Lab 8: Uno32/PIC32 IO

Olexiy Burov

June 5, 2015

TA: Hany Fahmy hfahmy@ucsc.edu
Section 2
Due: 8 June, 2015
E-mail: oburov@ucsc.edu

Lab Description and Objective

The lab consisted of three parts: A, B and C. Part A required us to write a PIC32 assembly program, which would based on the state of 4 switches and 4 buttons turn on the corresponding LED's on the IO Shield of the micro-controller. Part B requires us to write an assembly or C program which would play with bouncing LED patterns using software delay. Part was extra-credit and required us to use hardware timer of PIC32 for the same purpose as part B.

Extra Credit Description for part B

For extra credit there were a few additional tasks, which were applicable to part B:
Just use a few other LED patterns and add a functionality to change between them using a button.

Lighting up an LED

Eight LEDs provide discrete digital outputs. The LEDs are connected to the low eight bits of micro-controller PORTE and all eight can be written at the same time by writing to PORTE. In order to be able to write to LED one should put them in the "input-mode" by setting least significant bit of TRISE to 0.

Reading the switches

There are 4 switches, which are connected to the PORTD of the micro-controller. The value of SW1 is corresponds to 8th bit of PORTD. The values of SW2,SW3,SW4 are bound to 9,10,11th bits of PORTD respectively. In my implementation 4 switches control first 4 LEDs, SW1 corresponding to LD1 accordingly. So in order to light up LEDs in accordance to switches, I load the value of PORTD, shift it to the right by 8bits using srl(shift right logical). As a result of that operation, bits 8,9,10,11 become 0,1,2,3, which greatly simplifies the process because now we can simply OR PORTE with this value and bits 0,1,2,3 of PORTE would be set to the values of switches lighting up corresponding LEDs.

Reading the buttons

In contrast to switches, buttons are distributed over two different ports, PORTD (bits 5,6,7) and PORTF (bit 1). The technique of reading the value is similar to reading from switches, but one must do a little of bit shifting and masking here. So, to get the value of button 1, we need to and the contents of PORTF with 0b10 (binary 2 - first bit). After that I perform left logical shift (sll) by 3 bits to put this value into 4th bit and then OR it with LED register, which would result in first LED lighting up if the BTN1 is pressed. The same kind of manipulation is performed on BTN2,BTN3,BTN4. The bits 5,6,7 of the PORTD are masked out, then they ORed with LED registers. As a result bits 5,6,7 light up LED 6,7,8.

Part B using software timer

For part B we need to be able to control the delay of the software value by using 4 switches and implement some basic as well as custom LED patterns, the later is for extra credit though. In order to get the delay multiplier from 0-15 one can use the following line of code in C:
int switch_value = PORTD >>8;
In this way we get the 4 bits of the 4 switches, which are able represent 16 different values e.g. levels of delay. In order to use a software timer one just needs to declare some constant which is a delay and then have a for loop, which does nothing in it's body except incrementing a software timer.0 would mean a standard delay, while 5 would mean standard delay += standard delay * 5 in pseudo code.

LED patterns

In order to perform different blinking-bouncing patterns for LEDs one needs to enforce some rules on how the LEDs change.

Bouncing Rotate Pattern

This is the simplest pattern and the one required for the part B. We need to turn on the first LED, wait for the delay to pass and then turn off the first LED and turn on second LED instead and so on. When we reach the last LED, we need to start from the beginning again.

In this case LEDs depict what is happening to the bits of the PORTE register.
So if we start with 1 and keep shifting it to the left by 1 bit we will get the behavior we need. However, after 8 shift, one would be go beyound LEDs bit space, so we need to include a condition that would bring it back to 1.

Bouncing Back-Forth Pattern

This one is a little more advanced because once we reach the last LED, we need to start going in the opposite direction(shifting right) and when we reach 0, we need to go back to 1 and change the shift direction to left again. For direction I use a boolean flag as an int. 0 means that LED is going in left direction, while 1 means that it is going in right direction. Once one of the boundre values such as 0b0 or 0b10000 0000 is reached the direction gets changed.

Two Bouncing Back-Forth LEDs Pattern

For this one I created a pattern that has two LEDs starting from the middle and going to opposite direction and then "bounce" back into the middle and then "bounce" back again etc. This part also has a boolean flag that controls the direction. In this part, however, I chose to use two different masks on which I perform shift operation. These mask represent the positions of LED's on the board. They get shifted simultaneously into different directions as well.

Accumulating pattern

This one performs the following:
The first LED lights up, then after a delay the second lights up, but the first one stays lighted up. It happens until all of the LEDs are not lighted up. After that it jumps back and starts over again. This one again required me to use two masks: the one for storing the result and the one for shifting. So if we start with 0 and 1, 1 get written and the shifted to

2 etc. So one mask turns on the next LED, while the other one keeps track of all the LEDs that were turned on.

Hardware timer

Software delay is somewhat less efficient in terms of power use and accuracy because we initialize a variable, increment it every cycle, perform a branch instruction, etc. Hardware timer however is always there, so it doesn't need to be initialized and it gets incremented automatically without programmer supervision. Therefore it is more efficient and accurate. Hardware timer uses interrupts to count.

Conclusion

The lab provided a nice introduction to the concept of micro-controller, pins, shield and so forth. I've never had any experience in working with hardware, not saying about such low level control. However, I found it fascinating and insightful. It got really interesting when we were able to control the parts of the micro-controller and contemplate the results of our work and programing. This lab required us to do a lot of shifting, masking and ORing of different memory locations, so I reviewed mu knowledge and understanding of these operations. It also served as the final step in the course, which brought us from assembly to C and tought us how one can incorporate those two.