DSD F18

Lab

31 x 8 ROM

In this lab you will design a Verilog Module for a ROM. The rom needs to be able to hold 32 unsigned Integers each 8 Bits. Thus it must have 5 address lines 25 = 32. This seems to be fairly straight forward to do; however, when I wrote this lab code last year and looked at the elaborated schematic post synthesis, I was shocked. It was very complicated and did not work very well. I had implemented the design without a clock, since it is a combinational part and really does not need a clock. But, the implementation was horrendous and it did not work quite like it should, so I added a clock to control the update with the address lines and the parallel load. Bam, everything was now very happy, so here is how you can do this with a clock. First set up the rom with this statement:

**reg** [7:0] **Store [0:31]**; //Note the 8 bit words organizes hi-lo Left to Right but the rows organized 0 to 31 Left to Right. You can order the rows opposite if you want, but this makes most sense to me. The columns really do need to be in normal byte order though.

Functionally, the ROM should have:

1. A parallel **Load** function using the 8 bit values set in the slide switches 7:0 labeled as **D 7:0** in the port list and in the constraint file and pressing the Left button to latch in the data to the ROM.

2. A **Clear** that should zero the contents of 1 byte that is addressed as described below when the **Right** button is pressed.

3. Address lines to address each of the 32 locations in the “ROM”, using the high 5 slide switches, switch 15:11 which are called **A 4:0** in the port list and in the constraint file.

4. As the address lines are set, the address should display in the high 5 LED 15:11, called **Ao 4:0** in the port list and the constraint file and the contents of the ROM at that address should also display as noted below.

5. Whatever address is input, the low 8 leds, LED 7:0 should display the value stored in the ROM at the addressed location\

6. And finally, use a clock to trigger a real always block to make the Vivado’s synthesizer happy.

7. Write your own test bench for this project. Have it put a few values into the ROM and see if they are stored. Fill out the questions and paste in your code for turn in as well as have TA sign off on your demo of the solution on the NEYS4 board.

I have provided the shell only so that everyone can use the same signal names to make following code and debug easier for TA’s and me, but still have to write their original solution. So use the shell and finish the code. The shell uses a bunch of statements in the initial block to clear the array variables to zeros. This can be done more simply by using a for loop, just don’t forget to initialize the for index variable as an integer variable as shown below: for (i=0; i< 32; i=i+1) //remember there is no i++ syntax allowed.

DSD

Lab 4 ROM

Name\_\_\_\_\_\_Andrew Quick\_\_\_\_\_\_\_\_\_ abc123 \_\_\_qxt050\_\_\_\_\_ Date\_\_\_\_\_\_9/18/2024\_\_\_\_\_\_

1. How do you calculate the required size of a rom from the requirements of number of words and size of words?

**Total ROM size = Number of words × Word size**

2. What is minimum number of bits required to hold signed, 2’s complement intergers that can vary from -60 to plus 60

**7**

3. Look at the RTL schematic. Did you have any problems with errors referring to dual port memory limitations? What do you think these meant if you did?

**No, but that would mean you’re trying to read and write at the same time.**

4. Give 4 uses you know of that use ROMs in an embedded way.

**1. Store firmware**

**2. Bootloaders**

**3. Look Up Tables**

**4. Flash Memory**