Charles Tung Fang
EE 371
2019/10/15
Professor Rania Hussein

Lab 2 Memory Block Report

Procedure

I approached the problem by understanding each task at the very top level. After reading the specs, I examined the ram to make sure I know its input, output, write enable and address relationship. The major components in the system is reading and writing the data. Once I attained task 1, the rest had become easier since they are all further implementation using the same ideology.

Task 2 and task 3 are pretty much identical in terms of implementation.Task 2 is using the existing module in Quartus while Task 3 is using the original RAM I created myself. I have a r_data logic as the output, switch 4~8 as the address, switch 0~3 as the write data, and switch 9 as the write enable of the memory array. I then developed another module that simply display the result on the HEX.

Task 4 is a little more difficult but still has the same concept as the previous three tasks. I started from developed a 5 bit counter with reset function (KEY0) that can count up to 32 since we are reading and writing to a 32x4 RAM. Next, I created a clock divider module that make sure the clock increment in a 1 second interval. Further, I pass in the switch 0~3 as write data, counter's output as read address, switch 4~8 as write address, switch 9 as write enable, and additional r_data as the read data from the RAM. Lastly, I used the same HEX display function to make the result show on the DE1_SoC board.

I implemented the way I did because I have learned that each module should do one thing only. I have every single module carry on one simple tasks and my project simply came together when I instantiated them all.

Results

My result was successfully shown on both ModelSim and the FPGA board. In memArray testbench (Figure 1), one can easily see how the data is being written to the RAM only when the write enable is true. One can also tell that the RAM indeed output an updated value.

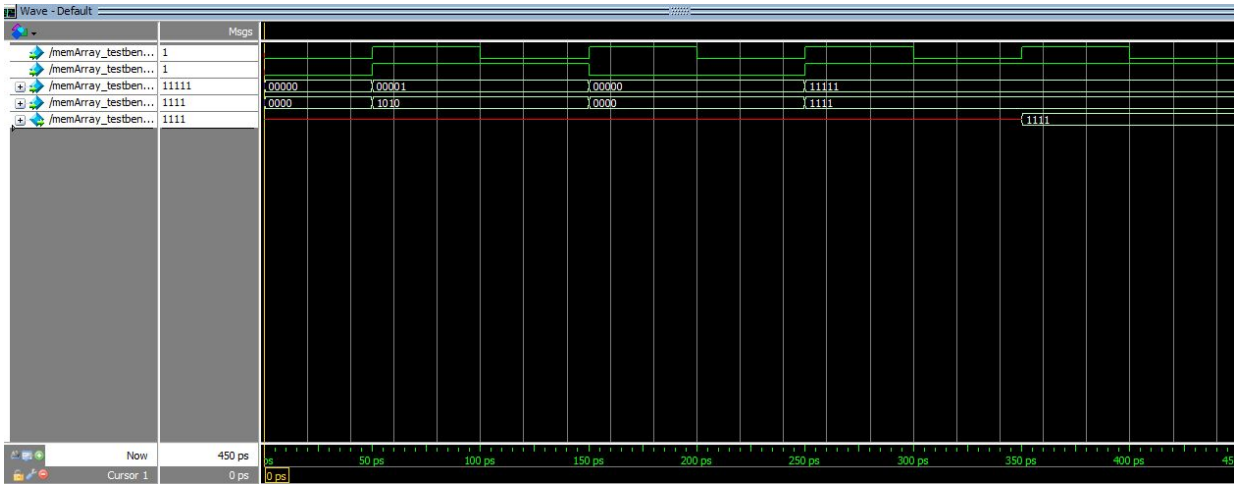


Figure 1. ModelSim for memArray module

The more advanced functions are tested in task3 ModelSim that we can see how the system allows user to use switches to change the reading and writing address (Figure 2). One can also tell that the address is written only when the write enable, which is switch 9, is true and the data updated immediately. The data stored in the address stayed the same if there is no further change in that address.

