ECE 375: Computer Organization and Assembly Language Programming

Lab 6 – Timer/Counters

SECTION OVERVIEW

Complete the following objectives:

- Learn how to configure and use the 16-bit Timer/Counters on the ATmega32U4 to generate *pulse-width modulation* (PWM) signals.
- Use the upper half-byte and lower half-byte (otherwise known as the upper and lower *nibbles*) of an I/O port for two different tasks.

BACKGROUND

Timer/Counters are one of the most versatile and commonly-used components of a microcontroller, and have many uses in embedded applications. In basic terms, a Timer/Counter is simply a register whose contents are regularly incremented (or decremented) at a specified, configurable frequency. If you know the rate at which this incrementing or decrementing is occurring, then you can compare the contents of a Timer/Counter's register at two different points in a program, and calculate how much time has elapsed!

Together, the width (in bits) of a Timer/Counter and the frequency of the clock supplied to the Timer/Counter determine its *range* (the largest interval that can be measured) and *resolution* (the smallest interval that can be measured). The ATmega32 microcontroller features one 8-bit Timer/Counters(TCNT0) and two 16-bit Timer/Counters(TCNT1/3), and one 10-bit High Speed Timer/Counter 4 (TCNT4) they all have multiple clock frequencies available.

Beyond simply measuring an interval of time, Timer/Counters can also be configured to take some action based on an observed interval, such as toggling an I/O pin after a certain amount of time has passed, generating a waveform. Depending on how the Timer/Counter is configured, changing this interval will change either the *duty cycle* or the frequency of the waveform. When the duty cycle is varied, this technique is known as *pulse-width modulation* (PWM).

To successfully complete this lab, you will need to have a solid understanding of how to configure Timer/Counters for PWM, and also how to modify the PWM duty cycle after initial configuration. For more details on how to make the ATmega32 microcontroller create PWM signals, please refer to Timer/Counter1's "Fast PWM Mode" subsection on page 124 of the ATmega32U4 datasheet.

PROCEDURE

For this lab, you will write an assembly program that allows a user to modify the speed of the TekBot by providing input via Port D. The speed itself will be modified by using the 16-bit Timer/Counter 1 in Fast PWM mode, and driving the right and left Motor Enable port pins with the PWM signals generated by the Timer/Counters. By varying the duty cycle of the PWM waveforms being generated, you will effectively be modifying the speed of the TekBot's motors.

You are not required to implement any BumpBot behavior in this lab, so the TekBot should just be configured to move forward indefinitely.

Some design decisions have been left up to you for this lab (such as how to detect user input on the Port D pushbuttons), but you **must** adhere to the following requirements for full credit:

1. The fast Pulse Width Modulation or **fast PWM mode** (WGMn3:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. TOP is equal to 0x00FF, 0x01FF, and 0x03FF for the 8-bit, 9-bit, and 10-bit Fast PWM mode, respectively.
   In this lab, you'll be using **8-bit** (WGMn3:0=5) Fast PWM Mode. Your TekBot needs to have **sixteen** equidistant speed levels, with Speed Level 0 being completely stopped, Speed Level 15 being full speed, and Speed Levels 1 through 14 in between. Refer to Table 1 for more details.

Speed Level	TekBot Speed
15	100% (255/255)
14	~93.3% (238/255)
13	~86.7% (221/255)
⋮	⋮
2	~13.3% (34/255)
1	~6.7% (17/255)
0	0% (0/255)

