University of Puerto Rico at Mayagüez
Department of Electrical and Computer Engineering
Microprocessor Interfacing Laboratory

Experiment 3: Timers and Applications

Objectives

- Become familiar with the use of timers in embedded applications
- Understand the timer operating modes
- Learn how to handle timers interrupts
- Learn how to program timer modules

Duration

- 2 Hours in laboratory and extra time for homework.

Materials

- Your Microcontroller IDE application
- Your Microcontroller development board
- Buzzer
- LCD
- Switches

Introduction

Timers are present in most embedded applications, so it is important to know how they work and how to use them in our applications. The core of a timer is a free-running binary counter fed by a clock of known frequency, as illustrated in Figure 1. Timers are used to generate time bases, count time between events, count external events, generate PWM signals, real time clocks, etc.

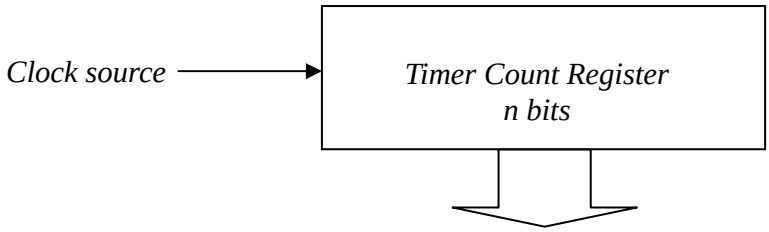

Figure 1, Timer basic counter structure

Timer modules can be of 8-, 16-, 24-bit, etc. The number of bits in the timer's counter determines the maximum value it can count to, i.e., the maximum value the timer count register can hold. Timers use a known clock source to increment its count and can count. The clock source can be chosen from the internal clock circuit or from an external clock generator.

The timer count register increases with each clock cycle. When the count reaches its maximum value (2^n-1) the timer generates an overflow signal as illustrated in Figure 2, and restarts counting at 0. The overflow signal can be polled by software or used to trigger an interrupt request (Timer Overflow).

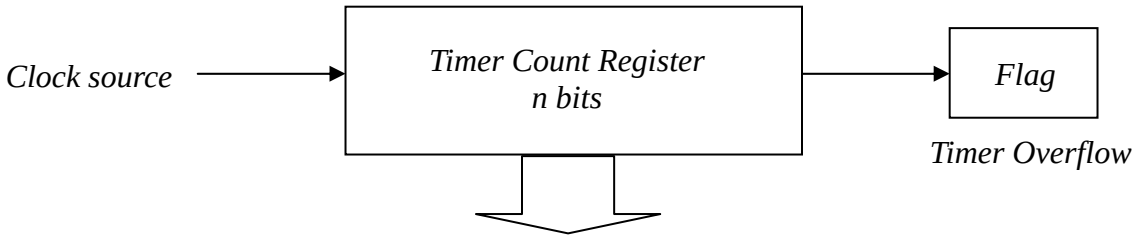

Figure 2, Timer overflow structure

Timers can also have a pre-scaler. This is just a chain of flip-flops that can divide the source frequency by values specified through a configuration register. Most typical values are 1, 2, 4 and 8, although this might change from one timer to another. Pre-scalers are very useful when you need to extend the length of time between timer overflows.

Another important part of a timer module is the terminal count register. This register, typically of *n* bits as the timer count, can be loaded with any value *TermCount*. When the timer counter reaches *TermCount*, the timer is reset and a *Top* count signal is generated. The *Top* signal can be either polled by software of enabled to generate an interrupt request. This is a very useful feature in a timer that can be used for many applications, such as generating periodic signals or pulses of predetermined width. Figure 3 shows a complete timer block diagram with both, pre-scaler and terminal count registers.

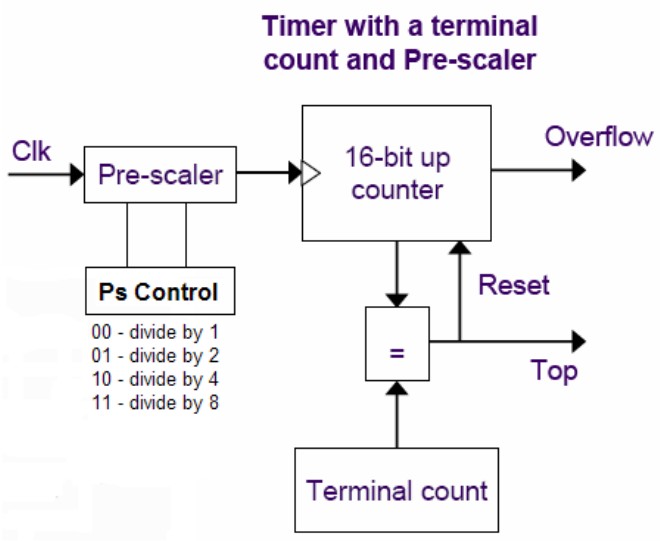

Figure 3, Timer basic structure

Procedure

Part I: Using Timer by Polling

1. Connect a buzzer as shown in Figure 4. Choose R to not exceed the MCU PIN maximum current I_{OL}. Indicate your selected clock source and frequency.

