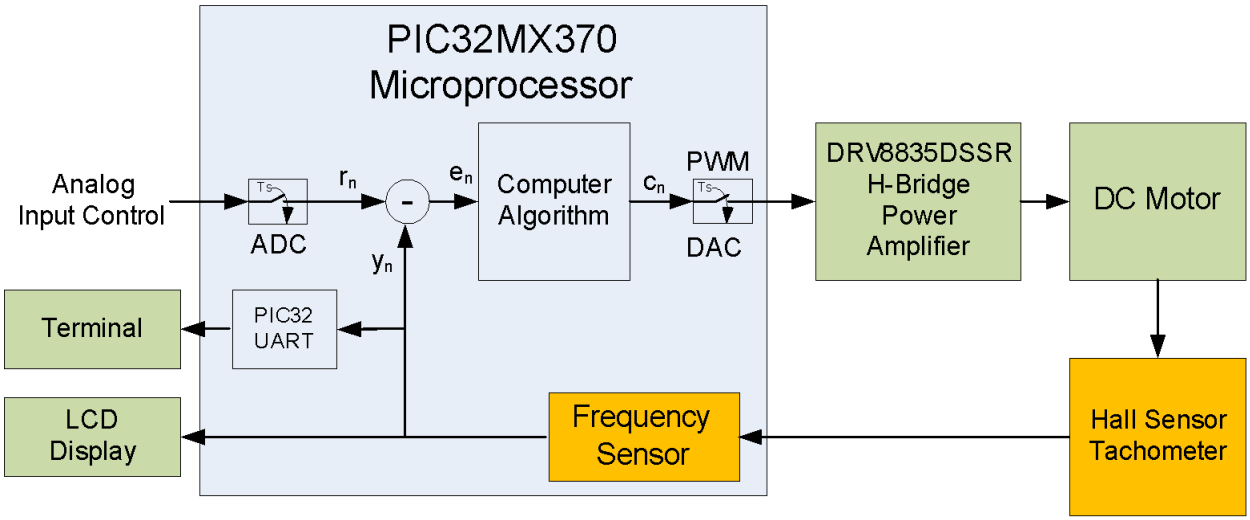

Figure 5.1. Closed-loop motor control block diagram.

Besides changes in mechanical loads on the motor, the major sources of noise or disturbances into the control loop is from the output amplifier, the process converting electrical energy to mechanical energy, and the tachometer used to measure the mechanical output. Noise can also be introduced on both the analog and discrete inputs. In our system we will consider only disturbed signals at the output of the tachometer and the output of the signal conditioning comparator.

5.2 Closed-loop Motor Control

Negative feedback generally promotes faster settling to an equilibrium condition and reduces the effects of perturbations, as opposed to positive feedback. Systems employing negative feedback loops, in which just the right amount of correction is applied with optimum timing, can be very stable, accurate, and responsive.

Considering the problem of controlling the speed of a DC motor, we will use negative feedback to cause the motor speed to be linearly dependent on the Analog Input Control voltage. Since the speed of the motor is roughly proportional to the applied voltage, it would seem that one would simply need to construct an equation that describes speed as that particular function of voltage, and then set the motor voltage based on the desired speed. This was the approach we took in Lab 6a. However, the motor speed is also a function of the load applied to the motor and dynamic effects such as friction and inertia. Closed-loop controllers are used to compensate for those issues that affect the motor speed and provide a linear response to linear inputs.

The issue now becomes choosing the algorithm that the microprocessor must implement to provide the desired speed control under varying load conditions. As described in Appendix C of Unit 6, classical controls uses proportional plus integral plus derivative (PID) control, which works well for controlling many dynamic systems. Other types of controllers use fuzzy logic and artificial neural networks (ANN) and are most suited for controlling extremely nonlinear systems.

Because of the difficulty of "tuning" PID controllers to avoid instability, this lab will use just proportional plus integral (PI) control. Classical control designs use mathematical equations to describe continuous systems and define the controller action, specifically differential equations and Laplace transforms. For digital computers to implement the controller action, the continuous equations must be transformed into a form that can be converted to digital code, as discussed in Unit 6.

6 Problem Statement

As illustrated in Fig. 5.1, closed-loop control incorporates a feedback signal along with the analog and discrete inputs to the control process. The tachometer frequency will be measured by using the Input Capture of the PIC32 to determine the signal period which is then inverted to yield the frequency. The tachometer period measurements are filtered with a moving averaging low pass digital filter as described in Lab 6a.

