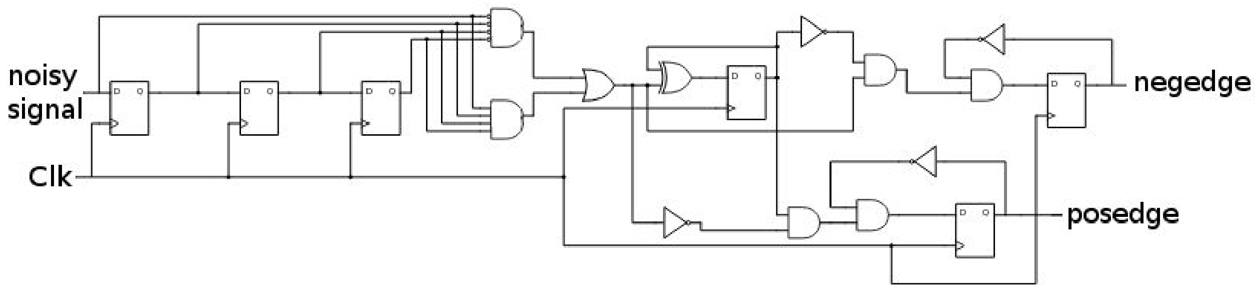
Lab 2: SPI memory

Anna Buchele, Jee Hyun Kim, Apurva Raman

Input Conditioner structural circuit



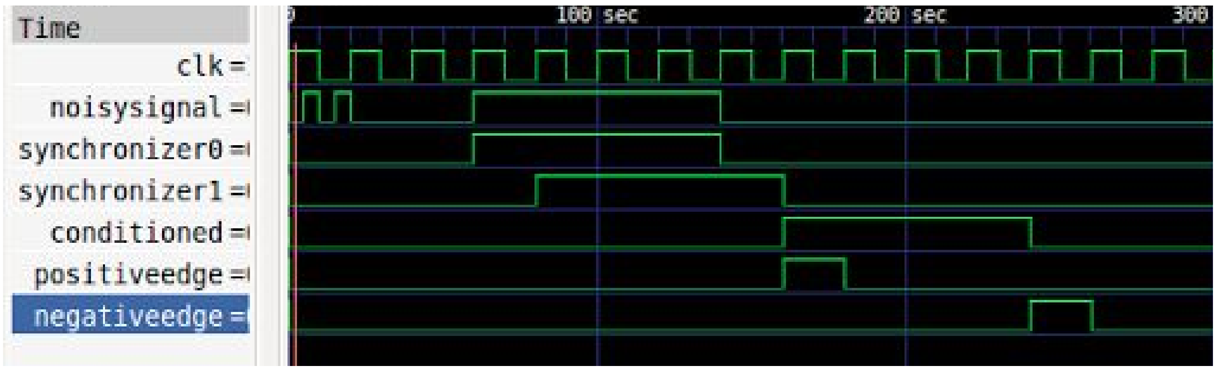
The first segment of the input conditioner (with the first 3 D-Flip Flops) accounts for the wait time. If the output values are all 0 (true for the And gate with the inverters) or all 1 (true for the and gate without inverters), they continue to the logic afterward. The next segment is the toggle, or T-Flip Flop, which we constructed with an XOR gate and a D-Flip Flop. From there we used D-Flip Flops and inverted the relevant signals to check if it was a negative or positive edge.

Input Conditioner timing analysis

If the clock runs at 50 MHz, the period is 20 ns (for each cycle). For a wait time of 10, which is 10 clock cycles, the minimum length of a signal for it to be considered valid and not a glitch is 200 ns. Therefore, the longest input glitch length is 199 ns.

Input Conditioner testing

To test the input conditioner, we used GTKwave to see if noise in the original noisysignal was not present in the conditioned signal. We also checked that the positive and negative edge detection was working as expected. To do this, we generated a signal that was noisy in the beginning and stable for a while afterward and checked that the waveform matched the expected behavior.

