Eugene Ngo: 1965514

1/20/2023

EE 371 – Lab 2 Report

Procedure

The lab comprises of three tasks. The first task was to design a 32x4 single port RAM that can takes an input 5-bit address and has a 4-bit data port. Then a write control input, its address, data to write and data to read will be displayed on the HEX display in hexadecimal form. The second task was to design a 32x4 2-port RAM using the IP catalog in Quartus that uses one port supplying a 5-bit address for read operations and the other port is used for a 5-bit address for write operations. The read address should be internally updating by a designed counter on an approximately 1 Hz frequency, and all addresses, the 4-bit data to read and data to write, should be display on the HEX display in hexadecimal format. The third and last task was to design a FIFO that uses a 16x8 2 port RAM to store data, and display the "queue's" full/empty status on LEDR9/8, and display read data, write data on the HEX display in hexadecimal form.

Task 1

By using built-in memory blocks, I designed a 32x4 single port ram that can takes inputs of a 5-bit address (SW[8:4]), a 4-bit data to write (SW[3:0]), a write control (SW[9]), and a user-operated clock (KEY[0]) and output a 4-bit data to read. The read/write address was displayed on HEX5 and HEX4, data to write is displayed on HEX2, and data to read is displayed on HEX0. These are all displayed in hexadecimal form. The RAM always reads the data in the specified address, but only when write control is enabled, the input data can be written into the address on the next self-operated positive clock edge.

Task 2

Task 2 built upon task 1 but is slightly different since task 1 uses 1 port, and task 2 uses 2. Using the IP catalog in Quartus Design, I designed a 32x4 2-port RAM as directed by the lab spec that is able to use one port supplying 5-bit address for read operations where the address is specified by a internal counter and scrolls through the memory locations at a rate of approximately 1 Hz and the other port is a 5-bit address controlled by SW[8:4] for write operations. It also takes in 4-

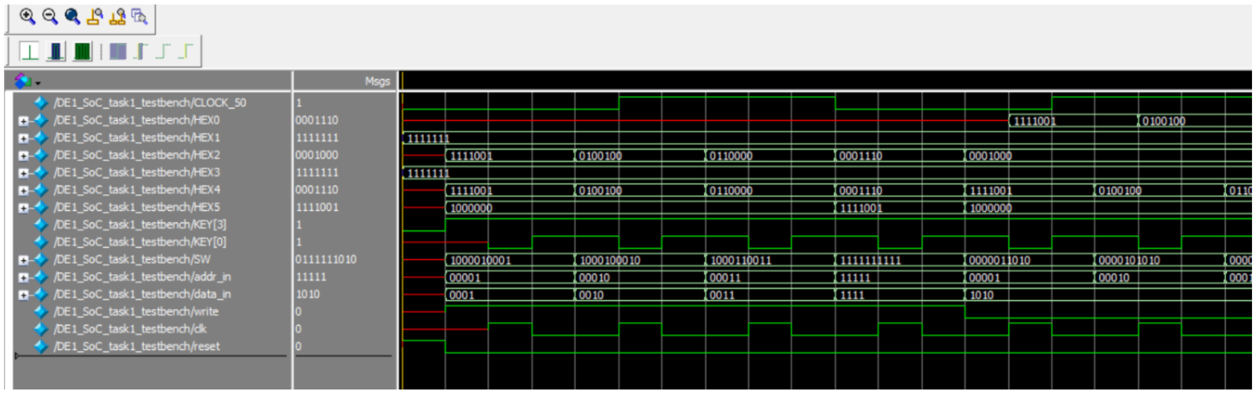
bit data to write and a write control using KEY[3]. The write address is displayed on HEX5 and HEX4, the read address is displayed on HEX3 and HEX2, the data to write on HEX1 and the data to read on HEX0, all in hexadecimal form. So the read and write operations are completely separate: counter and read works on a set frequency independently and the write operation is only working when wr_en, which is KEY[3], is pressed.

Task 3

For task 3, the final task, I designed a FIFO, using the given skeleton, that uses a 16x8 2 port RAM to store data, and operates like a 'Queue;.' It displays the "Queue" status, full or empty, on LEDR9 and LEDR8, respectively. When read is enabled, it reads the data that was first input and displays it on HEX1 and HEX0; it follows the rule: "first in, first out." When write is enabled, it writes the data specified using SW[7:0] to the newest address and displays this value on HEX5 and HEX4. All HEX displays are in hexadecimal form.

