Connor Aksama
EE 371
January 19, 2023
Lab 2 Report

Procedure

This lab is comprised of three tasks. The first task involved creating a 32x4 RAM module, simulating inputs using peripherals.

Task 1

I approached this task by first studying the block diagram given in the lab specification for the task. From this schematic, I extrapolated the input and output ports for the RAM module, then began working to define the internal logic. This involved creating the internal memory array, clocking inputs, and defining write logic.

After this module was developed and tested in ModelSim, a separate top-level module was created to instantiate an instance of the previous RAM module, and to define connections between board peripherals, and HEX displays. These modules were made separate as to make the RAM module more flexible – the control lines for the RAM module are not tied to any particular type of input source.

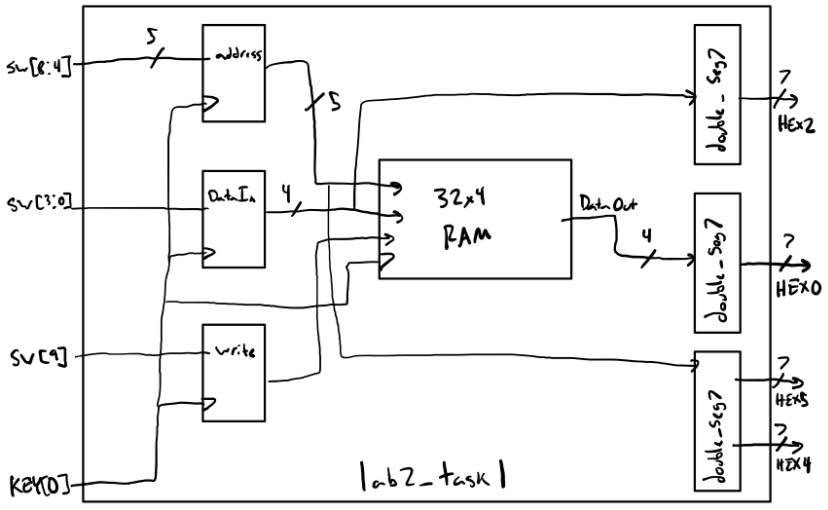

Figure 1. Top-Level Block Diagram for Lab 2 Task 1.

Task 2

I approached this task by first following the directions to create an on-chip, dual-port, 32x4 RAM module in Quartus along with a memory initialization file to provide initial values for instances of this RAM module. I then created a top level module in Verilog to instantiate this RAM module and define connections between board peripherals and displays. Additionally, I included a five-bit counter in this top-level module that would continuously increment on every cycle of a divided clock. This was done to achieve the one second, periodic cycling through of reading each memory address in the RAM module.

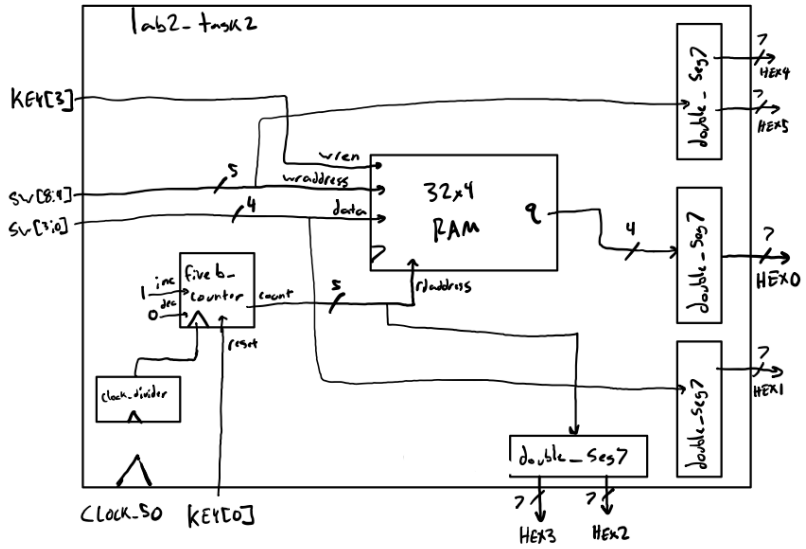

Figure 2. Top-Level Block Diagram for Lab 2 Task 2.

