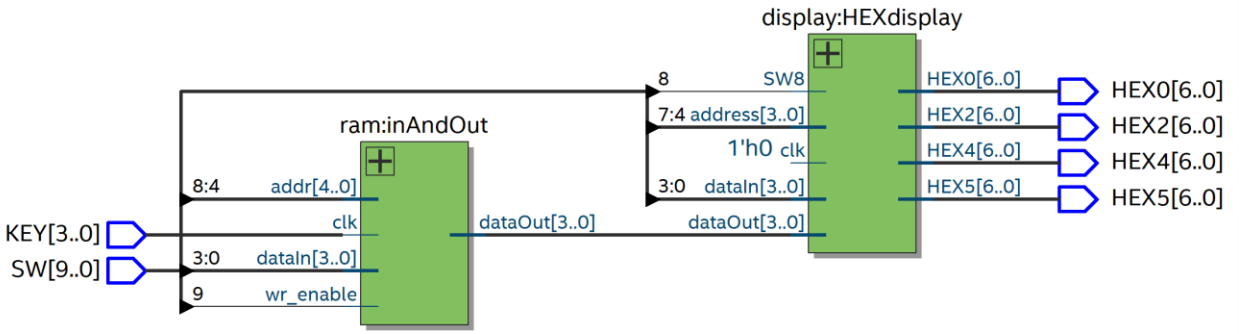
Alan Li
EE 371
January 15th, 2022
Lab 1 Report

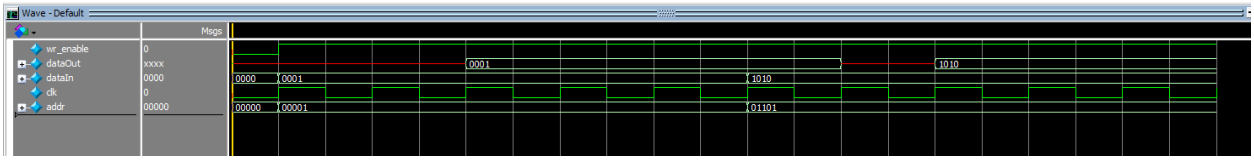
Task1

Section 1: Procedure

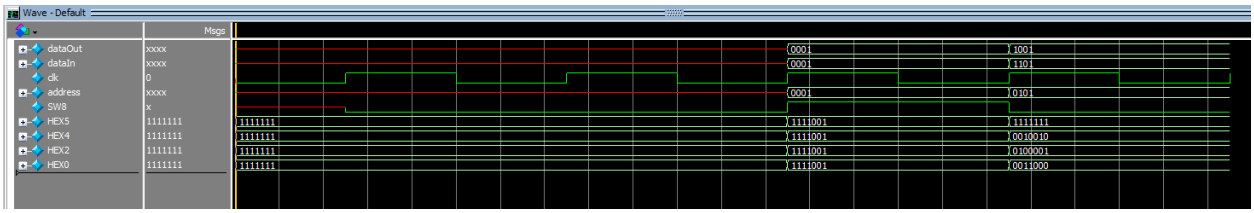


The lab description for task 1 is straightforward, write data into the ram when wr_enable is pressed and read data when clk input is toggled on. I designed three modules for the system. Ram module will handle the data and address from user input, as well as a control module to tell system when to store data and when to read data. The hex display is designed to display data from the ram module. DE1_SoC serves as top-level module which connects all the modules.

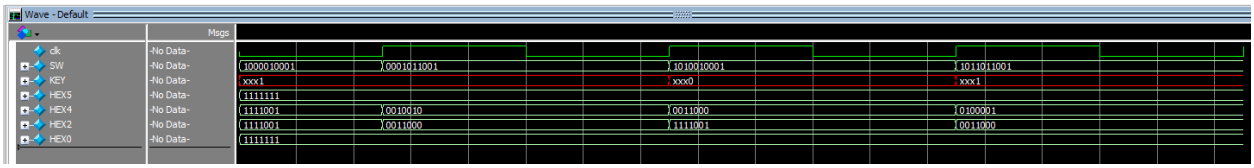
Section 2: Results



For the ram module, at first the wr_enable is off, so there are no data going into the ram. After user selected the data and address, the dataOut start to showing result. The reason that it takes two clock cycle to finally output the data is because writing the data takes one clock cycle and reading the data takes another clock cycle.

