CompArch Lab 2

Vivien Chen, Lauren Pudvan

October 18, 2018

1 Input Conditioner

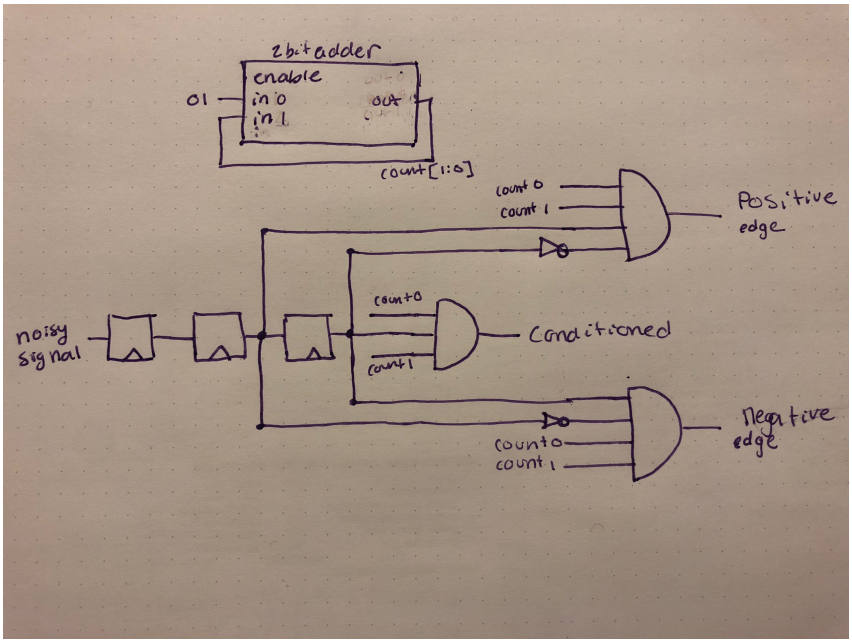

Figure 1: The circuit diagram for the input conditioner. We are utilizing an adder to be our counter. Whenever the output of the adder is 11 we want to allow positive edge, negative edge and conditioned the opportunity to be high. We used flip flops to make sure noisy signals do not get thrugh.

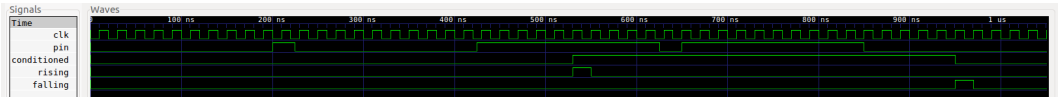

Figure 2: Input conditioner waveform

If the main system clock is running at 50MHz, the maximum length input glitch that will be suppressed by this design for a *waittime* of 10 is 200 ns.

$50MHz = 5 \cdot 10^7 Hz = \frac{1}{5 \cdot 10^7 s^{-1}} = 2 \cdot 10^{-8} s$

$2 \cdot 10^{-8} s \times 10[waittime] = 2 \cdot 10^{-7} s = 200ns$

2 Shift Register Test Bench

We tested each of the following cases:

- peripheralClkEdge is 0 and parallelLoad is 1 (in which the output of the shift register equals to the parallelDataIn value we give it)
- peripheralClkEdge is 1 and parallelLoad is 0 (in which the shift register advances one position
- serialDataIn is loaded into the LSB and the rest of the bits shift up by one)
- peripheralClkEdge is 1 and parallelLoad is 1 (in which the behavior of parallelLoad "wins" out)
- peripheralClkEdge is 0 and parallelLoad is 0 (in which nothing is written to the shift register)

3 Finite State Machine

3.1 Diagram

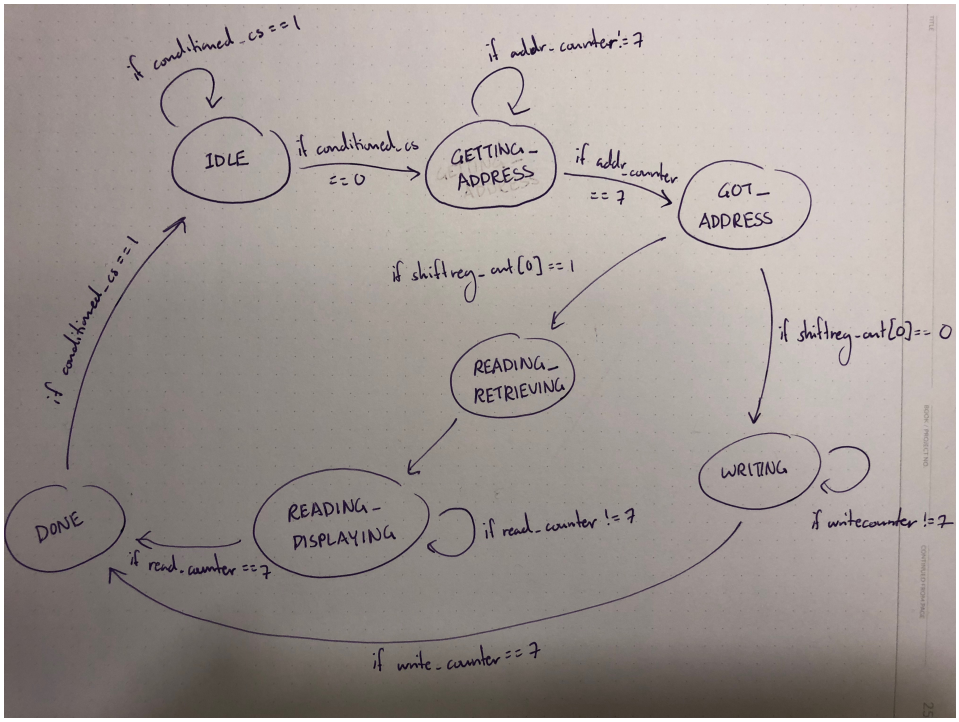

Figure 3: Diagram of the finite state machine.

