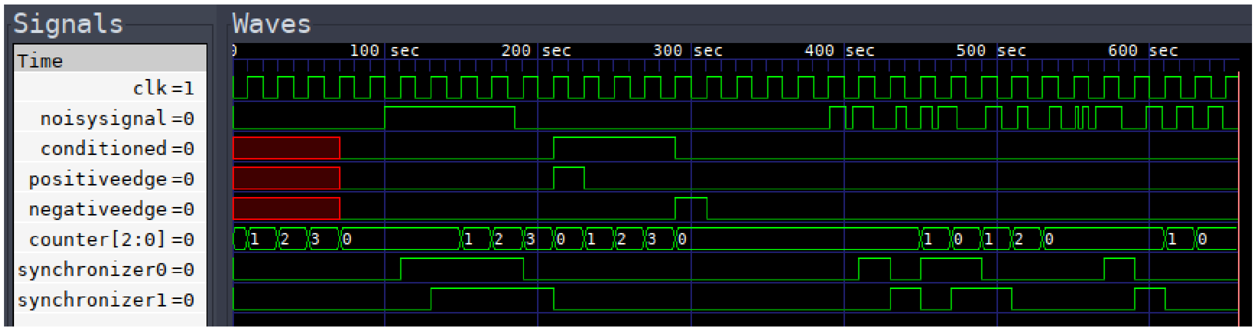
CompArch Lab 2: SPI Memory

Adam Selker, Nathan Yee

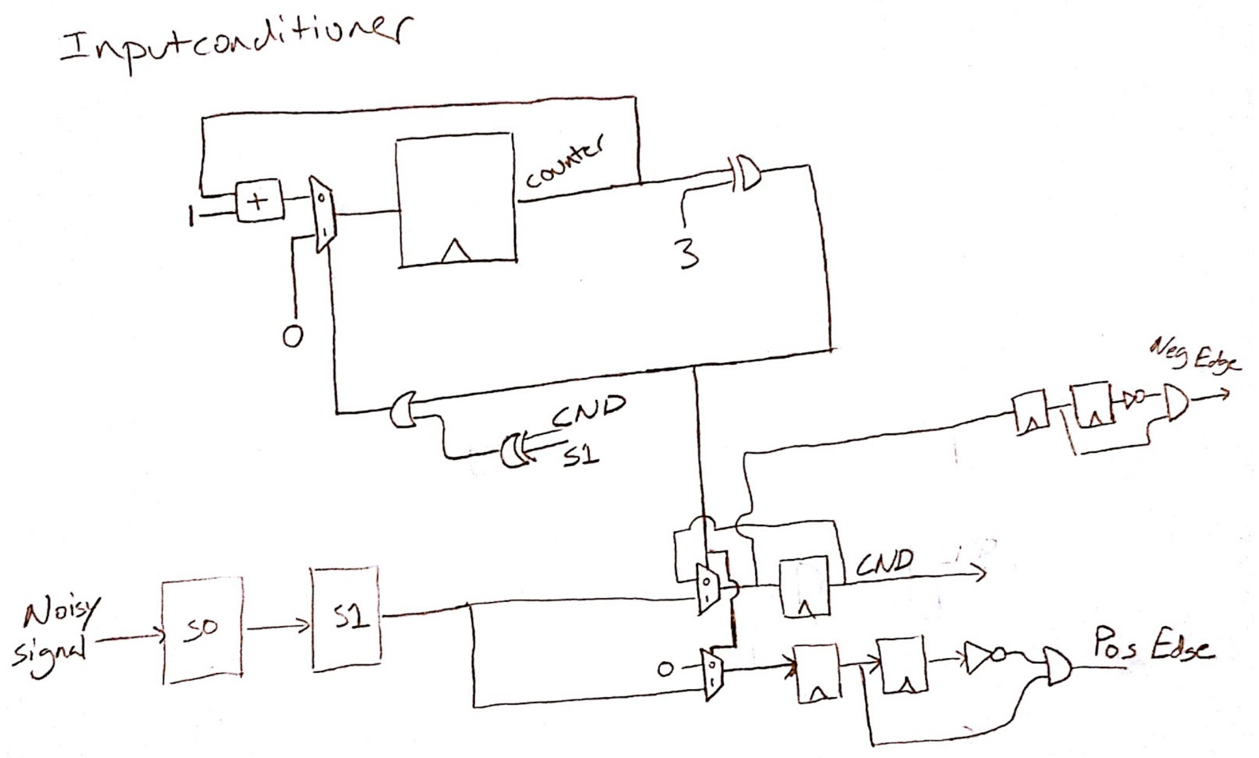
Input Conditioner



GTKWave output from the input conditioner test bench. The waves from 100 to 310 demonstrate input synchronization and edge detection. The noisysignal turns on at 100 and turns off at 185 - for a total of 85 units of time. The conditioned signal generated from this noisy signal turns on at 210 and turns off at 290 - for a total of 80 units of time. This slightly shortened duration showcases signal synchronization to the clock of the internal domain.

Positive edge detection is seen at 210 when the positiveedge turns on with the conditioned signal. Negative edge detection is seen at 290 when the conditioned signal turns off and the negativeedge turns on for a single clock cycle.

Input debouncing is tested between 390 and 650. The noisysignal is rapidly turned on and off for various short durations of time. Even though synchronizer0 and synchronizer1 pick up on the noisy signals, they do not stay on for long enough such that the conditioned signal remains off.



