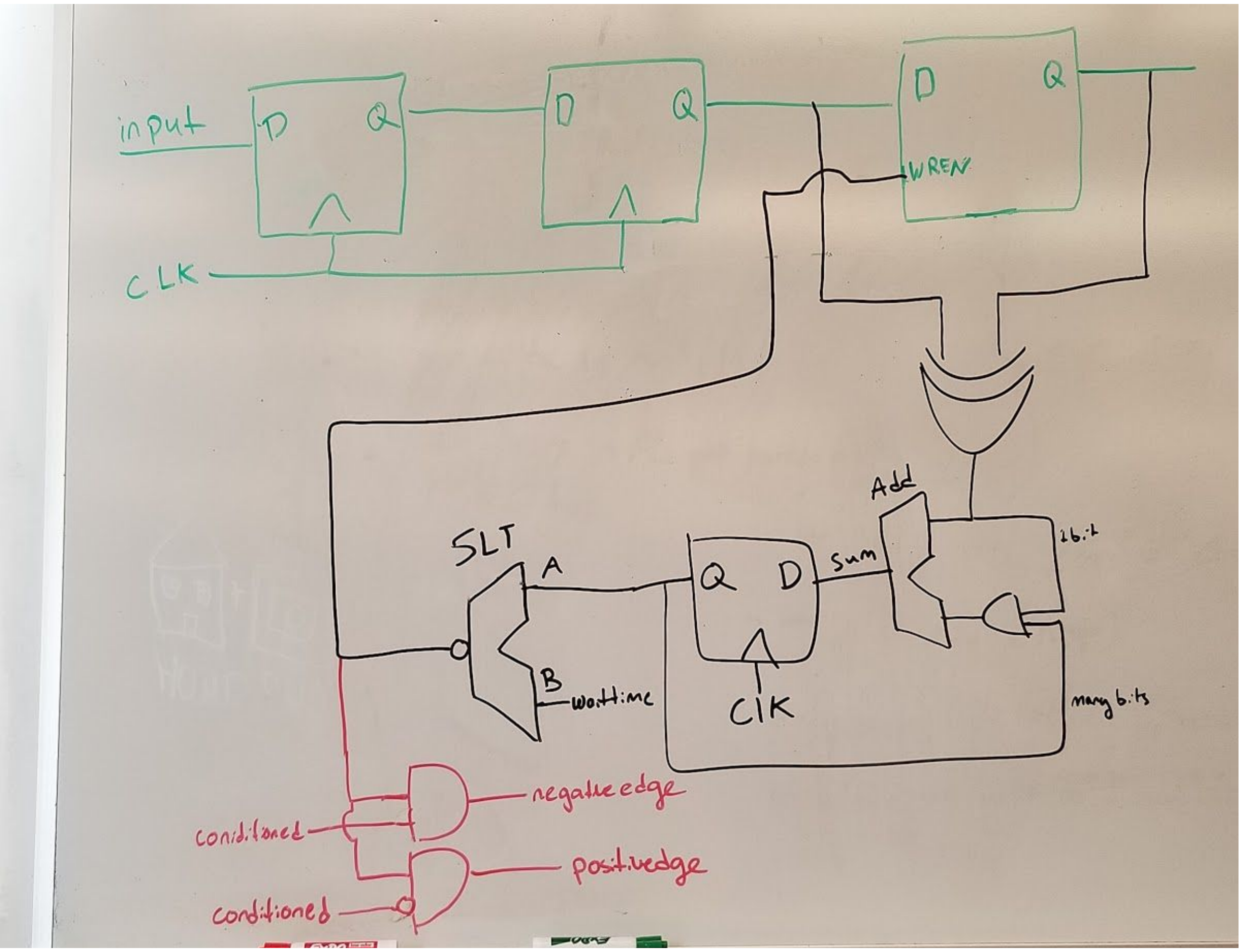
Lab 2 Writeup

Coleman Ellis & Paige Pfenninger

Input Conditioner

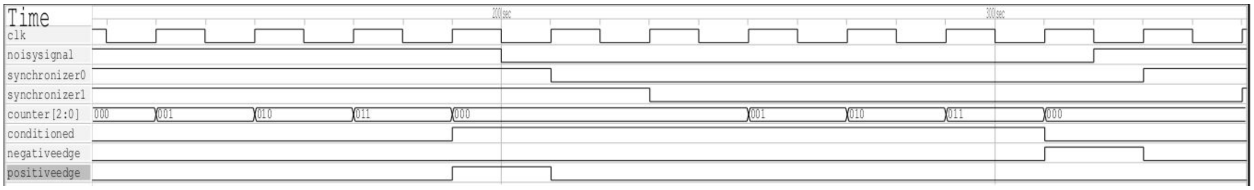
1. Circuit Diagram



The first two d flip flops propagate the input when the positive edge of the clock triggers. The when conditioned (the signal where the green and black lines connect on the far right) is different from the signal coming out of the last d flip-flop, the adder begins adding one to its current sum every clock cycle. When the sum is no longer less than the wait time, the final latch triggers and the conditioned value is then assigned to the input value. The negative and positive edges are set if the sum is no longer less than the wait time and either conditioned is true or false depending on the value.

We had 3 main tests for the input conditioner. The first test was designed to make sure that the input conditioner changed values after the appropriate number of clock cycles (in this case 6), and didn't change values before that point. The second test was to test debouncing. In this test, the raw input changes values quickly, and we want to make sure that the conditioned value doesn't change. We could have designed this test better because with our current wait times the value of the input at the positive edge is always the same which means that if you wait long enough it will switch. Our test does show that fast changes are not noticed by the input conditioner. The third test is to make sure that positive and negative edge are triggering properly. It makes sure that they both go high and low when needed.

The period of a clock cycle is 20ns for a clock that is running at 50Mhz. In our circuit, the input needs to be steady for six positive edges of the clock in order for the conditioned value to be set to the input. If the value changes right after the positive edge, it can remain steady for just shy of 6 clock cycles, and then change back and the conditioned value won't change. Therefore, the longest glitch that will be suppressed by a waittime of 3 is less than 120ns. With a wait time of 10, the value has to remain steady for 13 clock cycles, so the longest glitch that can be suppressed is less than 260ns.



The waveform shows the input conditioner working. The wait time for this conditioner is 3 clock cycles. Based off of this waveform, everything in the input conditioned appears to be working properly.