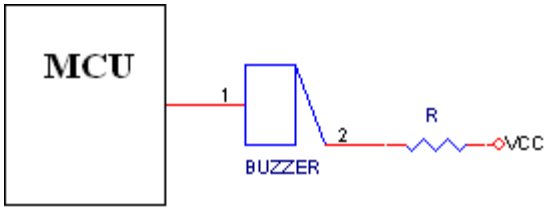

Figure 4, buzzer connection

2. Calculate the terminal count value to delay to generate an audible frequency (f = 1024Hz).
3. Determine the appropriate mode of your MCU timer to generate the above frequency and configure the timer in that mode.
4. Load the terminal count value you calculated to the timer.
5. Wait for the timer overflow flag.
6. Upon timer overflow, toggle the buzzer pin with a 50% duty cycle.

Figure 7 provides a sample flowchart illustrating a procedure to toggle the buzzer I/O pin.

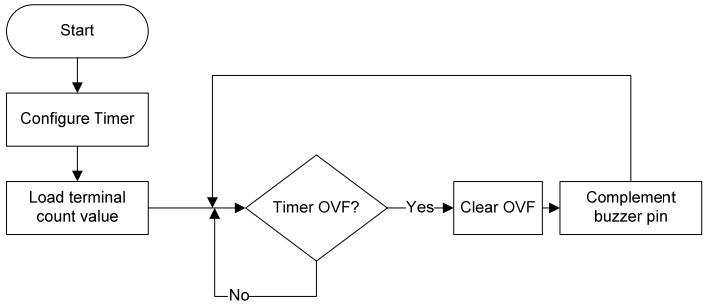

Figure 5, Flowchart for operating the timer by polling.

Exercise 1:

Add two switches to increment and decrement the frequency by a factor of 10.

Turn-in:

- Produce a demo to the TA.

Part II: Using Timer Interrupts

1. As before, connect a buzzer as shown in Figure 4.
2. Calculate the terminal count value to delay to generate an audible frequency (f = 1024Hz).
3. Configure the timer mode
4. Write a timer ISR to generate the audible frequency on buzzer see figure 6.
5. Enable timer interrupt and global.

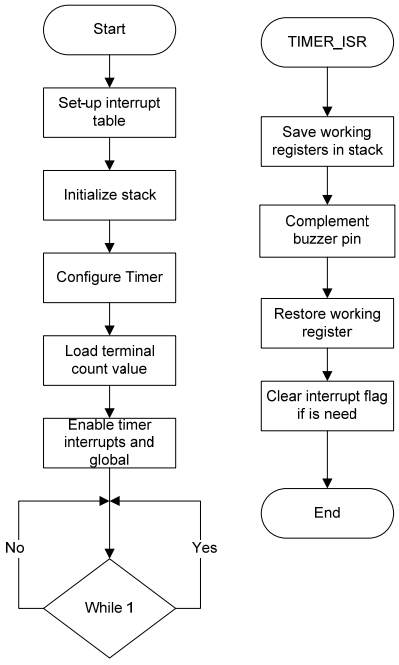

Figure 6, Timer with interrupt flowchart

Part III: Generating Random Numbers with a Timer

1. Connect the LCD and one switch to the MCU as you did on previous experiments.
2. Configure the timer in continuous mode to count from 0000h to FFFFh.
3. Each time the switch is pulsed, read the timer counter register.
4. Discard the 8 most significant bits and divide the less significant by ten and choose remainder. This will be your random number (between 0 and 9).
5. Display in the LCD the randomly generated number.

Exercise 2:

Write a program to ask the user to guess a random number between 0-9. If the user does not guess the number in tree trials display a game over message on LCD an restart. Use two switches as in Experiment 1 to increment/decrement the user input. Depress both switches to accept.

Turn-in:

- Produce a demo to the TA.

Homework

Digital Tachometer

Modify your homework last week to implement a tachometer for the encoder wheel. Use the LCD for displaying in the first line a message with the speed: “Speed = ### RPM”. In the second LCD line indicate whether the rotation direction is clockwise or counterclockwise. Internally, use the timer and interrupts to determine the speed of rotation of the wheel.

Turn-in:

- Software plan and explanation (pseudo code or flowchart)
- Connection schematic with components calculations and wheel assembly.
- Listing of documented code.
- Have your circuit assembled and produce a demo to the TA or professor.