Task 3

I approached this task by first configuring the dual port 16x8 RAM module to use for the buffer. After this, I designed the functionality by which the FIFO modules would keep track of the read and write addresses, which would act as the front and back pointers of the queue. To this end, I implemented both of these as five-bit counters, which would each increment on a read and write operation, respectively. I also decided to include a third five-bit counter, which would keep track of the number of elements in the buffer. This was done to distinguish between whether the buffer was full or empty.

Following this, I created the top level module that would instantiate the FIFO buffer modules and defined connections between peripherals and displays. I also decided to include an extra output signal “error” which would be raised when the user tries to read while empty, or to write while full. This was done to increase the level of feedback given by the system given inputs from the user.

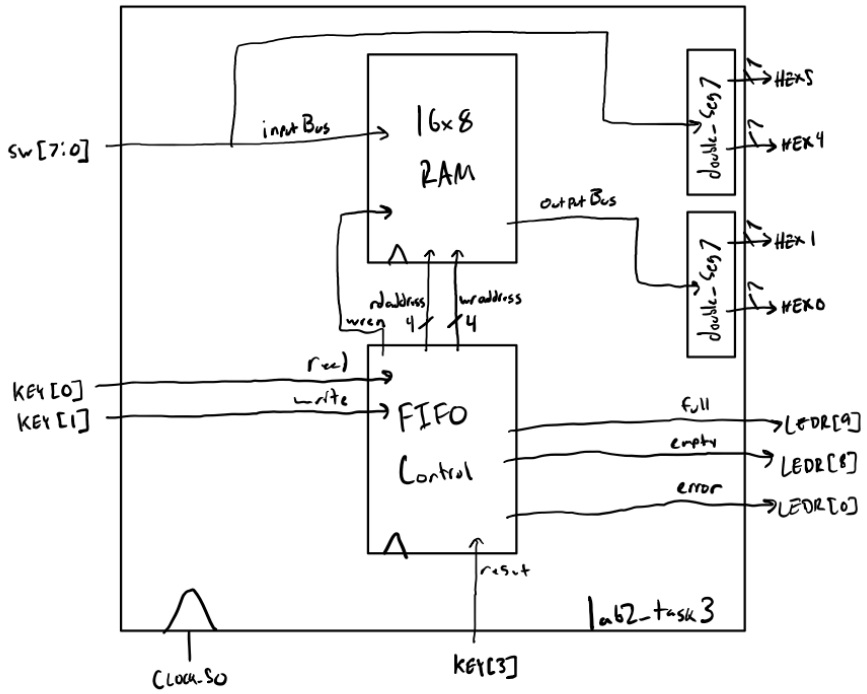

Figure 3. Top-Level Block Diagram for Lab 2 Task 3.

Results

Task 1

The following are screenshots from ModelSim simulations for each module used in the simulated RAM module.

For conciseness, HEX displays are encoded in hexadecimal. Use the following table to convert from hexadecimal code to the corresponding seven segment display:

ModelSim Hexadecimal Code	Seven Segment Display Character
40	0
79	1
24	2
30	3
19	4
12	5
02	6
78	7
00	8
10	9
08	A
03	b
27	c
21	d
06	E
0e	F

Table 1. Conversion between hexadecimal values shown in ModelSim and the character shown on a seven segment display.

