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

Lab 6a: Open-loop Process Control

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

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 open-loop and closed-loop motor control.
4. Develop 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 monitoring devices.

2 Basic Knowledge

1. How to configure pins on a Microchip® PIC32 PPS microprocessor.
2. How to implement a real-time system using preemptive foreground – background task control.
3. How to generate a PWM output with the PIC32 processor.
4. How to configure the Analog Discovery 2 to display logic traces.
5. 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
7. Digital Multimeter

3.2 Software

The following programs must be installed on your development workstation:

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 (such as Microsoft Excel)

4 Project Takeaways

1. How to read 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 and Input Capture to implement a tachometer.
4. Fundamental analog and filtering concepts for data smoothing and open-loop control.

5 Fundamental Concepts

Open-loop control involves one or more inputs, some type of conversion or means of combining the values of the inputs, and one or more outputs represented by the block diagram in Fig. 5.1. As was demonstrated in Unit 2, stepper motors are often used for open-loop control of position. A stepper motor rotates to one of a number of fixed positions, according to its internal construction. Sending a stream of electrical pulses to it causes it to rotate by exactly that many steps, hence the name. Such motors are often used, together with a simple initial datum sensor (a switch that is activated at the machine's home position), for the control of simple robotic machines such as the commonplace inkjet printer head. The drawback of open-loop control of steppers is that if the machine load is too high, or if the motor attempts to move too quickly, then steps may be skipped. The controller has no means of detecting this and so the machine continues to run slightly out of adjustment, until reset. For this reason, more complex robots and machine tools instead use servomotors rather than stepper motors, which incorporate position encoders and closed-loop controllers.[1]

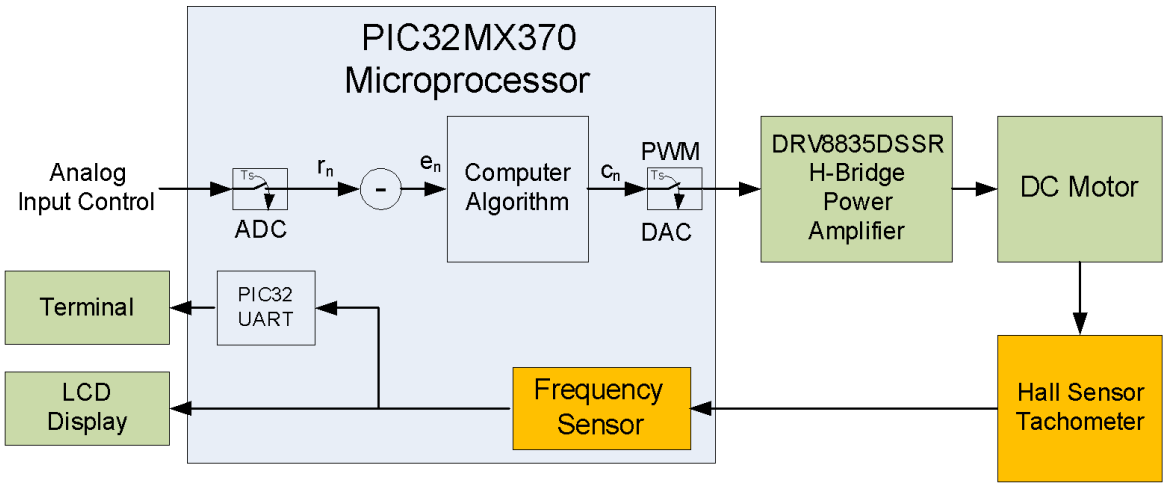

Figure 5.1. Open-loop motor control block diagram.