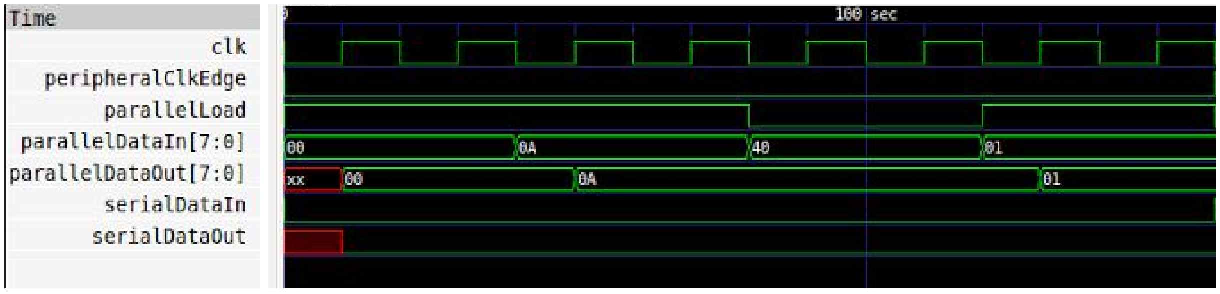


Shift Register test bench strategy

For our test bench, we tested all possible representative cases for shift register functions. irst, we made sure that when the inputs were set to zero, the outputs would also be zero. Then, we tested the parallel operation and serial operation by checking if we were getting the right values when the parallel enable was on and when it was off. We repeated this for the series operation.

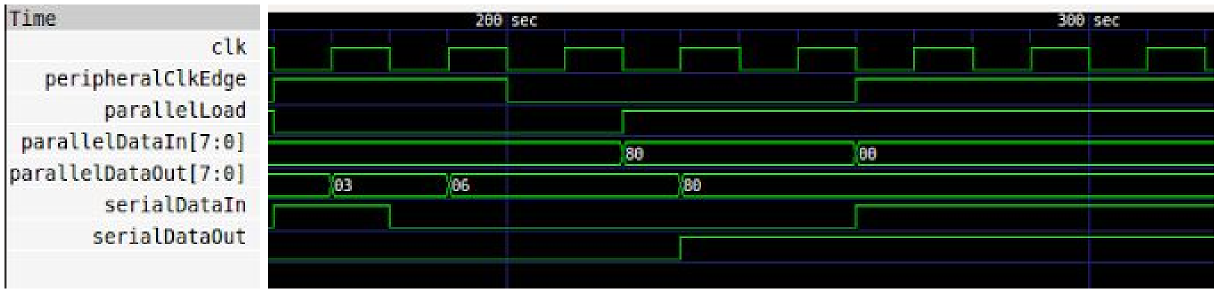
Testing parallel operation:

Then, we tested to see if, when the parallel option was enabled, the parallel output would be equal to the parallel input. Then, we turned the parallel enable off and sent different values to parallelDataIn, and read to make sure that the parallel out was the previous parallel out. Then, we set the parallel option to true, and input 1 to the parallel, making sure we received 1 from the output and also setting up for the next portion of testing. This period of testing can be seen in the waveform below.



Testing serial operation:

Next, we tested the serial operation with similar strategy as the parallel operation. With parallelDataOut set to 1, we put serialDataIn up for one clock cycle, and checked to make sure we recieved an output of 3 (11b), and a serialDataOut of 0, since serialOut should be the most significant bit. Then, with serialDataIn set to 0, we got an output of 6 (110b). Then, we put set parallelData to true, and put in 128d. We received 128d, so we knew it was working. Then, we set serialDataIn to true, and checked to make sure we were getting 128d from parallelOut, and also 1 from serialOut (since serialOut should be the most significant bit). With this series of tests, we were able to make sure that each operation of our shift register was working properly. This second half of testing can be seen in the waveform below.



By design choice, we chose to ignore the case when both serial and parallel enable were switched on. We set both to be enabled and checked that there was no change in the outputs.

SPI Memory:

SPI Memory testing strategy
First, we learned how spi timing work using the timing diagram. Then, we created data variables and address variables for testing. The address variable was divided into address_read and address_write variables for clarity.

For SPI memory testing, we wanted to test

1. If we could write different data to a different addresses in the data memory and retrieve the corresponding data properly
2. If we could overwrite data for the same address
3. If we can get the same data repeatedly if we gave in the same address repeatedly

Starting out, we wrote a data to an address and read from the same address to see if we got the same data back. Then, we wrote two different data to two different address and confirmed that reading data from the specific addresses gave the data we wrote in.

To see if we could overwrite different data into a address already containing a data, we wrote data1 to address1 and then wrote data2 to address1. Then, we checked by reading the data from address1 and confirming that the output data was data2 and not data1.

Then, we tried reading from same address twice to see if it works.
We were not able to get working results from our testing. Our FPGA stopped working unexpectedly and we were not able to find another one to use, so we do not have results from that, either. We believe that it is the test bench that is broken, rather than the SPI memory, but we cannot be certain of that. However, individual testing of our modules has shown that they are each working individually, so if there was an error in our program it would be in the SPI memory, and likely to do with timing of the inputs.

Analysis of work plan:
Coming from Lab 1, we knew that the lab was going to take longer than our expectations. Therefore, we intentionally overestimated the time we would spend on each of the subparts and assigned a long time for debugging each components. Initially, we were on schedule and the time we spent matched the work plan. Further into the lab, we encountered a problem in timing the inputs and outputs and it took much longer than expected to debug.