This lab is to use a digital implementation of proportional plus integral (PI) control in the control loop. The speed of the motor will be linearly dependent on the voltage controlled by the Analog Input Control potentiometer. This is described by Eq. 6.1 which results in linear motor speed control that will vary between 30% and 90% of maximum motor speed.

$$Motor\ Speed = \left[\left(\frac{Analog\ Control\ Voltage * 0.6}{Maximum\ Analog\ Control\ Voltage}\right) + 0.3\right] \cdot Maximum\ motor\ speed \qquad \text{Eq. 6.1}$$

The range of 30% to 90% is selected on the characteristics of the DC motor we are using for Lab 6a and 6b. By experimentation, it was determined that when the motor supply is set to 5.0 V, the %PWM must be set above 25% to 28% or the motor will not turn as shown in Fig. 8.3 of Lab 6a. If the motor is not turning, then there are no input capture interrupts to set the RPS variable in the control algorithm.

7 Background Information

7.1 Closing the Control Loop

Because of ease of implementation and stability, we will implement proportional plus integral (PI) control in this lab. The control algorithm for the PI control is implemented by the pseudo code of Listing 7.1. This code is executed at the frequency established by the PWM PERIOD. Any update to the PWM duty cycle is made when Timer 2 resets.

The software shown in Listing B.6 of Lab 6a specifies where to add the PI control code for the pseudo code in Listing 7.1 below. (This code is reproduced from Listing C.1 of the Unit 6 tutorial.) Using the motor specified in the Equipment List, the constant (KP+KI) is set to 60/1024 and (KP-KI) to 60/1024. These constants were selected by experimentation solely for demonstration purposes and may need to be adjusted for other types of motors.

Listing 7.1. Pseudo Code for Digital PI Controller

```
#define GetPeripheralClock() 10000000                  // PBCLK set in config_bits.h
#define PWM_MAX (GetPeripheralClock())/10000)          // Set output range
#define PWM_MIN 0

static int ERROR = 0;                                  // Initial value for e_n-1
static int CTRL = 0;                                   // Initial value for cn_n-1

TACH = Read motor speed (Hz);                          // Global variable set by InputCapture ISR
// The SET_SPEED reference input is scales using a first order polynomial in the form y = ax + b to set the slope, a, with
//  units rps/ADC bit and "b" being the minimum speed when the ADC input is zero
SET_SPEED = Scaled potentiometer setting                // Determine speed set point from ADC

ERROR_LAST = ERROR;                                    // Save previous error and control values
CTRL_LAST = CTRL;
```

```
ERROR = SET_SPEED - TACH;                                   // Compute new error value

CTRL = (KP + KI)*ERROR - (KP - KI)*ERROR_LAST + CTRL_LAST; // Compute new control output value

if(CTRL > PWM_MAX) then PWM_CONTROL = PWM_MAX; else  // Limit output range to prevent windup
{
      If (CTRL < PWM_MIN)  then PWM_CONTROL = PWM_MIN;  else
      PWM_CONTROL = CTRL;                                    // Output PWM value
}
```

Figure 7.1 was generated by monitoring the motor tachometer using the SPI_CK DIO 4 pin on the Analog Discovery 2 connector that is toggled in the input capture ISR, as shown in Listing B.4 of Lab 6a. Examining the plot for the tachometer in Fig. 7.1, we see than the PI algorithm requires about 150 ms to achieve steady state.

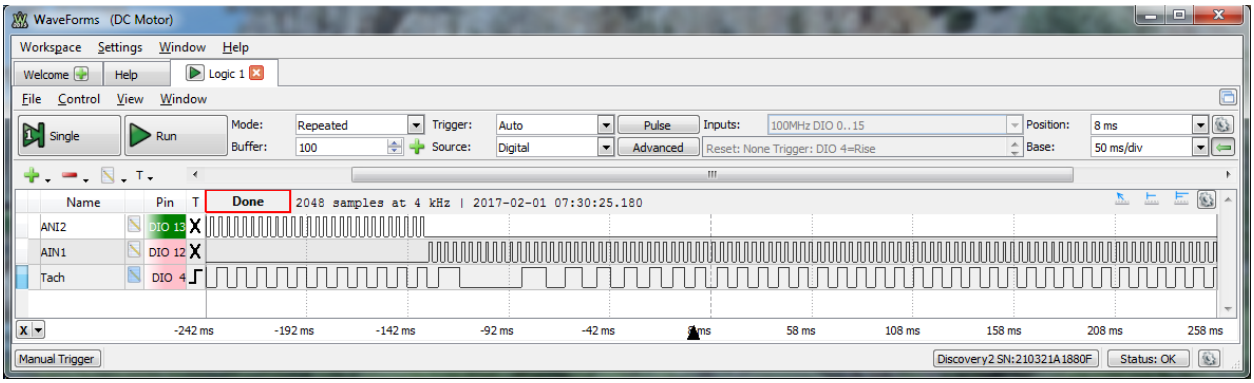

Figure 7.1. Motor speed response for a step input from CW to CCW at 30% PWM for PI closed-loop control.