In the IDLE state nothing is happening and conditioned is high. Then when conditioned goes low, it enters the GETTING_ADDRESS state. Then when the counter is at 7, it enters the GOT_ADDRESS state where it reads the bit to say if it will be reading or writing. If it is WRITING, it will write the 8 bits of data to memory then enter the DONE state. If it is reading it will retrieve the data then display the 7 bits (one at a time) then go to the done state. There is one more clock cycle to read data than to write because it has the additional READING_RETRIEVING state. Once in the DONE state, it will stay there until conditioned goes back to high. The control tables for the FSM is below.

3.2 Control Table

State	MISO_BUFE	SR_WE	DM_WE	ADDR_WE
IDLE	0	0	0	0
GETTING_ADDRESS	0	0	0	0
GOT_ADDRESS	0	0	0	1
WRITING	0	0	1	0
READING_RETRIEVING	0	1	0	0
READING_DISPLAYING	1	0	0	0
DONE	0	0	0	0

4 SPI Memory Testing

We start by initializing chip select to 1 (false) and the MOSI pin to 0 while simulating one clock cycle. We check that the state is IDLE. From here on, we simulate a clock cycle after every CS or MOSI pin entry.

We tested a good write, a good read, a CS high write, and a CS high read. The formatting of the tests are largely the same.

For the good write, we first set CS to 0 (true) for the SPI to begin receiving the address through the MOSI pin. After entering the first 7 bits, which comprise the actual address, we check that the state is GETTING_ADDRESS. The 8th bit determines R/W (read or write). Here, we check that the state is now GOT_ADDRESS, the address is as expected, and R/W is 0 or write. Then we store the 8 bits of data and check that the state is WRITING and that the data memory is as expected. Then we set CS to 1 (false), check that the state is DONE after one clock cycle and then IDLE after another.

For the good read, we again set CS to 0 and check the address states, address, and R/W as we enter the address, which is the same address that was written to. *Due to the structure of our FSM, we have to wait a clock cycle before reading data.* We check after this clock cycle that the state is READING_RETRIEVING. Then we read the MISO pin after each subsequent clock cycle for 8 cycles and check that the data matches what was written. After these 8 bits are displayed, we check that the state is READING_DISPLAYING before setting CS to 1 and checking that the state is DONE, then IDLE.

We tested a write and read case that was identical to the ones previously de-

scribed with one exception, we kept the chip select high. This meant that no matter what else happens to the inputs the state stayed in IDLE.

This test bench covers reading and writing when CS is 0 and CS is 1. While this far from exhaustively tests everything, it tests the overall functionality and the different states.

5 Work Plan Reflection

Below is the expected work plan next to the actual times:

Input Conditioning	Expected Time	Actual Time
Complete the module	1.5 hours	1 hour
Test Bench	0.5 hour	0.5 hour
Debugging	1 hour	1 hour
Circuit Diagram	0.5 hour	1.5 hour
Time Calculations	1 hour	0.5 hour

Shift Register	Expected Time	Actual Time
Complete the module	2 hours	1 hour
Test Bench	0.5 hour	0.5 hour
Debugging	1 hour	0.5 hour
Report Writing	0.5 hour	0.25 hour

Midpoint FPGA	Expected Time	Actual Time
Write Midpint	0	0.75 hour
Load to FPGA	0.5 hour	0.75 hour
Testing by hand	0.5 hour	0.75 hour
Submitting	0	0.5 hour

SPI Memory	Expected Time	Actual Time
Making SPI Memory	0	1.5 hours
Draw finite state machine	2 hours	2 hours
Implement Design	1 hour	0.5 hours
Test Bench	1 hour	1 hours
Debugging	1 hour	2.5 hours
Writing Report	2 hours	3 hours

We failed to identify some things that we needed to complete that could be time sinks. For example writing the midpoint module, submitting our midpoint check-in, and making the SPI memory module. These took a lot more time than we expected, so overlooking them worked against us.

For the input conditioner, we were faster than expected for some sections and slower for others. Overall we were within 1 hour of all of our estimations. Also, this section took the exact amount of total time that we planned for it! We think this section was scoped out pretty well.

For the Shift register, everything we did was on time or under time. For this section it took 1.75 hours less than we expected.

For the midpoint, we went way over time on every section. We took 1.75 hours more than planned to do this part of the lab.

For the SPI memory, most tasks took the expected time or went over. Overall, for this section, we went 3.5 hours over the expected time. This is the worst scoped section. We learned that we should plan for things to take more time debugging when it is a more complicated file.