EE128: Data Acquisition, Instrumentation and Process Control
University of California, Riverside

Lab 5: Timer and PWM Operations

Spring 2019

Objective

To exercise programmable timer operations and pulse-width modulation techniques on Dragon12-JR and Arduino boards, and to improve skills in designing and debugging microcontroller-based systems.

References

- Dragon12-JR Trainer User's Manual
- Freescale MC9S12DG256 Device User Guide
- Freescale HCS12 Core User Guide
- Reference Guide for D–Bug12 Version 4.x.x
- ATmega328 datasheet
- Online Arduino Reference

Equipment

- PC running MS Windows
- Digital Multi-Meter (DMM)
- Dragon12-JR 9S12DG256 EVB
- Arduino Uno
- **Breadboard** (you need to bring one)

Parts

- 2 each 7-segment display, common cathode
- 2 each 390Ω x 8, independent resistor network (DIP)

Software

- Freescale CodeWarrior for HC12 v5.1
- Arduino IDE
- RealTerm

Background

The Dragon12 board equipped with an MC9S12DG256 microcontroller (MCU) provides a high-speed timer subsystem with powerful features. On the other hand, Arduino equipped with an ATmega328 MCU provides an easy-to-use programming interface for basic timer operations including pulse-width modulation (PWM). In this lab, we investigate how to generate PWM on Arduino and how to use the timer subsystem on Dragon12 for measuring pulse widths and duty cycles.

Please refer to the Arduino programming reference (online) and the MC9212DG256 user manual for details on the timer subsystem.

Specification

This lab is composed of several parts. At first, you will generate Pulse-Width Modulation (PWM) signal using the Arduino board, and drive an LED with the PWM signal. Next, you will take that PWM signal and use it as input into the input-capture pin of the Dragon12 (9S12) board. Using this input capture, you will measure the duty cycle of the PWM and display it on the 7-segment displays.

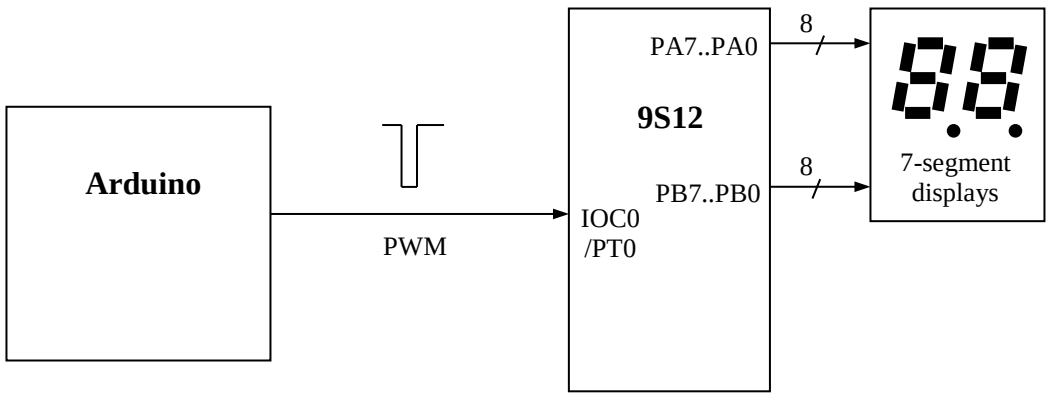

Figure 1. Overall Block Diagram

Step 1) Pulse Width Modulation on Arduino

There is a PWM example program for the Arduino. Start the Arduino programming environment and then go to the **File->Examples->Analog** menu of the Arduino software. There you will find a "**Fading**" example that demonstrates the use of analog output (PWM) to fade an LED. In the way of background, PWM can be used to generate analog output using digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off.

The duration of "on time" is called the pulse width. To get varying analog values, you change or modulate that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5V controlling the brightness of the LED.

In Figure 2, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with the Arduino's default PWM frequency at 490Hz, the green lines would measure about 2 milliseconds each. **analogWrite(pin, value)** is a function to generate a PWM signal on a digital output pin. The first parameter 'pin' is the output pin number, and the second parameter 'value' is the duty cycle of the signal, ranging from 0 to 255. For example, analogWrite(9, 255) requests a 100% duty cycle (always on) on Pin 9, and analogWrite(9, 127) is a 50% duty cycle (on half the time). See https://www.arduino.cc/en/Reference/AnalogWrite for more details.

