CMPE 12L Lab 6 - Fall 2014
Ready, Set, Go!

Prof. Matthew Guthaus

Due: December 12, 2014 5pm

100 Points (25 Report, 75 Work)

1 Overview

In this lab, you will use a hardware timer (Timer 1) and an interrupt to implement an accurate stopwatch that is controlled by the switches (SW1 and SW2). The stopwatch starts at "0:00" and SW1 enables/disables the timer which increments every second. Whenever SW2 is on, the time should clear and stay cleared until the switch is returned to its original state. Clearing should take priority over the time incrementing.

2 Prerequisites

- Read through this **entire** lab assignment.
- Read Section 13 (Timer1) of the PIC32 reference manual (PIC32MX320F128.pdf) and Section 14 (Section14_Timer.pdf) of the PIC family reference manual.
- Read Section 8 (Section8_Interrupts.pdf) of the PIC32 family reference manual
- Review the slides on PIC32 and PIC32 interrupts from lecture.

3 Tutor/TA Review

Your lab tutor/TA will cover the following items in the first portion of the first lab:

- Questions about C programming
- What's required

4 OLED Displays

This project uses an 128x32 pixel OLED (Organic Light-Emitting Diode) display to show a stopwatch time value. The time is displayed on a single line in the form M:SS where SS is the two digit number of seconds (00-59) and M is the number of minutes (0+). You do not have to track hours and the minutes can overflow into more digits as necessary.

You are given a set of library source files to control the OLED display. You may call these functions from C or assembly. While there are many functions to draw abitrary things, you will likely use these functions:

- OledInit : Initializes the OLED display (called once in main.c)
- OledDisplayOn : Turns the OLED display on (called once in main.c)
- OledClear : Clear the OLED display contents (called whenever you want to update the display with a new time).
- OledPutString : Takes one argument, the starting address of a string, and displays all characters until the first null character on the OLED display.
- OledPutChar : Takes one argument, a single ASCII character, and displays it on the OLED display.
- OledSetCursor : Takes two integer arguments which are the X,Y locations to put the next character or string. This is used after a clear to reset the location of where strings and characters will be drawn on the display.

Every time you display a character or string, the position of the cursor is moved to the right. It will be necessary to set the cursor back to (0,0) after you clear the display to prevent the time from "traveling" across the display.

OLED displays are rather slow, so it is not very well suited to sub-second displays. In addition, this means that refreshing the display too quickly may result in "flicker" of the display. You should pay careful attention to only clear and refresh the display when values change in order to avoid unnecessary flicker. To do this, check if you are going to display the same value before doing a clear and display (also called a "refresh").

5 Timer and Interrupts

In this lab assignment, we will use an on-chip timer module that counts peripheral bus clock pulses and causes an interrupt when a specified number have occurred. The clock frequency of the chip is 80MHz and the peripheral bus (PBUS) is 1/4 that frequency, 20MHz.

5.1 Memory Mapped I/O Registers

As a reminder from lab 5, each of the device registers actually has four addresses that perform different functions. The base register can be read or written to change all of the bits at once. Three additional registers are used to atomically modify one or more bits at a time: clear (offset +4), set (offset +8), and invert (offset +12). The ones in a mask specify which bits to clear, set or invert. So, for example, if you want to clear bit 0 of the TRISF register (0xBF886140), you can write a binary mask of 0x1 to address 0xBF886144 (0xBF886140 + 4). In order to clear this one bit without the special register you would need to do a read, mask, and write sequentially.

5.2 Timer Configuration

As shown in lecture (and in the PIC32 manual), the configuration registers for the timer module are T1CON, TMR1, and PR1. These are configurable just like the digital input/output registers in the last lab using direct write, clear, set and invert register offsets. TMR1 is the current timer count and PR1 is the period register. After TMR1 reaches the count in PR1, it will trigger an interrupt when configured to do so. You should clear TMR1 and set PR1 according to how frequently you want to trigger interrupts.

T1CON is the timer configuration register. You should use the T1 clock prescaler (T1CKPS[1:0]) to divide the peripheral bus clock even further to a slower one so that the counter operates slower. You should configure this to use the maximum prescalar of 1:256 by writing 0b11 to T1CKPS[1:0] (T1CON[5:4]).

