CSCE 236 Embedded Systems, Spring 2014
Lab 2

Monday, February 10, 2014

Names of Group Members:

1 Instructions

This is a group assignment to work on during class. You only need to hand in one copy of this, but make sure that the names of all of your group members are on this sheet to receive credit. Complete all of the sections below and make sure to get the instructor or TA to sign off where required. You should keep your own notes on what you complete since parts of future homework will build on this lab.

2 Timers

In the pre lab you detected button bouncing. You detected this by looking at the counter which counted the number of falling edges on the button input pin and seeing if this value increased by more than one within a small window of time. The assumption was that any oscillations would calm down within that time period. In this section, you will reconfigure the `Timer1` to time how fast bouncing occurs. This will let you minimize the delay in the previous sections so that you can detect very fast button pressing, while still ignoring bounces.

Start by reviewing section 16.6 of the datasheet on the input capture unit (you should have already read this section before the lab). The way the input capture unit works (when it is configured) is that when a transition on pin `ICP1` (labeled on the Arduino schematic simply as `ICP`) occurs, the current value of the counter, `TCNT1` is copied into the register `ICR1`. An interrupt can be triggered whenever this occurs, but do not worry about using interrupts at this point. Instead, modify your code so that when the button is pressed you reset `TCNT1` and `ICR1` to zero. After 100 milliseconds (or there about) read the value in `ICR1`. If any bouncing occurred (or if you released the button within this time), the register `ICR1` will contain the number of ticks that occurred between the initial press and the last bounce. Specifically, you need to:

- Connect your button to `ICP` (physically with a wire)
- Configure the timer clock so it counts up continuously (clocked from the internal clock, not the input pin). You need to pick the right divider for the clock. If you run the clock too fast, you will overflow the 16-bit counter before a bounce occurs. If you run it too slow, you will not have very good resolution.
- On button press, clear `TCNT1` and `ICR1`
- Delay for a little while before reading the value of `ICR1`
- If `ICR1` is non-zero a bounce occurred. Compute how long the bounce took.

Checkoff: *If (and only if) a bounce occurs, print out the time (in micro or milliseconds) since the initial press of the button occurred. If no bounce occurs, simply print out the number of times the button has been pressed. Note: Do not use the input noise canceler, since we are trying to measure the noise!*

3 Counting Button Presses

Checkoff: *Build on the previous code and create a program that will turn on the red LED if the button is pressed once, green if pressed twice, blue if pressed three times. For each of these you should only turn on the LED after the button pressing has completed (after a short timeout) and you should turn it off after 2 seconds. Also make sure that it functions properly with both short and long button presses, but also avoids bounces.*

4 More Bounce Detection

Checkoff: *Our button tends to bounce more on release. Update your code to time how long bounces are on release. For this to work well, you may need to hold your button for a while before releasing.*