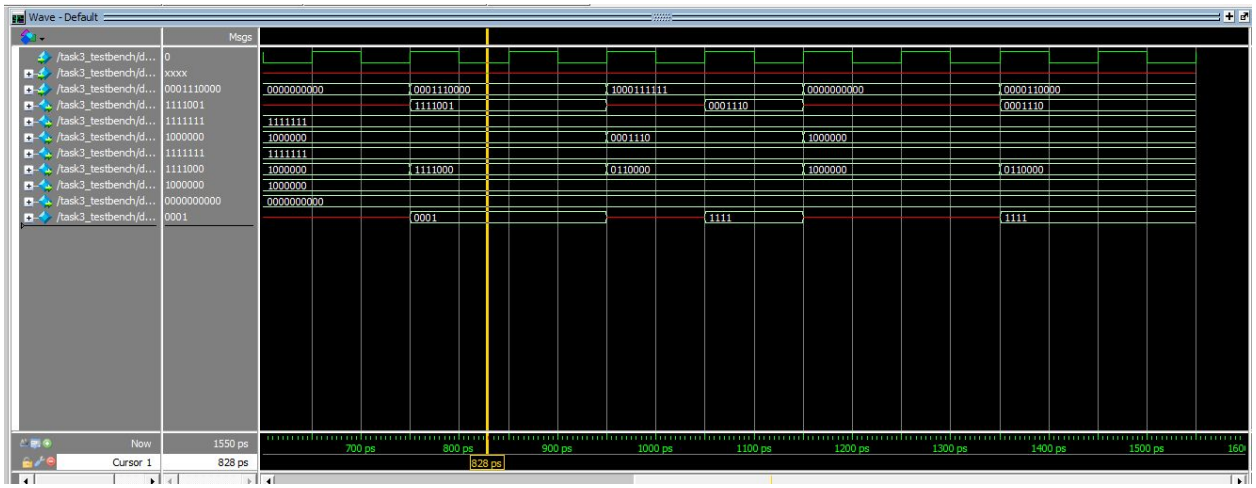


Figure 2. ModelSim for task3

The actual design of task 3 can be seen in Figure 3~5. Initially the board stores nothing in address 5'b00000 (Figure 3). Then I wrote 4'b0011 into address 5'b01000. One can see the data out on the HEX0 changed to 4'b0011 immediately (Figure 4). When I remove the write enable (switch 9) and the input data(switch 0~3), the value is still stored in the address 5'b01000 (Figure 5).

