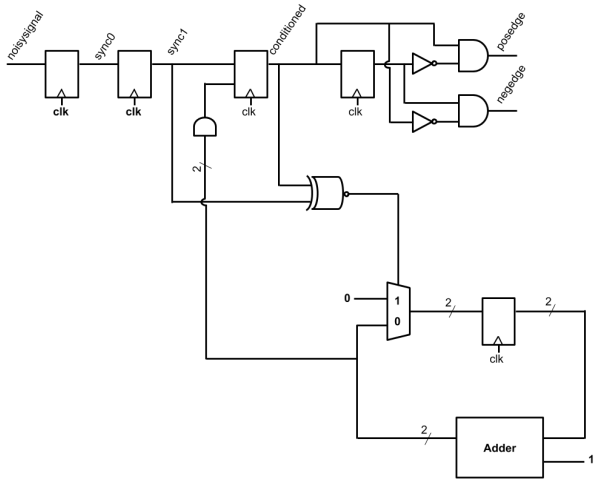
Lab 2: SPI Memory

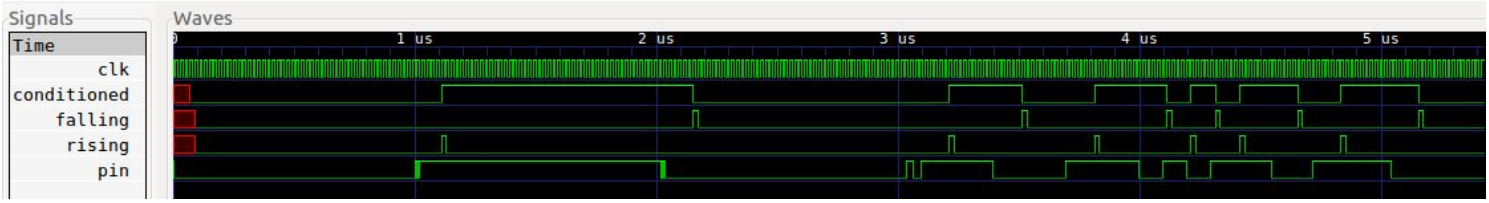
Annie Kroo and March Saper

Input Conditioner Structural Circuit

Our implementation of our input conditioner had both debouncing and conditioning behaviors that we accounted for with the following design.



Input Conditioner Testbench



The image above shows the three tests that we performed to verify the input conditioner. The first set of tests verified that we would filter noisy signals. We wanted to see if the conditioned signal switched during periods of noise or didn't switch (when it should) after the noise subsided. The second set of tests verified edge detection. We wanted to see if the conditioned signal switched without the positive or negative edge detections being high or vice versa. This would have indicated edge detection failure. The third set of tests verified synchronization. We wanted to see if the positive or negative edges ever indicated high while the clock was actually low as this would have indicated we were not synchronizing correctly.

If the main system clock is running at 50MHz, the maximum input glitch that will be suppressed with a waittime of 10 is 2e-7 seconds.

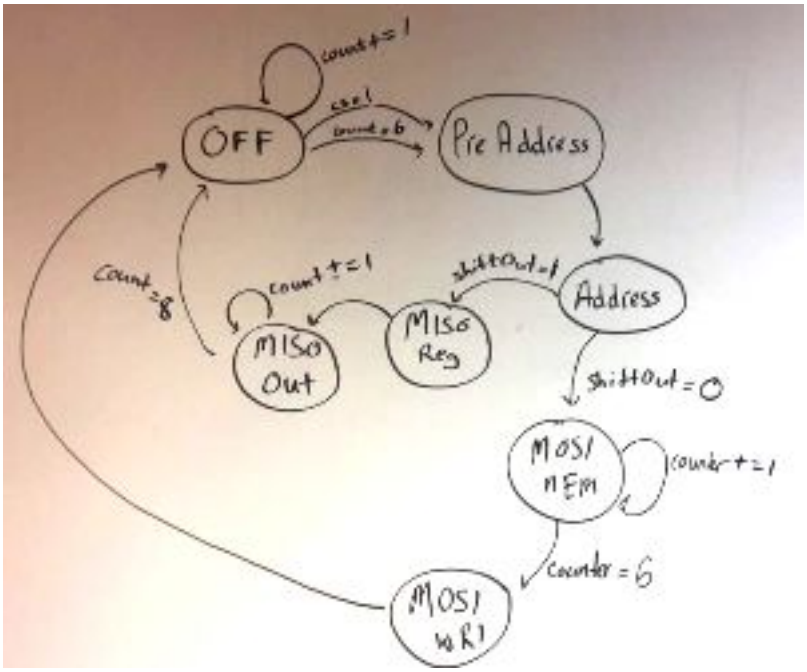
Shiftregister Testbench

To test the shift register we used a test bench that utilized 8 tests. Each of these tests were designed to test a specific functionality of the shift register.