8 Lab 6b

The DC motor used in Lab 6a and 6b has a 19 to one reduction gear on the output shaft. The Hall Effect sensor used for the tachometer is mounted to the motor shaft and hence will be rotating 19 times faster than the output shaft. All motor speed measurements are based on the tachometer.

This lab will be divided into two phases. The first phase uses open-loop control with proportional control that includes the tachometer input to the processor. The tachometer input will be filtered with a moving average digital filter before being displayed on the LCD and sent to the UART.

Phase two incorporates proportional plus integral control of the motor speed control. The reference input, r_n, will be derived from the Analog Control Input using the relationship previously described in Eq. 6.1.

8.1 Requirements

1. Phase 1: Background Tasks
    a. Same as background tasks for Lab 6a.
    b. Set peripheral bus clock for 10 MHz in config_bits.h.
    c. Timer 2 is to be used for the PWM generation
        i. Set the Timer 2 input clock for 10 MHZ
        ii. Set the Timer 2 period for 10000 counts
    d. Timer 3 is to be used as the time reference for input capture.
        i. Set the Timer 3 input clock for 625000.

ii. Set the Timer 3 period for maximum counts (65536).

2. Phase 1: Foreground Tasks
    a. Same as foreground tasks for Lab 6a.
    b. Basys MX3 JA-10 (PIC32MX370 Port G pin 9) is to be used for the Input Capture channel 1.
    c. Input Capture ISR
        i. An interrupt is to be generated on each positive transition of the tachometer signal.
        ii. Read the captured Timer 3 value.
        iii. Compute the period of the tachometer signal as the difference between two successive Timer 3 captures.
        iv. Implement a fourth order Moving Averaging low-pass filter to reduce noise on the period measurement.
        v. Compute the motor speed in RPS to be used in the PI control loop and the background display tasks, assuming that the motor speed in revolutions per second (rps) is the same as the tachometer frequency.
    d. Timer 2 ISR
        i. Read the ADC
        ii. Compute the set speed using Eq. 6.1 above
        iii. Compute the actual motor speed as a percentage of maximum speed when PWM is set to 100%.
        iv. Implement the control algorithm following the pseudo code shown in Listing 7.1 and using the control constants provided in Unit 6 background text.
3. Phase 2: Background Tasks
    a. All background tasks are to be serviced every 250 ms.
    b. The motor speed setting (spd), the tachometer measurement (rps), PWM control output (cn), and the control error (en) is to be displayed on the Basys MX3 LCD as shown in Fig. 8.1.

Figure 8.1. LCD display for closed loop operation.

    c. In addition to the data displayed on the LCD in Fig. 8.1, display the measured Analog Input Control voltage on your workstation terminal as shown in Fig. 8.2.

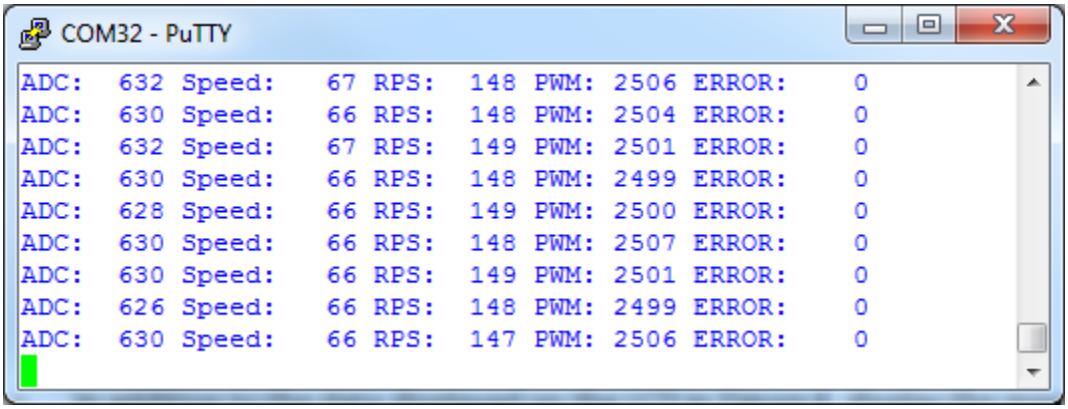

Figure 8.2. Terminal screen for closed loop motor control.

d. Using the slide switches on the Basys MX3 board, control the motor operations as listed in Table 8.1.