[1] Open-Loop Controller, https://en.wikipedia.org/wiki/Open-loop_controller

6 Problem Statement

As illustrated in Fig. 5.1, open-loop control does not use feedback to the control processing that indicates how the output is actually working. The open-loop portion of this lab will use the potentiometer labeled *Analog Control Input* identified as R134 on the Basys MX3 schematic, as shown in Fig. 7.1. It will be used to set the percent PWM to operate the speed of the motor from 0 rpm to the maximum the motor can achieve given the motor performance and the supply voltage. Monitoring of specific signals will validate the control of the PWM output by the *Analog Input Control* potentiometer using the Analog Discovery 2 and a digital multimeter.

7 Background Information

7.1 Analog Speed Control

The potentiometer labeled the Analog Input Control (R134) on the Basys MX3 trainer board, as shown in Fig. 7.1, will be used as the input for controlling the motor speed. The potentiometer wiper terminal is connected to the PIC32MX370 processor Port B pin 2. Using the PIC32 analog-to-digital converter (ADC) is rather straightforward with the aid of the example code provided by the MPLAB X Help for the XC32 Peripheral Libraries. Listing B.1 in Appendix B shows how to initialize an ADC channel to continuously sample and convert analog data. Listing B.2 is example code to read the most recent ADC conversion and call functions to set the motor direction of rotation and speed. The motor control is discussed in section 7.4 below.

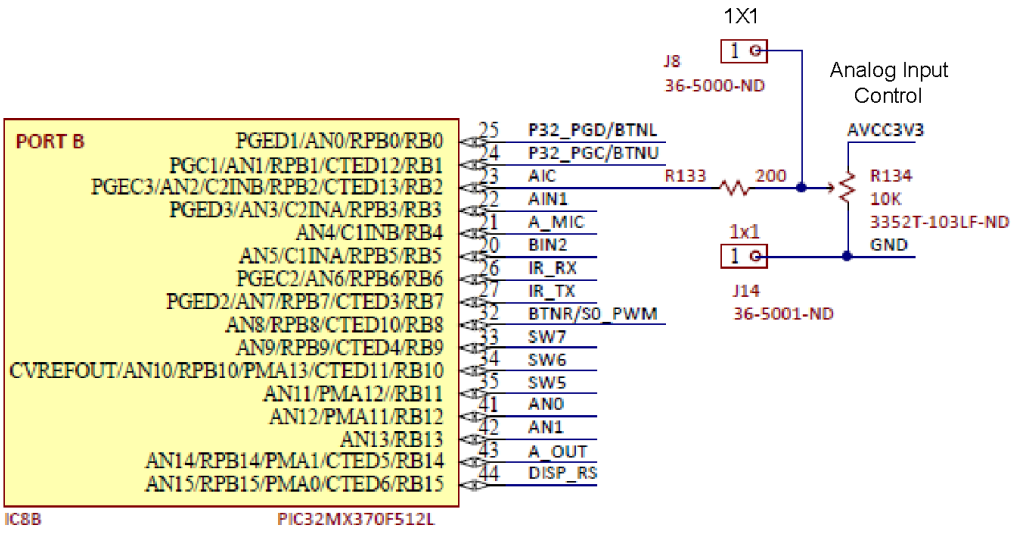

Figure 7.1. Analog Input Control schematic diagram.

7.2 Motor Speed Indication

Section 7.2.1 of the Unit 6 tutorial describes the details of measuring the period of the pulse output of a digital tachometer. Listing B.3 and B.4 of this text present the code for initializing the input capture to generate an interrupt on every positive transition and the interrupt service routine that computes the period between successive captures. Since the period measurement is instantaneous, it is very sensitive to input noise. One means of reducing the effects of noise on a measurement is to implement a low-pass filter using a moving average (MA) algorithm. This common smoothing algorithm is expressed in Eq. 7.1. The smooth output, *y[i]*, requires the N-1 past inputs to be summed and that sum to be divided by the value of N. The filter expressed in Eq. 7.1 is a non-

recursive implementation that only uses past inputs to generate the newest output average. It requires N addition operations and one division as well as N-1 memory locations.

