EECE 5141C/6041C
Introduction to Mechatronics
Lab 3: Switch Debouncing and Edge Counting

Name: _____

In this lab, you will design, program, and construct a simple microcontroller system capable of debouncing a single switch input and counting rising edges.

DELIVERABLES

1. Completed Worksheet
2. Annotated Source Code

SYSTEM DESCRIPTION

The microcontroller must monitor one digital input supplied by the provided spinner assembly. The spinner assembly utilizes a single-pole, double-throw switch. However, this switch can also used as a single-pole, singe-throw switch by connecting only one of the throw terminals. (Note: One throw is normally open, and one throw is normally closed.) In this lab, you will debounce the switch signal and count switch transitions using two methods: one software method and one hardware method. The microcontroller, should also display the current number of counted edges in hexadecimal format using two seven-segment displays.

Preliminary Switch Characterization

Construct a pull up switch circuit as shown in Figure 1 using the normally open throw of the spinner switch. Using the oscilloscope, measure the output signal of the circuit as you rotate the spinner and trigger the switch. Sketch the measured signal in Question 1 of the Design Questions Section. Be sure to indicate the approximate settle time as well as the approximate pulse length of the bouncing signals. Using these measurements, determine the amount of time your software should measure a constant signal before declaring the switch settled and answer Question 2.

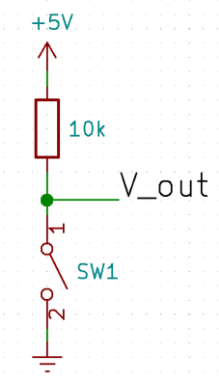

Figure 1: Pull-Up Switch Circuit

Method 0: No Debouncing

In order to demonstrate the need for switch debouncing, we will first implement a simple edge counting algorithm that only looks for pin transitions without any debouncing. This can be accomplished with a simple modification of your code from Lab 2. The first modification requires the addition of edge detection logic within the main `while(1)` loop. This is accomplished by comparing the previous pin value with the current pin value for each iteration of the `while(1)` loop. Each time there is a 1 to 0 transition, a falling edge has occurred and an `edge_count` variable should be incremented. Now, instead of displaying the value of the 8 input switches from the previous lab(which we are no longer using), we will display the current value of an `edge_count`. Once implemented, slowly rotate the spinner to trigger the switch. Record your observations in Question 3.

