CSCE 236 Embedded Systems, Spring 2016
Lab 2

Thursday, February 11, 2016

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 Counter/Timer

Section 16.6 and 16.11 of the datasheet provides the key details for the counter/timer, although there is a lot of other useful information throughout section 16. Refer to these sections for additional details.

2.1 Counting Button Presses with Counter

Download the code from the course website. Pay close attention to the definitions at the top as to where the pins are connected. Look at the comments to see how the counter is configured. Most important is the line that configures the counter to count when the pin T1 changes from high to low (falling edge). This is all performed in the setup.

Now, modify the code in the main loop so that it only prints the value of the register `TCNT1` when it changes. Be aware that TCNT1 may change at any point, so you should only read it once in each iteration of your main `loop()` to ensure that the value you are using is consistent.

One way to detect that the button bounced is if `TCNT1` has increased by more than one since the last iteration. However, since the main loop runs very quick, you should add a delay in the main loop so that you only check it periodically, perhaps every 100 milliseconds to give the button time to bounce. Implement this approach and print a warning every time the button bounces.

Checkoff: *Show your code that only prints the value of TCNT1 when the value changes and also that it can detect bounces.*

2.2 Avoiding Button Bounces with Delays

There is a preexisting Arduino sketch `Examples->Digital->Debounce` which implements debouncing by using the command `millis()` to record when the button was pressed and then to only check the state again after a fixed delay period. Look at this code to see how they implemented it and make sure you understand the code. Ask the instructors if you have questions on how it functions, but there is no need to get this checked off. The problem with this is approach is that different buttons may require different delay times, so now we will figure out this delay using the timer/counter hardware.

2.3 Continuous Counting with Clock

The first step to determine the length of the button bounce is to change the configuration of the counter/timer unit to increment with the main CPU clock. Download a new copy of the lab2 code and now modify the

configuration of TCCR1B such that the CPU clock with some divider (1, 8, 64, 256, or 1024). This will increment the clock and it will eventually roll over the counter once it reaches the maximum value of 0xffff. **Checkoff:** *How many seconds does it take to reach the top for each divider? How did you figure this out?*

2.4 Using Counter to Measure Button Bounce

Now we will be able to figure out how long the bouncing lasts. This will let you minimize the delay in the example Arduino debounce code so that you can detect very fast button pressing and not bounces.[1]

Section 16.6 of the datasheet describes the details of the input capture unit. 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.

Unlike the first problem, we are not using TCNT1 to count button presses, but rather we are using it to time the button bounces by storing the value of TCNT1 in ICR1 when the last bounce occurs.

Specifically, you need to:

- Connect your button to ICP (physically with a wire), check the schematic to verify which pin this is.
- Configure the timer clock so it counts up continuously (clocked from the internal clock), like in the prior problem. 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. Use the information from the prior problem to help you figure this out.
- In your main loop code you should check for a button press by using digitalRead. Then, on button press, clear TCNT1 and ICR1
- Once the button is pressed, 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!*

2.5 More Bounce Detection

Checkoff: *Buttons also bounce 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.*

2.6 Timing Button Presses

Checkoff: *Build on the previous code to create a program that will time how long you hold down your button before releasing.*

[1] In practice you will probably never press a button fast enough for this to matter, but if you are using a button or similar device to measure engine RPM or something along these lines it will be useful.