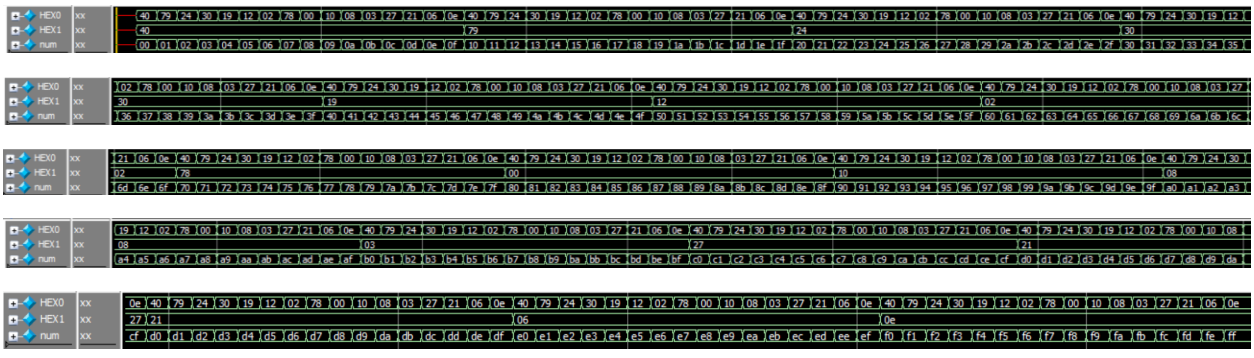

Figure 4. ModelSim Waveform for the Double Seven Segment Display module testbench.

This waveform demonstrates the two digit, hexadecimal display given any number from 0x00 to 0xff. Refer to Table 1 to decode the hexadecimal values to their corresponding characters.

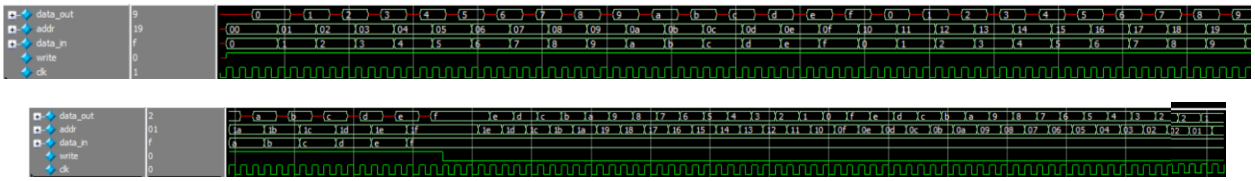

Figure 5. ModelSim Waveform for the 32x4 RAM module testbench.

These screenshots first show the process of writing data to every address in the module. Notice that it takes two clock cycles for the data on “data_in” to appear on the “data_out” line. Next, the waveform shows the process of reading from every address in the module. Notice that the value read from each address matches with the one previously written.

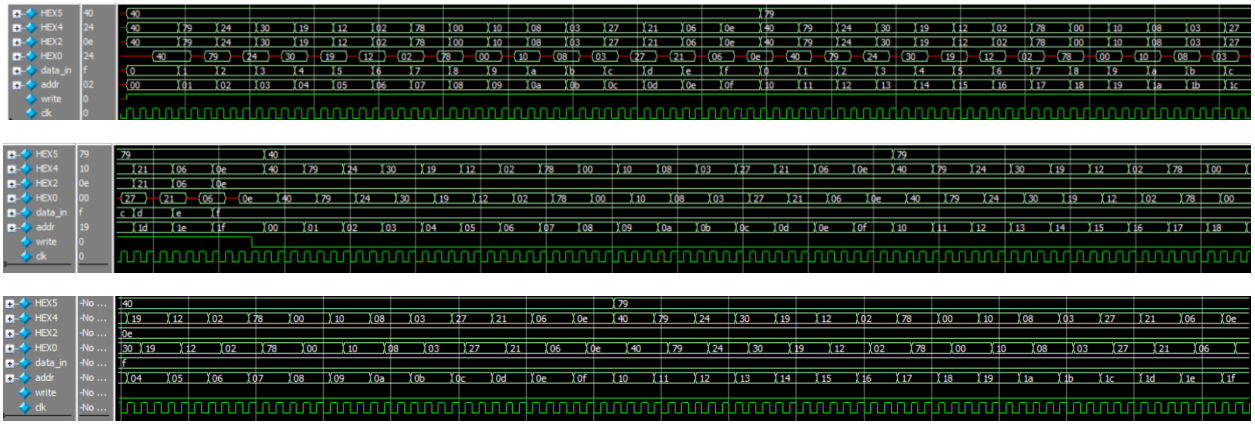

Figure 6. ModelSim Waveform for the Task 1 top-level module testbench.

