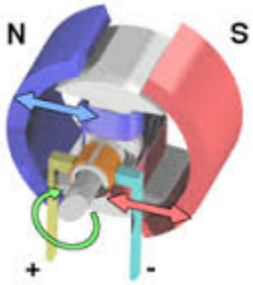
Revision: 05 Nov 2017 (JFF)
Richard W. Wall, University of Idaho, rwall@uidaho.edu

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Project 9: PWM using Output Compare



Purpose

The purpose of this project is to learn how to generate a proportional output using the output compare resource on the PIC32MX processor to implement digital to analog conversion with pulse width modulation (PWM). The proportional output will be used to control the speed of a DC motor.

Minimum Knowledge and Programming Skills

1. Knowledge of C or C++ programming
2. Working knowledge of MPLAB IDE
3. IO pin control
4. Understanding of PWM principles
5. Understanding of Fourier Series fundamentals
6. Use of logic analyzer or oscilloscope

Equipment List

1. Cerebot MX7cK processor board with USB cable
2. Microchip MPLAB ® X IDE
3. MPLAB ® XC32 Compiler (documentation support)
4. Digilent PmodCLP Parallel Character LCD
5. Digilent H-Bridge driver PMod
6. Digilent DC Motor
7. Oscilloscope / Logic analyzer (Digilent Analog Discovery)

Software Resources

1. Microchip XC32 C/C++ Compiler Users Guide
2. PIC32 Peripheral Libraries for MBLAB C32 Compiler
3. PIC32 Family Hardware Reference Manual Section 16 Output Compare
4. Cerebot MX7cK Board Reference Manual
5. MPLAB ® X Integrated Development Environment (IDE)\

References

1. Using PWM to Generate Analog Output
2. C Programming Reference
3. PIC32 Output Compare

Fundamentals of D2A

The information Digital IO works well for generating and detecting discrete events. But the real world is continuous hence, for microprocessors to have value, they need to have capability to input and output analog signals. Project 9 investigates one method that a microprocessor can generate a varying amplitude signal. Project 10 investigates a method for sensing a time varying signal. A digital to analog converter (DAC) converts a digital or binary value to an analog voltage. An analog to digital converter (ADC) converts a voltage in a specified range to a binary value that represents the magnitude of the signal.

There are several different technologies used to implement DACs. To say that a continuous time varying signal can be generated from a sequence of binary values is an overstatement. In reality, the output of the DAC can only output discrete voltage levels over a limited range of voltages. Three parameters define a DAC operating characteristics: the resolution, the dynamic range, and the DAC order. The DAC resolution is defined as the change in output level for a change in the least significant bit of the binary input. The dynamic range of the DAC is defined by the maximum output level minus the minimum output level. Bipolar DACs can output both positive and negative voltages. Unipolar DACs output either positive or negative voltages. The order of a DAC is defined by the log base 2 of the number of discrete values that the DAC can output. For example, a 10 bit DAC can output 1024 discrete values.

The DAC performance is measured with these following parameters: conversion speed, differential nonlinearity, integral nonlinearity total harmonic distortion plus noise, zero offset,

monotonicity, and missing codes. A thorough discussion on DAC performance is beyond the scope of this project but it is obvious that better performance comes at a higher cost.

PWM as an D2A Converter

In communications theory, modulation is used to translate the frequency spectrum of information signals to a frequency spectrum centered around a higher fixed frequency. The purpose of modulation is to allow multiple information signals to be simultaneously communicated on a single medium such as wire, fiber optic cable or through the air as electromagnetic or acoustic waves using frequency division multiplexing. When discussing modulation, two terms frequently arise: the information or modulating signal and the carrier signal. The carrier is a constant higher frequency that is modulated or modified by the information or modulating signal. After transmission, the information signal must be recovered or extracted from the carrier signal using a demodulator.

PWM Modulation

Pulse width modulation (PWM) (also referred to as pulse duration modulation or PDM) is commonly used as a method of generating a signal with a microprocessor for two reasons: the PWM signal is easily generated using processor timers and digital comparators, and the PWM signal easily demodulated with little or no external circuitry. Readers who are unfamiliar with PWM concepts are referred to one of the many web based tutorials. The bottom trace in Figure 1 is an example of a PWM signal showing that it is a constant period rectangular wave with a varying duty cycle. The inverse of the constant PWM period is called the PWM cycle frequency or the carrier frequency. The PWM duty cycle is the ratio of high output to total PWM period and is expressed in percent. Hence a 50% PWM duty cycle results in a square wave.

If the modulating signal is a constant, the demodulated PWM signal is a DC level. When working with bipolar modulating signals, the signal level must be offset such that the zero level generates a PWM output with 50% duty cycle. The offset must be removed by the demodulator circuit to recover the bipolar signal. Demodulation circuits are discussed in paragraphs below.