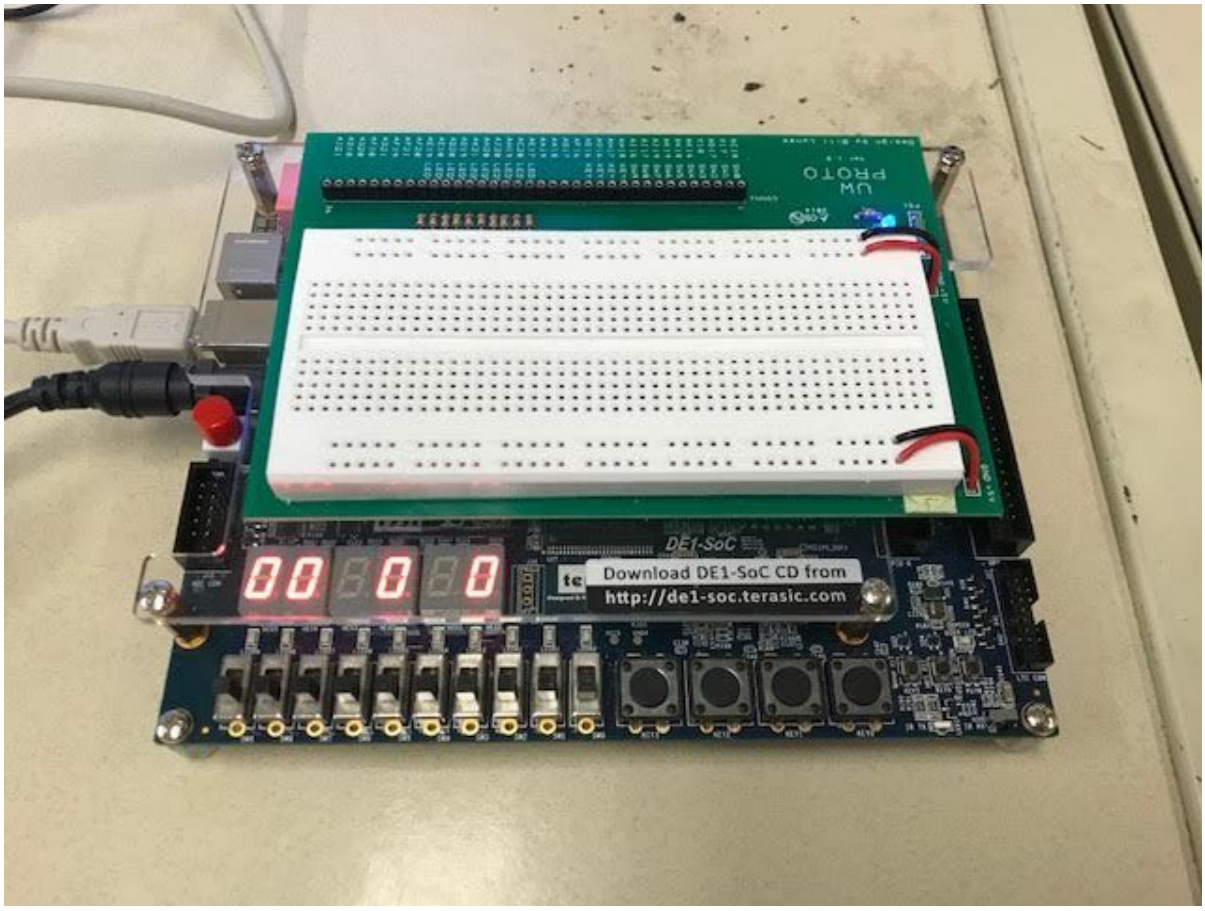


Figure 3. Initial state of Task 3

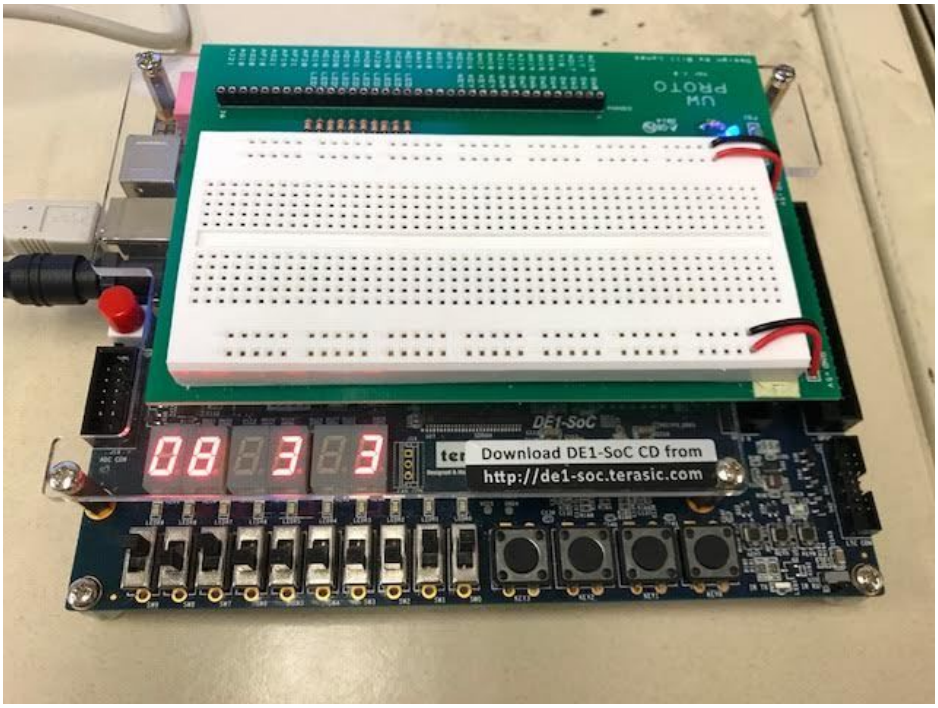


Figure 4. Write 4’b0011 to address 5’b01000

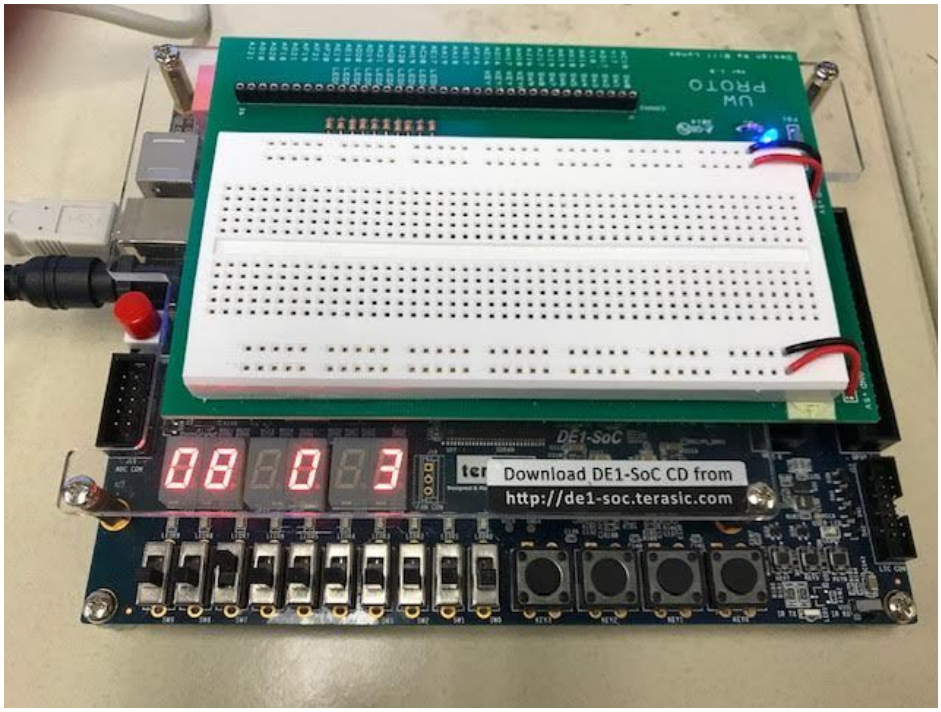


Figure 5. Value stayed in address 5’b01000

Task 4's actual result is shown in Figure 6~10. Figure 6 is me pressing the