Table 8.1. Motor control modes

Slide Switch	LOW	HIGH
SW5	Run the motor according to the direction set by SW7	Coast operation
SW6	Run the motor according to the direction set by SW7	Brake (STOP)
SW7	Rotate CCW at speed set by analog input control	Rotate CW at speed set by analog input control

4. Phase 2: Foreground Tasks
    a. Input Capture ISR
        i. Compute the difference in Timer 3 ticks between two successive positive transitions of the tachometer.
        ii. Implement a 4th order moving average on the tachometer period.
        iii. Compute the motor speed in RPS.
    b. The Timer 2 will serve the following functions:
        i. The PIC32 ADC will read the digitized value of the “Analog Input Control” voltage.
        ii. Scale the analog control voltage as described by Eq. 6.1.
        iii. Compute the PWM output from the proportional plus integral (PI) algorithm.
        iv. Write to the output compare secondary registers (OCxRS) that generate the PWM output.

8.2 Design Phase

This design will be completed in two phases.

1. Phase 1
    a. Complete the design for Phase 1 and 2 of Lab 6a.
    b. Modify the Input Capture function that measures the tachometer period to implement the moving average low pass filter specified by the Requirement 4.ii.

c. Create the control flow diagrams that describe the process to implement the design requirements for Phase 1.

2. Phase 2
   a. Create a data flow diagram that modifies the one created for Phase 1 but adds the following tasks.
      i. LCD display as specified in the requirements
      ii. UART output as specified in the requirements
      iii. Add the PI control and motor PWM control to the Timer 2 ISR
   b. Create the control flow diagrams that describe the process to implement the design requirements for Phase 2.

8.3 Construction Phase

1. Since the motor must be turning before the tachometer will generate the pulses needed to cause an input capture, I recommend that the motor operate in open-loop mode for a few hundred milliseconds so a valid motor speed reading can be used in the closed-loop algorithm.
2. Phase 1
   a. Complete Phase 1 and 2 of Lab 6a.
      i. Develop background and foreground functions.
   b. Proceed to Phase 1 testing
3. Phase 2
   a. Attach the workstation monitor and launch the terminal emulator application.
   b. Port the LCD code from Lab 3a or 3b into this project and display the motor Analog Control Input value, the %PWM and the motor speed in RPM.
   c. Port the UART code from Lab 4a into this project and initialize for 38000 BAUD with no parity.
   d. Add the code necessary to initialize the input capture channel 1 (IC1) that uses Timer 3.
   e. Modify the background and foreground functions developed in Construction Phase 1 to report the motor operations.

8.4 Testing

1. Phase 1
   a. Set the Analog Input Control potentiometer for approximately 1 V.
   b. Run the project to make the DC motor spin.
   c. Measure the tachometer frequency using the oscilloscope measurements as shown in Fig. 8.3.

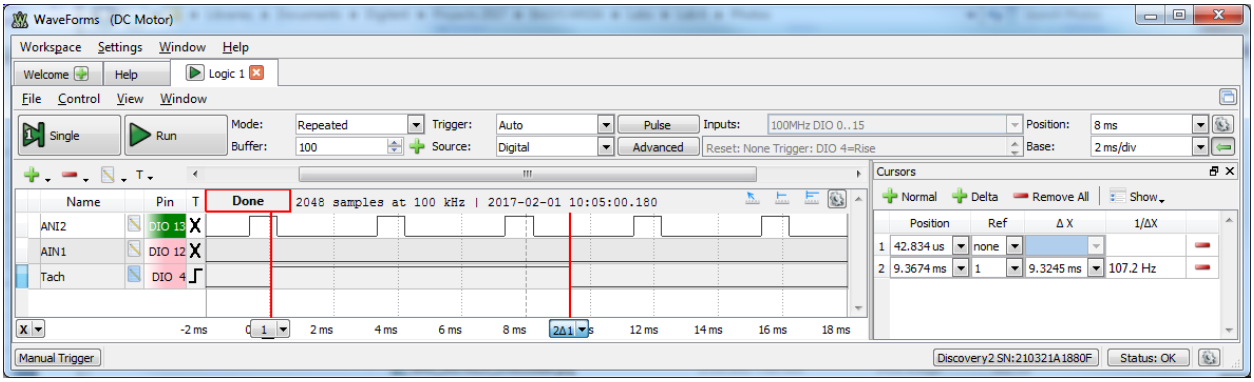

Figure 8.3. Tachometer timing for 106 Hz Display on LCD.

d. Verify that the frequency measurement on the LCD and that reported to the UART terminal match.

2. Phase 2.
   a. Maintain the oscilloscope connections as Phase 1 testing step c.
   b. Set the oscilloscope time base for 0.05 seconds per division
   c. Using the oscilloscope single trace, capture the tachometer transition just when the motor rotation direction is reversed using SW7, as shown in Fig. 8.4.