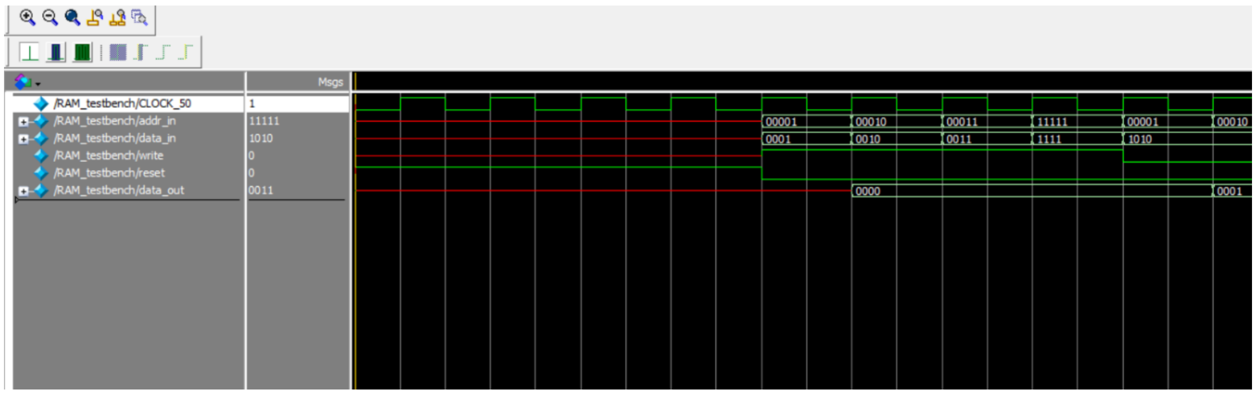
Results

Task 1

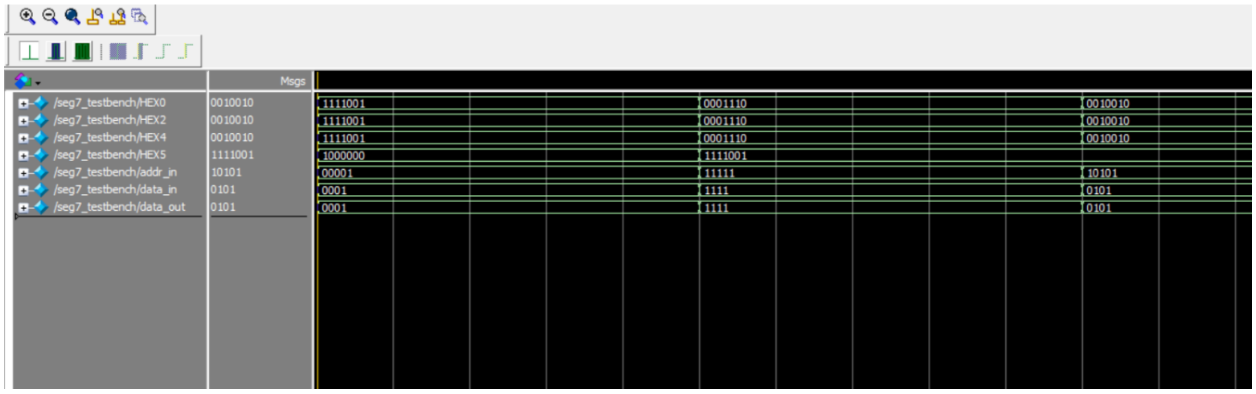


```
                                              Msgs
/DE1_SoC_task1_testbench/CLOCK_50             1
/DE1_SoC_task1_testbench/HEX0                 0001110
/DE1_SoC_task1_testbench/HEX1                 1111111
/DE1_SoC_task1_testbench/HEX2                 0001000
/DE1_SoC_task1_testbench/HEX3                 1111111
/DE1_SoC_task1_testbench/HEX4                 0001110
/DE1_SoC_task1_testbench/HEX5                 1111001
/DE1_SoC_task1_testbench/KEY[3]               1
/DE1_SoC_task1_testbench/KEY[0]               1
/DE1_SoC_task1_testbench/SW                   0111111010
/DE1_SoC_task1_testbench/addr_in              11111
/DE1_SoC_task1_testbench/data_in              1010
/DE1_SoC_task1_testbench/write                0
/DE1_SoC_task1_testbench/clk                  0
/DE1_SoC_task1_testbench/reset                0
```

