1. Description

The purpose of this lab is to code and implement a digital counter with an interrupt switch. Whenever the interrupt switch is triggered, the counting is halted and the interrupt service function is called to print the word “Interrupt_” on the LCD.

A phenomenon known as “bouncing,” which is the result of the switch’s dynamics, was then corrected as it would lead to erroneous interrupt triggers. The base system was improved upon by adding a debouncing circuit to eliminate the switch’s undesirable transient dynamics. This debouncing circuit was implemented through the use of two NAND gates.

Hierarchy:

```
Main
|_MyRio_Open()                                          // Opens a session with the MyRio
|_Irq_RegisterDiIrq()                                   // Register the interrupt
|_pthread_create()                                      // Create a new thread for the interrupt
|       |_DI_Irq_Thread()   (called as an argument)     // Interrupt service function
|               |_while()                               // Interrupt until thread stopped by main
|               |       |_Irq_Wait()                    // wait for IRQ to assert or time out
|               |       |_if()                          // if the IRQ is asserted
|               |               |_printf_lcd()          // Interrupt message printed to LCD
|               |               |_Irq_Acknowledge()     // Interrupt acknowledged to the scheduler
|               |_pthread_exit()                        // Terminate the new thread
|_while()                                               // Counting loop
|       |_printf_lcd()                                  // Prints the current time count on the LCD
|       |_wait()                                        // wait() called 200 times to consume ~1s
|_pthread_join()                                        // Part of the interrupt termination sequence
|_Irq_UnregisterDiIrq()                                 // Interrupt unregistered
|_MyRio_Close()                                         // Closes the session with the MyRio
```

Note 1: Some functions are called multiple times per hierarchy tier, but only one call is shown above.

2. Testing

Initial testing was focused on verifying that the counter was executing properly and that the switched triggered the interrupt and executed the interrupt service function. This testing involved running the code and checking that the LCD showed incrementing counter values and verifying that when the switched was depressed, the word “Interrupt_” was displayed. The bounce phenomenon was observed during this initial testing and can be seen in Figure 1. The testing procedure was repeated after the addition of the debouncing circuit and the impact on the output can be seen in Figure 3.