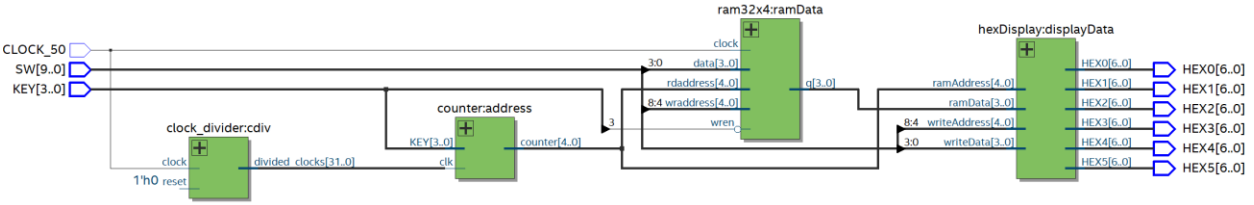


For the display module at first I set the wr_enable as off so in that way nothing is displayed. All four hex display is set as 11111111 which means they are off. Then I enabled wr_enable signal, then it starts to convert the binary input to hexadecimal and then output the data in hexadecimal format on display. So you can see the corresponding value on HEX display.



For the top level module I test 2 situation, first is both wr_enable and clock is on, in this case both dataIn and dataOut is updated and displayed on HEX display. When I disable wr_enable only data stored previously will be updated on HEX display.

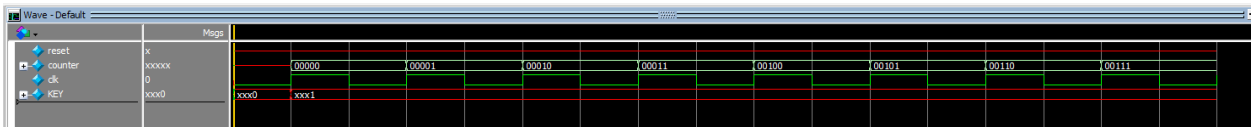
Task2



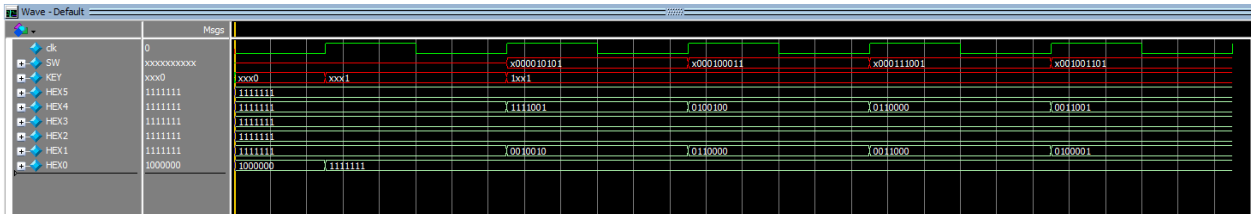
Section 1: Procedure

The way I approach to task2 is similar to task1. Compared to task1, we need another input which is the address that you want to read data in the ram. There are 6 files inside the design, hexDisplay is the same as hexDisplay I used in task1, the only difference is I need to rearrange the content on hexDisplay. I followed the tutorial and created ram32x4, clock_divider is used to generate 0.75hz clk which will display each data for about one second, counter is used to cycle through all the data inside the ram. Ram32x4.mif stores the initial value for the ram, the data are random. DE1_SoC serves as top-level module.

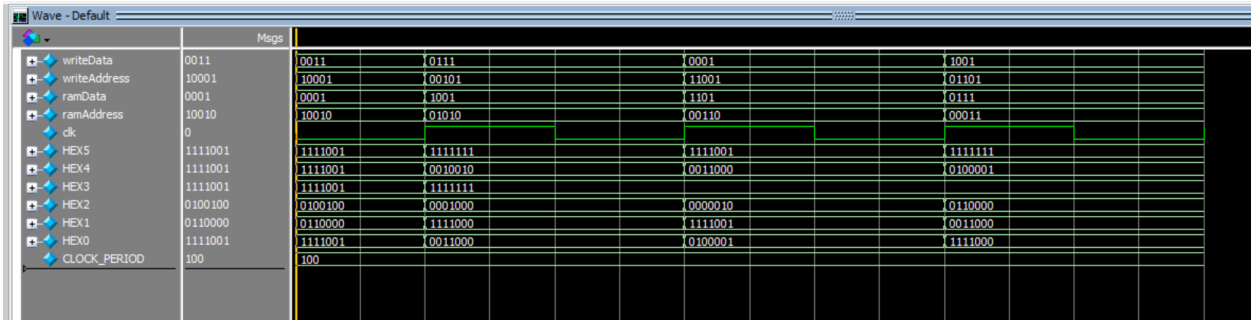
Section2: Results



For counter you can see the value has a increment by 1 at rising clock edge.