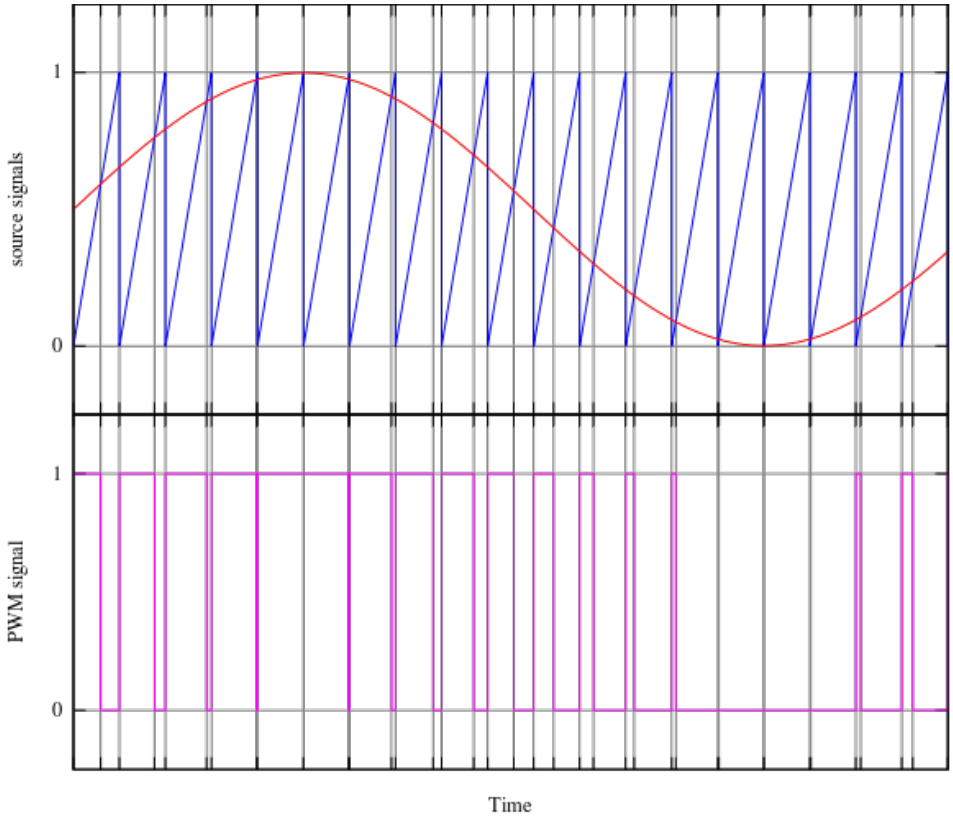


Figure 1. Generation of a PWM signal from an AC source signal

As illustrated in Figure 1, the PWM carrier frequency signal is modulated by changing the duty cycle of the carrier rectangular wave in proportion to the amplitude to the information signal. The top trace in Figure 1 shows the information or modulating sine wave and the sawtooth wave used by the modulator. When the modulating signal has amplitude that is higher than the sawtooth signal, the PWM signal is high as shown in the bottom trace of Figure 1. Conversely, the PWM output is low whenever the sawtooth signal is higher than the information signal.

Microprocessors can generate a PWM output signal using a counter and a digital comparator. The counter synthesizes the sawtooth wave and a data stream synthesizes the modulating wave. Many microprocessors including the PIC32 contain dedicated hardware that can generate a PWM output with minimum software support.

As described above, PWM is a modulation scheme where the information signal is used to modulate a carrier signal. The result of modulation translates or shifts the information signal frequency spectrum to a spectrum that centered on the PWM cycle frequency. The PWM signal has multiple frequency components. Spectral analysis reveals that the PWM signal has the spectrum of the original information signal as well as frequency components at the carrier frequency as well as harmonics of the carrier signal. The bottom trace in Figure 2 shows the frequency spectrum of a PWM signal with a 1KHZ PWM cycle (carrier) frequency and modulated at 65% PWM duty cycle. The amplitude of each of the frequency spikes is a function

of the percent duty cycle. The signal at zero Hz is the spectrum of the information signal. It is recovered by low pass filtering of the PWM signal to remove the carrier frequency at 1 KHz and the harmonics of carrier signal. One can observe the reduced amplitude spikes of the harmonics at multiples of 1 KHz. The harmonics are the frequency components that are needed to generate the square waves of the PWM signal and are not needed to demodulate the PWM signal.

