ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #8: Pulse Width Modulation and Servo Control

Week of 28 March 2022

Goals

1. Understand the concept of Pulse Width Modulation (PWM)
2. Learn how to configure and start a timer.
3. Use PWM to control LED brightness.
4. Use PWM to control a servo motor.

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.

Lab Procedure

The end goal of this lab is to have the Green LED controlled by Pulse-width Modulation (PWM), then to additionally control a servo motor via PWM.

Part A – Initial Setup

1. Make a copy of a previous C lab to use as a base, Lab#7 is probably a good one to use.

2. If you're using Linux, I've posted an updated template to the website that has some more definitions in the header file that will be helpful with this lab.

Part B – Get the Green LED controlled by TIM1

1. This mostly involves getting the values in your prelab put into `main.c`.

2. We will be using a **4MHz** MSI clock in this lab. So either modify your `System_Clock_Init()` routine from Lab#7 to be 4MHz, or remember that the board comes up in 4MHz mode by default so in theory you can just comment out the call to that function.

3. Set up the Green LED (GPIOE8)

   (a) Remember to enable the GPIOE clock in the `RCC->AHB2ENR` register.

   (b) Set the values in the prelab to get the GPIOE8 pin set to Alternate Function Mode `TIM1_CH1N`.

4. Set up the TIM1 timer to run at period of 10ms (0.01s) and a duty-cycle of 50%

   (a) Again these settings are found in the pre-lab, based on values from Chapter 30 of the STM manual as well as Example 15-4 in the textbook and the Flowchart in Figure 15-7.

(b) Put this code in a function `TIM1_Init(void)` that you will call from `main.c`

(c) Enable the TIM1 clock in the `RCC->APB2ENR` register.

(d) Set `TIM1->CR1` DIR to up-counting

(e) Set `TIM1->PSC` to divide the 4MHz clock down to 100kHz

(f) Set `TIM1->ARR` to reload every 1000 counts, or 10ms (remember that it takes one count to take effect)

(g) Set `TIM1->CCMR1` OCIM field to PWM Mode 1 (0b110)

(h) Set `TIM1->CCMR1` OP1PE field to enable preload

(i) Set `TIM1->CCER` output polarity

(j) Set `TIM1->CCER` to enable the complementary output of Channel 1 (CH1N) which is what is hooked to GPIOE8.

(k) Set `TIM1->BDTR` to enable the main output (MOE)

(l) Set `TIM1->CCR1` to have a duty cycle of 50%

(m) Finally set `TIM1->CR1` to enable the counter.

(n) Once you have done all of this, flash the program and your green LED should be blinking very fast with a duty cycle of 50%. (It will look solidly on, but a bit dimmer than full brightness).

Part C – Make the Green LED pulse

1. Now put an infinite loop in your `main.c` function that makes the green LED PWM duty cycle change from 0% to 100% and back (causing it to pulse).
2. You do this by changing the `TIM1->CCR1` value with a short delay between each update.
3. Example 15-5 in the textbook gives an example on how to do this.

Part D – Servo Motor

1. For this part we will connect a servo motor to the board. The servo motor maintains its position using feedback.
2. We will be using an SG90 servo motor (shown in Figure 1). It can rotate approximately 180 degrees (90 degrees in each direction). The operating speed is 0.12sec/60 degrees (4.8V, no load). The motor has three wires (see Figure 2): red wire for +5V, brown for ground, and orange for the PWM control signal (hook this to PE8 on our board).
3. The servo motor position is controlled by the pulse width as seen in Figure 3:
   - a pulse of 1.5 ms makes the motor return to the middle (0 degrees)
   - a pulse of ~1 ms makes it rotate all the way to the left (-90 degrees)
   - a pulse of ~2 ms makes it rotate all the way to the right (90 degrees)
4. The servo comes with various adapters you can snap onto the shaft that will make seeing the position a bit easier.

Figure 1: The SG90 Servo Motor.

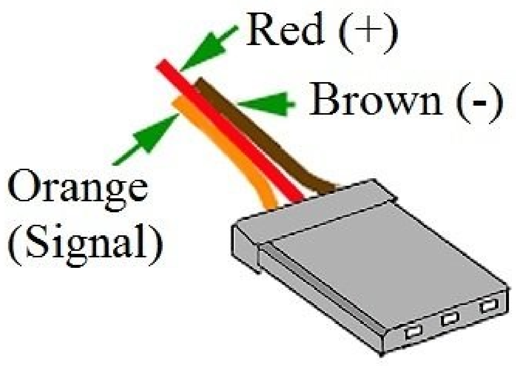

Figure 2: SG90 Servo Motor wires.