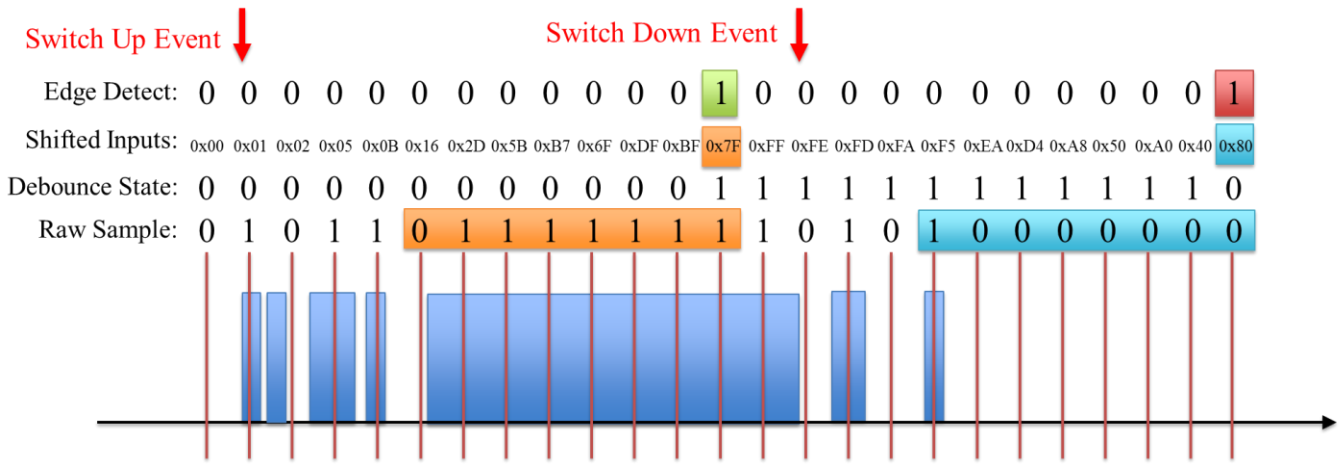

Figure 2: Shift and Match Method

Method 1: Software Debounce and Software Edge Counting

For this method, your system should use Timer 1 to generate an interrupt at a regular interval (which will be calculated based on switch characteristics). Within the corresponding ISR, your system should sample and debounce the switch circuit signal using the shift and pattern match method as shown in Figure 2. We will utilize a 16-bit long match pattern. Therefore, you will need to use a `uint16_t` variable to store the shifted input bits. You must determine the appropriate sample period (Timer 1 match value) based on the switch bounce characteristics observed in the previous section. For example, if you have determined that a switch is stable after no changes in the past 5ms and you are using a 16-bit pattern, your sample rate would be 5ms/16. Document these calculations and the corresponding clock source and `OCR1A` register values in your answer to Question 4. Once a falling edge is detected, the ISR should increment an `edge_count` variable. Lastly, the total number of counted edges should be displayed using two seven-segment displays. Recall from Lab 2, your Timer 1 ISR alternated activation of the two digits in order to time multiplex the segment outputs. This program should also execute the same alternating behavior. However, your Timer 1 match rate is selected to provide the appropriate pin sample rate for your debouncing functionality. This match rate may be too fast for your seven segment circuits to respond. Therefore, you will need to use an additional count variable to reduce the digit toggle rate. For example, if Timer 1 ISR triggers every 1ms, but you want to toggle your digits every 30ms, you will need to toggle every 30 ISR calls.

Method 3: Hardware Debounce and Counting via Timer 0 Clock

As an alternative to software debouncing, we can also construct hardware debouncing circuitry external to the microcontroller. For this lab, you will construct a hardware SR latch debouncing circuit as shown in Figure 3. You will be provided with a quad NAND gate IC. In order to utilize this circuit, you will need to connect both throws terminals of the spinner switch to pull-up resistors and connect the single pole terminal to ground. The output of the SR latch debouncing circuit can then be used to clock an internal timer/counter in order to count the debounced edges. In this lab, you should utilize Timer 0 as the edge counter. You will still use Timer 1 to generate match events to display the current count value (which will now be stored in `TCNT0`) using the two seven-segment displays.

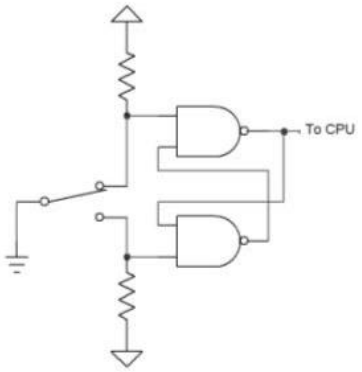

Figure 3: SR Latch Debounce Circuit

DESIGN QUESTIONS

1. Sketch the measured output signal from your pull-up resistor circuit. Indicate the approximate settle time as well as the approximate pulse length of the bouncing signals.

2. Based on your observations in Question 1, what is a reasonable amount of time to wait until declaring the switch signal stable? Why?

3. When implementing Method 0, record the behavior of the counted value as you slowly rotate the spinner.

4. When implementing a 16-bit long match pattern, what sample rate would your Timer 1 need to generate in order to detect a settled signal based on your answer of Question 2? What Timer 1 clock source and `OCR1A` values are needed to generate this sample rate.

5. Suppose you needed to observe and debounce 64 input signals. From a system cost perspective (assuming your design and programming time is free), would you implement hardware or software debouncing? Why?

6. Are there any reasons we should not use arbitrarily long match patterns for our shift and match algorithm? For example, we could use 256-bit long patterns. Why might this be a bad idea?

7. Conversely, we could also use 2-bit long patterns (for example `0b10` for falling edge and `0b01` for rising edge). Why might this be a bad idea? Is this similar to any other methods we have discussed?

8. When using a hardware debouncing circuit (such as the SR latch debouncer), you could implement a software edge detector similar to the shift and pattern match method. How long (how many bits) of a pattern should you use? Why? Is this similar to any other methods we discussed?