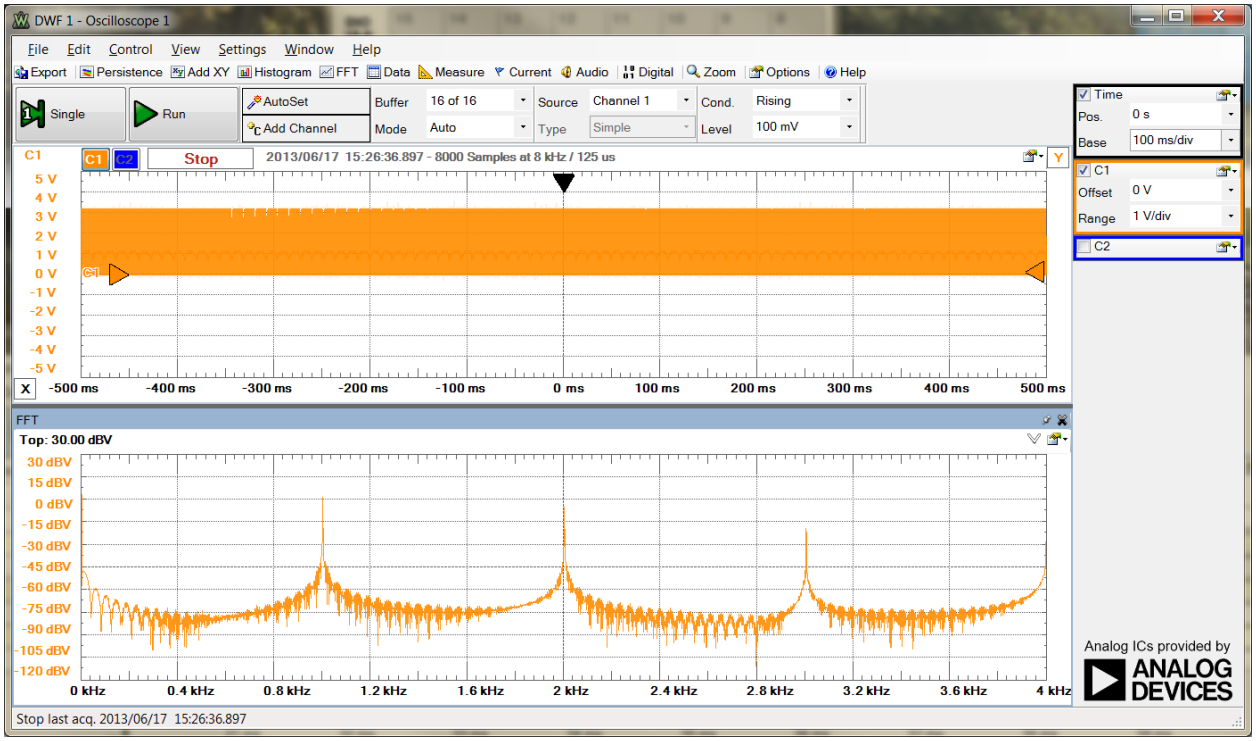


Figure 2. Frequency spectrum of a 1KHz PWM signal with constant 65% duty cycle.

PWM Demodulation

Demodulation is the processes that separates the information signal from carrier signal and returns the signal to its original format. For PWM, the demodulator is an integrator implemented with a low pass filter. If we are reconstructing a bipolar signal, then a circuit must be added to removes the offset or uses a series capacitor to block the offset DC component.

The information signal used to modulate the PWM rectangular wave for this project is a DC signal of different levels. Figure 3 shows the block diagram of a circuit that can be used to demodulate the microprocessor's PWM output to control the speed of a DC motor that will rotate in one direction only. Since the PWM signal has only two states, on and off, the output power can be amplified with a single transistor.