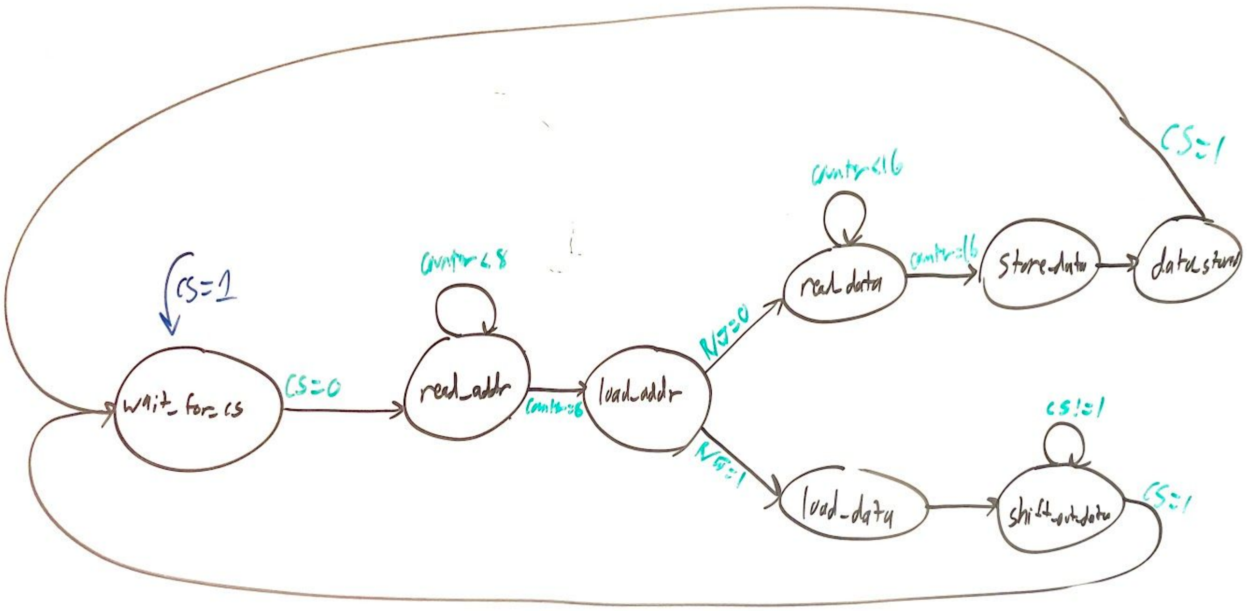
Shift Register

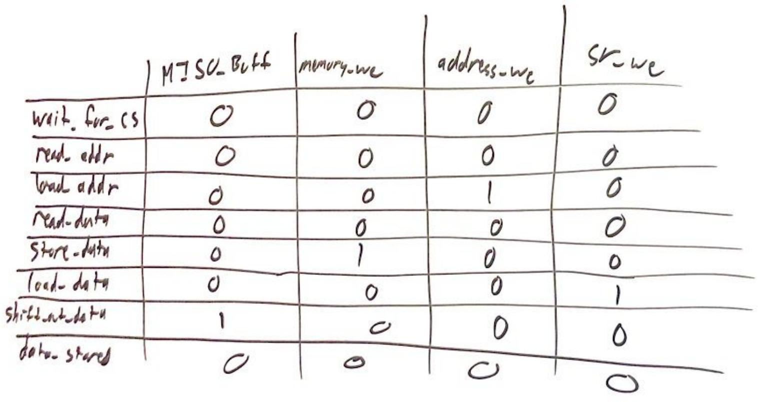
We ran the following tests for our shift register:

1. Load b1010 with the parallel load function, then confirm that the register's parallel_out was correct
2. Make sure that the register's serial out was b1 (i.e. the most significant bit)
3. Shift forward one clock cycle and shift in a b0, then confirm the parallel output is b0100
4. Ensure that if parallel load and peripheral clock are asserted on the same clock cycle, that parallel load takes priority.

We also tested this with both 4 bit (which is the current version of the test bench) and 8 bit shift registers. We felt these test cases covered the entirety of the normal operation of the shift register.

Finite State Machine