Displayed above is the DE1_SoC testbench which shows the data input being displayed on HEX2 and the specified address on HEX4 and HEX5. HEX1 and HEX3 are off. Then when written, using SW 9, and then read, the read data is displayed on HEX0.

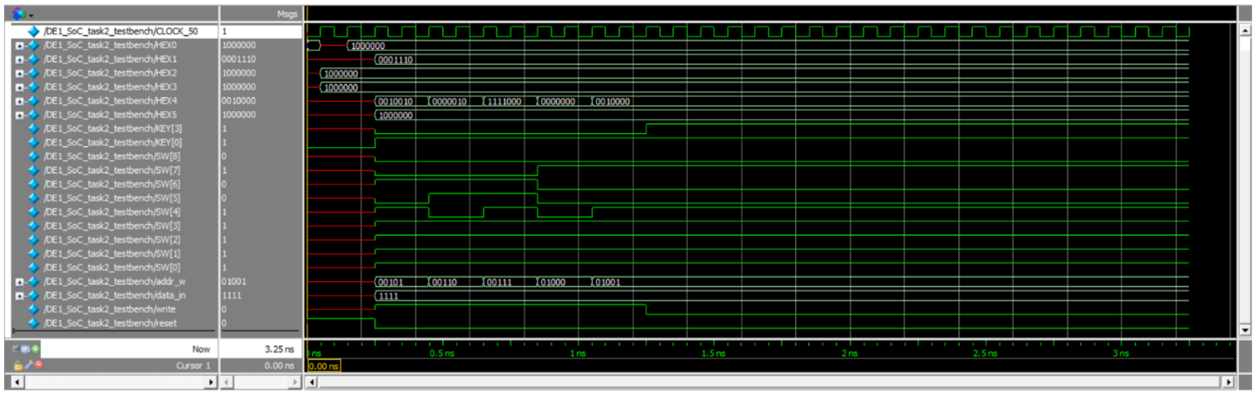


This RAM test bench shows the data, data_in, being written in at a specified address addr_in, and then shows the output in data_out.

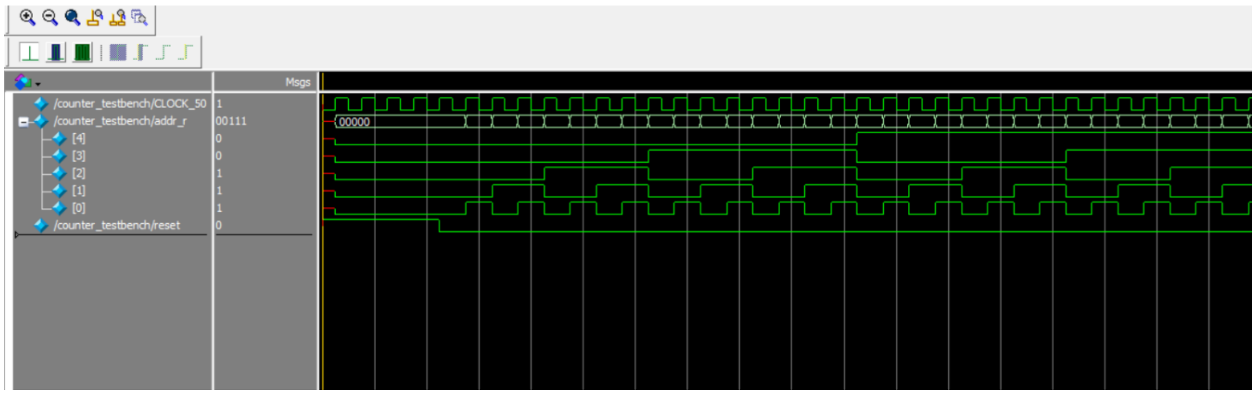

Msgs
/seg7_testbench/HEX0 0010010
/seg7_testbench/HEX2 0010010
/seg7_testbench/HEX4 0010010
/seg7_testbench/HEX5 1111001
/seg7_testbench/addr_in 10101
/seg7_testbench/data_in 0101
/seg7_testbench/data_out 0101
1111001 0001110 0010010
1111001 0001110 0010010
1111001 0001110 0010010
1000000 1111001
00001 11111 10101
0001 1111 0101
0001 1111 0101

The seg7 testbench displays the HEX displays functioning properly to display data to the appropriate HEX displays.