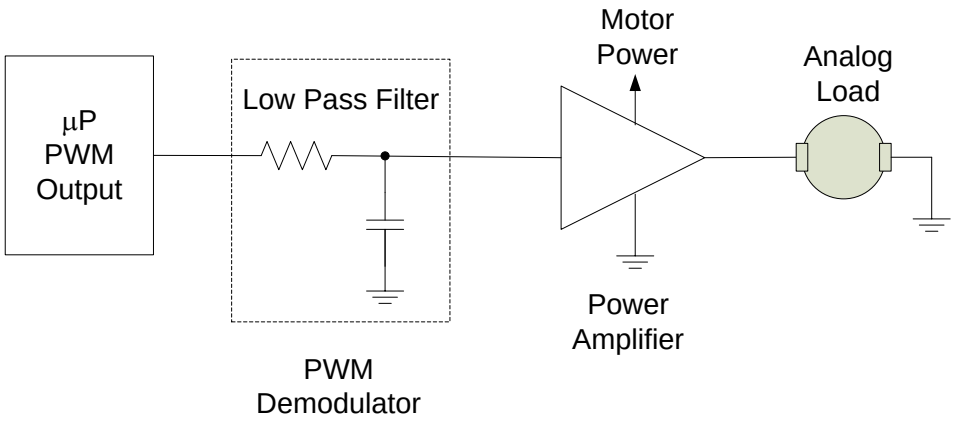


Figure 3. Block diagram of PWM demodulator

However, the DC motor driver circuit provided by the PmodHB5 allows control of the motor speed by connecting the Output Enable pin to the PIC32 PWM output as well as motor direction control by connection to another PIC32 IO pin. Figure **4** shows a conceptual block diagram of an interface of an H bridge driver such as the PmodHB5 with the PIC32 processor and the DC motor. The power amplifier outputs a high voltage level when the enable pin is set high and the direction control pin is also high. When the direction pin is set low and the output enable pin is set high, the power amplifiers outputs a low voltage level. A voltage difference across the terminals of the DC motor causes the motor to turn. Whenever the enable pin is low, the output from the power amplifier is high impedance.

The circuit shown in Figure **4** is one means of implementing bipolar control where the PWM signal contains the signal magnitude and a separate binary signal indicates the sign on the magnitude.

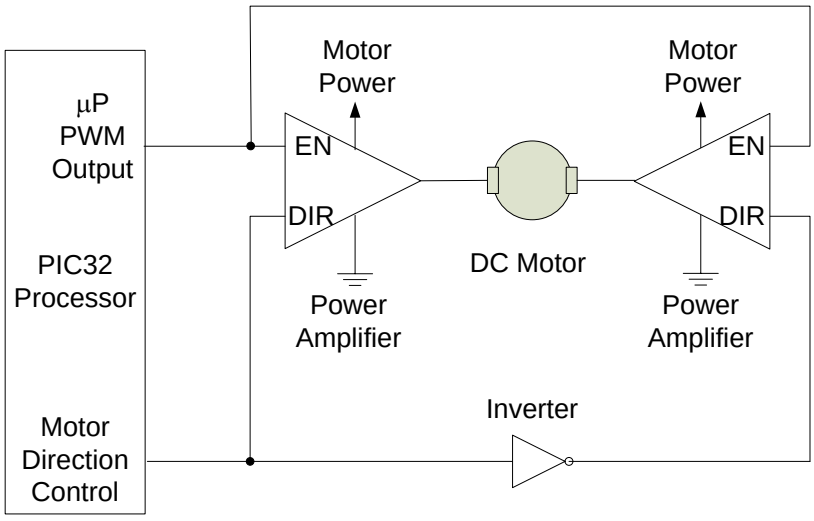


Figure 4. DC motor bidirectional control used in Project 9

For this project, the PWM output will be used to power a DC motor as shown in Appendix A. The PmodHB5 H bridge driver shown in block diagram serves as the power amplifier. There is no need for a low pass filter because the DC motor will respond only to the modulation signal and reject the carrier frequency signal and its harmonics.

PWM with Output Compare

The PWM pulse width is controlled by the PIC32MX processor output compare (OC) module. The reader may wish to review the web PIC32 PWM example as a starting point for this design. Even though there are five PWM outputs available, we are constrained to using either Timer 2 or Timer 3 to generate the PWM cycle frequency. For Project 9, we will use Timer 2. The DC motor is connected to JD on the Cerebot MX7cK board. This connection provides access to output compare channel 3 (OC3) that we will use to generate the PWM output. Although the complete programming details to set up a PWM output are provided below, there are two key parameters that must be specified. The first, PWM_CYCLE_FREQUENCY, is the inverse of the PWM cycle period. The value written to the Timer 2 period register, PR2, determines the PWM frequency, as shown in Equations (1) and (2). ***(Note: PR2 is a PIC32 register and should never be declared as a variable or a constant.)***

PWM_CYCLE_COUNT = PBCLK / (T2_PRESCALE * PWM_CYCLE_FREQUENCY) (1)

PR2 = PWM_CYCLE_COUNT – 1 (2)

The second parameter, PWM_DUTY_CYCLE, is a value between zero and 100. The duty cycle of the PWM output is determined by the ratio of the value written to the output compare register, *OC3RS,* to the value of the Timer 2 period register, PR2, as shown in Eq. 3.

OC3RS = (PWM_DUTY_CYCLE * (PR2+1) / 100) (3)

For example, to generate a 30% PWM duty cycle output when the Timer 2 period register, PR2, is set to 999, the value 300 must be written to the *OC3RS* register. Note that Eq. 3 uses only integers in the math equation resulting in faster processor computation. Integer math always truncates the fractional portion of the result of a divide operation. To preserve resolution, multiply operations should be completed prior to divide operations. Use parenthesis to dictate the order of evaluation of operations.