	MISO_Buff	memory_we	address_we	sr_we
wait_for_cs	0	0	0	0
read_addr	0	0	0	0
load_addr	0	0	1	0
read_data	0	0	0	0
store_data	0	1	0	0
load_data	0	0	0	1
shift_out_data	1	0	0	0
data_stored	0	0	0	0

Not pictured in this diagram is the “counter” that also exists in the fsm module. It counts the number of rising sclk edges, is used as a condition for transitioning between states, and is reset to 0 every clock cycle while in the wait_for_cs state.

At any point in time, if cs is set to 1, the state gets reset to wait_for_cs.

Memory Module Architecture

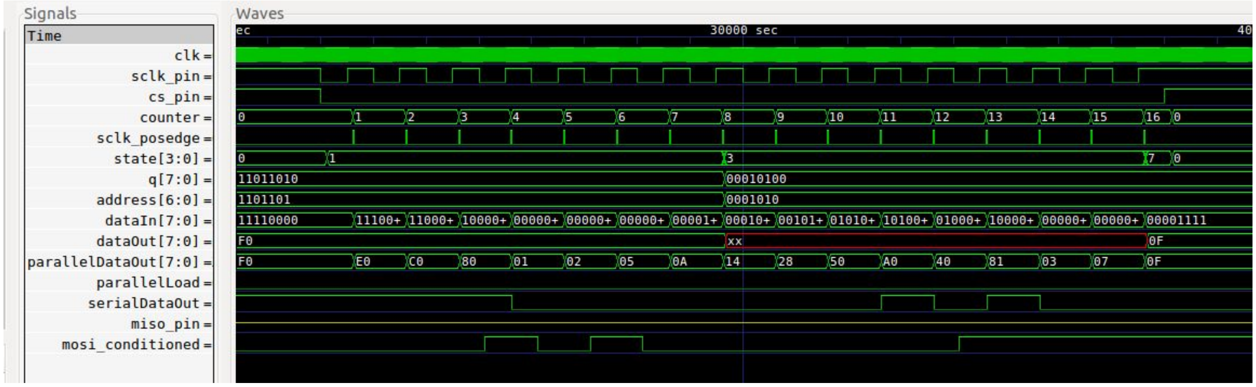
We modeled our memory module on the example provided in the Lab 2 document. It’s almost identical, except that we combined the dff and tri-state buffer for miso into one module (dff_buffer.v).