A possible structural diagram for the input conditioner based on our behavioral verilog. S0 and S1 synchronize the noisysignal to the internal clock domain. The counter determines the amount of time the signal must remain one value. In this diagram, the counter must reach the value of 3.

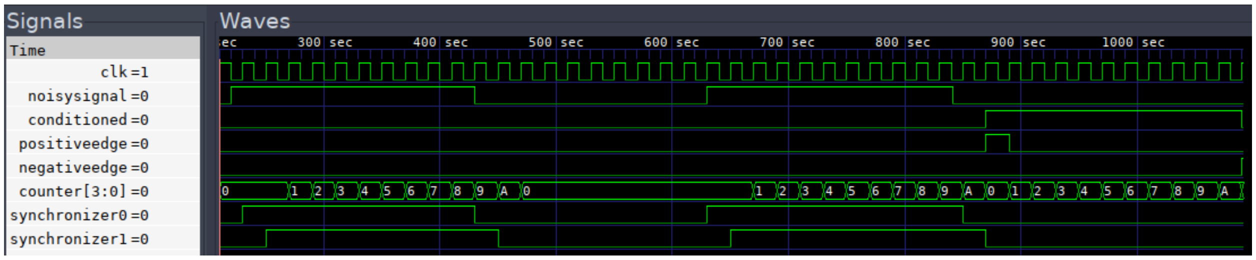


Figure 3. Maximal length suppression when WAITTIME = 10

WAITTIME controls how many cycles the input conditioner waits after a rising or falling edge before passing the signal on to its outputs. If WAITTIME = 10, and the system clock runs at 50 MHz, the maximum length glitch that will be suppressed is 210 ns. We verify this in figure 3, as the noisy signal activates at time 220 for 210 ns and the conditioned signal never gets activated. However at 630, the noisy signal activates for 211 ns and the conditioned output is activated.