The resolution of the PWM output compared to an equivalent digital to analog converter (DAC) is the inverse timer PWM_CYCLE_COUNT. The resolution relates to the smallest increment that the output voltage can be changed. For example, if PR2 is set to 999, then the resolution of the PWM output is one part in 1000. This is equivalent to a N-bit DAC where N = $\log_2$(PWM_CYCLE_COUNT) rounded off to the nearest integer. Consequently, for our example, one part in 1000 results in N equal to 10.

For a given timer input clock frequency, the higher the PWM cycle frequency, the lower the DAC resolution. Consider the case where the desired PWM cycle frequency is 20 KHz when the PBCLK is 10 MHz and the Timer 2 prescale set to one. The PWM_CYCLE_COUNT value equals (10E6/20E3) resulting in PR2 being set to the value of 499. A PWM_CYCLE_COUNT of 500 is approximately equivalent to a 9 bit DAC (2^9 = 512).

Appendix A shows a block diagram of the equipment used in Project 9. Appendix B provides the pin connections between the Pmod HB5 and the Cerebot MX7cK. The EN (enable) pin is the PWM input to the Pmod HB5 motor driver circuit. Whenever the EN pin is high, power is applied to the motor. The DIR pin controls the direction of the motor and will be set to zero for this project. The SA and SB pins should be set as inputs but will not be used until Project 10.

Programming the Output Compare

Figure 16.1 of the PIC32 Family Hardware Reference Manual shows the block diagram of the compare output module. This diagram shows that the timer count value is compared with the output compare register. This reference describes multiple modes of the operation output compare module. Readers are directed to read section 16.3.3 of this reference manual. We will not be using the fault protection capability in this project.

The PWM output using OC3 can be set up and controlled using four XC32 peripheral library functions. The process is defined by the following steps.

1. *mOC3ClearIntFlag();* // Clear output compare interrupt flag (not using this interrupt)
2. *OpenTimer2(Set_up_bits, PR2_value);* where PR2_value is the period register value computed for the PWM cycle frequency. Initialize Timer 2 to generate an interrupt at the rate of the PWM cycle using the programming concepts provided in Project 5. In the Timer 2 ISR, toggle the LEDA bit for timing instrumentation.
3. *OpenOCx(OC_configure_bits, nOCxRS, nOCxR);* where nOCxRS is the initial value written to the OCxRS register and nOCxR is the initial value written to the OCxR register. "x" designates the specific OC register being configured in the range of 1 through 5. Refer to the Section 13 of the C32 Peripheral Library Guide for option details.
    a. *OC_configure_bits* (bits are inclusive):
        i. OC_ON //Enables the Output compare processor resource
        ii. OC_TIMER_MODE16 // Timer uses 16 bit mode
        iii. OC_TIMER2_SRC // Selects Timer 2 as input timer
        iv. OC_PWM_FAULT_PIN_DISABLE // set so the fault pin is not used
    b. *nOC3R* – This is a constant or a variable that specifies the initial PWM compare value. This is set to the initial PWM duty cycle setting.
    c. *nOC3RS* – This is a constant or a variable that specifies the next PWM compare value. This should be set to the same value as *nOC3R*.
    d. The range of values for nOC3RS and nOC3R is 0 to PR2+1. PR2 is specified in step 2. If these parameters are set to a value greater than the PRx, the PWM output will be set to a constant high (100% duty cycle).

4. After initialization, the last function, "*SetDCOC3PWM(nOC3RS)*", can be used to change the PWM duty cycle at any time while code is executing.

For this project, the DC motor driver module is connected to Cerebot MX7cK PmoD jack, JD. JD pin 7 is connected to the PIC32 (PORTD) RD1 IO pin and controls the HB5 PMod DIR signal that sets the direction that the motor rotates. RD1 should be configured as a digital output and cleared. The PWM output is on JD pin 8 that is connected to the PIC32 Output Compare 3 (OC3). Hence, we will use the instruction "*OpenOC3(OC_configure_bits, nOC3RS, nOC3R);*" to initialize the PWM.

JD pins 9 and 10 are the tachometer inputs. Although these pins will not be used until Project 10, RD3 and RD12 should be initialized as digital inputs.