Go ahead and get the example of this fading going, running at 490 Hz. In this part of the lab, use the PWM output to drive a single LED (using a current-limiting resistor of course). Verify its operation, and check the brightness of the LED as the duty cycle changes.

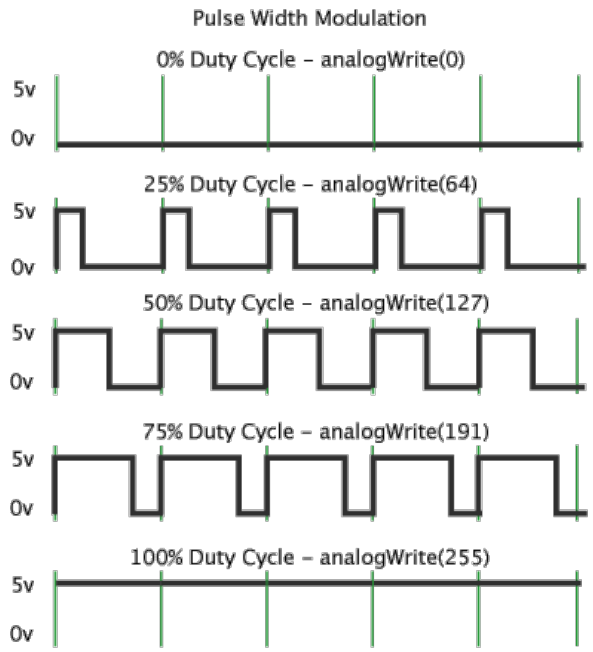

Figure 2. PWM Duty Cycle

Step 2) Input Capture and Display

Now you can run the code on lecture note to measure the duty cycle of a PWM signal coming into an input-capture pin of the 9S12 microcontroller. Modify the code appropriately so that it measures the rise and fall times of the PWM signal, and displays the duty cycle on two 7-segment displays. "00" should correspond to 0% duty cycle, "50" should be 50% duty cycle, and "99" should be 99% duty cycle.

Step 3) Modifications

Modify your Arduino source code so that the frequency of the PWM is 245 Hz instead of 490 Hz (hint: http://playground.arduino.cc/Main/TimerPWMCheatsheet). Check the PWM signal frequency with an oscilloscope. Verify that your 9S12 program still reads the correct duty cycle.

Pre-lab Questions

1. What is the notion of **common ground** in electronic circuits? Do Arduino and 9S12 need common ground in this lab?

2. If the Arduino is set up to do PWM at 490 Hz and the duty cycle is 30%, what is the duration that the signal is high and the duration that the signal is low for one cycle?

3. Design the required circuit for the experiment.

4. Write and compile an initial version of your source code for both Arduino and 9S12.

Questions

1. When the PWM circuit modulates the LED, what is the minimum frequency you can use before you start to see the LED blink?

2. For the input capture circuit, what prescale values did you choose to use? When would you make the prescale value higher or lower?

3. Arduino provides `analogWrite()` which is a very convenient function to generate PWM signals. Let's implement a similar function for HCS12. Your `analogWrite()` for HCS12 should follow the following format:

Syntax
`void analogWrite(int pin, int value)`
Parameters
pin: the PWM pin to be used, e.g., if pin is 1, PWM1 (PP1) is used. value: the duty cycle: between 0 (always off) and 255 (always on).

The frequency of the PWM signal is fixed to **approximately 980 Hz** (e.g., 976 Hz is acceptable). This function can be called **multiple times on different pins** in order to generate multiple PWM signals on different pins simultaneously. For example,

```
void main()
{
     analogWrite(0, 40);
     analogWrite(1, 60);
     analogWrite(2, 30);
     while(1);

}
```

Provide the **source code** of your implementation of the analogWrite() function for **HCS12** which generates a waveform with the frequency of 980 Hz.

Lab Demo (50 points)

Demonstrate Steps 2 and 3 to the TA and get a confirmation of completion (Step 1 demo is not required).

Lab Report (50 points)

Make sure you include the following in your report:

1. Abstract (a short paragraph stating the objectives and accomplishments)
2. Experiment system specification (what has been designed and implemented) – **10 points**
   - Flowchart diagram (show how your system works)
   - Photos of your boards and circuits
3. Hardware design – **10 points**
   - Draw schematics of your own; do not copy and paste from the handout
4. Software design – **10 points**

- High level description of the software
- Program listing (including comments)

5. Technical problems encountered, and how they are solved
6. Answers to the questions – **10 points**
7. Conclusion
    - A very short concluding remark
    - Summary of the contributions of each member