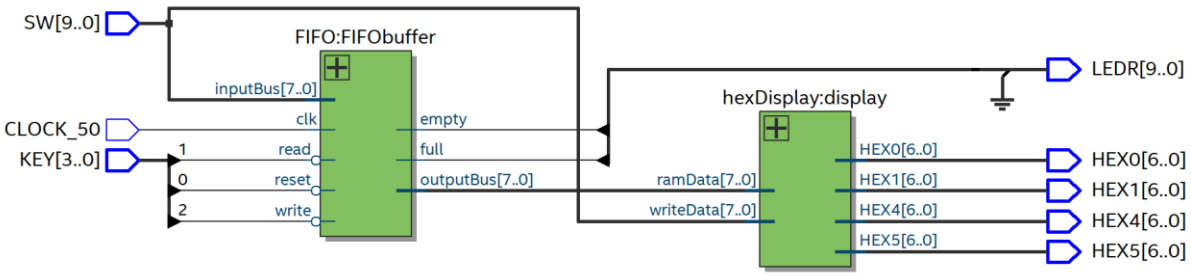


For DE1_SoC, the simulation results show the correct output data on HEX display. The address is increasing by one on every rising clock edge and the value is assigned to the corresponding address and then later being displayed.



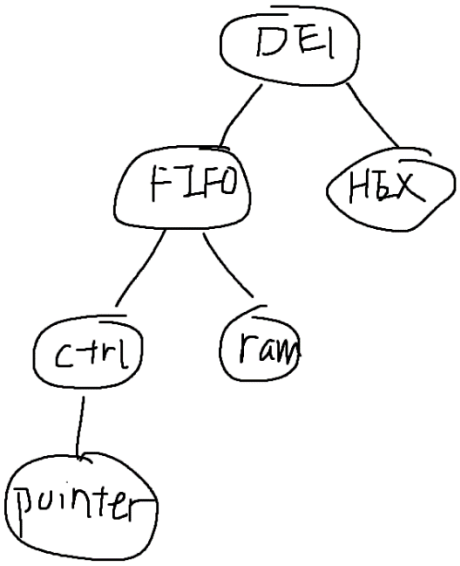
For hexDisplay, the testbench include some random data and address. The wave from shows that it displays the correct number/letters.

Task3

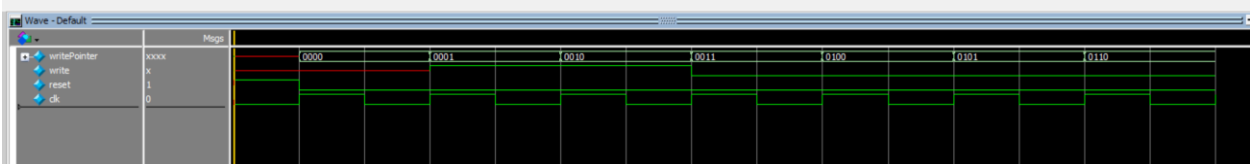


For this task I first create ram16x8 module and its mif file to assign random data. I design the FIFO as a “circular” ram, and it contains two pointers. One for read and another for write. Both pointers will cycle through the ram when read or write signal is given. The two pointers is like playing a chasing game, when read pointer reaches write pointer, that means all the data in FIFO has already been read and

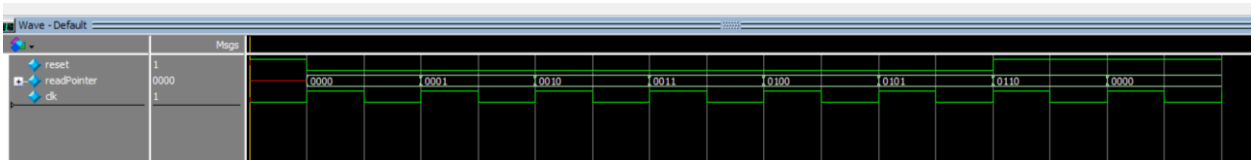
there are no new updated data since write signal was never given. Likewise, when write pointer reaches read pointer that means all the data in FIFO has already been updated and there are no room for another data, this time the FIFO is full. For my design, apart from the ram, hexDisplay and DE1_SoC, I added read and write pointer, FIFO and FIFO control. This is my structure for the whole task, I build my project based on this scheme.