$$y[i] = \frac{1}{N}\sum_{j}^{N-1} x\,[i+j] \qquad \text{Eq. 7.1}$$

Eq. 7.2 is a recursive implementation of the same moving average. The newest average is a function of only two inputs, the present input and the input captured N samples back, and the last computed moving average, y[n-1]. The advantage of Eq. 7.2 is that only two additions (one add and one subtract) and one division is needed regardless of the size of N, while requiring N+1 memory locations.

$$y[n] = \frac{(x[n]-x[n-N])}{N} + y[n-1] \qquad \text{Eq. 7.2}$$

Software timing tests reveal that 24 µs is required for an 8-point MA filter µs using Eq. 7.1, while only 8 µs is required for Eq. 7.2. The advantage of Eq. 7.2 is even greater when one realizes that the execution time of Eq. 7.2 does not depend on the value of N.

7.3 DC Motor Power Control

In this lab we will be using a permanent magnet brushed DC motor, as shown in Fig. 7.2, where the speed of a DC motor is proportional to the applied DC voltage. References 3 and 4 provide insight on the speed control characteristics of this type of electric motor. The characteristic of interest to us is that the speed of the motor is roughly proportional to the applied DC voltage. The challenge for Lab 6a is how best to generate a variable DC motor supply using the PIC32 processor.

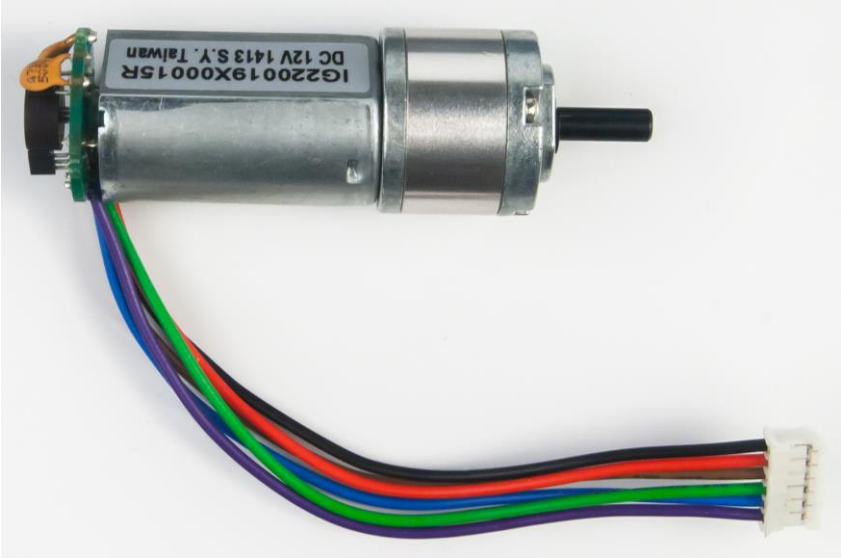

Figure 7.2. DC Motor with digital tachometer.

The motor driver IC on the Basys MX3 processor platform is shown in Fig. 7.3. The DC motor with a Hall sensor tachometer that will be used for this lab is shown in Fig. 7.2. The motor wires used for power input and tachometer output are identified in Fig. 7.3. A 5.0 V external power supply is required to be connected to J11, marked as VBAR on the Basys MX3 processor board. A jumper must be used on JP1 to connect the 5V power, VBAR, to the motor supply, VM.