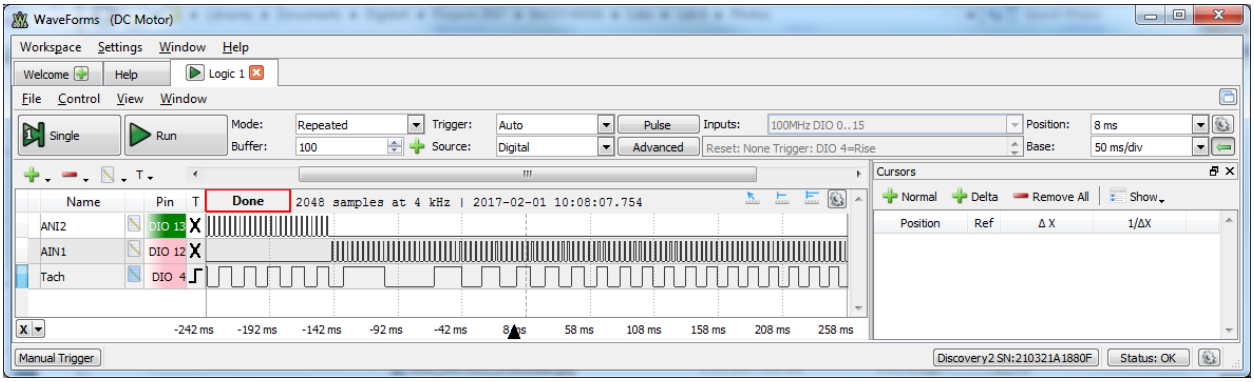

Figure 8.4. PWM output timing.

d. Determine the time for the motor to reach steady state rps in the new direction, marked by a uniform tachometer frequency.
e. Adjust the Analog Input Control potentiometer for an ADC value of 0 to 1024 in steps of 100. Complete Table 8.2, recording the specified entries.

Table 8.2. Closed loop motor speed performance.

Analog Input ADC	% PWM	Motor Speed - RPS
0		
100		
200		
300		
400		
500		
600		
700		
800		
900		
1000		

f. Using a spreadsheet program, plot the results for both the open loop control measured in Lab 6a and for the closed-loop control measured above. This plot should look similar to Fig. 8.5.

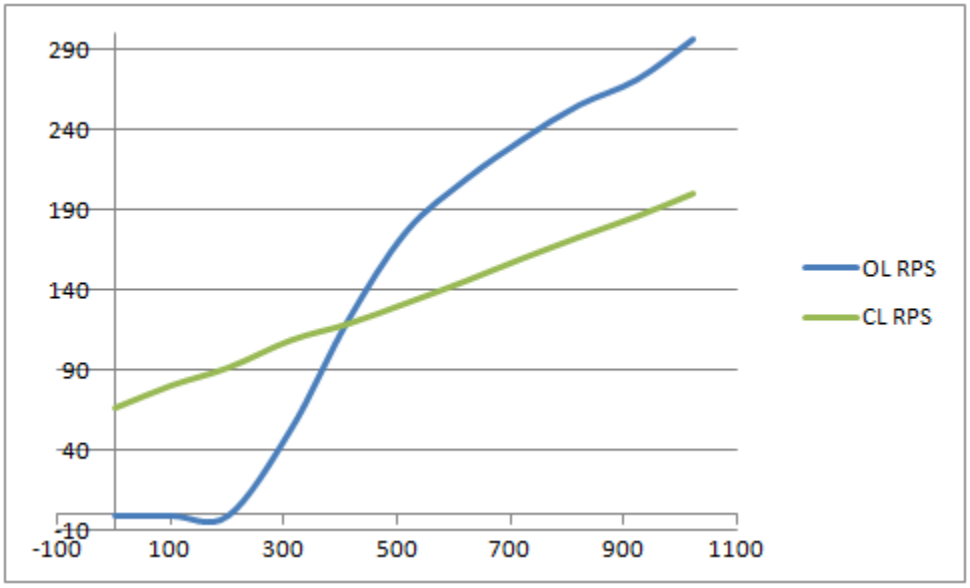

Figure 8.5. Motor speed vs Analog Input Control ADC value for open-loop (OL RPS) and closed-loop (CL RPS) control schemes.

9 Questions

1. What do you conclude from the plots in Fig. 8.5?
2. Why does Fig. 7.1 show that the motor decelerates more quickly than accelerate?
3. What affect does the order of tachometer moving averaging low pass filter have on the motor response time?
4. What effect does the execution time for updating the LCD and sending the UART message have on the tachometer period measurements using the PIC32 capture and compare feature?
5. What effect does the execution time for updating the LCD and sending the UART message have on the PI control algorithm?