3. Results

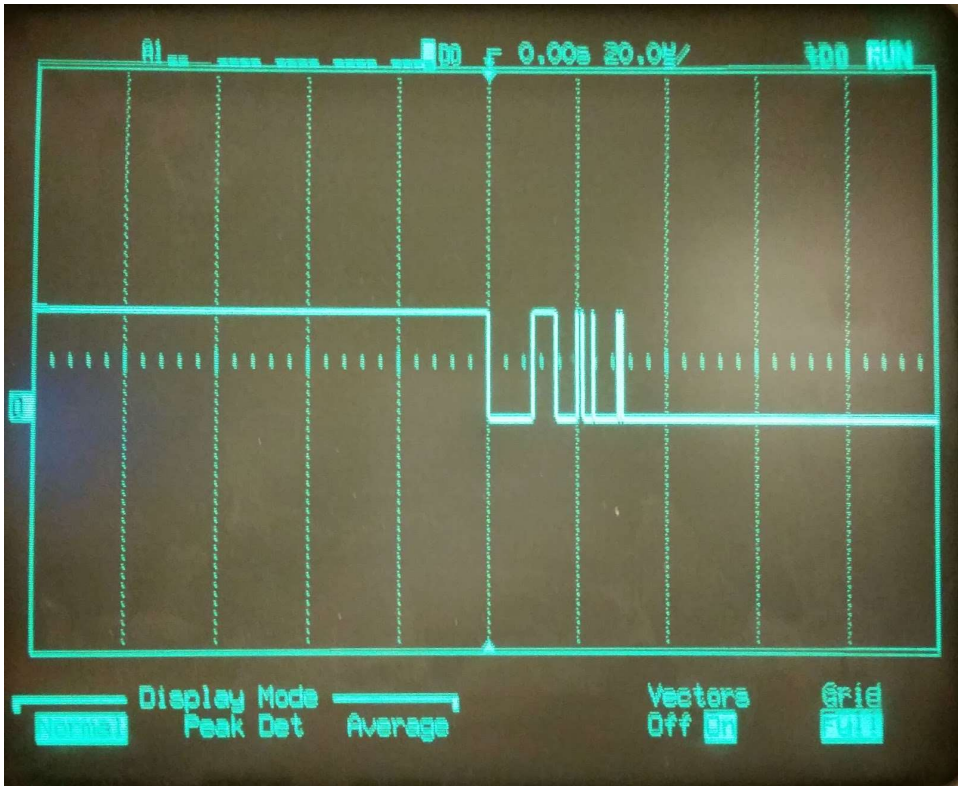

Figure 1: Oscilloscope Output for Uncompensated System (Bounce)

The scale in Figure 1 is in 20 microseconds and the initial high value represents the switch being depressed. The additional high/low fluctuations in the signal represent the bounce phenomenon. This transient behavior took approximately 20 to 24 microseconds to settle and between 2 and 10 peaks were seen during this interval. The debouncing circuit seen in Figure 2 was added to the system in order to remove these additional pulses.