Similar to Figure 5, these screenshots demonstrate the process of writing to every address in the RAM module, followed by reading from every address in the module. Additionally, the screenshots demonstrate that the correct values for the correct input and output values are shown on the HEX displays. (Refer to Table 1.)

Task 2

Task 2 uses the same Double Seven Segment Display module from Task 1. Refer to Figure 4 for waveform screenshots.

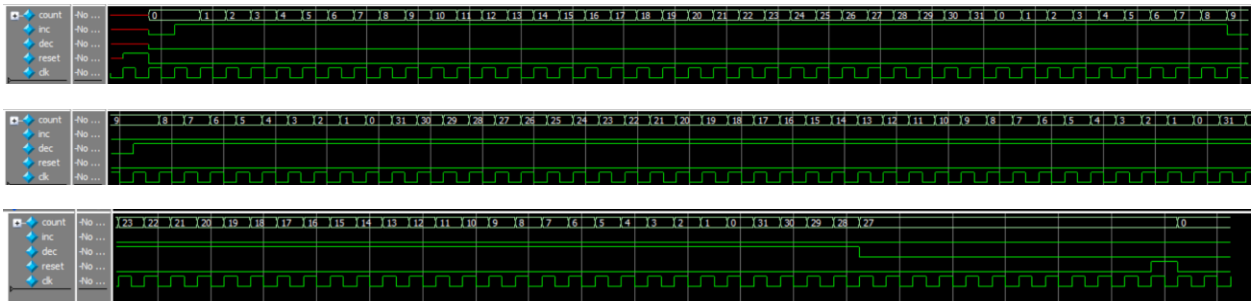

Figure 7. ModelSim Waveform for the five-bit counter module testbench.

These screenshots demonstrate incrementing the counter and overflowing, decrementing the counter and underflowing, holding the counter constant, and resetting the counter to zero.