10 References

1. "Open-vs. closed-loop Control", Vance VanDoren, Control Engineering, Aug. 28, 2014
2. DRV8835 Data Sheet, https://www.pololu.com/file/0J570/drv8835.pdf.
3. "Brished DC Motor Basics Webinar", John Moutton, Microchip Technologies Inc., http://www.microchip.com/stellent/groups/SiteComm_sg/documents/DeviceDoc/en543041.pdf.
4. "AN905 Brushed DC Motor Fundamentals", Reston Condit, Microchip Technology Inc, Aug. 4, 2010, http://ww1.microchip.com/downloads/en/AppNotes/00905B.pdf.
5. AN538, "Using PWM to Generate Analog Output", Amar Palacheria, Microchip Technology Inc., http://ww1.microchip.com/downloads/en/AppNotes/00538c.pdf.
6. "AB-022: PWM Frequency for Linear Motion Control", Precision Microdrives ™, https://www.precisionmicrodrives.com/application-notes/ab-022-pwm-frequency-for-linear-motion-control.
7. Scientists and Engineer's Guide to Digital Signal Processing, Dr. Steven W. Smith, http://www.dspguide.com/.

8. "Implementing a PID controller Using a PIC18 MCU", Chris Valenti, Microchip Technologies Inc., 2005, http://ww1.microchip.com/downloads/en/AppNotes/00937a.pdf.
9. Real Time Systems Design and Analysis 4th Edition, http://droppdf.com/v/s2EZM.

