ECE 218
EXERCISE 5. PULSE WIDTH MODULATION

In this exercise you learn about PWM using the output compare module. The main steps will be described and you will write a program from scratch that outputs a PWM signal on OC1. We will examine the output on the logic analyzer, and then use the signal to drive a positional and a continuous motion servo motor.

OUTPUT COMPARE MODULE REVIEW

The PIC24 Output Compare modules can be used to generate single "one-shot" pulses, a continuous string of fixed pulses, or a pulse-width modulated (PWM) signal. Here we will focus on the PWM signal, which is a signal with a fixed period, and a variable pulse width, as illustrated in Figure 1.

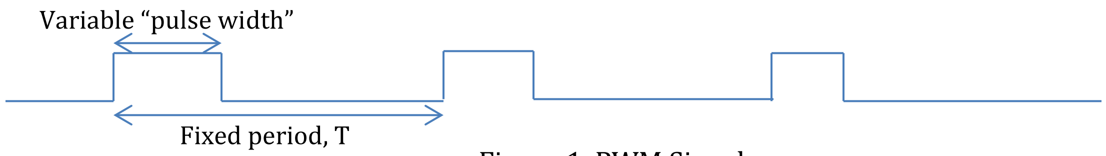
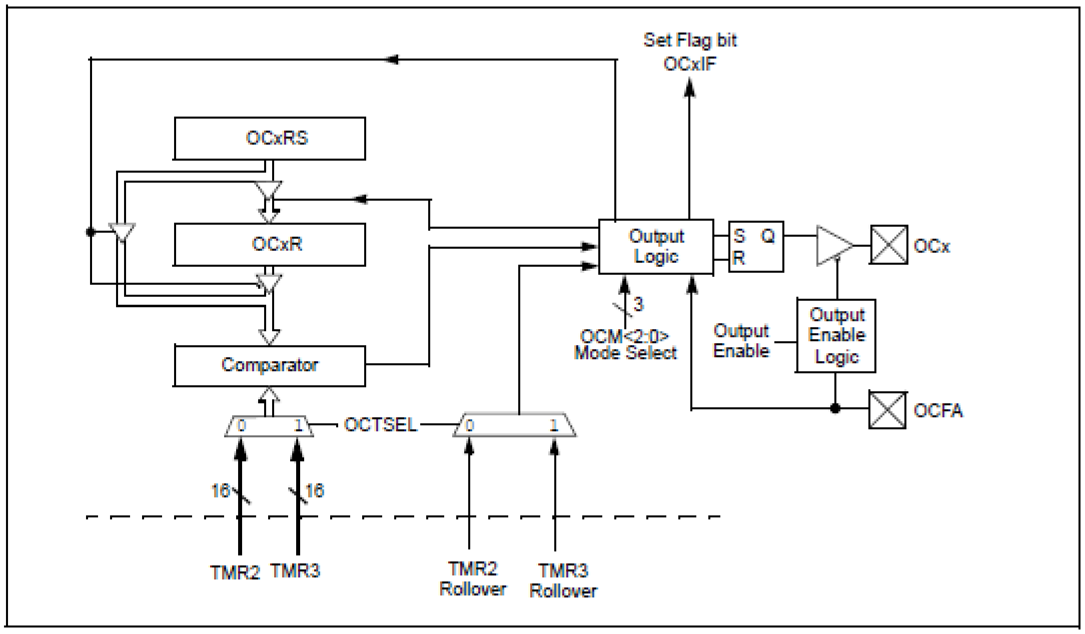


Figure 1. PWM Signal

The "duty cycle" of a PWM signal is defined as the percentage of time that the signal is high.

There are four Output Compare channels on the PIC24 we are using, OC1-OC4. Each Output Compare channel can use either Timer2 or Timer3 as its time reference, and it compares the timer value to one or both of the Output Compare registers, OCxR and OCxRS, depending on the mode used. Figure 15-1 is a block diagram of an Output Compare module from the datasheet.