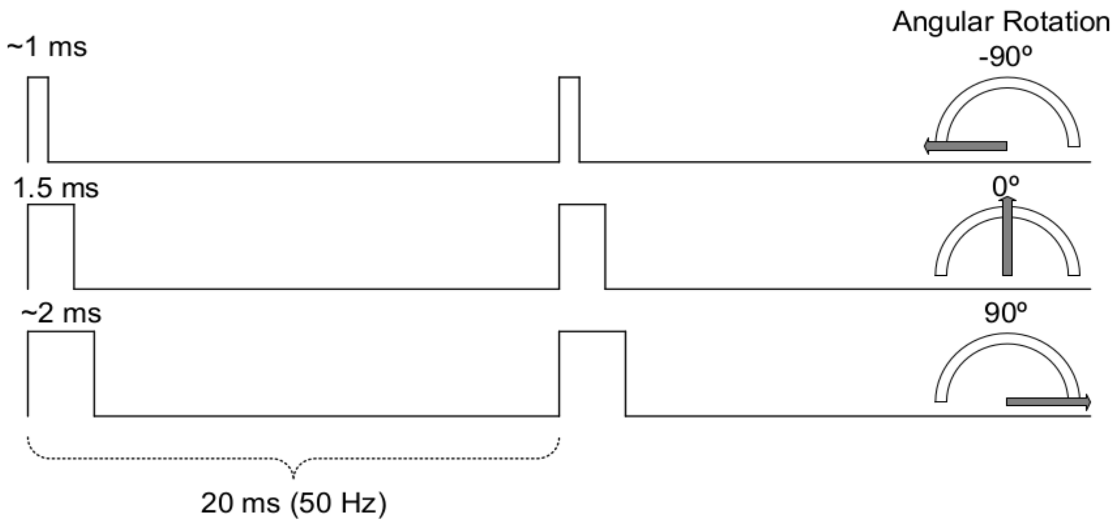

Figure 3: PWM Waveforms.

5. Modify your code so it can control the servo motor. Be sure you preserve the LED pulsing code so you can demonstrate it to the TA (have it in a separate function of file, as we will change some of the code to get the servo to work).
6. Modify your TIM1 initialization code so the ARR period happens at 50Hz (20ms)
7. Calculate the values you need in the CCR1 register to end up with a duty cycle of 1ms, 1.5ms, and 2ms. Create a function you can call that sets the CCR1 register in this way so you can rotate to 90, 0, and -90 degrees.
8. You will demo to the TA the pattern of 0 degrees, wait 1 second, move to 90 degrees, wait 1 second, move to -90 degrees, wait 1 second, then move to 0 degrees.
9. You may find that for your servo 1ms / 2ms might not be the exact right values to rotate 90 degrees. Adjust the values until you get a good 90 degree rotation and document the values you used in the Lab demo section.

Part E – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. Use the joypad to control the brightness of the LED.
2. Use the keypad to control the servo motor.

Lab Demo

1. Submit your code
    - Complete a README with the post-lab answers.
    - Make sure the code is properly commented.
      This includes a header at the top of your main.c with your name and a brief summary of the lab.
    - Check your code and README into your gitlab tree.
2. Demo your implementation to your lab TA.
    (a) Display the code that uses PWM to pulse the green LED.
    (b) Demonstrate the control of the servo motor. Show you can do this pattern:
    - Rotate to position 0 degrees.
    - Pause 1 second
    - Rotate to position 90 degrees.
    - Pause 1 second
    - Rotate to position -90 degrees.
    - Pause 1 second
    - Rotate to position 0 degrees.
    - Stop

    (c) What were the final values for CCR1 you used to get the proper 90 and -90 degree rotation?

Position	Pulse Width	Value of $TIM1->CCR1$
-90 degrees		
90 degrees		

Post-Lab

- Place your answers to the question in a file Readme.md
- Submit with your code via the gitlab server.
- Questions: Assume you set your board to use an 8MHz MSI signal
    1. To generate a 1 Hz square wave with a duty cycle of 50%, how should we set up the timer? Indicate your counting mode and show the value of ARR, CRR, and PSC registers.
    2. What is the smallest PWM frequency that can be generated?