reset key to initialize the system. As the system is running, I store value F to three continuously address 0, 1, 2 (Figure 7).

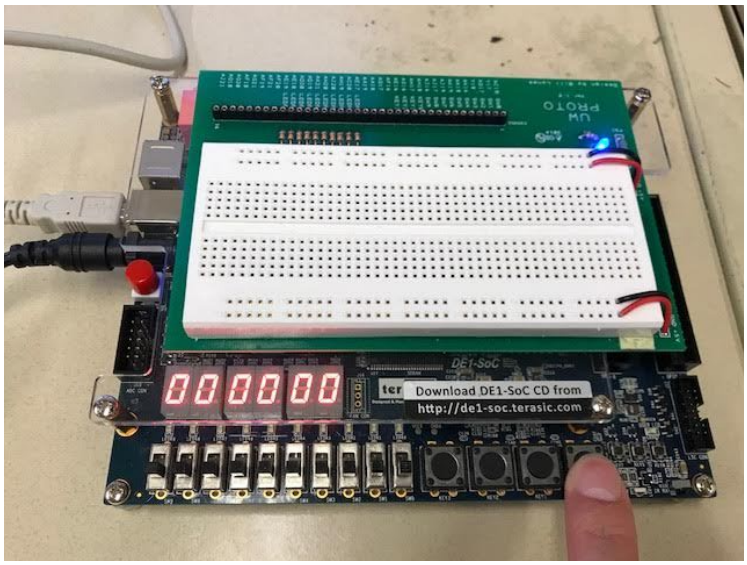


Figure 6. Task 4 demo

Then reset again to see if they replaced the value that was stored and changed them to F. Figure 8 shows my design work perfectly.