FIGURE 15-1: OUTPUT COMPARE MODULE BLOCK DIAGRAM



In the PWM mode of the Output Compare, the period of the PWM signal is set by writing to the PR2 or PR3 register for the timer that is selected as a time reference. The pulse width is set by writing to the OCxRS register. The pulse width value can be written at any time, but the pulse width value is not latched onto the OCxR register until the timer resets on a period match. In PWM mode, OCxR is a read-only register. In this exercise, we will be using Output Compare channel 1, and Timer 2. The registers we will deal with are:

• T2CON	TMR2 control register
• PR2	Timer 2 preset register for setting the PWM signal period
• OC1CON	OC1 control register
• OC1R and OC1RS	registers for setting the PWM signal duty cycle

Hardware Setup

In this exercise we will generate the PWM signal on remappable pin, RP1 (pin 5). The signal will be used to set the position of a standard RC servo, and to set the speed/direction of a continuous RC servo. We will also use a logic analyzer to view the PWM signal. Make the connections shown in Figure 2, being careful to connect the power source with the correct polarity, and connect the ground of the Microstick to the ground of the battery pack. The two types of servo motors have color coded signals:

Dark brown=GND, Dark orange=Vdd, Yellow=PWM

Black = GND, Red = Vdd, White = PWM

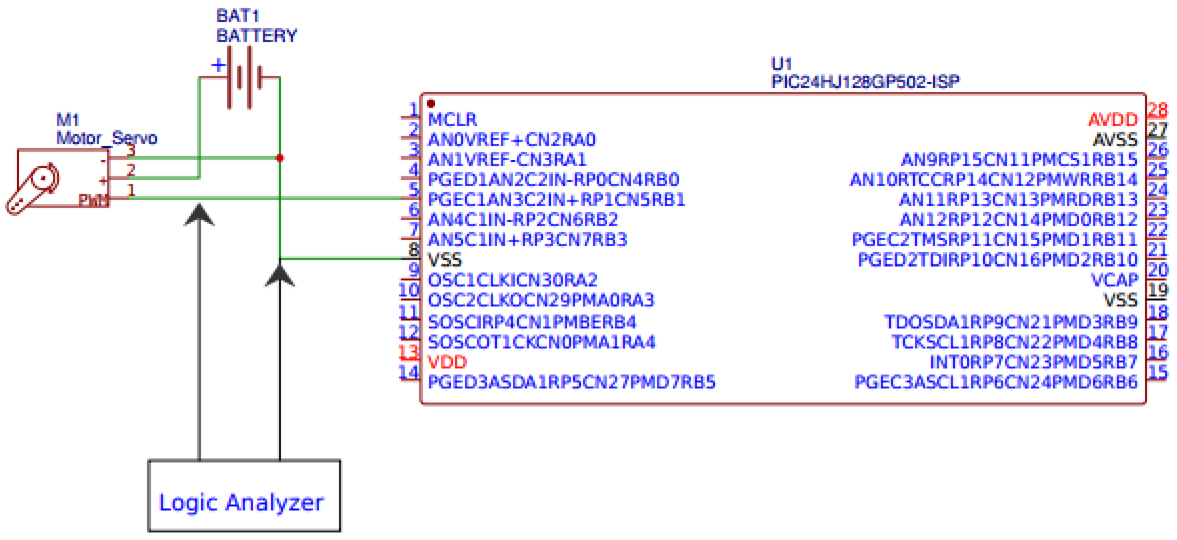
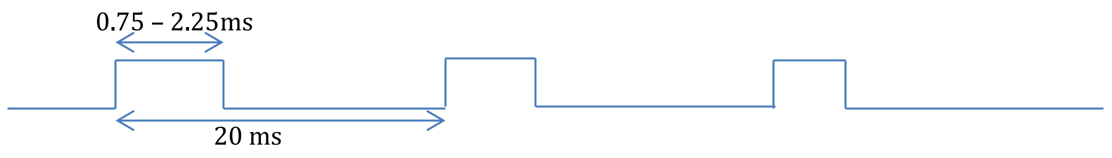


Figure 2. PIC24 Connections to Servo and Logicport

Standard servo motors convert PWM signals to positions. The positional servo used in this exercise has the following specification, and the continuous rotation servo is similar:

Communication: Pulse-width modulation, 0.75–2.25 ms high pulse, 20 ms intervals



Timer2 will be configured with a 1:256 prescale to create a clock period, Tck, of 6.4us. Convert the specifications of the servos to number of Tcks in the table below.

Servo parameter	**Positional Servo**	**Continuous Rotation servo**
p_min (minimum pulse width	0.75ms = 117.1875 Tck	1.3ms = 203.125 Tck
p_max (maximum pulse width)	2.25ms = 351.5625 Tck	1.7ms = 265.625 Tck
middle	1.0ms = 156.25 Tck (midway)	1.5ms = 234.375 Tck (stop)
Period	20ms = 3125 Tck	

Software for Generating Constant PWM Signal

The goal of the first program is to generate a PWM signal with a pulse width of about 1.5ms and a period of 20ms. There are five parts to this program. Start a new project using the textbook library (steps found in Exercise 3) and use the template (using the text library) C file on Nexus.