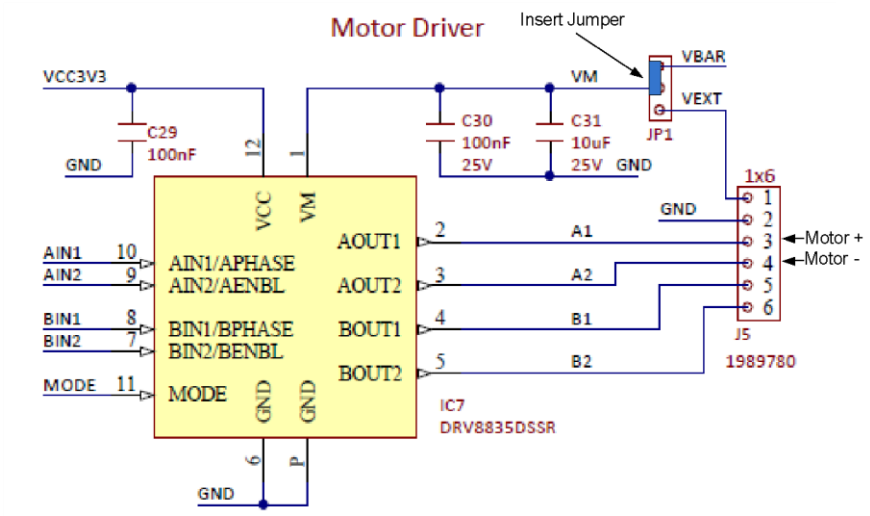

Figure 7.3. Basys MX3 Motor driver circuit.

Table 7.1. Motor driver IC7 connection to PIC32 processor.

Function	IC7 Pin	PIC32 Port Pin	Function	Motor Connector
AIN1	10	RB3	Motor Supply +	Red
BIN1	8	RE9	Not Used - Set to zero	
AIN2	9	RE8	Motor Supply -	Black
BIN2	7	RB5	Not used – Set to zero	
MODE	11	RF1	Set to 0	
	Pmod JA			
Vdd	JA-12		+3.3V	Brown
Gnd	JA-11		Gnd	Green
SA	JA-10	RG9	Hall Sensor A	Blue
SB	N/C		Hall Sensor B	Purple

For the motor to turn clockwise, the A1 output must be held low and the A2 output must be PWM modulated. The opposite is true of turning the motor in the counterclockwise direction. A PWM output can be held at zero level simply by setting the output compare value to zero. Similarly, a PWM value can be held high by setting the output compare to a value equal to or greater than the timer period. Example code for configuring and implementing a bidirectional variable speed control of a DC motor is shown in Listing B.6 through B.8.

7.4 DC Motor Speed Control

PWM output from a PIC32 requires the Output Compare and a reference timer. The OCMP example provided in the MPLAB XC32 Peripheral Library documentation provides an example of setting up a PWM output using Output Compare channel 1. For Lab 6a, we need only a 16-bit timer, hence we will use only Timer 2. If we want the PWM carrier frequency to be 1 kHz, we must set the PWM period to 1 ms.

The Output Compare channel that implements the PWM uses Timer 2 to set the period and duty cycle. To allow the motor to rotate in a given direction, PWM output for signal AIN1 will be set to the desired duty cycle while the duty cycle for AIN2 will be set to zero. To reverse the direction of rotation, AIN1 will be set to zero and AIN2 will be set to the desired duty cycle using the macro instructions shown in Listing B.6 as *SetDCOC2PWM(a);* where the variable "*a*" equals *PWM_PERIOD * duty_cycle/100* and *%duty cycle* ranges in value from 0 to 100. Since the

analog input varies between 0 and 1023 while the motor control varies between zero and 100%, the duty cycle variable is computed from the following equation.

$$\%\ duty\ cycle = \left(\frac{AN2*100}{1023}\right) \quad \text{Eq. 7.3}$$

Note that since we are using integer math, truncation errors will be minimized if multiplication is completed prior to the division operation, which is the C inherent precedence unless forced to do otherwise, provided that the multiplication does not exceed the maximum value for an unsigned integer. Listings B.6 through B.8 show the code needed to setup and control the two PWM channels.