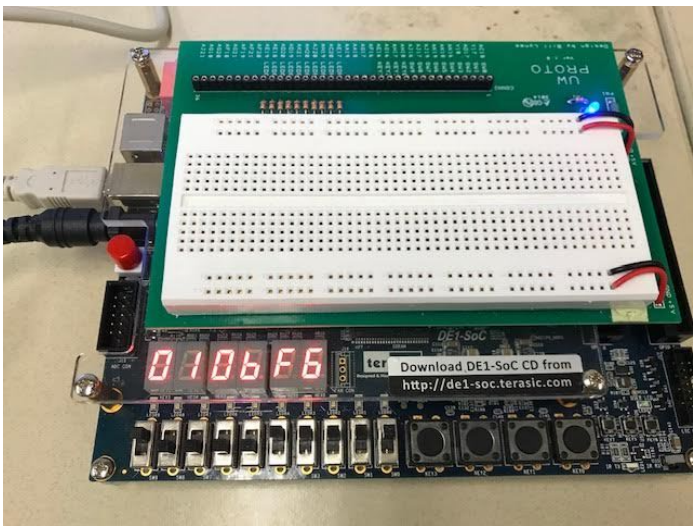


Figure 7. Set address 5'b0001 to F

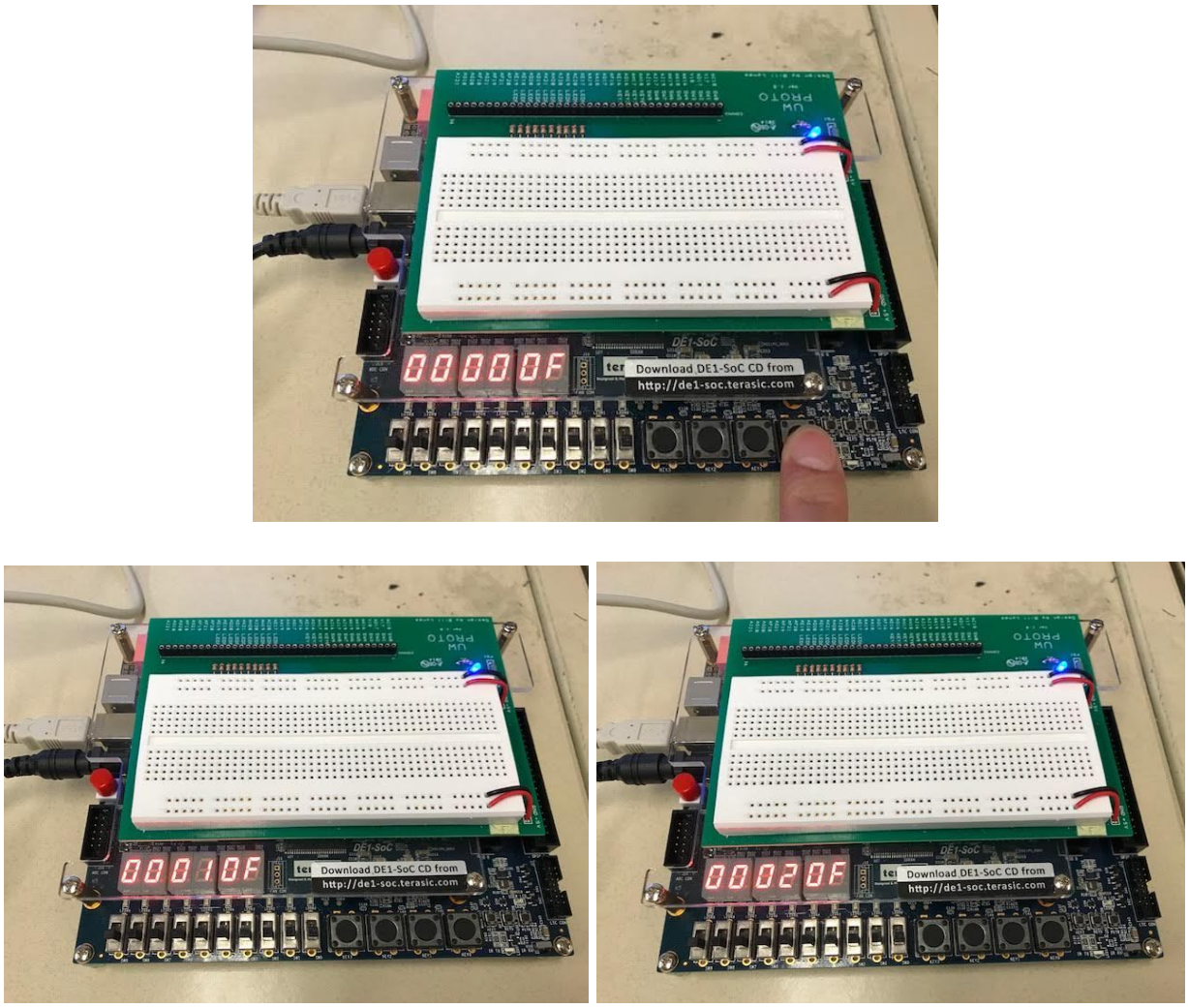
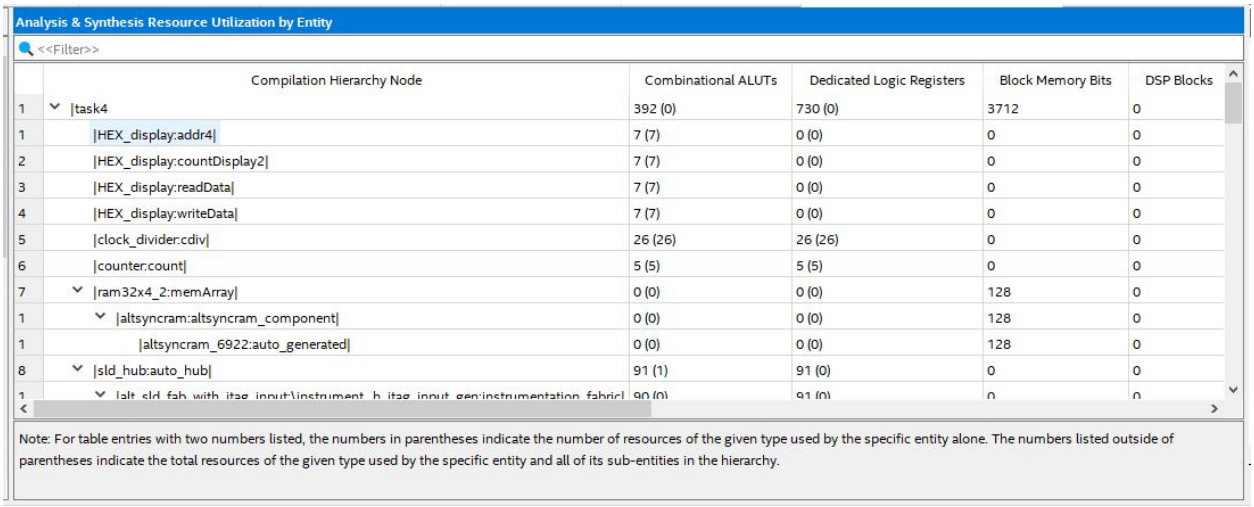