After you have set the prescalar and configured the interrupts (in the next section), you can enable the timer by setting bit the ON configuration bit T1CON[15]. Disabling is done by clearing this bit.

5.3 Interrupt Configuration

The interrupt configuration registers are IFS0/IFS1 and IEC0/IEC1. T1IF is the timer 1 interrupt status flag in the IFS0 register. T1IE is the timer 1 interrupt enable flag in the IEC0 register. These bits behave exactly like the LC-3 interrupt configuration bits.

The PIC32 also has programmable interrupt priorities in the IPC0-IPC11 registers. T1IP[2:0] which is the IPC1 register is the timer 1 interrupt priority configuration register. You should set it to 0b100 (4). T1IS[1:0] is the interrupt subpriority which is used if you have more than one interrupt matched to the same priority. We don't care about that.

Details on interrupt handlers are in Section8_Interrupts.pdf of the PIC32 Family reference manual. Specifically, section 8.6 talks about the address of the handler. Our PIC32 is using multi-vectored interrupts similar to the LC-3 vector table. One difference, however, is that our vector table contains a small set of instructions rather than addresses. If the first instruction is a jump, it is similar in behavior to the LC-3 interrupt table.

The exception (interrupt) base address (EBASE) is set to 0xBFC0_1000 which is the configurable location of this vector table. The vector offset is computed according to Section 8.6.1 as

```
Computed Offset Vector Address = Vector * Vector Spacing + EBase + 0x200
```

Our vector spacing is set as 32 bytes which means we could directly put 8 instructions in the table. We will use just one 4 byte jump instruction. Timer 1 uses interrupt vector 4 since we are setting the priority to 4. To make things simpler, our linker is smart enough to know where to place this in memory using the following "section" directive and the jump instruction:

```
.section .vector_4, code
j T1_ISR
```

This will jump to a service routine that you implement at the label T1_ISR.

Your ISR should be as short as possible. It should increment the time count only and not perform any OLED display functions. Otherwise, it will have a very inaccurate time!

6 Requirements

In this lab, you are required to write at least 3 assembly routines: T1Start, T1Stop, T1Setup. In addition, you must write one assembly Interrupt Service Routine (ISR)

The rest of the code can be written in assembly or C. It will be easier to write in C, but it is not overly complicated that makes writing in assembly impractical. You are welcome to try either approach depending on what you feel most comfortable doing. You are free to make additional subroutines in C or assembly beyond those that we require. However, you may not use any other library routines except for the OLED display and standard C libraries.

6.1 T1_ISR

You must write an assembly interrupt service routine to handle the Timer 1 interrupts. This routine should keep track of the time counter in units of your choice (seconds, milliseconds, etc.). This integer count should be in global memory so that your other functions can also access it. Using the peripheral bus clock frequency and the timer prescalar, the count can be converted to a time value. You may store a converted time and not just the count. Beware, that every instruction in your ISR adds additional delay overhead and can make your time incorrect! You shouldn't call the OLED display functions in the ISR because these are very slow functions. You should derive what you think the count should be to achieve approximately a 1 second period (or another value, if you want) and then calibrate it later when you can actually see the time display. Personally, my counter was in units of about 100ms.

6.2 T1Setup

Timer 1 should be used to implement the time count. You must set up the timer and timer interrupts in this routine when you first start your program. The configuration options were discussed in Section 5 and the lecture slides on PIC32 interrupts.

The period register (PR1) will determine how often your ISR is called. Given the peripheral clock frequency and the prescalar value that you set for the timer, you can calulate the time period for the interrupt to occur. It is simply

$$Time = PR1 * \frac{1}{\text{Frequency}} \tag{1}$$

However, any additional instructions in the ISR can make this time less accurate. Therefore, it is important to make the ISR as short as possible.

6.3 T1Start/T1Stop

These two assembly functions should simply start and stop the timer, respectively. In order to save power and not keep counting when the SW1 switch is off, we need to disable interrupts from happening.

6.4 Volatile Memory

If you chose to implement the non-required portions of your program in C, you need to understand the concept of volatile types. The ISR will track the time in memory each time it is called, but your C program will also want to read this to display it. The C program needs to know that the variable can be modified by instructions other than itself (the ISR). The keyword “volatile” tells your C code that the value is always changing and it should do a load instruction every time it needs the value. If you do not do this, the compiler could load the value once, keep it in a register, and never notice that it changes in memory! The volatile keyword is typically only used when interfacing directly with hardware (such as data input registers written by a peripheral), by multi-threaded programs with shared memory, or in ISRs like in our case. For your convenience, I have left this in the main.c program if you chose to write in C.