Table 1: Equidistant Speed Levels

2. A user must be able to modify the speed in 3 different ways: (1) increase speed by one level, (2) decrease speed by one level, (3) immediately increase speed to the highest level (full speed).
3. A single button press on Port D must result in a single action; for example, if the user presses the "decrease speed by one level" button, your program should smoothly decrease the speed by just one level, **not** several levels in rapid succession. Keep in mind, the correct way to meet this requirement can depend on some of your previous design choices, especially regarding how to detect user input.
4. The speed levels **must not** wrap around from 15 $\rightarrow$ 0 or 0 $\rightarrow$ 15. In other words, if the user requests a speed increase and the TekBot is already at full speed, the speed must not wrap around to a lower speed (i.e., overflow). In a similar fashion, the TekBot must not go from stopped to a higher speed (i.e., underflow) if a decrease is requested by the user.
5. In order to use the 16-bit Timer/Counter 1 properly, you need to check Timer/Counter Control Registers(TCCR1A & TCCR1B) with Waveform Generation Mode (WGM), Compare Output Mode (COM), and Clock Selection (CS). See 16-bit Timer/Counter Register Description and Alternate Functions of Port B in ATmega32 Datasheet.
6. To allow the user to visually assess the current speed level of the TekBot, you must use the `ATmega32` LEDs connected to the lower nibble (pins 3:0) of Port B to display a 4-bit indication of the current speed level. For example, you can use "0000" (all 4 LEDs off) to represent Speed Level 0, and "1111" (all 4 LEDs on) to represent Speed Level 15. This 4-bit indication of speed level **must not** interfere with the "Move Forward" motor control signals, which are also being output to Port B at the upper nibble (pins 7:4).

LAB REPORT

For every lab, you will be required to submit a write-up report that details what you did and why. As a requirement, your lab report should consist of an Introduction, Design Document, Program Overview, Testing, Questions, Difficulties, and Conclusion. **You must fill the program overview section with a summary of what you have understood by reading/programming comments/code of your AVR code. A detailed description of each section is also required to be elaborated (please refer to the description povided in the report submission page in Canvas). You also need to answer the study questions given in this document.** You must provide your answers using the **"Lab Report Template"** provided in the Canvas webpage. Additionally, you must include your source code at the bottom page of your report. All the submissions including a write-up report and a source code should be done via **Canvas** before the start of the following lab. **NO LATE WORK IS ACCEPTED**.

Note, code that is not well-documented will result in a **severe loss of points** of your lab grade. For an example of the style and detail expected of your comments, look at the provided skeleton code you downloaded. Generally, you should have a comment for every line of code. In terms of Study Questions, some are related to the current lab tasks and some are related to the next lab tasks. As is often the case in engineering courses, there will be some occasions when you are exposed to information that has not been covered in class. In these situations you will need to get into the habit of being proactive and using your study skills to research the answers. This can involve strategies such as reading ahead in the textbook, checking the datasheet, searching online, or reviewing other documentation from the manufacturer.

Study Questions

1. In this lab, you used the Fast PWM mode of 16-bit Timer/Counter, which is only one of many possible ways to implement variable speed on a TekBot. Suppose instead that you used Normal mode, and had it generate an interrupt for every overflow. In the overflow ISR, you manually toggled both Motor Enable pins of the TekBot, and wrote a new value into the Timer/Counter's register. (If you used the correct sequence of values, you would be manually performing PWM.) Give a detailed assessment (in 1-2 paragraphs) of the advantages and disadvantages of this new approach, in comparison to the PWM approach used in this lab.
2. The previous question outlined a way of using a single 16-bit Timer/Counter in Normal mode to implement variable speed. How would you accomplish the same task (variable TekBot speed) using in CTC mode? Provide a rough-draft sketch of the Timer/Counter-related parts of your design, using either a flow chart or some pseudocode (but **not** actual assembly code).
3. In the next lab, you will be utilizing Timer/Counter1, which can make use of several 16 bit timer registers. The datasheet describes a particular manner in which these registers must be manipulated. To illustrate the process, write a snippet of assembly code that configures OCR1A with a value of

0x1234. For the sake of simplicity, you may assume that no interrupts are triggered during your code's operation.

4. Each ATmega32U4 USART module has two flags used to indicate its current transmitter state: the Data Register Empty (UDRE) flag and Transmit Complete (TXC) flag. What is the difference between these two flags, and which one always gets set first as the transmitter runs? You will probably need to read about the Data Transmission process in the datasheet (including looking at any relevant USART diagrams) to answer this question.
5. Each ATmega32U4 USART module has one flag used to indicate its current receiver state (not including the error flags). For USART1 specifically, what is the name of this flag, and what is the interrupt vector address for the interrupt associated with this flag? This time, you will probably need to read about Data Reception in the datasheet to answer this question.