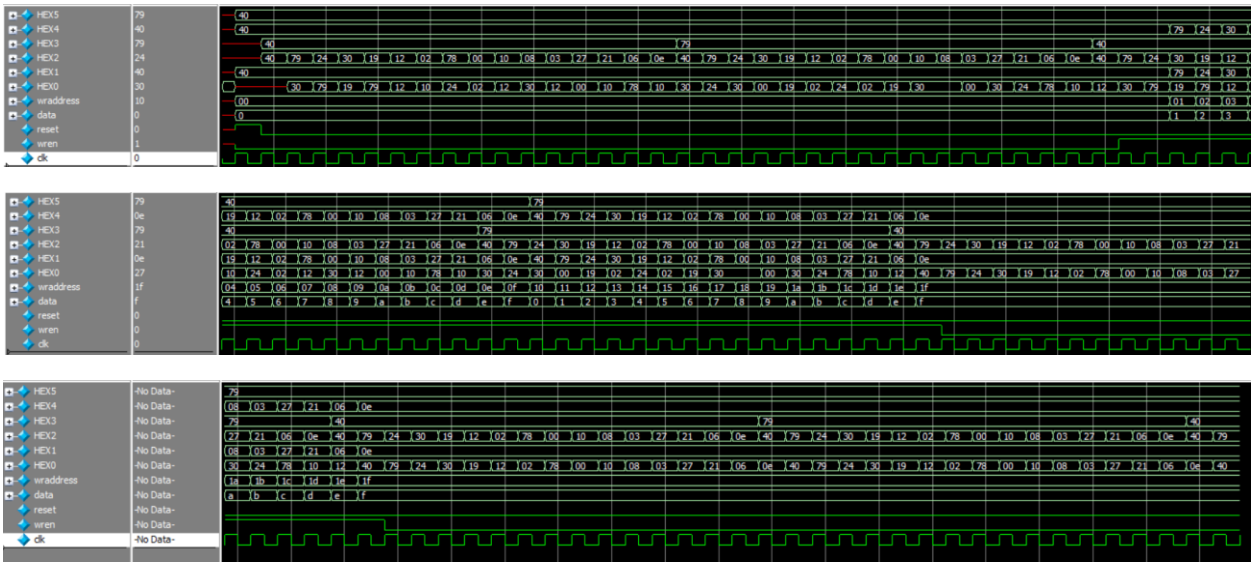

Figure 8. ModelSim Waveform for the Task 2 top-level module testbench.

These screenshots demonstrate the process of reading from every address in the RAM to ensure the correct initial values were loaded, then the process of writing a new value to every address, then reading every value to ensure the correct values were written. Note: some signals are given aliases for better clarity.

Task 3

Task 3 uses the same Double Seven Segment Display module and five-bit counter module from Task 2. Refer to Figure 4 and Figure 7 for waveform screenshots.

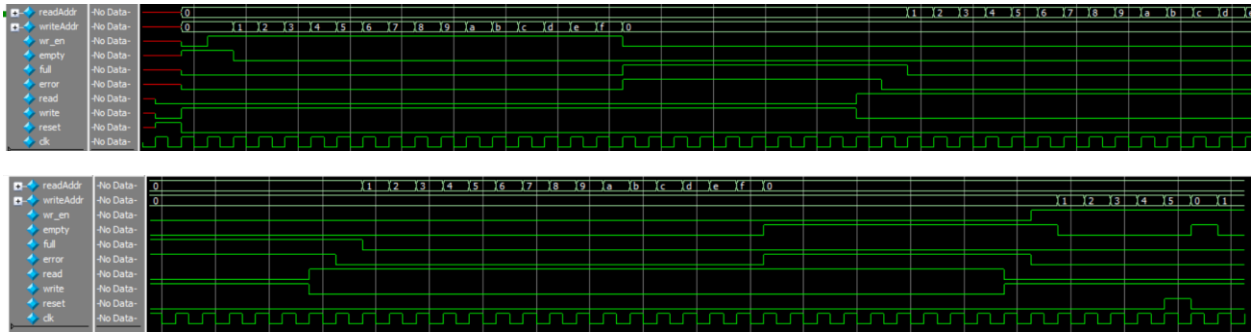

Figure 9. ModelSim Waveform for the FIFO Control module testbench.

These screenshots demonstrate the process of writing into the buffer until full, then attempting to write while the buffer is full. Then the process of reading values from the buffer until empty is shown, followed by attempting to read while the buffer is empty. These screenshots additionally demonstrate that the correct output signals are raised in the correct conditions (i.e. “full” when the buffer is full, “empty” when the buffer is empty, etc.).

