Computer Architecture Lab 2

Joseph Lee, Wilson Tang, and Bryan Werth

November 2017

1 Introduction

For this lab we were tasked with designing and implementing a SPI Memory and creating an automated test harness and use it to verify your memory (and possibly the memories of the other groups in the class). We split the project into it's components and finished a midpoint FPGA implementation + test scenario with the input conditioner and shift register. Then we created the Finite State Machine for controls and linked it to the rest of SPI.

2 Implementation

2.1 Input Conditioner

The input conditioner edge detection functionality was implemented by buffering the conditioned output using a register and checking for disparities between the signals at the input and output of the buffer register. To ensure clarity, we drove the edge detector outputs low for each clock cycle that the input and output lines of the buffer register were the same.

The maximum glitch length that can be handled with a waittime of ten is ten. When there is a new input, the counter initializes and counts on every clock cycle until the waittime is reached before setting the output equal to the input. The glitch can be any length up to the waittime when the output is driven. If there is still a glitch at that point, the output isn't accurate.

We tested the three different functions of the input conditioner by driving the noisy signal input with a series of pulses of varying lengths. Functionality was confirmed by observing the output waveforms, looking for a debouncing, internal clock synchronization, and edge detection.The block diagram of the completed input conditioner is below:

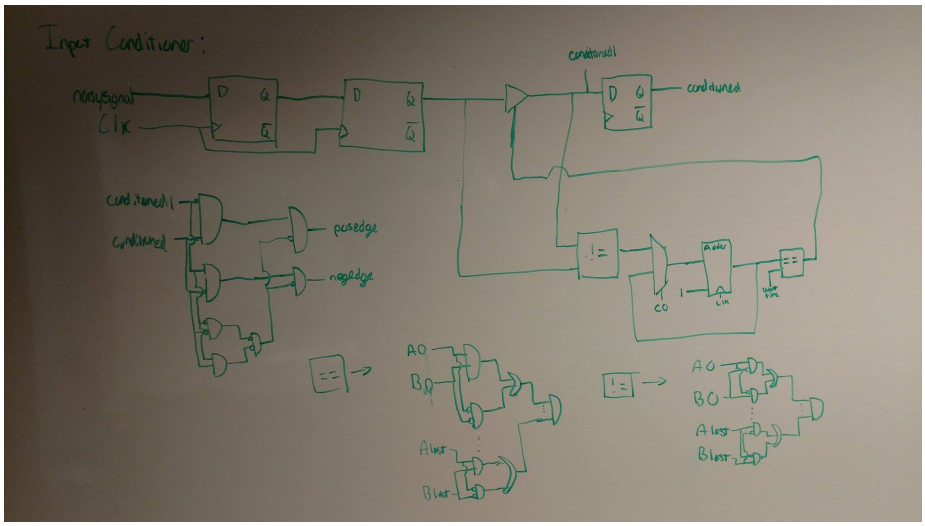

Figure 1: The block diagram for our input conditioner.

2.2 Shift Register

Our implementation of the shift register is fairly straightforward. Essentially at each positive clock edge we check to see if parallelLoad is asserted. If parallelLoad is high then shiftregistermem is loaded with parallelDataIn. If parallelLoad is low then at each peripheralClk edge we shift a new bit into the shiftregister. Serial output is just the bit the gets bumped off the end of the shiftregister each shift and parallelDataOut is simply the current state of the register.

To test the shift register we wrote a simple testbench that generates a clock signal and then does a parallel load followed by a serial load. We simulated our testbench in ModelSim to observe the behavior.

2.3 Finite State Machine

The finite state machine was the last built part of the system. As such, by the time we were ready to test this block, we knew everything else was working. Considering this, we tested the SPI slave subsystem as a whole instead of testing the fsm on its own. First we executed a write operation. Following this, we executed a read operation at the address of the initial write operation. Finally, we performed a read operation at an address that had not been written to to confirm that the data memory was not pre-written for some reason. When we received the outputs that we expected, we knew the fsm was working properly.

3 Testing Strategy and Results

To Test our completed SPI Memory we built a test bench that would simulate the proper clock signals and input an appropriate signal (16 bits for Write and 8 bits for Read) into our spimemory.v. We tested the basic capabilties with the following sequence: Write at Address 7'b0000001 with Data: 8'b11111111

Read at Address: 7'b0000001 Check that the miso output at each time step equals the appropriate value we should've written to it with the previous write.

Then we Read at a different Address: . Confirmed that we did not get any data written to it and that it was undefined data.

Finally we tested the cs pin by raising it high in the middle of a write operation then reading it. Write at Address: 7'b0000010 with Data: 8'b11111111.

Read at Address: 7'b0000010

We can see that the data is only written/defined up to a certain point and the rest is tri-stated.

4 Reflection

Our work plan was generally pretty accurate at estimating how much time everything would take. Our input conditioner took about an hour and a half to write, compared to the 2 hours that we predicted it would take. Our shift register took about 2 hours to write, which is how long we expected it to take. Implementation of the midpoint testing on the FPGA took considerably faster than we expected. We expected to spend 2 hours on it, but spent around an hour because it worked on the first try. The testbenches for the input conditioner and shift register took about an hour and a half for each, which was a little longer than we anticipated, but not significantly longer. Our SPI memory took about 2 hours to write, which is longer than the hour we expected it to take. One of the main reasons that this took longer is that we had a tendency to over-complicate our implementation trying to catch edge cases that we later realized were not part of the specification.

Our SPI memory testing took longer than expected because we didn't have the most efficient debugging strategy. In the future, we will need to be more aware of how long debugging usually takes. The report took about as long as we would have expected it to, about two hours in total.