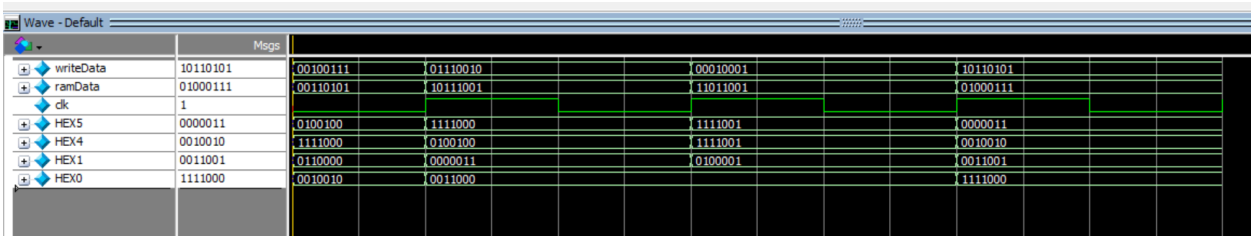
Section2: Procedure



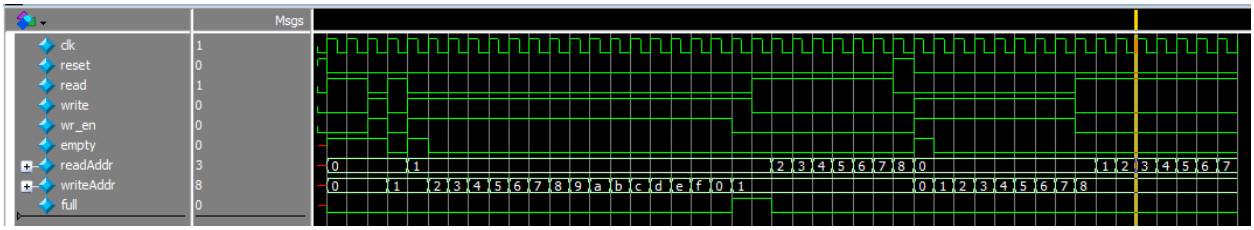
For writePointer testbench the clk is the write signal in the main module, at rising clock edge you can see that the address of the pointer increase by 1 every time.



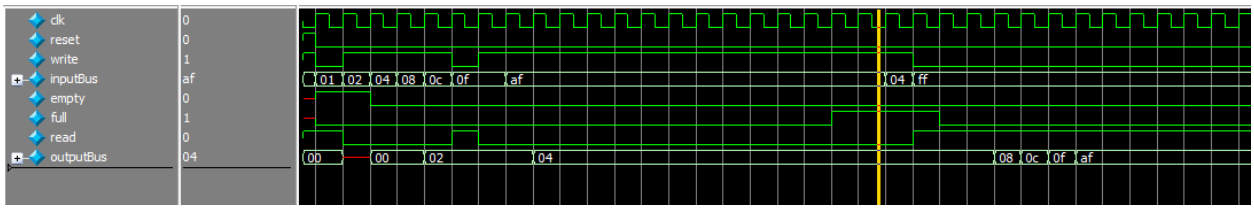
For readPointer testbench the clk is the read signal in the main module, at rising clock edge you can see that the address of the pointer increase by 1 every time.



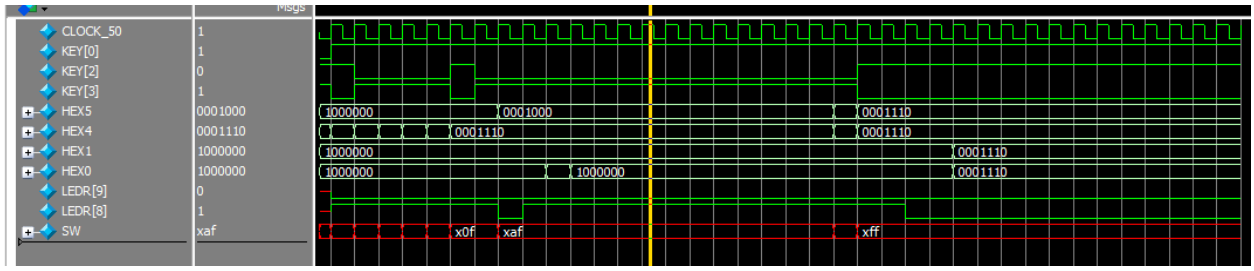
For hexDisplay, the testbench include some random data and address. The wave from shows that it displays the correct number/letters.



For FIFO_control, you can see after write 16 times the FIFO is full, then I read 16 times the FIFO is empty. The waveform also shows the correct address and data. w_en is on when (wr and empty) is true



For FIFO, you can see that when wr_en is on, the data in FIFO starts to update. The output bus will stay idle until the read signal is on. Then it starts to display the value in FIFO(08, 0c, 0f)



For DE1_SoC, the testbench will show the same result for data input/output and their address, the only thing that's added to the waveform is the hexdisplay which I have already tested in hexDisplay module.