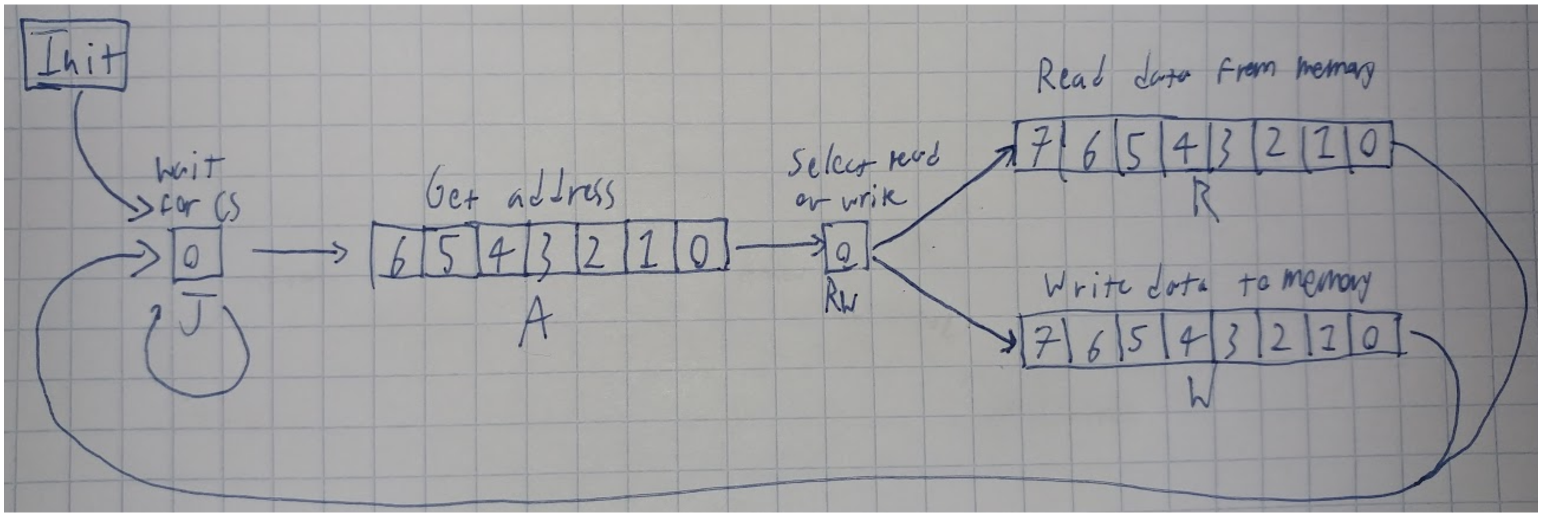
Shift Register

To test the shift register, we built a simple test bench which tested each of the four key behaviors: serial in, serial out, parallel in, and parallel out. It did this in a few steps:

- Load all 0’s in parallel, to put the SR in a known state.
- Load the pattern 0b10101010 in series.
- Check the parallel output.
- Load the pattern 0b01010101 in parallel.
- Check the parallel output.
- Shift by one, and check both outputs.

Midpoint Check In video https://photos.app.goo.gl/s4GjNxpAkXamRoHh6

Finite State Machine



The states are grouped into five sets: J, A, Rw, R, and W.

- J is the “default” state, where the memory waits for the CS pin to be pulled low. This is also where the finite-state machine starts. It has just one sub-state.
- A listens for the address to read from or write to. It has 7 sub-states, corresponding to the 7 bits of the address (the memory has a depth of 7 bits).
- Rw chooses whether to read or to write. It has just one sub-state.
- R reads data from data memory and writes it to the MISO pin. It has 8 sub-states, corresponding to the bits in each register (the memory’s width).

- W writes new data to data memory, from the MOSI pin. It also as 8 sub-states.

All transitions occur on rising edges of the SCK pin. All are unconditional, with the following exceptions:

- If the CS pin is high, then J goes to J. Otherwise, it goes to A6.
- If the MOSI pin is high, then Rw goes to R7. Otherwise, it goes to W7.

The finite-state machine has four outputs:

- miso_en enables the MISO output. When it is low, the MISO pin is put in a high-impedance state. At the moment, this is always high, because this is the only SPI device on its bus, so there is no possibility of jamming other communications.
- dm_we enables writing to the data memory. When it is high, the data memory reads whatever is in the shift register and saves it to the register referred to by the address latch. This is set high for exactly one clock cycle as the FSM transitions from the W state to the J state..
- addr_we enables writing to the address latch. When it is high, the address latch reads the seven low bits in the shift register, saves them, and sends them to the data memory's address port. This pin is set high for exactly one clock cycle after the FSM enters the Rw state, with a one-cycle lag first to let the shift register finish shifting.
- sr_we enables the shift register's parallel input. When it is high, the shift register reads the output of the data memory into its internal memory. It is high for exactly one clock cycle as the FSM transitions from the Rw state to the R state.

We decided to test the "core logic" of the SPI memory separately, to simplify the process. The core logic consists of all parts except for the input conditioners, output D flip-flop, and output tri-state buffer. To test the core logic, we send four SPI signals;

- Write 0x00 to address 0x2A
- Read from address 0x2A, ensure that it is 0x00
- Write 0x22 to address 0x2A
- Read from address 0x2A, ensure that it is 0x22

The choice of an arbitrary register (not 0x00 or 0xFF) should generalize to the rest of memory, since the memory is homogeneous. Testing two different signals in two different cases should ensure that the memory is actually re-writable, and can switch states more than once.