Testing in Software

Our main testing in software only has two different tests. Each of these tests have several subtests that will print out an error if anything is incorrect. Our first test just writes to two memory addresses and then reads the values stored in those memory addresses. During the reading part, if any of the read values are incorrect or if MOSI is being driven at any point the test outputs an error.

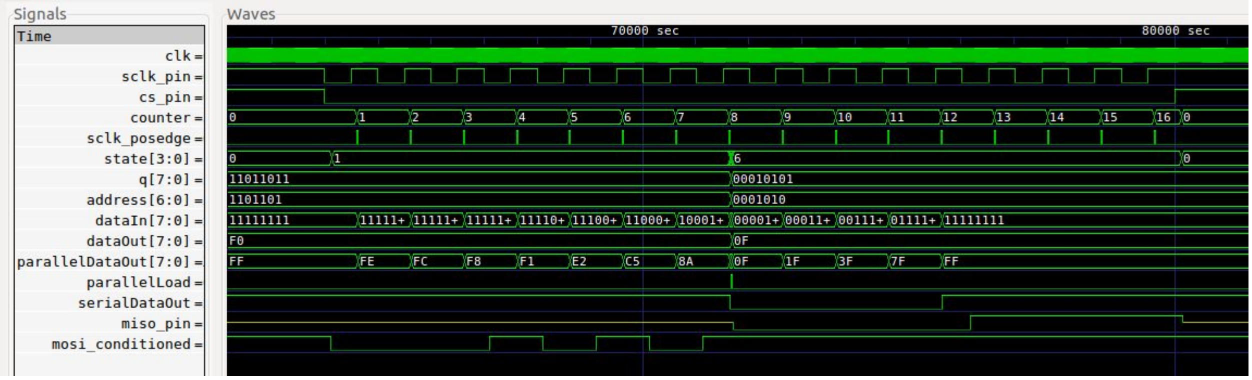
The second test is designed to make sure that the chip select line works. It attempts to write when chip select is still high. The test then reads the value from the register it attempted to write to and makes sure that the register didn't change.

Write



Here’s an example of a standard write operation. Cs_pin is pulled low and sclk_pin starts to clock in pulses. As sclk’s positive edges are reported by the conditioner, the fsm’s counter increases, and its state transitions into waiting for the first 8 bits to have been shifted in. Once 8 bits have, the address register’s q is rewritten to those bits, and the address is changed to the one that was shifted in. Since the operation is a read as mosi is low for the 8th bit, the fsm waits until the rest of the data has been shifted in, then writes the data to memory (dataOut gets changed to match dataIn). Then, cs goes high and the fsm’s counter and state return to 0.

Read



Here's an example of the corresponding read operation. Again, cs is pulled low and sclk begins clocking in pulses. The fsm counts pulses and waits for the 8th bit to be shifted in. Then, once the address register is changed, the data is loaded from memory into the shift register (parallelDataOut changes to match dataOut, with a parallelLoad pulse). Then, miso is taken out of tri-state and shifts out the data from memory. Once all 8 bits have been shifted out, cs goes high, the fsm's counter and state return to 0, and miso is tri-stated once again.

Work Plan Analysis

Everything in this lab took a lot less time than we thought it would and because of that, we were able to complete everything by the dates we set in the beginning. We budgeted many hours for the initial parts of the lab (input conditioner and shift register), but each part only took about two hours including making the test bench. The second part of the lab also took a lot less time than we thought. We each spent about two hours on writing the test bench and putting the pieces together. Because we were able to do things so quickly, we were able to spend a few hours trying to connect our spi memory to an Arduino (we spent ~2 hours attempting to get the fpga board to interface with an Arduino, but were not successful. We reached a point in debugging where we would have to write different images to change the output tied to the leds, and since each new image takes ~10 minutes to write, we decided to give up for the sake of time). The only thing that we did budget the right amount of time for was the final writeup.