8 Lab 6b

8.1 Requirements

1. Phase 1: Background Tasks
    a. All background tasks will be executed once each 250 ms, which includes reading the following actions: Reading the Analog Input Control voltage, reading the states of slide switches SW5 through SW7, and changing the PWM secondary compare registers.
    b. 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

2. Phase 2: Foreground Tasks
    a. The PWM period is set for 1 ms using Timer 2.
    b. The Timer2 ISR is to complete the following:
        i. The “Analog Input Control” on the Basys MX3 is to adjust the percent PWM from zero to 100 %.
        ii. The percent PWM must be converted to motor speed based on the percent of the PWM period.
3. Phase 3: Background Tasks
    a. The tachometer speed in revolutions per second (RPS) is to be displayed on the Character LCD. The first line displays “PWM = ##%”. The second LCD line displays “RPM=####”. (The motor speed in RPS is equal to the tachometer frequency in Hz.)
    b. The same data that is displayed on the LCD is to be sent to the UART channel as a single line of text at 38200 BAUD with no Parity.
4. Phase 4: Foreground Tasks
    a. The motor speed is adjusted by writing to the PWM output control register each time the Timer 2 interrupt occurs.
    b. Basys MX3 JA-10 (PIC32MX370 Port G pin 9) is to be used for the Input Capture channel 1.

c. The tachometer frequency is to be updated each positive transition of the tachometer input signal.
d. Timer 3 is to be used as the time reference for input capture.
e. An interrupt is to be generated on each positive transition of the tachometer signal.
f. The input capture ISR is to compute the motor speed in RPS and toggle Port F pin 6. (This pin is assigned to be the SPI CLK pin but since we are not using SPI communications in this lab, we will use it as an indication of a tachometer transition.)

8.2 Design Phase

This design will be completed in two phases. The first phase results in the open loop motor control using the Digilent Analog Discovery 2 as the instrumentation to verify correct motor control. The second phase adds the tachometer to the design to allow the PIC32 to display the motor performance on the LCD.

1. Phase 1
    a. Create a data flow diagram that partitions this assignment into task functions as follows:
        i. A system hardware and resource initialization.
        ii. Motor operation control using switches SW5 through SW7.
        iii. PWM duty cycle control using the Analog Input Control potentiometer.
    b. 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. Tachometer period measurement
        ii. LCD display
        iii. UART output
    b. Create the control flow diagrams that describe the process to implement the design requirements for Phase 2.

8.3 Construction Phase

1. Phase 1 Construction
    a. Monitor the PIC32 PWM outputs using the Analog Discovery 2 set to display digital channels 12 and 13 that are connected to AIN1 and AIN2.
    b. Develop code for Phase 1 of the project design.
        i. Initialize the ADC10, Input Capture, and the Output Compare as per Listings B.1, B.3, B.6, and B.7.
        ii. Configure the ADC10 for continuous sample and conversion as per Listing B.1.
        iii. Configure the output compare to generate PWM signals for AIN1 and AIN2 that is updated each PWM cycle. (See Listings B.6 and B.7.)
        iv. Configure the Input Capture Channel 1 to trigger an interrupt on every positive transition of tachometer signal connected to JA-10 (Port G pin 9) as per Listing B.3. Note: Pin 6 of Port F is to be toggled each timer capture interrupt. This pin is assigned to the Analog Discovery 2 DIO 4 and is not used for the SPI clock. The pin is configured in Listing B.3 and toggled in Listing B.4.
        v. Complete the Timer2 ISR to read the Analog Input Control and set the PWM output as per Listing B.2.

vi. Complete the background coding that has an infinite 250 ms loop with no other functionality operations (at this point).

c. Proceed to Phase 1 testing.

2. Phase 2 Construction
   a. Attach the workstation monitor and launch the terminal emulator application.
   b. Port in the LCD code from Lab 3a or 3b into this project and display the motor Analog Control Input value, the percentage 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. Modify the background task to update the LCD and UART once each 250 ms.

8.4 Testing

1. Phase 1
   a. Set SW5, SW6 and SW7 low.
   b. Connect a digital voltmeter from the AIC terminal to the GND terminal.
   c. Adjust the Analog Input Control for a digital voltmeter reading of approximately 1.7 V for 60% PWM.
   d. With the project running, set the Analog Discovery 2 to capture the waveforms for AIN1 and AIN2. The display should be similar to Fig. 8.1 when SW7 is off (low) and similar to Fig. 8.2 when SW7 is on (high).