Figure 8~10 show F is stored in address 0~2

On the other hand, the Resource Utilization by Entity page (Figure 11) shows that the size of the program is 392+730 = 1222



Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks
1	|task4	392 (0)	730 (0)	3712	0
1	|HEX_display:addr4|	7 (7)	0 (0)	0	0
2	|HEX_display:countDisplay2|	7 (7)	0 (0)	0	0
3	|HEX_display:readData|	7 (7)	0 (0)	0	0
4	|HEX_display:writeData|	7 (7)	0 (0)	0	0
5	|clock_divider:cdiv|	26 (26)	26 (26)	0	0
6	|counter:count|	5 (5)	5 (5)	0	0
7	|ram32x4_2:memArray|	0 (0)	0 (0)	128	0
1	|altsyncram:altsyncram_component|	0 (0)	0 (0)	128	0
1	|altsyncram_6922:auto_generated|	0 (0)	0 (0)	128	0
8	|sld_hub:auto_hub|	91 (1)	91 (0)	0	0
1	|alt_sld_fab_with_jtag_input|instrument_h_jtag_input_gen:instrumentation_fabric|	90 (0)	91 (0)	0	0

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

Figure 11. Resource Utilization by Entity page

Problems Faced and Feedback

I think the most difficult part of this lab is understanding the counter represent the address in RAM. I was not so sure initially what is the difference between the read and write address so I passed in the same value to the RAM. But I then realized the read and write should be two different ports. I resolved the problem by passing in the counter module's output and complete the rest of the design.

I really like this lab because every task is built on the previous one. I learned a bit more gradually. Moreover, I feel accomplished when I used the new skill I learned from the prior task instantly in the next task. I also feel more comfortable to develop program related to memory after this project.