1. Define the PWM_PERIOD, p_min and p_max constants to be the Tck values you calculated for the positional servo.

2. Define a global variable pulse_width, which will be the 16-bit integer used to define the pulse width of the PWM signal.

3. Write a function, configOC1(), to configure the output compare peripheral. No parameters are needed, and it does not return a value. Here are the steps to take:
   a. Turn **off** the timer used as a time base (Timer2) during the configuration. Remember you will do this with the T2CONbints.TON bit.
   b. Map the OC1 output to the remappable pin, RP1. Recall there is a library macro for this: CONFIG_OC1_TO_RP(RB1_RP);
   c. Clear the OC1RS and OC1R registers.
   d. Set the output compare module to use Timer 2 as the clock source, and to operate in PWM mode with fault pin disabled. You will use the OC1CON register, defined in the PIC24 datasheet and the prep slides.

4. Define a function, configTimer2() to set up Timer 2. No parameters are needed and it does not return a value. You will refer to the PIC24 datasheet, slides, or the textbook library to find the particular settings, but the necessary steps are as follows:
   a. Turn off Timer2
   b. Set Timer2 prescale to 1:256
   c. Set Timer2 to use the instruction clock, Tcy (T2CONbits.TCS = 0)
   d. Set the PR2 compare register to the PWM_PERIOD value.
   e. Clear the Timer 2 register and the Timer 2 flag.

5. Define an interrupt service routine for Timer2 (see prep slides from Exercise 4). When Timer 2 is interrupted, we want to update the OC1RS register with the global pulse_width variable value that will be set in the main program. The only other task is to reset the Timer 2 interrupt flag before exit.

6. Main program: There are four configurations to do in the first part of the main program. The clock and heartbeat, and then the Timer2 and OC1 configurations. Then you will enable the Timer 2 interrupts, turn Time 2 on, and set the pulse_width variable to the "midway" number of Tck cycles to generate a 1.5ms pulse. Conclude your program with an empty infinite while loop.

Testing Constant PWM Signal with Logic Analyzer and Servos

Compile and download your program and start the logic analyzer application to view the signal[1]. Once you verify it is correct, show the resulting PWM signal to your instructor.

This PWM signal with a 1.5ms pulse width will hold the positional servo at a particular rotational position, near the middle, but closer to one side. When you connect the continuous motion servo, it should cause the servo to stop. It may turn slowly if it has not been calibrated. You can calibrate it by gently adjusting the potentiometer with a small screwdriver through a small access port in the servo while the 1.5ms pulse is being applied until it stops completely. Now the servo is calibrated.

Software for Generating PWM Signal with Variable Duty Cycle

Next we will add on to the main part of the program to vary the pulse width. After setting the pulse width to 1.5ms, we will delay for half a second (delay functions found in the *pic24_delay.h* library). In the infinite while loop, use a "for" loop to increment the pulse width from its minimum value to its maximum value, with a delay of 50ms between each increment. Use the min and max values for the positional servo. Now decrement with a second "for" loop. These two loops are in the infinite outer loop so that they repeat.

Watch the resulting PWM signal on the logic analyzer and observe the positional servo motor moving through its range of motion. Real time monitoring is only possible with the Intronix logic analyzers, with the LA2016 logic analyzer you will have to keep capturing the signal to see it change.

Testing Varying PWM Signal with Servos

The positional servo should move from its minimum to maximum position. Servos can vary, so these minimum and maximum values may need to be adjusted for your particular servo to get a full range of motion (close to 180 degrees).

Now we will modify your program to control the continuous motion servo by simply changing the minimum and maximum values. At the maximum 1.7ms, it will rotate at full speed in one direction, and at 1.5ms it will run full speed in the opposite direction. At 1.5ms, it will stop.

Check for Understanding

How often does the Timer 2 interrupt service routine execute?
What would happen if the _T2IE bit is not set in the main program?
What would happen if the _T2IF is not reset in the interrupt service routine?

[1] For 16-channel LA2016 logic analyzer, the software is "Kingst VIS". For the 34 channel Intronix logic analyzer, the software is "LogicPort".