CPE 490 Lab 4

## Goals

1. Learn how to use interrupts in C language program

## Overview

This lab will give you experience in writing an Interrupt Service Routine ISR in a C language program. You will write an ISR for timer 1. Timer 1 will be connected to an external 32.768KHz watch crystal. The ISR will service a real time clock. You will keep variables that hold the number of seconds, minutes and hours of the current time. The time will be displayed using the LEDs on the explorer 16 board.

## Design

### Configuration bits

Set up the bits as did in the previous lab, lab3. This includes setting up the main osc to function at 16MHz

### Write the ISR

Like any other function the ISR should be declared outside the main loop. To declare and ISR for the timer1 interrupt the following code should be used (note where double underscore is used vs single underscore, also note the space between closing double parenthesis and \_T1Interrupt():

void \_\_attribute\_\_ ((interrupt, no\_auto\_psv)) \_T1Interrupt (void)

{

//ISR goes heres

}

To understand how the semantics consult the chapter 14 of the XC16 Compiler user guide (posted on blackboard). To use this for other interrupts you will need to use the correct name. For tables of interrupt vectors by device family:

• In MPLAB X IDE, for newer versions of the compiler, open the Dashboard window and click on the Compiler Help button. (Question mark in blue, circular back ground)

* Find the section called “16-Bit Language Tools Readme and Reference. In this section click on the link “Interrupt Vector Tables Reference”. Next click on the “dsPIC33F DSC and PIC24H MCU Interrupt Vector Table”. Now find the correct name for the interrupt or trap you want to program.

Since you can’t pass parameters with an ISR function you will need to declare global variables to contain the seconds minutes and hours that will be modified in the ISR. Since these global variables are changed by an interrupt you must tell the compiler to not optimize code by assuming the value has not changed between consecutive needs in a code module by using the type modifier “volatile” (see section 8.10.2 in XC16 C compiler user guide). In addition it is nice to initialize values for you global variables for testing the code. In the following example code the variables ‘example1’ and example 2’ are declared to be integers that are volatile, and are initialized to 52 and 0:

int volatile example1 = 52, example2 = 0;

Set up your interrupt so that it occurs every time a second passes. At that point you will need to increment the seconds register. If the second register is at 59d then it will have to roll over and then increment the minutes register, and when the minute register is 59 then it will roll over and increment the hours register, and when the hour register gets to 12 it will rollover to 1.

Make sure the ISR takes care of any flag that has to be reset in order to not have an endless loop of calling the interrupt routine. Bits are defined in the header file so for example \_T1IF is defined as:

#define \_T1IF IFSObits.T1IF

and furthermore the structure that defines IFSObits.T1IF is also in the header file. All you have to do is set the bit so for example \_T1IF = 0;

### Write the main code

The main code will first initialize everything that needed for:

* Timer1 to use the 32.768 kHz crystal and set the PR1 value correctly.
  + See chapter 9, Figure 9-1 in the data sheet. To use the secondary oscillator you must drive it with an inverter. This inverter is turned on by writing bit LPOSCEN in the OSCCON SFR. Since this control register also controls the main oscillator it is protected from accidental writes. The protection is in the form of some code words that must be written in sequence to the OSCCONL SFR before it is changed. Fortunately, the C compiler provides a macro to do this function for us. Use the following code to setup the secondary oscillator:

\_\_builtin\_write\_OSCCONL(2);

This macro inserts assemble code into your program as described in appendix C of the XC16 C compile user guide. The sequence to allow a write to the OSCCONL is described in the dsPIC33F family reference manual section 7 that is posted on blackboard.

* + To set up timer 1 to use the secondary oscillator see chapter 12 of the data sheet. This means that you will have to modify T1CON and PR1. Remember that since the timer starts at 0 you want to load a value into PR1 that is one less than the time periods you want to count.
* Port A setup to drive the LEDS (recall previous labs)
* Then do the following to set up for the interrupt

\_T1IF = 0; //clear the interrupt flag

\_T1IE=1; //set the interrupt enable

\_IPL =0; //set the processor priority level to 0

Now write your code that will forever loop. This code will display the current time though the 8 LEDs on the demo board. Show the number of minutes that have elapsed using the bottom 4 LED to form lower BCD digit and the top 4 LED to show the BCD most significant digit. To show seconds, simultaneously flash all the LEDs that should be on to represent minutes for one second and then off for one second based upon the second’s timer.

### Verify

Verify that the minutes change correctly with a stop watch (most cell phones have this function). Initialize the hours minutes and seconds so that you can watch them roll over correctly using the in circuit debugger to verify your code. **Show an instructor your working program.**

**Witness Signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

## Extra Credit 10% extra (if you got time)

Use the switch S3 as an input to switch the display from the minutes to the seconds in the LEDs while holding in the switch down. Use the switch S6 to do the same with hours. **Show an instructor your working program.**

**Witness Signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**