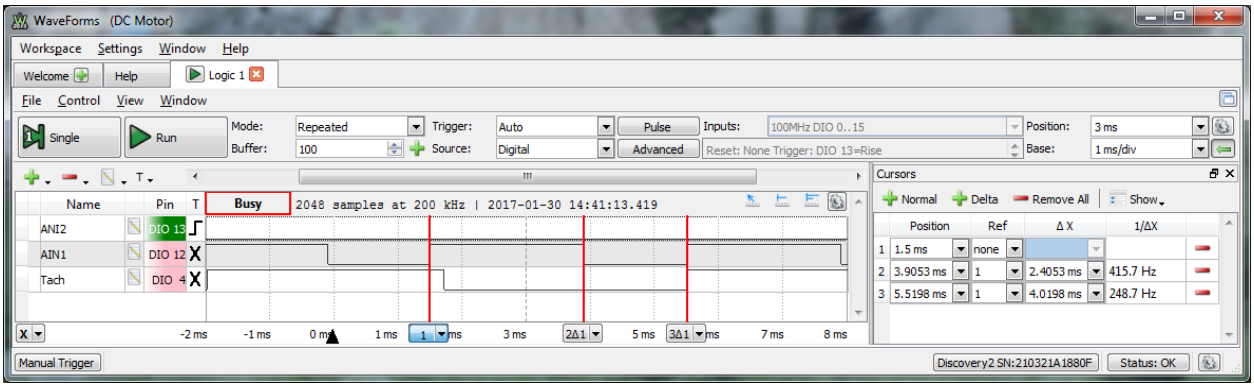

Figure 8.1. H-Bridge inputs for CW motor rotation for 60% PWM.

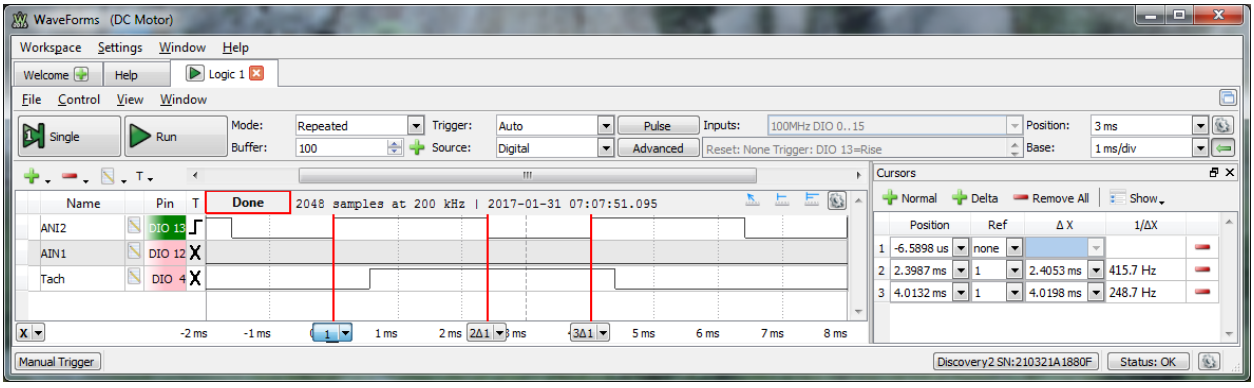

Figure 8.2. H-Bridge inputs for CCW motor rotation for 60% PWM.

e. Capture a waveform as shown in Fig. 8.1 or Fig. 8.2 to determine the PWM period and duty cycle. Record the PWM period and duty cycle values.

e. Observing the signal traces on the Analog Discovery 2 Logic Analyzer, verify that the AIN1 and AIN2 outputs conform to the four operational modes listed in Table 8.1 above.
f. Repeat Testing Steps 1.a, 1.c, 1.e above for the Analog Control Voltage set for 1.0 V and 3.0 V.