Project Tasks

The objective of this project is to implement an open-loop motor speed control. Open loop control is similar to you driving your car down the highway with the accelerator at a fixed position. The car speed will vary depending on the grade of the highway. Similarly, the speed of a DC motor is a function of the DC voltage and the mechanical load. The motor speed will decrease as the mechanical load increases if the applied voltage remains fixed. For this project, the motor speed control will be set to the four fixed PWM duty cycle values shown in Table I. These duty cycle values are selected using the Cerebot MX7cK BTN1 and BTN2 push buttons. The percentage PWM duty cycle will be displayed on line 1 of the LCD. Specifically, the project tasks are as follows:

1. Write and verify the C code to complete the button control and LCD display design specified below.

2. Capture the four logic analyzer screen captures for the PWM for the four settings specified in Table I

3. Capture the LEDA, LEDB and PWM EN signals to demonstrate that the PWM output and Timer 2 interrupts continue to function while the CN interrupt is being served.

Table I. Button controlled PWM

BTN2	BTN1	PWM
OFF	OFF	40%
OFF	ON	65%
ON	OFF	80%
ON	ON	95%

Project Specifications

1. Setup the hardware as shown in Figure 6 of Appendix A.
    a. Be careful to observe the polarity of the 10V power connection to the Pmod HB5 motor driver circuit as shown in Figure 8 of Appendix C.
    b. Attach the DC motor pictured in Figure 9 of Appendix D to the Pmod HB5 motor driver.
    c. Connect the PmodCLP to the Cerebot MX7cK JC and JD. (Refer to Project 6)
    d. Adjust the motor power supply for 10VDC.
2. Set the PWM output as a function of the states of buttons BTN1 and BTN2 as specified in Table I.
3. Detect button uses the change-notice interrupt similar to the Project 5 implementation. The CN interrupt is to use a 20 ms software delay for switch debounce.
4. Toggle LEDA in the Timer 2 ISR every millisecond to indicate that the PWM cycle period is 1 ms. This results in a 500 Hz square wave at probe point LEDA.
5. Writes the PWM percent duty cycle to line 1 of the LCD whenever the buttons change the PWM duty cycle as per Table I.
6. The following are the required functions and descriptions of operations:
    a. "main"
        i. Calls application initialization
            1. Configure Cerebot MX7cK board BTN1 and BTN2 as inputs
            2. Configure PmodSTEP LEDA through LEDD as outputs
            3. Configure Timer 2 to generate an interrupt each millisecond. Group priority level 2 and subgroup priority level 0.
            4. Configure the PWM output channel as follows
                a. Use output compare 3 (OC3)
                b. Uses Timer 2 for the time base for the output compare
                c. PWM cycle frequency of 1000 Hz
            5. Initialize a CN interrupt for button status detection at group priority level 1 and the subgroup priority level 0.
            6. LCD initialization with a function added to position the LCD cursor at any position. (Refer to Project 6)
        ii. Executes a *while(1);* loop that does nothing
    b. Button Detect ISR (refer to Project 5)
        i. Sets LEDB on entry and clears LEDB on exit
        ii. Remove button contact bounce with 20ms software delay
        iii. Reads button state
        iv. Decodes buttons and sets PWM in accordance with Table I
        v. Sets the motor PWM
        vi. Updates only line 1 of the LCD reporting the percent PWM duty cycle using the format: PWM = ##%
        vii. Clears CN interrupt flag

c. Timer2 ISR
    i. Toggles LEDA
    ii. Clears T2 interrupt flag

Project Testing

As shown in the connection diagram shown in Appendix B, the PWM output on the OC3 output pin is connected to the motor enable (EN) pin. After developing the program that meets the project specifications, connect the logic probe to the EN pin (JD pin 8). As shown in Figure 5, verify the PWM output for the four settings listed in Table I. Also verify that the PWM cycle time is 1ms by connecting a logic probe to the LEDA output pin. Connect LEDB pin to the second logic analyzer probe to observe the operation of the PWM when a button is pressed to change the motor speed as shown in Figure 5.