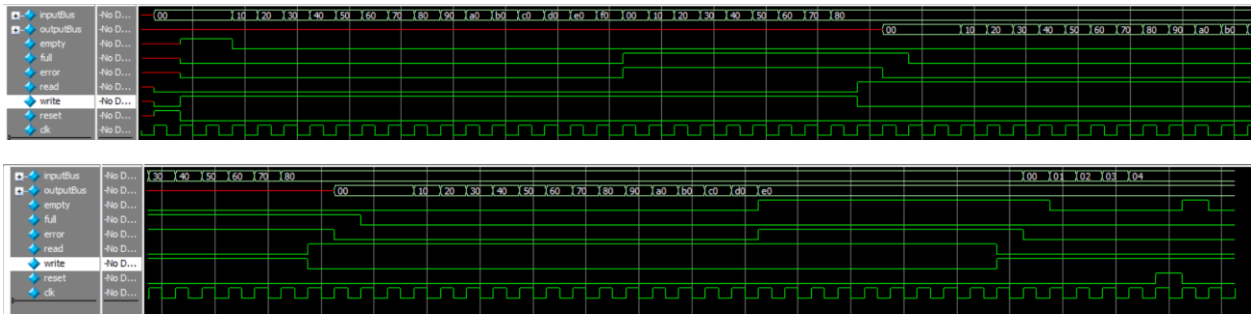

Figure 10. ModelSim Waveform for the FIFO module testbench.

These screenshots demonstrate the process of writing specific values into the buffer until full, then attempting to write while the buffer is full. Then the process of reading those same values from the buffer until empty is shown, followed by attempting to read while the buffer is empty. These screenshots additionally demonstrate that the correct output signals are raised in the correct conditions (i.e. “full” when the buffer is full, “empty” when the buffer is empty, etc.).

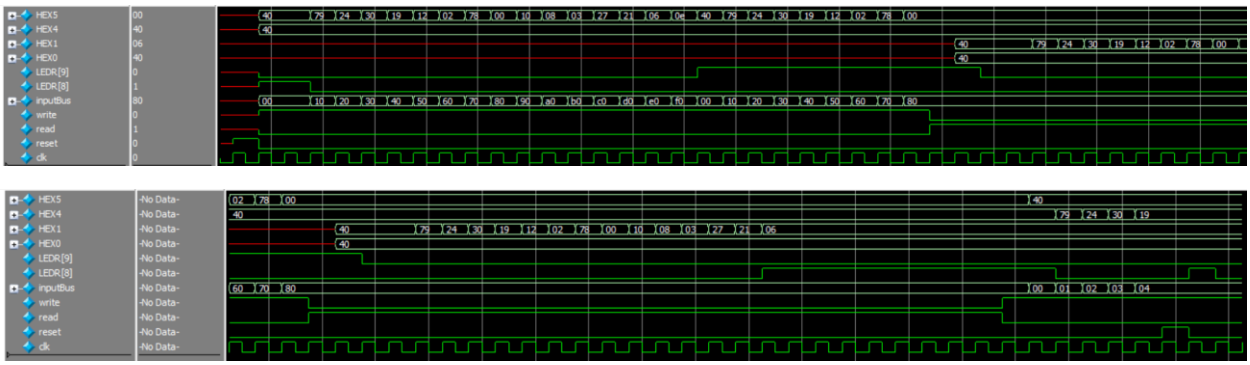

Figure 11. ModelSim Waveform for the Task 3 top-level module testbench.

These screenshots demonstrate the process of writing values into the buffer until full, then attempting to write while the buffer is full. Then the process of reading values from the buffer until empty is shown, followed by attempting to read while the buffer is empty. These screenshots additionally demonstrate that the correct output signals are raised in the correct conditions (i.e. “full” when the buffer is full, “empty” when the buffer is empty, etc.), as well as that the correct displays are shown in the correct conditions (i.e. HEX displays, LEDs, etc.). Note: some input signals are given aliases in these screenshots for better clarity of signal function.

Final Product

The overarching goal of this lab was to gain practice with interacting with RAM modules in various ways. Task 1 necessitated building the RAM module mostly from scratch while still utilizing on chip memory. This task was completed fully in line with the specification. In Task 2, I used a RAM module configured from the IP catalog, then built a lightweight system around it to cycle through its contents and be able to write to a specified address. This was also completed in line with the specification. And in Task 3, I again used a configured RAM module from the IP catalog, but this time to build a more involved FIFO buffer. The results of this task were more open ended, so there were more individual decisions made about timing constraints, edge case behavior, and overall data flow.

Appendix: SystemVerilog Code

(See next page)