Lab #8 – Uno32/PIC32 IO

Final Lab Assignment!!!

40 points + 10 points lab report
20 points extra credit
Submit all source by, December 10th by 11:55pm

It is assumed you went through the lab # 8-tutorial lab which will familiarize you with the Uno32 and the software flow. In this lab you will be working with the input and output (IO) of the Uno32.

Part A – IO Overview

The input/output in our microcontroller is based on device registers mapped to certain addresses. Specifically, LED 5 is connected to pin 43 on the board. This is pin 58 on the microcontroller which is digital IO port F. Specifically, bit 0 (RF0) is the bit that controls the LED. This information is available in the Uno32 document from Digilent (**chipKIT-Uno32-RevC_rm.pdf**) and the PIC32mx reference guide from Microchip (**PIC32 MX320F128.pdf**). Both are in the class resources on eCommons in the UNO32 area.

The port F address starts at 0xBF88 6140 and is shown in Table 4-31 of PIC32mc reference guide. There are 4 registers that let us configure the input/output: TRISF (0xBF88 6140), PORTF (0xBF88 6150), LATF (0xBF88 6160), and ODCF (0xBF88 6170) which are described in Section 12.1 (starting at page 102). The document describes all the ports as TRISx where x can be A to G for each of the IO ports (A to G) at different addresses.

Each of the above 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 modify one bit 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 set bit 0 of the TRISF register, you can write a binary mask of 0x1 to address 0xBF88 6148 (0xBF88 6140 + 8).

For example to work with LED 5

You will only need to use the TRISF register and PORTF register. First configure PORTF so that it will be an output port by setting TRISF register bit 0 to a 0 (active low). Once you have TRISF bit 0 set to a 0, you can now write data to bit 0 of PORTF. Setting PORTF bit 0 to a 1 will turn on LED 5 and setting to a 0 will turn off LED 5.

What to do for Part A:

Program a continuous loop which does the following:

- Read the value of each of the 4 switches (SW1 – SW4) and turn on LD1-LD3 if the corresponding switch is “on”.
- Read the state of the 4 buttons (BTN1 – BTN3) and turn on LD4-LD8 if the corresponding button is being pressed. Unlike the switches, the buttons are “momentary”, meaning they are have to be held down to be active.

Pretty straightforward. This gives you the basics of how use a microcontroller to read input and set output values.

Part B – Software Delay

In this part you will be working with the delays (software and hardware based) of the Uno32 to come up with 16-levels of delay for the cycling through the LEDs. Start by making a function called `mydelay` which uses a software loop to “waste” time. Send in an argument in `$a0` register which will be a multiple of the base delay. Basically make a for-loop in your `mydelay` which repeats `$a0` times.

Use this delay function to do the following:

- Read in the value of the 4 switches (0-15 to match to 1-16 multiplier)
- Turn on LED1. Wait for a 1-16x multiple of the `mydelay`, so a lower setting on the switches will result in a faster turning off. After the delay turn the LED1 off.
- Turn on LED2. Repeat.
- Got through all the LEDs.

If you like, get creative, and do additional patterns in addition to the basic one. Here are some other ideas:

- Cycle back and forth through the LEDs or do like a Cylon from Battlestar Galactica.
- Start with the two middle LEDs on and then go outwards and bounce back in.
- Come up with your own patterns
- Use a button(s) to change your pattern

Extra credit will be given for different patterns.

Part C – Hardware timer (EXTRA CREDIT)

Using a software loop is very wasteful of resources. In a real embedded system you would need more exact timing or need to be conservative of a batter power supply. You might put the processor into a very low power sleep mode and have it be woken up when it needs to change an IO. This is where a hardware timer comes in.

Refer to this web link for some information on the timer hardware of the PIC32:

http://www.johnloomis.org/microchip/pic32/timer/timer.html

Create a new delay procedure called `myhwdelay`. There is even an example of a delay function in the link above. You can model yours after that. Just as with Part B, you will read in the 4 switches and delay your LED pattern based upon a 1-16x delay. It should be as simple as replacing your `mydelay` called with `myhwdelay`.

Lab Write-Up Requirements

In the lab write-up, we will be looking for the following things. 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.

- The flow charts you created for your program.
- How did you go about designing your program?
- What sort of bugs did you encounter while writing your program?
- Describe how the registers for the ports works? Don't just repeat what is in the reference manual, give a description that a non-computer person might understand.
- What is a hardware timer? Why is it better than a just putting in some instructions in a while or for loop?
- What happens if you forget to put a nop after your branch?

Happy Embedded Coding!!