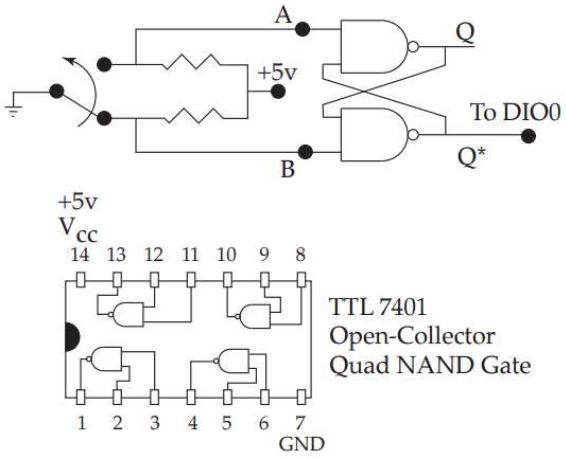

Figure 2: Debouncing Circuit

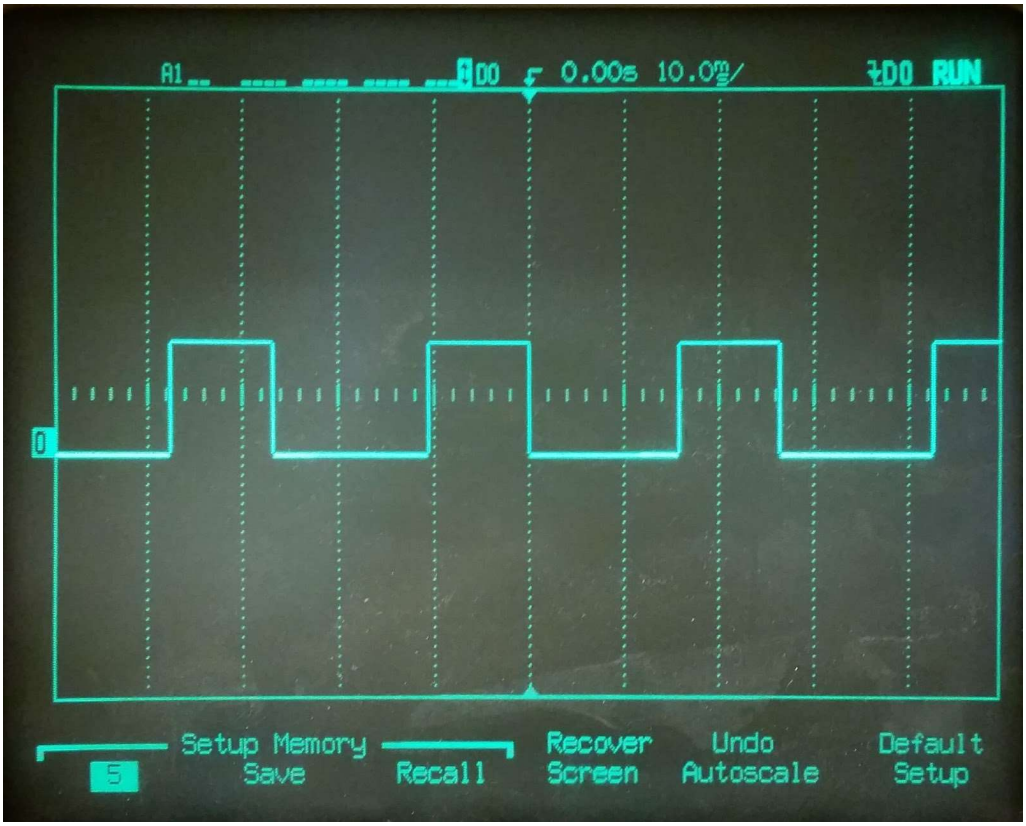

Figure 3: Oscilloscope Output for Compensated System (Debounced)

Figure 3 above shows the output of the system after the addition of the debouncing circuit. Notice that the time scale is on the order of milliseconds instead of microseconds. Each of the displayed pulses is the result of a single depression of the switch.

Below is an example of the output of the debouncing circuit given an assume initial condition of Q*=1.

Step	A	B	Q	Q*
@B	1	0	0	1
Transition	1	1	0	1
@A	0	1	1	0
Transition	1	1	1	0
@A	0	1	1	0

Table 1: Input/Output for Debouncing Circuit

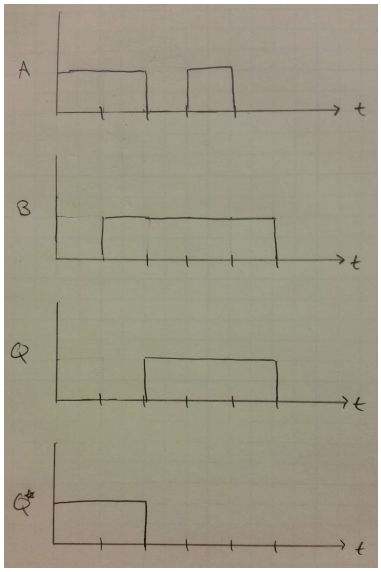

Figure 4: Sample Input/Output for Debouncing Circuit

Table 1 and Figure 4 show an example execution sequence for a switch experiencing bounce. The graph of Q and Q* show that the oscillation seen in the A signal is removed and all that is left is a smooth signal. The NAND gate circuit in Figure 2 was used to create this table and a 1 condition was based on the given lead being at 5v and 0 was associated with being at 0v. The operation of the NAND circuit required an initial state assumption for Q* and historical states of Q and Q* were then used to determine current states of Q and Q* given the A and B states at that step.

The *main()* function configures the interrupt thread through the call to *pthread_create()* where the interrupt thread conditions and the interrupt service function are passed as arguments. When the interrupt condition is raised then the *main()* thread will pause and the interrupt thread will be executed. The interrupt will be serviced as long as the main function has not terminated the thread via setting irqThread0.irqThreadRdy to NiFpga_False, which will only occur when the *main()* function has finished executing in this particular case. If this condition is not set (i.e. irqThread0.irqThreadRdy = NiFpga_True), then the interrupt will continue to be serviced every time the interrupt flag is detected. When the interrupt flag is detected and as long as irqThread0.irqThreadRdy is set to NiFpga_True, the interrupt will be serviced by the *DI_Irq_Thread()* function. This function executes as a result of the interrupt thread being processed in lieu of the *main()* function thread.

The test results received were as expected and the addition of the debouncing circuit had the desired effect of smoothing the signal. One potential source of error in this experiment is how the switch and debouncing circuit are connected. Different waveforms were seen amongst the different stations and could be a result of inverted switch connections. In these cases, the signal remained high until the switch was depressed, at which point the signal went low until the switch was released. This lab would be much improved with a more detailed explanation of how interrupts are implemented in a general case. Interpretation of experimental results would have been aided by additional instruction in circuit logic and reasoning through cases like the Q and Q* graphs.