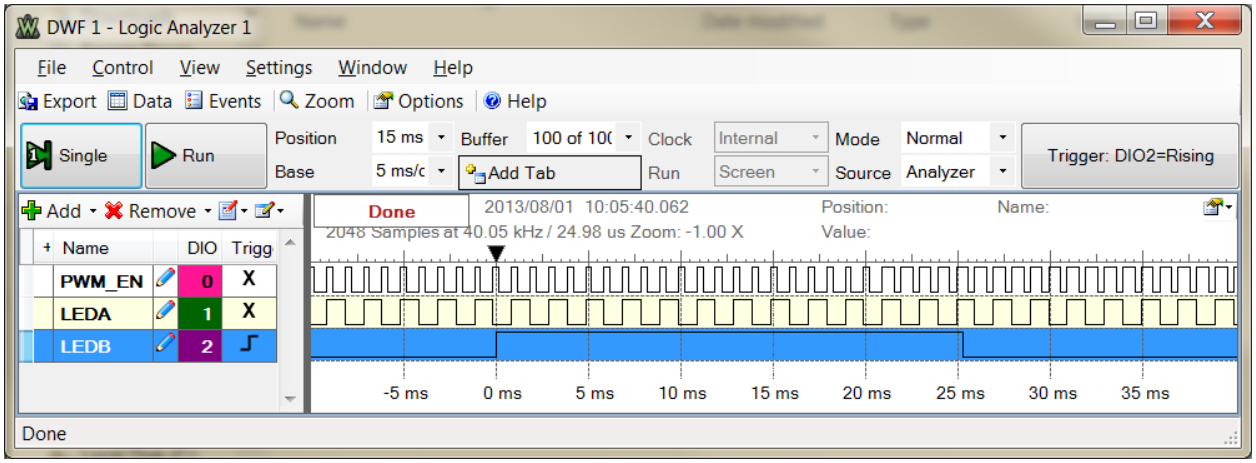


Figure 5. PWM EN signal for 33% duty cycle, Timer 2 Interrupt (LEDA) , and CN interrupt in-progress (LEDB)

Appendix A: Project 9 Parts Configuration

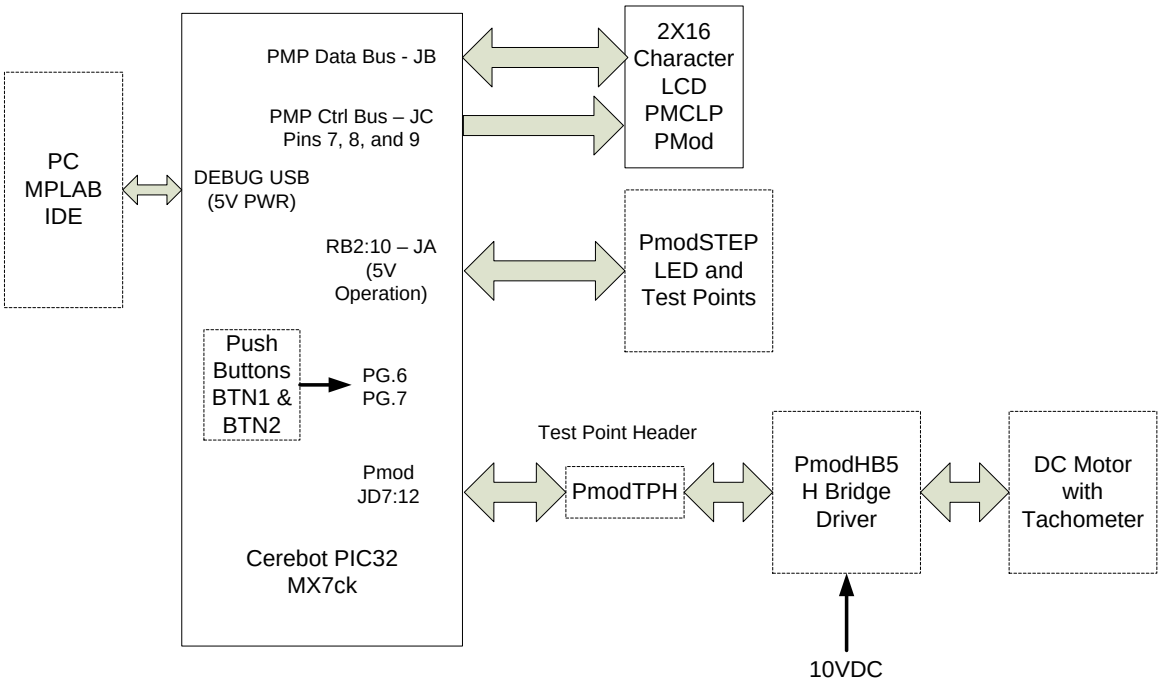


Figure 6. Block diagram of the equipment used in Project 9.