Task 2

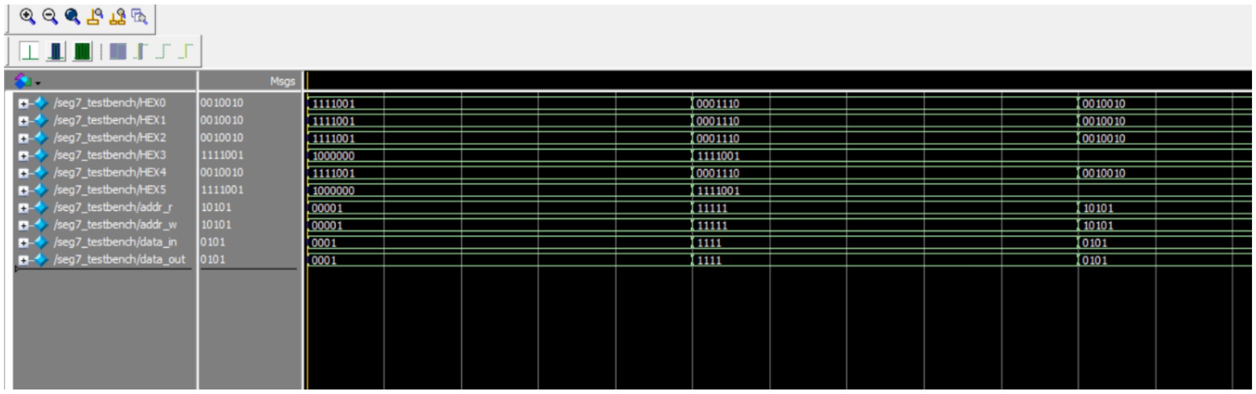

Msgs
/DE1_SoC_task2_testbench/CLOCK_50 1
/DE1_SoC_task2_testbench/HEX0 1000000
/DE1_SoC_task2_testbench/HEX1 0001110
/DE1_SoC_task2_testbench/HEX2 1000000
/DE1_SoC_task2_testbench/HEX3 1000000
/DE1_SoC_task2_testbench/HEX4 0010000
/DE1_SoC_task2_testbench/HEX5 1000000
/DE1_SoC_task2_testbench/KEY[3] 1
/DE1_SoC_task2_testbench/KEY[0] 1
/DE1_SoC_task2_testbench/SW[8] 0
/DE1_SoC_task2_testbench/SW[7] 1
/DE1_SoC_task2_testbench/SW[6] 0
/DE1_SoC_task2_testbench/SW[5] 0
/DE1_SoC_task2_testbench/SW[4] 1
/DE1_SoC_task2_testbench/SW[3] 1
/DE1_SoC_task2_testbench/SW[2] 1
/DE1_SoC_task2_testbench/SW[1] 1
/DE1_SoC_task2_testbench/SW[0] 1
/DE1_SoC_task2_testbench/addr_w 01001
/DE1_SoC_task2_testbench/data_in 1111
/DE1_SoC_task2_testbench/write 0
/DE1_SoC_task2_testbench/reset 0
1000000
0001110
1000000
1000000
0010010 0000010 1111000 0000000 0010000
1000000
00101 00110 00111 01000 01001
1111
Now 3.25 ns
Cursor 1 0.00 ns
0 ns 0.5 ns 1 ns 1.5 ns 2 ns 2.5 ns 3 ns
0.00 ns