Appendix A: Lab 6a Motor Configuration

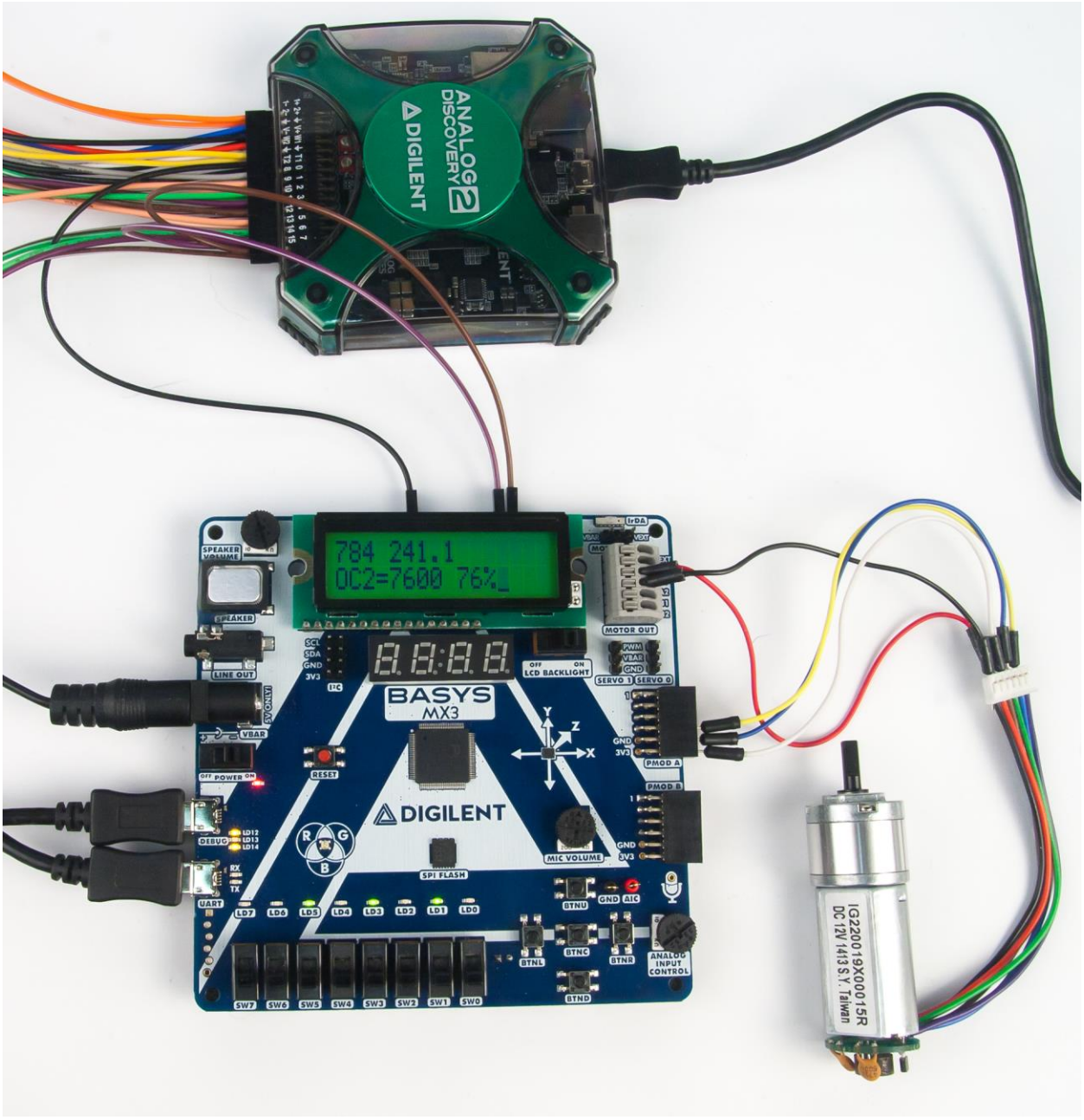

Figure A.1. DC Motor connection to the Basys MX3 processor board.

Appendix B: Lab 6a Code Listings

Listing B.1. Initialize ADC10 to Read Analog Channel 2

```
void ADC10Init()
{
// define setup parameters for OpenADC10
              // Turn module on | output integer  | trigger mode auto | enable auto sample
#define ADC_PARAM1  ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_AUTO | ADC_AUTO_SAMPLING_ON

// define setup parameters for OpenADC10
                    // ADC ref external | disable offset test | disable scan mode |
                    //perform 2 samples | use dual buffers | use alternate mode
#define ADC_PARAM2  ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_OFF | \\
                   ADC_SAMPLES_PER_INT_2 | ADC_ALT_BUF_ON | ADC_ALT_INPUT_ON

// define setup parameters for OpenADC10
                  // use ADC internal clock | set sample time
#define ADC_PARAM3  ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_15

// define setup parameters for OpenADC10
                  // do not assign channels to scan
#define ADC_PARAM4    SKIP_SCAN_ALL
// define setup parameters for OpenADC10
                  // set AN2 as analog inputs
#define ADC_PARAM5    ENABLE_AN2_ANA Motor driver output pin assignments

// wait for the first conversion to complete so there will be valid data
// in ADC result registers
   while ( !AD1CON1bits.DONE );

// Start Timer 2 interrupts
   OpenTimer2((T2_ON | T2_SOURCE_INT | T2_PS_1_4), PWM_PERIOD-1);
   ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_1);

   INTConfigureSystem(INT_SYSTEM_CONFIG_MULT_VECTOR);
   INTEnableInterrupts();

}
```

Listing B.2. Timer 2 ISR to Read ADC10 Channel 2 and Set PWM

```
int speed;    // variable is required to be declared global

void __ISR(_TIMER_2_VECTOR, IPL1SOFT) Timer2Handler(void)
{
unsigned int offset;
unsigned int channel2;
int
// Determine buffer offset
   offset = 8 * ((~ReadActiveBufferADC10() & 0x01));

// Read the result of channel 2 conversion from the idle buffer
   channel2 = ReadADC10(offset);             // Read the analog buffer
   speed = (channel2 * 100) / ADCMAX; // Convert to PWM in %

// User supplied code to determine required motor control mode as per
// Requirements and "MOTOR_CTRL" declaration in Listing 6 below.

   motor(motor_control_mode, speed);         // Set motor rotation direction and speed

   mT2ClearIntFlag();                // Clear Timer 2 interrupt flag
}
```

Listing B.3. Input Capture and Timer 3 Initialization

```
#define T3_TICK               0              // Maximum Timer 3 period
#define TACHout               BIT_6          // RF6 is for tachometer instrumentation
tachInit(void)
{
   PORTSetPinsDigitalOut(IOPORT_F, TACHout);   // Instrumentation only

// Enable Input Capture Module 1
   // - Capture every rising edge
   // - Enable capture interrupts
   // - Use Timer 3 source
   // - Capture rising edge first
   ANSELGbits.ANSG9 = 0;                     // Set RG9 as digital IO
   TRISGbits.TRISG9 = 1;                     // Set RG9 as input
   ConfigCNGPullups(CNG9_PULLUP_ENABLE);     // Enable pull-up resistor
   IC1R = 0b00000001;                        // Map RG9 to Input Capture 1
   OpenCapture1( IC_EVERY_RISE_EDGE | IC_INT_1CAPTURE | IC_TIMER3_SRC |\
                 IC_FEDGE_RISE | IC_ON );
   ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_3 | IC_INT_SUB_PRIOR_0);


// Timer 3 initialization
   mIC1ClearIntFlag();
   ConfigIntTimer3(T3_INT_ON | T3_INT_PRIOR_2 | T3_INT_SUB_PRIOR_0);
   OpenTimer3( T3_ON | T3_PS_1_16, T3_TICK-1);
   mT3IntEnable(1);                          //  EnableIntT3 - ISR for Timer 3 is required


}
```