Test Number and Description	parallel-Load	parallel-DataIn	serial-DataIn	serial-DataOut	parallel-DataOut
1) Serial Data in, parallel and serial data out where serial data out equals 1	0	00010000	10001111	1	10001111
2) Serial Data in, parallel and serial data out where serial data out equals 0	0	00010000	01100011	0	01100011
3) Parallel Data in, parallel and serial data out where serial data out equals 0	1	00010000	11111111	0	00010000
4) Parallel Data in, parallel and serial data out where serial data out equals 1.	1	10000000	11111111	1	10000000
5) Parallel Data in, all zeros	1	00000000	11111111	0	00000000
6) Serial Data in, all zeros	0	01100101	00000000	0	00000000
7) Parallel Data in, all ones	1	11111111	00000000	1	11111111
8) Serial Data in, all ones	0	01100101	11111111	1	11111111

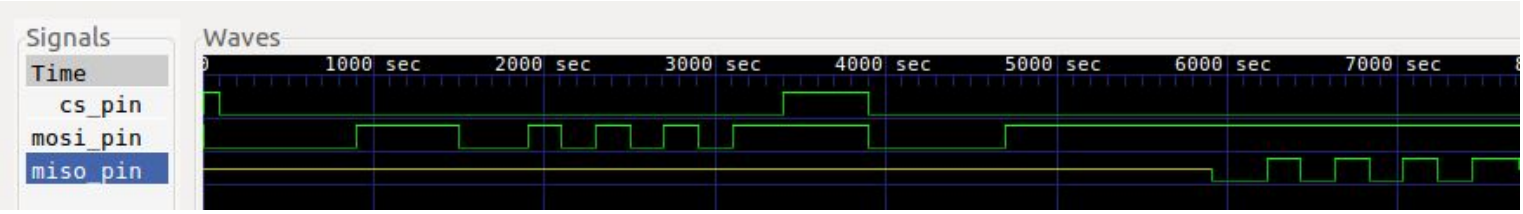
The midpoint check in of loading and implementing the input conditioners and shift register onto an FPGA and testing it provided additional affirmation that our shift register was functional and behaved as we expected.

Finite State Machine



State	add_WE	dm_WE	sr_WE	Miso_Buff
PreAddress	1	0	0	0
Address	0	0	0	0
Miso Reg	0	0	1	0
Mosi mem	0	0	0	0
Miso out	0	0	0	1
Mosi write	0	1	0	0
Off	0	0	0	0

SPI Memory



We verified our SPI Memory by writing to and reading from the address 0001111. We successfully wrote and read the value 01010101. We chose this test as it allowed us to verify that we could pass both zeros and 1s in and was asymmetric so ensured that we were getting out the bits that we put in in the way that we intended. To test this we used a verilog test bench to toggle the clock and drive a signal into our SPI module. Through the use of GTKwave and print statements we were able to verify the functionality of our SPI memory module. In the GTKwaveform above one can see when the chip select pin is enabled, the MOSI pin begins to input the bits to be saved to our data memory. The first bits input are the address. The address is written in at which time our system checks if it is to go into MISO or MOSI mode and reacts accordingly. In the first portion of the test (up to around 3500 sec), our SPI memory module recognizes that it is saving the data input and writes the input data to the data memory in the location specified by the address. During this entire process the MISO buffer prevents our system from driving the MISO pin so that it can be potentially used by other slaves in the network. To verify that we correctly took in the data and wrote the data to the correct address in data memory, we again input the address but instead go into MISO mode and read out the data that we previously saved to Data Memory[address]. The above waveform affirms that the data input into the system was saved and was readable out of the system. As such our module passed our tests and is shown to work.

Workplan Reflection

We began by following our work plan closely and got our midpoint check in submitted on time to our deadlines. Unfortunately from here it derailed and we did not follow our workplan at all. It was difficult to catch back up after that. Although we had all of the subcomponents done on time according to our work plan we massively underestimated the amount of time debugging would take and ended up taking many many times longer to do so than we accounted for in our work plan.