CompArch Lab 2: SPI Memory

Kimberly Winter, Andrew Pan, and Robbie Siegel

November 2, 2017

1 Input Conditioner

1.1 Circuit Diagram

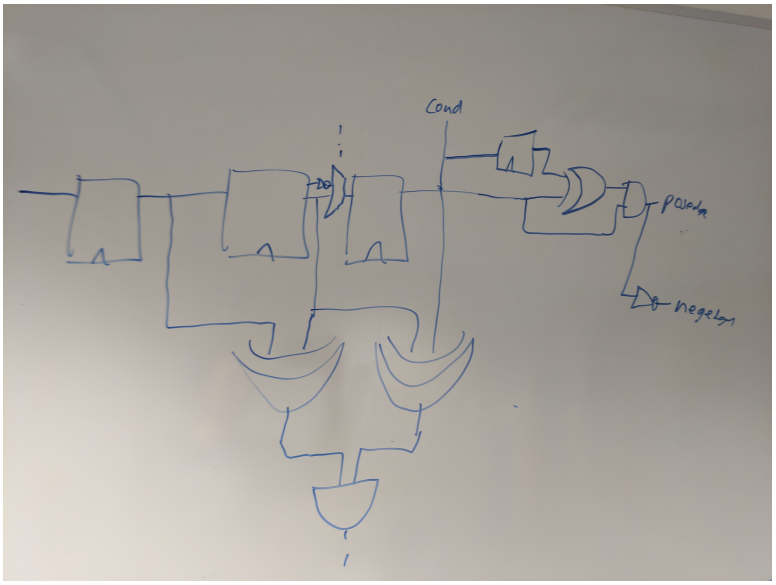

Figure 1: The waveform output from our input conditioner test bench

*Note: The output of the and gate from the two xor gates is acting as the control signal to the mux, where it selects the normal signal of the previous D-Flip-Flop gate when low, and its inverse when high

1.2 Waveforms

Input	Time	Conditioned Signal	Clock	Posedge	NegEdge
0	0	0	0	0	0
1	110	1	1	1	0
1	150	1	1	0	0
0	160	1	0	0	0
1	170	1	1	0	0
0	290	0	1	0	1
1	392	0	0	0	0
1	400	0	1	0	0
1	410	1	0	1	0

Figure 2: The results of our input conditioner test bench

We tested the input conditioner for synchronization, debouncing, and edge detection. The results of our test bench is shown above in Figure 2. We created waveforms based on these tests, shown in Figure 3.

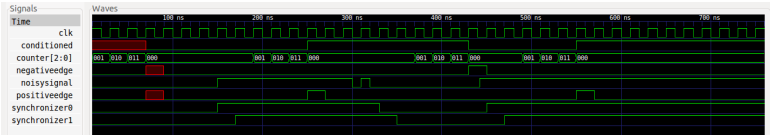

Figure 3: The waveform output from our input conditioner test bench

We checked synchronization by inputting the signal at an offset from the clock edge and making sure that our output was stil synchronized with the clock edge. We tested debouncing by changing noisySignal from high to low and back to high within a very short time period. The input conditioner allowed conditioned to remain constant in this situation. Finally, we tested edge detection by ensuring that positiveEdge and negativeEdge changed as conditioned rose to high and fell to low.

1.3 Glitch Suppression Question

If the main system clock is running at 50MHz, then that means each clock cycle takes $\frac{1}{50MHz} = 20ns$. The waittime of 10 in the system indicates that any glitch under 10 clock cycles in length will be suppressed. Therefore, any glitch that is less than 200 ns will be suppressed.

2 Shift Register

2.1 Test Bench Strategy

The test bench we used to test our shift register before loading it onto the FPGA is included in the file shiftregister.t.v.

There were three sections to this test bench. In the first, we tested the capabilities of parallel in, serial out. We tried to input 8 bits with parallelLoad set to low and high to ensure that serialDataOut would only respond to a high parallelLoad.

We then tested serial in, parallel out. We tried to input a serial bit with peripheralClockEdge set to low and high to ensure that parallelOut would only shift when peripheralClockEdge is high.

We then ran one test with both peripheralClockEdge and parallelLoad set to high to ensure that serial in, parallel out would take priority. This was a decision we made when designing our shift register. Although we did not see a way the priority affected our system, we felt it would be important to test nonetheless.

3 SPI Memory

We followed the suggested schematic from the assignment description to design our SPI Memory, but also added the system clock to the finite state machine so that it could be updated within the serial clock cycle.

3.1 FSM

We first designed a block diagram for our FSM, shown below in Figure 4.