Appendix B: Motor Controller Wiring Diagram

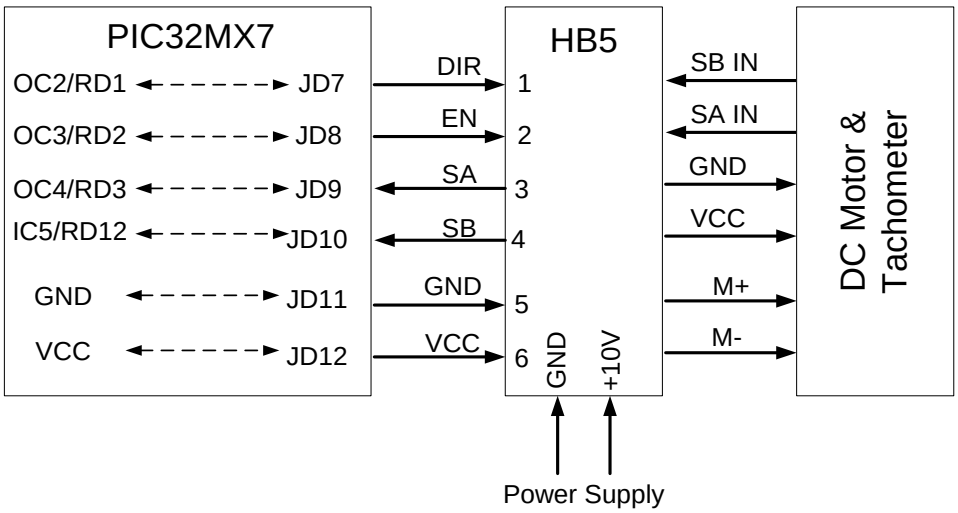


Figure 7. Motor connection diagram

Appendix C: PModHB5 Half H-Bridge Drive

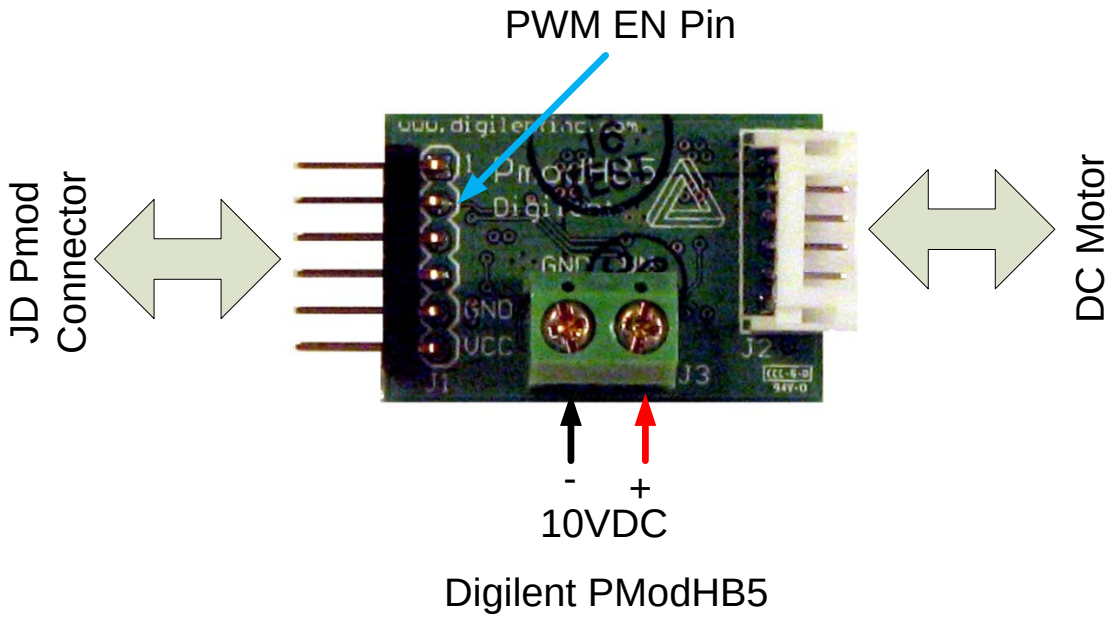


Figure 8. PmodHB5 instrumentation connections

Appendix D: Geared DC Motor

Geared DC Motor with
Tachometer

Figure 9. Digilent DC Motor

Appendix E: PmodCLP

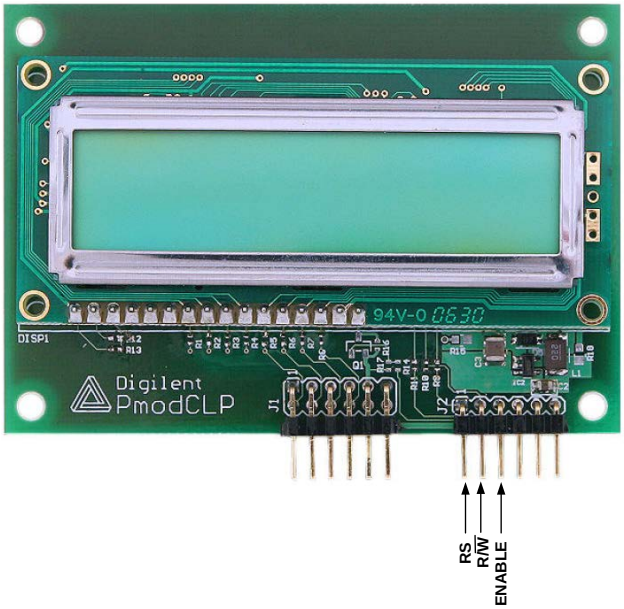


Figure 10. PmodCLP Character LCD pin identification