This testbench shows that the HEX displays are functioning properly as the input data is being specified using SW 3-0 and shown on HEX1 and the address is specified using SW 8-4 and shown on HEX5-4. The data is then written using KEY3 and displayed on HEX0.



```
                                 Msgs
/counter_testbench/CLOCK_50      1
/counter_testbench/addr_r        00111
  [4]                            0
  [3]                            0
  [2]                            1
  [1]                            1
  [0]                            1
/counter_testbench/reset         0
```

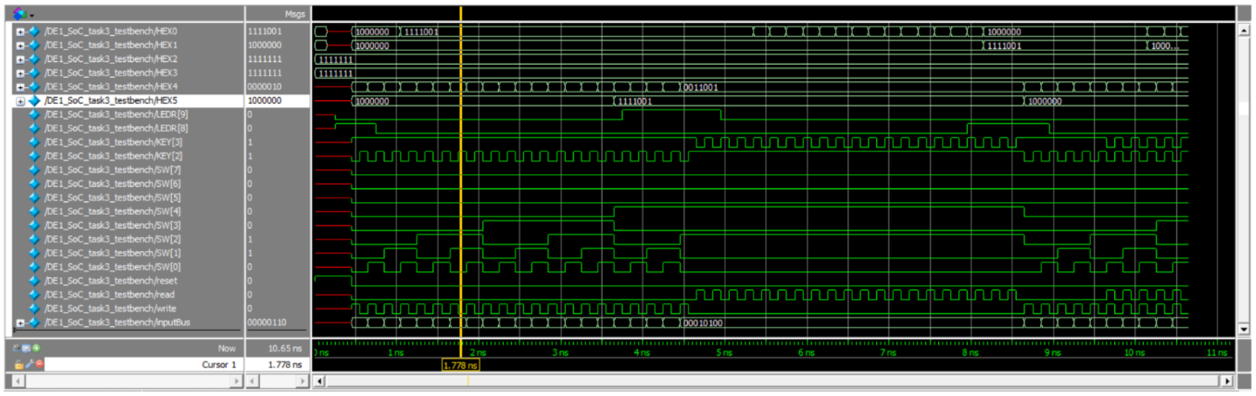
The counter testbench shows that the 5-bit addresses are being scrolled though incrementally.



```
                                 Msgs
/seg7_testbench/HEX0             0010010
/seg7_testbench/HEX1             0010010
/seg7_testbench/HEX2             0010010
/seg7_testbench/HEX3             1111001
/seg7_testbench/HEX4             0010010
/seg7_testbench/HEX5             1111001
/seg7_testbench/addr_r           10101
/seg7_testbench/addr_w           10101
/seg7_testbench/data_in          0101
/seg7_testbench/data_out         0101
```

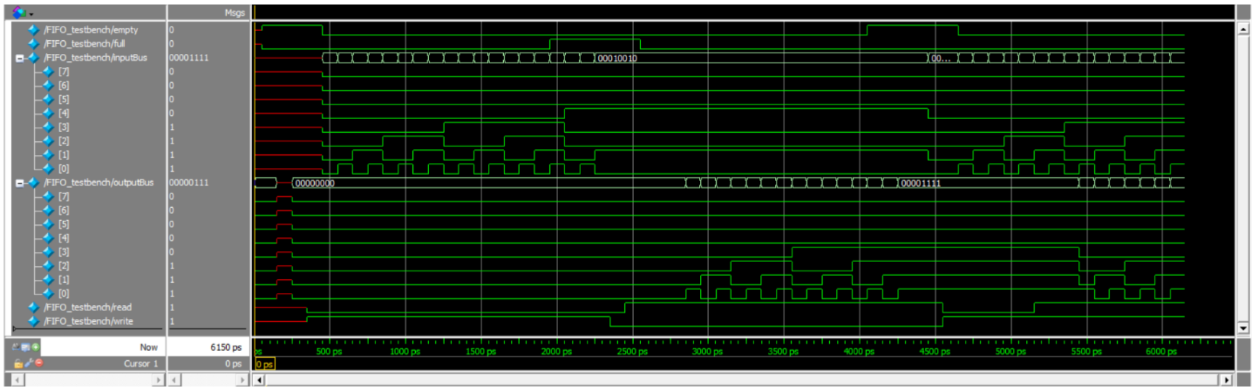
The seg7 testbench is testing random inputs of addr_r, addr_w, data_in, and data_out, and is displaying is on the appropriate HEX displays as seen above.