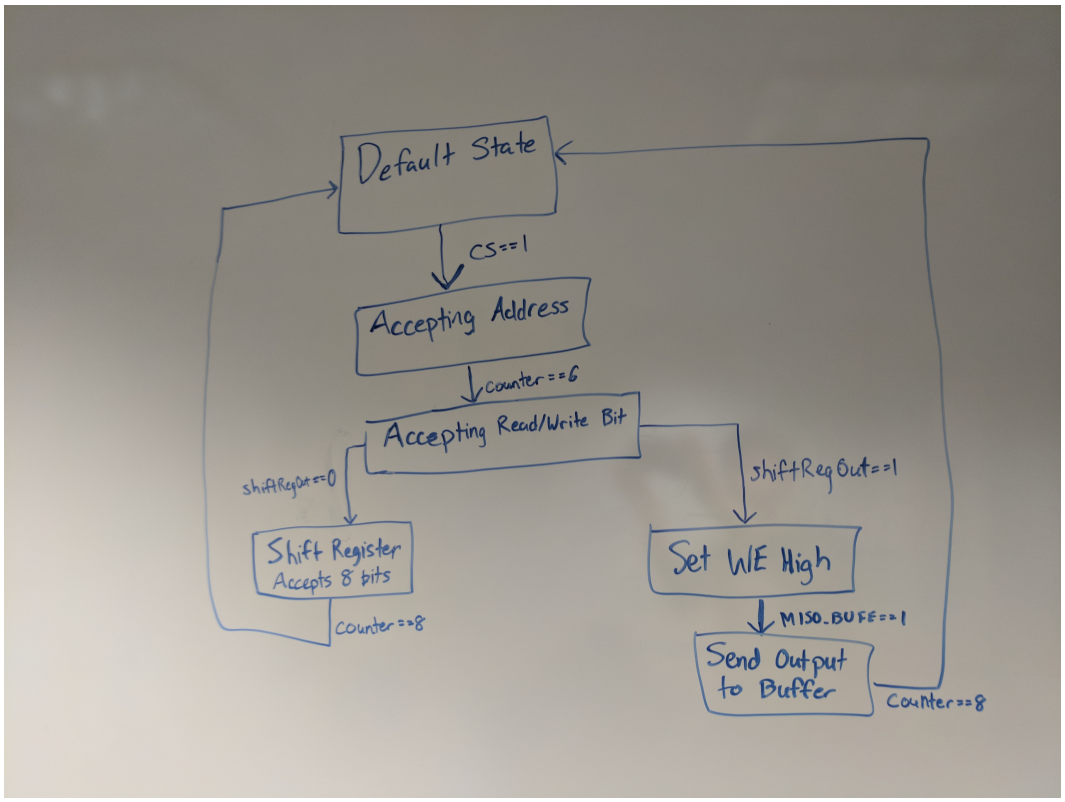

Figure 4: A simplified block diagram of our Finite State Machine

We based our design of the FSM off of a master-slave interaction between the FSM and the rest of our SPI Memory module. The FSM handled controls that allowed the functioning of the other memory components, mainly accounting for the timing with which all components had to operate alongside the shifting of bits from the shift register. We drafted our design by breaking down the necessary steps required for the read and write functions of the SPI Memory in terms of the write enable inputs for Data Memory, Shift Register, and Address Latch. After determining what write enable lines had to be set to high, we planned out the number of clock cycles each state the FSM should remain in for, and began writing behavioral Verilog to replicate our expected behavior.
In our code, we used a switch statement and a series of cases based off of the "current state" of our FSM, and at the positive edge of each SCLK pulse, we would evaluate what state changes needed to be made. Our states were broken down into a few categories: the default waiting state, address accepting state, read output state, and data writing state. Each category consisted of one or more states based on the number of different discrete operations involved in each category, and handled the transition from state to state given certain conditions and the current state.

3.2 Timing

We began this lab with several timing issues and had to spend a significant portion of time debugging the timing of our pins. A fair bit of our issues were resolved by properly accounting for propogation delay and accordingly increasing delays in our testbench, but we also had to be very specific about our timing for the FSM. Small errors (and especially 1-off errors) would cause significant issues in the overall functionality of our memory, and failing to account for the timing of signals throughout the entire SPI Memory module when writing our FSM made debugging difficult. Eventually, we timed out how many clock cycles it took for each bit to get through each element of the circuit. We kept track of how many serial clock cycles had past by implementing a counter and creating a for loop that broke or changed conditions when a certain counter number was hit. This allowed us to control our timing more specifically within the system.

4 SPI Testing Strategy

For testing the SPI, we wanted to test that the machine could perform its basic functions, reading, writing, and reading another address after it has read a different address. This was implemented by first writing "11111111" to the address "0000000", one of the simplest tasks one could perform and read the input to memory to check that it had performed its task, and it had. Next, we wrote "10101010" to the address "0000001" and read that. It also had worked. After this, we read the original address to check that our SPI could hold a state. After testing these basic tasks, we determined that our SPI worked.

5 Work Plan Reflection

Based on our two previous labs, we realized that we needed to anticipate longer times to accomplish tasks than we expected. The only part of the lab that took far longer than we expected was testing and debugging our SPI Memory. The time estimates for the other tasks in this lab were relatively accurate.

We also fell behind on the dates that we intended to get things done by. This was largely due to the fact that we were incorrect about when we would be able to meet as a group when we planned this lab out a week and a half in advance. As such, most of the tasks we laid out for the second half of this lab were completed a few days behind schedule.