Listing B.4. Input Capture ISR

```
void __ISR( _INPUT_CAPTURE_1_VECTOR, IPL3SOFT) Capture1(void)
{
   ReadCapture1(con_buf);          // Read captures into buffer
   PORTToggleBits(IOPORT_F, TACHout); // For tachometer instrumentation


// User supplied code to determine the period between two successive interrupts


   mIC1ClearIntFlag();
}
```

Listing B.5. Timer 3 ISR

```
void __ISR(_TIMER_3_VECTOR, IPL2SOFT) Timer3Handler(void)
{
// User supplied code to blink LED 3 once each second for indication only
   mT3ClearIntFlag();
}
```

Listing B.6. PWM Constants and Macros Definitions

```
// Motor driver output pin assignments
#define AIN1bit         BIT_3                       // RB3
#define AIN2bit         BIT_8                       // RE8
#define ENAbit          AIN2bit                     // RE8
#define MODEbit         BIT_1                       // RF1

// Motor drive pin control macros
#define setPHASEA1(a);  {if(a) LATBSET = AIN1bit; else LATBCLR = AIN1bit;}
#define setPHASEA2(a);  {if(a) LATBSET = AIN2bit; else LATBCLR = AIN2bit;}
#define setENA(a);      {if(a) LATESET = ENAbit; else LATECLR = ENAbit;}

// IO pin mapping constants
#define PPS_RE8_OC2 0b00001011                      // Map RE8 to OC2 for PWM
#define PPS_RB3_OC4 0b00001011                      // Map RB3 to OC4 for PWM

// PWM period constants
```

```
#define PWM_PERIOD      (GetPeripheralClock()/1000)-1  // One ms PWM period
#define PWM100          PWM_PERIOD                     // 100% PWM duty cycle

// Motor control macros – the parameter "a" is the PWM percentage multiplied by the
// PWM period and divided by 100.
#define MOTOR_MODE(a)   { if(a) {LATFSET = MODEbit; else LATBCLR = MODEbit;}
#define MOTOR_COAST();  { SetDCOC2PWM(0); SetDCOC4PWM(0) }
#define MOTOR_CW(a);    { SetDCOC4PWM(a); SetDCOC2PWM(0) }
#define MOTOR_CCW(a);   { SetDCOC4PWM(0); SetDCOC2PWM(a) }
#define MOTOR_STOP();   { SetDCOC2PWM(PWM100); SetDCOC4PWM(PWM100); }
enum  MOTOR_CTRL {COAST=0, CW, CCW, BRAKE} ;
```

Listing B.7. PWM Initialization

```
void motor_init(void)
{
// Set all motor driver outputs zero
   setPHASEA1(0);
   setPHASEA2(0);

// Make motor driver pins outputs
   PORTSetPinsDigitalOut(IOPORT_B, AIN1bit );  //RB3
   PORTSetPinsDigitalOut(IOPORT_E, AIN2bit );  //RE8


// Map Port pins to output compare channels
   RPB3R = PPS_RB3_OC4;            // Map RB3 to OC4 for PWM
   RPE8R = PPS_RE8_OC2;            // Map RE8 to OC2 for PWM

// Set motor driver IC for parallel outputs
   setMOTOR_MODE(0);


// Initialize two PWM channels
   OpenTimer2((T2_ON | T2_SOURCE_INT | T2_PS_1_1), PWM_PERIOD);
   ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_1);
   OpenOC2((OC_ON|OC_TIMER_MODE16|OC_TIMER2_SRC|OC_PWM_FAULT_PIN_DISABLE),
           0, 0);
   OpenOC4((OC_ON|OC_TIMER_MODE16|OC_TIMER2_SRC|OC_PWM_FAULT_PIN_DISABLE),
           0, 0);


// Enable all interrupts
   INTConfigureSystem(INT_SYSTEM_CONFIG_MULT_VECTOR);
   INTEnableInterrupts();
}
```

Listing B.8. Code Listing Motor Direction and Speed Control

```
void motor(enum MOTOR_CTRL mc, unsigned int speed)
{
unsigned int pwm;
   pwm = ((speed * PWM_PD)/100) - 1;    // Compute PWM setting values for Lab 6a only
/* Insert code that implements the specifications for Lab 6b here */
   switch (mc)                 //  Determine direction of rotation
   {
       case COAST:
           MOTOR_COAST();      // Set all outputs off
           break;
       case CW:
           MOTOR_CW(pwm);      // Set CW speed
           break;
       case CCW:
           MOTOR_CCW(pwm);     // Set CCW speed
           break;
       case BRAKE:
           MOTOR_BRAKE();      // Set all outputs on to short motor inputs
           break;
   }
}
```