Task 3

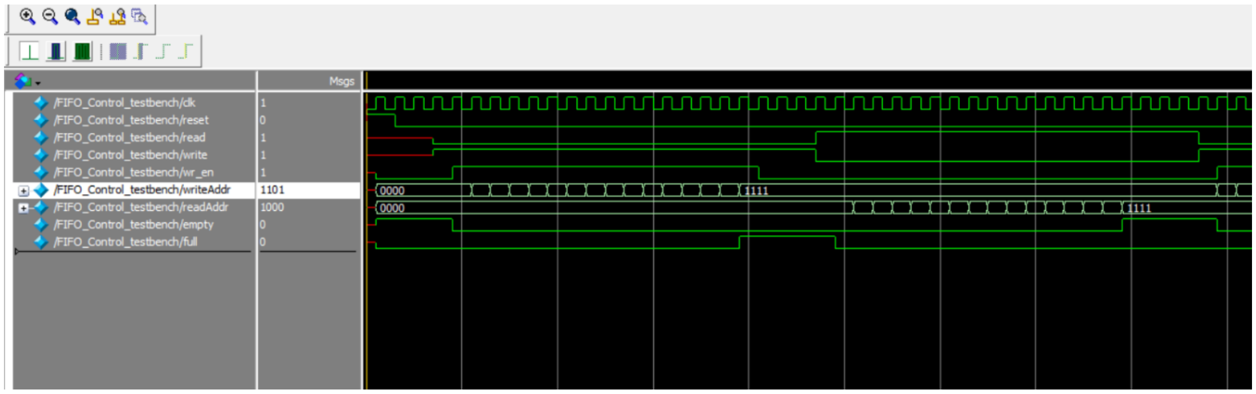


```
                                            Msgs
/DE1_SoC_task3_testbench/HEX0               1111001
/DE1_SoC_task3_testbench/HEX1               1000000
/DE1_SoC_task3_testbench/HEX2               1111111
/DE1_SoC_task3_testbench/HEX3               1111111
/DE1_SoC_task3_testbench/HEX4               0000010
/DE1_SoC_task3_testbench/HEX5               1000000
/DE1_SoC_task3_testbench/LEDR[9]            0
/DE1_SoC_task3_testbench/LEDR[8]            0
/DE1_SoC_task3_testbench/KEY[3]             1
/DE1_SoC_task3_testbench/KEY[2]             1
/DE1_SoC_task3_testbench/SW[7]              0
/DE1_SoC_task3_testbench/SW[6]              0
/DE1_SoC_task3_testbench/SW[5]              0
/DE1_SoC_task3_testbench/SW[4]              0
/DE1_SoC_task3_testbench/SW[3]              0
/DE1_SoC_task3_testbench/SW[2]              1
/DE1_SoC_task3_testbench/SW[1]              1
/DE1_SoC_task3_testbench/SW[0]              0
/DE1_SoC_task3_testbench/reset              0
/DE1_SoC_task3_testbench/read               0
/DE1_SoC_task3_testbench/write              0
/DE1_SoC_task3_testbench/inputBus           00000110
                                     Now    10.65 ns
                                Cursor 1    1.778 ns
```

