Jiovany Soliman, 1951338

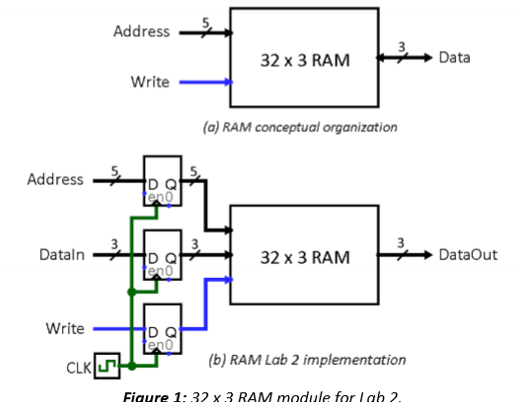
Chloe’ Miller, 2031648

EE/CSE 371 Lab 2 Report: Memory Blocks

October 13, 2023

**Design Procedure**

The purpose of this lab was to explore the general issues involved with implementing Random-Access-Memory (RAM) for systems that only employ FPGA technology by utilizing existing memory resources on the FPGA device. We first implemented a 32x3 RAM block with a 5-bit address input port, 3-bit data input port, a Write Enable input port, and a 3-bit data output port in Fig. 1(b)by using a prebuilt library module. Then, we recreated this 32x3 RAM block functionality using only SystemVerilog code and taking inputs from the onboard switches and keys. Lastly, we created a memory module that had independent read and write capabilities by adding separate ports for the addresses of read and write operations.



***Figure 1:*** *32 x 3 RAM module (from Lab 2 document).*

**Task #1:**

For Task #1, we implemented the 32x3 RAM block using the prebuilt library from the IP Catalog in Quartus called “RAM: 1 PORT” as seen in Fig. 2. We verified our instantiation of this memory block by creating a suitable testbench to read and write data to RAM. As seen in Fig. 3,data was written to RAM at a specific address in memory when the Write Enable (wren) signal was logic HIGH, and one clock cycle later the written data was read out. When the wren signal was logic LOW, we were able to verify that RAM retained the data written to it in the beginning of the simulation. A register-trasfer level (RTL) representation of the code can be seen in Fig. 4.

A screenshot of a computer program

Description automatically generated***Figure 2:*** *32 x 3 RAM:1-PORT instantiation.*

A screen shot of a computer

Description automatically generated***Figure 3:*** *ModelSim waveform for RAM:1-PORT functionality.*

A diagram of a computer

Description automatically generated

***Figure 4:*** *RTL view of RAM:1-PORT instantiation.*

**Task #2:**

For Task #2, we recreated the 32x3 RAM block functionality by using only SystemVerilog code (Fig. 5). To accomplish this, we needed to use multidimensional arrays for handling inputs and outputs of the memory block. Onboard switches SW3-SW1 were used to specify the input data, and switches SW8-SW4 were used to specify the address in RAM to write to. Switch SW0 was used as our Write Enable signal and KEY0 was used as our clock input. We also used the onboard 7-segment HEX displays to read out the write address, the write data, and the output data in hexadecimal on HEX5 – HEX4, HEX1, and HEX0, respectively.

A computer screen shot of a program

Description automatically generated***Figure 5:*** *SystemVerilog code for 32x3 RAM functionality.*

By using the same testbench as in Task #1, we verified that the behavior of our 32x3 RAM block matched the behavior of the 32x3 RAM block instantiated from the IP Catalog prebuilt library by running it in ModelSim (Fig.6). The ModelSim waveform for Task #2 is identical to that of Task #1. We further verified this behavior by uploading and testing our code on the DE1\_SoC in LabsLand.

A screen shot of a computer

Description automatically generated***Figure 6:*** *ModelSim waveform for SystemVerilog 32x3 RAM functionality.*

A computer diagram of a computer

Description automatically generated with medium confidence***Figure 7:*** *RTL view of SystemVerilog code 32x3 RAM instantiation.*

**Task #3:**

For Task #3, we needed to create a different type of memory block that had separate ports for the addresses of read and write operations. This type of memory block allows users to write to one address in RAM and read from another address simultaneously, compared to the single port RAM blocks that can only read or write to an address in memory at a time. We implemented this memory block by using the prebuilt module “RAM: 2-PORT.”

Other requirements for this task were to modify the Top-Level module to combine the single-port RAM from Task #2 with the dual-port RAM and give users the ability to select which RAM block to write to by using the switches. An important feature of this task was that the memories from Task #1 and Task #3 were independent of each other, such that writing to one RAM was not reflected in the other.

The 7-segment HEX displays were used to read out the write address, the input data, the output data, and to cycle through the selected memory’s contents. The displays were used similarly to Task #2, with the only difference being that HEX3 – HEX2 were used to display the cycled read address. We needed to create a new counter module that cycled through read addresses about one per second from 0 – 31 and implemented a clock divider module used in previous labs from EE 271 to manage the speed of this counter module.

Onboard switches and keys were used to read and write to the RAM blocks in this task. Just like in Task #1, switches SW3-SW1 specified the input data, switches SW8-SW4 specified the address in RAM to write to, and switch SW0 was the Write Enable signal. To choose between reading/writing Task #2 and Task #3 RAM blocks, switch SW9 was used to indicate Task #2 RAM if logic LOW, and Task #3 RAM if logic HIGH. We also needed a reset signal KEY3 for the counter module which restarts the counter at read address 0 (this made it easier to verify if data input had been properly written and stored). KEY0 was not used at all for Task #3 like it was in Task #2.

**Results**

Our completed system allows users to select between either a single-port RAM or a dual-port RAM to write to. All 9 switches are used to control where the desired input data is stored, and the HEX displays give visual feedback on user inputs as well as RAM outputs. One such specification that our SystemVerilog code does not meet is displaying the cycled read address and the associated data stored at that read address for the current selected RAM. There is a one-cycle delay between the read address and the stored data such that the read address is “ahead” of the stored data and gives a misrepresentation of the contents of the selected RAM. We were unable to resolve this bug in the code.

**DE1\_SOC (Task #2)**

A computer screen with green lines

Description automatically generated***Figure 8:***

A screenshot of a computer program

Description automatically generated***Figure 9:*** *Code for top-level.*

A diagram of a computer

Description automatically generated***Figure 7:*** *RTL view of SystemVerilog code 32x3 RAM instantiation.*

**DE1\_SOC (Task #3)**

A diagram of a computer

Description automatically generated

**Flow Summary**

A screenshot of a computer

Description automatically generated

**Experience Report**

We found that this lab was more difficult than expected, and that some lab specifications were not made clear or were confusing. It also took us a while to figure out a bug in our code that would not display the counter properly on the HEX displays (we forgot that KEYs are active-low). Our file management also set us back because sometimes we were dealing with older versions of code with subtle differences from the current versions we should have been using for testing. Lastly, sometimes Quartus, ModelSim, and LabsLand would not all agree on what errors to throw, which contributed to our debugging time immensely.

A helpful feature of Quartus that we utilized was the RTL viewer, which gave us a great visual representation of how our modules and wires were connected and helped us debug at times or understand how we should implement new code. Hanging out in the lab and asking the TAs a lot of questions also helped us work through our code.

This lab took us approximately 18 hours, broken down as follows:

* Reading – 30 minutes
* Planning – 30 minutes
* Design – 1 hours
* Coding – 3 hours
* Testing – 5 hours
* Debugging – 8 hours