

VANIER COLLEGE – Computer Engineering Technology – Autumn 2017

Introduction to Microprocessors (247-302)

Lab 6

Timer0 Interrupt

NOTE:

To be completed in TWO lab sessions of 6 hrs.

One report has to be submitted **not later than one week** after the last lab session.

This exercise is to be done **individually** except where specified in the procedure. **Each** student must submit a lab report with original design, observations and conclusions.

OBJECTIVES:

After completing this lab, the student will be able to:

1. Understand differences between polling and interrupt mechanism
2. Understand the operation of TMR0 as timer using simulation
3. To implement interrupt service routine for TMR0

THEORY:

A microcontroller can serve several devices. In general, there are 2 approaches where a processor acquires information from external or internal peripherals and devices, namely polling and interrupt.

Polling

This is the simplest method, where microcontroller periodically checks each device/interface to see if any of these requires its service. In our previous labs, *polling* algorithm (checking on flag/bit in a loop) was used to check status of pushbutton in order to turn on or off LEDs. The main drawback of this method are:

- Takes up processing time even when no requests pending
- Overhead may be reduced at expense of response

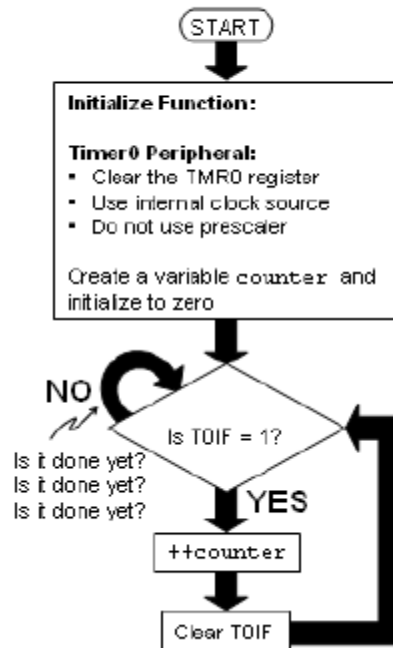
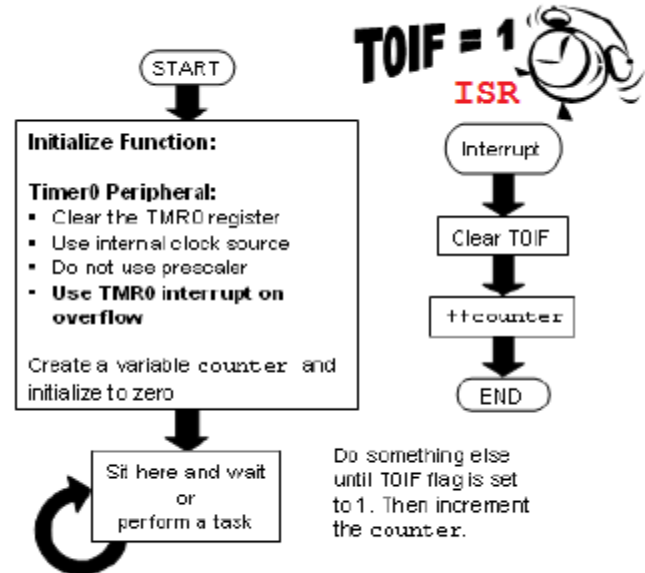
Interrupt

Interrupt is signal sent to microcontroller to mark the event that requires immediate attention. When interrupt happens, microcontroller stops performing the current program and executes a routine called an Interrupt Service Routine (**ISR**) or interrupt handler to deal with the interrupt. There is no overhead when no requests pending.

In this lab, a counter variable increments each time the TMR0 register overflows from 255 to 0. The diagram on the next page illustrates the flow of these 2 approaches.


1. To do this, a "Polling" algorithm would check Timer0 Interrupt Flag (T0IF) periodically to see if it was set to '1'. This indicated that the TMR0 register had overflowed and that the counter variable should be incremented.
2. Uses interrupts which serve as an alarm, signaling that a particular event has occurred (such as when the T0IF flag is set). When any interrupt occurs, and there could be more than one, the processor will immediately stop what it is doing and jump to the ISR to service the interrupt. Once completed, the processor returns to what it was doing in code, prior to being interrupted.

If multiple peripheral interrupts are used, a prioritization algorithm will need to be included in the ISR to determine which interrupt is serviced first.

POLLING**INTERRUPT****Lab procedures**

Note: For this lab, you will be using MPLAB IDE to perform simulation/animation, instead of MPLAB X IDE.

PART A: Animate a polling on TMR0 interrupt

1. Copy and unzip the attached tmr0_poll.zip file into your computer. Open the workspace using MPLAB IDE.
2. Read and understand the polling mechanism as implemented in tmr0_poll.asm.
3. Perform simulation/animation on the code by performing the following steps:
 - a) Make the project by pressing the Make icon. There should be no errors.
 - b) Select MPLAB SIM as the debugger.
 - c) Open a Watch window (View > Watch) and add the TMR0, INTCON and OPTION_REG SFR.
 - d) Add variable `counter` by specifying the address of memory location.
 - e) Press the Animate icon  in the Debugger toolbar and observe that the following occurs;
 - i. On a TMR0 overflow, the TOIF flag is set briefly (INTCON<2>)
 - ii. When TOIF flag sets, the `counter` variable increments by 1
4. Record your observation and findings in your lab report.

PART B: Animate an ISR for TMR0 interrupt

5. Start a new workspace and name it `tmr0_isr`.
6. Based on the code provided in part A, modify it to service the TMR0 interrupt using ISR instead of polling.
 - a) You must perform necessary context saving in the interrupt service routine
 - b) Variable `counter` should be incremented in ISR instead
7. Compile and perform animation as in Part A. Confirm and observe that the following occurs:
 - a) On a TMR0 overflow, the T0IF flag is set (INTCON<2>)
 - b) When T0IF flag sets, the ISR is executed and `counter` variable is incremented by one
8. Show your working animation and code to your teacher.

PART C: 1 second single digit 7-segment counter

9. Based on existing architecture (4 LEDs on RA0-RA3), implement a 1 second 4-bits LED binary counter using Timer0 interrupt mechanism.
10. Using oscilloscope/logic analyzer, capture necessary waveform with details to clearly illustrate that the counter is incremented at 1 second interval.
11. Now, modify your code and hardware to drive a single digit 7-segment display, instead of a simple 4 bits LEDs. The counter should count up at 1s interval, and display the counting from 0 →9, then round back to 0.
 - a) Request a single digit 7-segment display unit from your teacher. You will be randomly assigned one. Is it a common cathode and common anode unit?
 - b) Find out which BCD to 7 Segment Display Decoder would you need, and request it from your teacher.
 - c) Modify your hardware to drive the decoder chip using RA0-RA3.
 - d) Modify your code accordingly to meet the requirements.
12. Demonstrate your final working system to your teacher.
13. Include the following in your lab report :
 - a) A copy of all your source codes (Part B & C).
 - b) Flowchart for part C.
 - c) Schematic diagram for Part C.
 - d) Oscilloscope/logic analyzer waveforms as required.
 - e) Any observation, analysis if any.