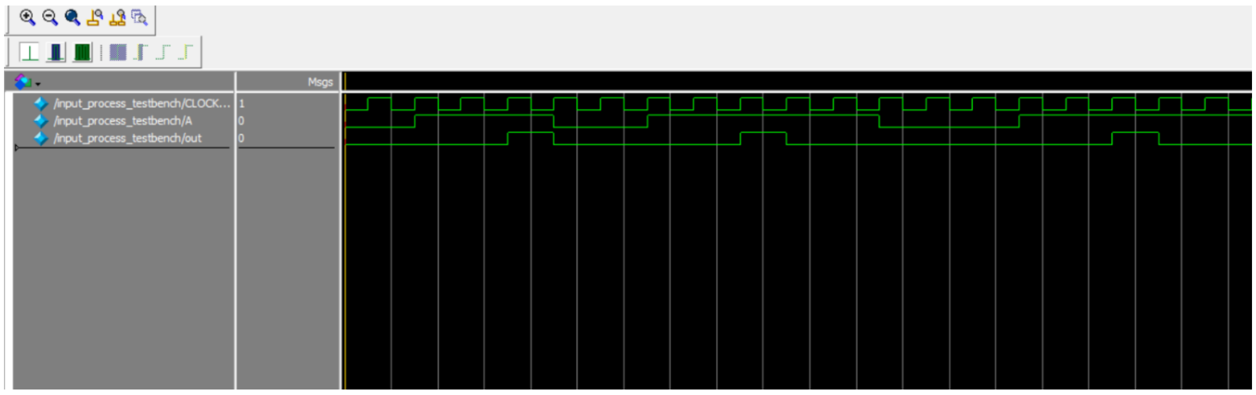
The DE1_SoC testbench writes in a bunch of data and fills the queue and once the queue is full, LEDR9 turns on to indicate such. Then excessive inputs is tested which will not be added to the queue as shown above and then the queue is emptied by reading out the entire queue in the order that it was entered in as displayed in the HEX displays. Once the queue is emptied, LEDR8 lights up to signal that the queue is now empty. Excessive reading is tested but once the queue is empty, no more data is read. Simultaneous reading and writing is tested at the end.



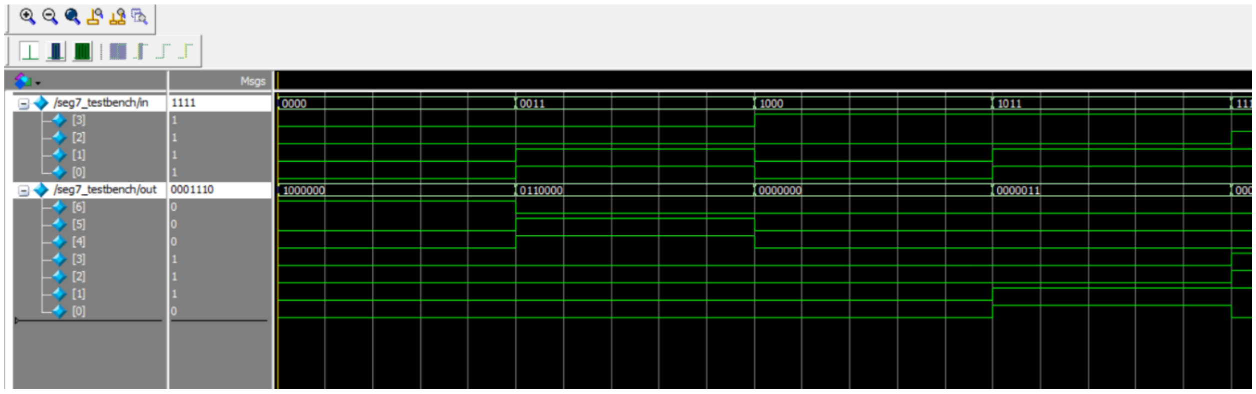
FIFO tests to ensure that data is stored in order and once the input bus is full, no more data is stored. Then it reads the entire queue and attempts to read excessively. The testbench also attempts a simultaneous read and write operation.


/FIFO_Control_testbench/clk 1
/FIFO_Control_testbench/reset 0
/FIFO_Control_testbench/read 1
/FIFO_Control_testbench/write 1
/FIFO_Control_testbench/wr_en 1
/FIFO_Control_testbench/writeAddr 1101
/FIFO_Control_testbench/readAddr 1000
/FIFO_Control_testbench/empty 0
/FIFO_Control_testbench/full 0
0000
1111

The FIFO_Control testbench ensures that this module is sending the correct signals out to the FIFO module such a when the queue is full or empty.


Msgs
/input_process_testbench/CLOCK... 1
/input_process_testbench/A 0
/input_process_testbench/out 0

The input process testbench shows that the input process module is controlling user inputs correctly and processing that input uniformly 2 clock cycles after the initial button press.



Seg7 demonstrates random inputs being displayed properly.

Overview

Overall, the lab taught me how to design RAM, reading and writing specified data to specified addresses, on multiple ports. The spec was easy to follow, and while long, was thorough in teaching us how to use the IP catalog to make the RAM.

Overall, the systems work as expected.

Appendix

See following code