2. Phase 2.
   a. Vary the percentage PWM from 0% to 100% in 11 steps using the Analog Input Control potentiometer on the Basys MX3 board and record the displayed Analog Control Voltage, the percentage PWM, and the motor speed in RPM.
   b. Enter the 11 measurements made in part a into a spreadsheet program, such as Microsoft Excel.
   c. Plot the motor speed vs. percent PWM as demonstrated in Fig. 8.3. All motors have different power-speed characteristics.

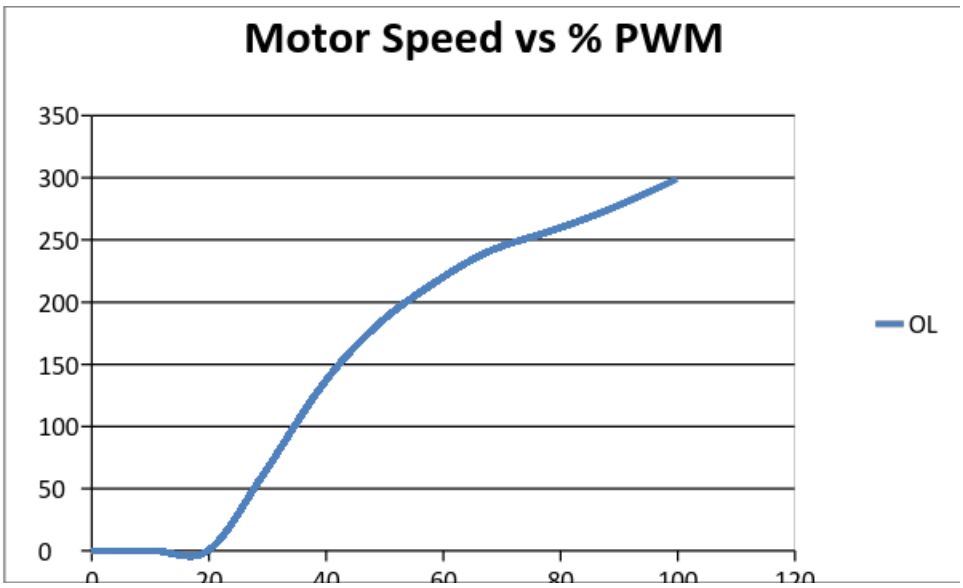

Figure 8.3. Example plot of motor speed vs. %PWM.

9 Questions

1. After plotting the data in Phase 2 testing part c, describe the resulting curve.
2. Explain whether Input Capture, Timer 2, or Timer 3 should be set to the higher interrupt level.
3. If Input Capture, Timer 2, and Timer 3 are set at the same priority level, explain which interrupt should be set to the higher sub interrupt level.
4. Explain the difference in interrupt operation between question 2 and question 3.
5. Consider the case when the motor draws more current than can be supplied by a single H-Bridge driver and the outputs from A1 must be in parallel with B1 and A2 in parallel with B2. What effect on motor operations does changing the primary output compare registers (OCxR) instead of the secondary output compare registers (OCxRS)?

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. "Brushed 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. "AN1353 Op Amp Rectifier, Peak Detectors and Clamps", Dragos Ducu, Microchip Technologies Inc., 2011, http://ww1.microchip.com/downloads/en/AppNotes/01353A.pdf.
8. "Simple Methods for Detecting Zero Crossing", Richard Wall, Proceedings of The 29th Annual Conference of the IEEE Industrial Electronics Society Paper # 000291, ISBN 0-7803-7906-3, Nov. 6, 2003.
9. AD8561 Single Supply Comparator, http://www.analog.com/media/en/technical-documentation/data-sheets/AD8561.pdf.
10. "Adding Extra Hysteresis to Comparators", https://www.maximintegrated.com/en/app-notes/index.mvp/id/3616.