7 Suggestions (Please read!!)

Please consider the following:

1. Be sure to use proper subroutine calling conventions. If not, it is easy to over-write registers when calling subroutines or in interrupts.
2. Make sure to put nop instructions after all branches and jumps.
3. Make sure to clear and reenable interrupts in your ISR.
4. Breakpoints are your friend when your program goes haywire!
5. To help debug the timer, I use the LEDs that were used in lab 5. I simply display the count from the timer so that I can see it incrementing in binary. This is very useful to figure out why an ISR might be misbehaving.
6. Consider using the sprintf library function in stdio.h to convert your minutes and seconds to ASCII for OLED display. Instead of printing to output, this will print to a memory location producing a formatted string. In particular, you can specify a a formatted time using the following format specifier: “%d:%02d”. You can call this from either C or assembly depending on how you are programming the assignment. Be sure to allocate a buffer (on the stack or in global memory) for the string!

8 Extra Credit

It is required that you complete the rest of the lab first and that you demo the extra credit during your lab check-off. You cannot get extra credit if your assignment is late and you cannot do a second check-off with the extra credit.

You can receive 10 points of extra credit for blinking the colon in the time every second. It should turn off for one half second and on for one half second. It should not cause extra flicker in the display. You can implement this as an extra refresh as long as it doesn’t cause flicker. However, you could also use the graphic drawing function sin the library to invert the OLED pixels of the display so that it doesn’t require an entire refresh.

9 Lab Submission

Your lab will be submitted via your eCommons account. Please log in to eCommons using your UCSC account and attach the following files to your "Lab 6" assignment submission:

- main.c
- lab6.s
- lab6_report.pdf

Note that the final report must be submitted in PDF format.

Make sure to confirm that your assignment is SAVED and SUBMITTED before the deadline. You may resubmit your assignment an unlimited number of times up until the due date.

9.1 Check-off

For this lab, as with most labs, you will need to demonstrate your lab when it is finished to the TA or tutor and get it signed off. You will also need to submit your lab files using eCommons.

9.2 Grading template

This is a suggested grading rubric. It is also a good general guideline before submitting your lab to check off these points.

9.2.1 Requirements

☐ (15 pts) Control Switches

- Does SW1 start/stop the timer? (In other words, is the timer disabled when SW1 is off?)
- Does SW2 clear the timer?
- Does SW2 take priority over timing/SW1?

☐ (25 pts) Timer 1

- Is the T1Start subroutine written in assembly?
- Is the T1Stop subroutine written in assembly?
- Is the T1Setup subroutine written in assembly?
- Is the T1 ISR written in assembly?
- Is Timer 1 properly calibrated to get an accurate second?
- Do the subroutines and ISR save/restore registers?
- Does the ISR clear and re-enable interrupts?
- Is the timer count saved in global memory?

☐ (35 pts) Display

  - Does the program display the cleared/initialized time of “0:00”?
  - Does the program always display two digits of seconds?
  - Does the display refresh when the timer is enabled?
  - Does the dispay flicker during timer enabled?
  - Does the display flicker when the timer is disabled?
  - Does the display flicker when the timer is being cleared?
  - Is a memory buffer properly allocated on the stack or in global memory for the string?
  - Can the timer count past 10s?

- ☐ (10 pts) Extra Credit
  - Does the display blink the “:” on and off every second?
  - How did you make it blink? (Refresh or individual library calls on pixels?)

9.2.2 Lab write-up requirements

In the lab write-up, we will be looking for the following things. The lab report is worth 25 points. We do not break down the point values; instead, we will assess the lab report as a whole while looking for the following content in the report.

Along with the usual items, you should answer the following questions:

- How did you go about designing your program?
- Where is your ISR placed in memory? What type of memory is it?
- How did you calibrate the timer routine? How far off was your “ideal” timer PR1 register based on frequency due to the ISR overhead?
- How did you ensure that your display didn’t flicker?
- What sort of bugs did you encounter while writing your program?