Appendix A: Lab 6a Motor Configuration

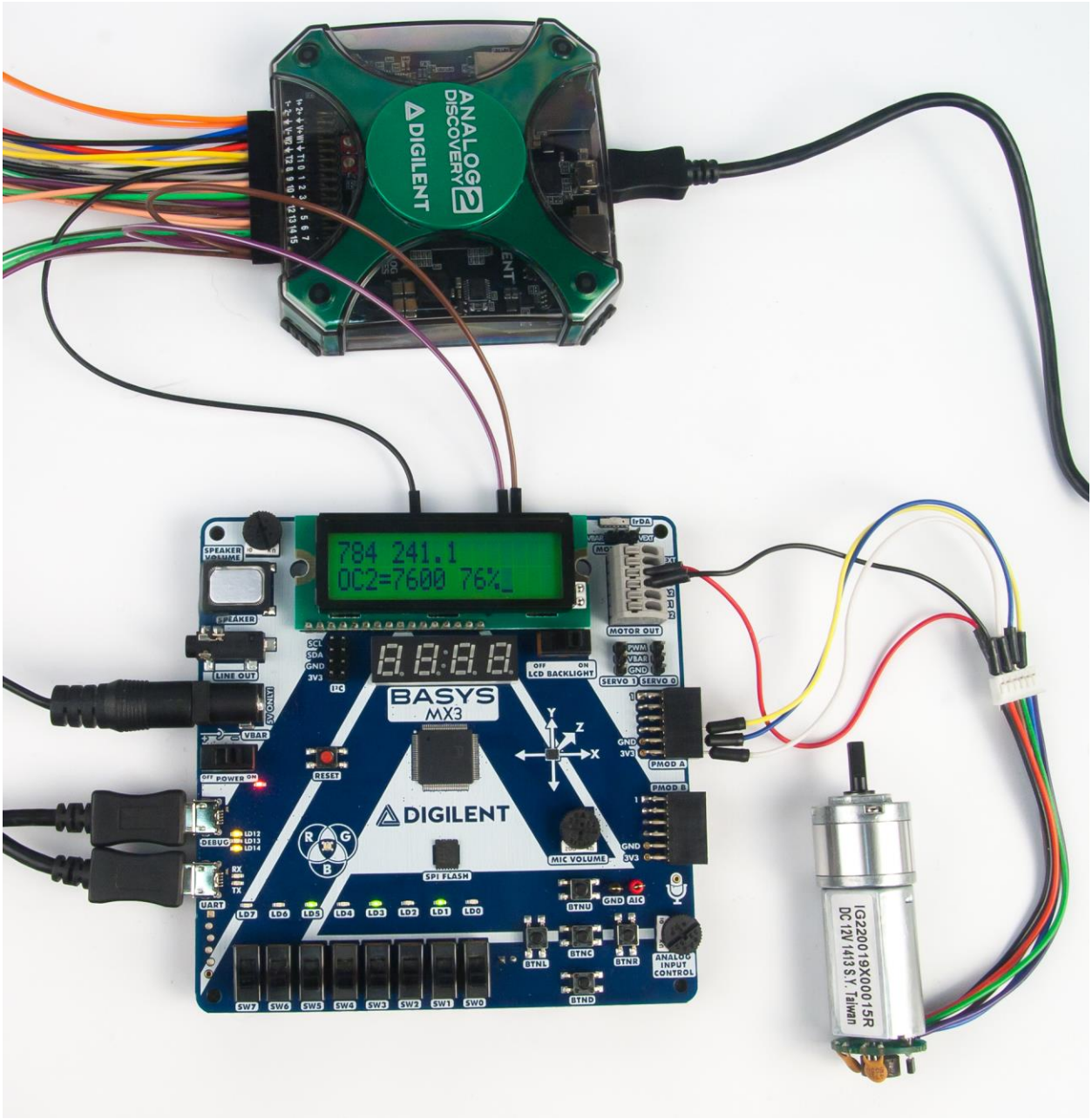

Figure A.1. DC Motor connection to the Basys MX3 processor board.

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;
   }
}
```