California State University, Northridge

Department of Electrical & Computer Engineering



ECE 526L

Lab 7 Report

By

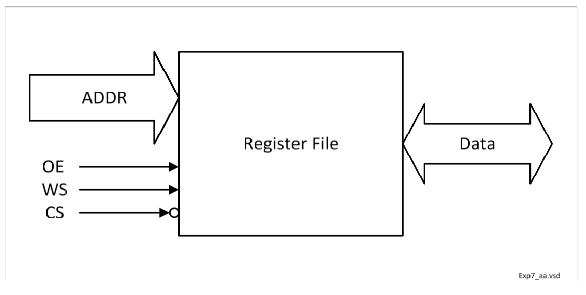
Avinash Damse

CSUN ID- 203131064

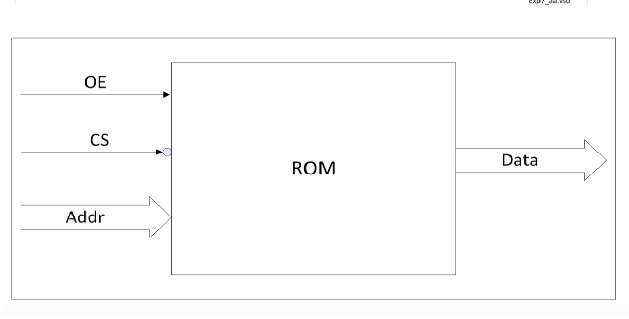
**1: Introduction**

The objective of this lab is to construct a scalable register file with a bidirectional data bus line and Initializing memory from file.

Here we have to use following diagram to build scalable register.



Which will be used as Random Access Memory.



**2: Procedure**

**a. Part 1: Creating REG\_F Module**

In this lab I have created a REG\_F module for scalable register. Inside the module I have assigned “WS, OE, CS and ADDR,” as input variables and “DATA” as bidirectional variable. Then I wrote the scalable register logic inside module. After completing the code I ended the module using and saved the file with name “REG\_F.v”.

**a. Part 1.2: Creating ROM Module**

In this lab I have created a ROM module for only read purpose. Inside the module I have assigned “ OE, CS and ADDR,” as input variables and “DATA” as output variable. Then I wrote the read logic inside module. After completing the code I ended the module using and saved the file with name “ROM.v”.

**b. Part 2: Creating REG\_TB1 Module**

I have written the test bench for the REG module. We require test bench just to make sure that the module we have created is working properly. Here, in this testbench I have performed write to and read from operation and demonstrated an individual and block read.

**c. Part 3: Creating REG\_TB2 Module**

I have written the test bench for the REG module. Here, in this testbench I have initialized the memory in hex using $readmemh system task. I put all the addresses and memory in INIT.txt file.

**c. Part 3: execution.**

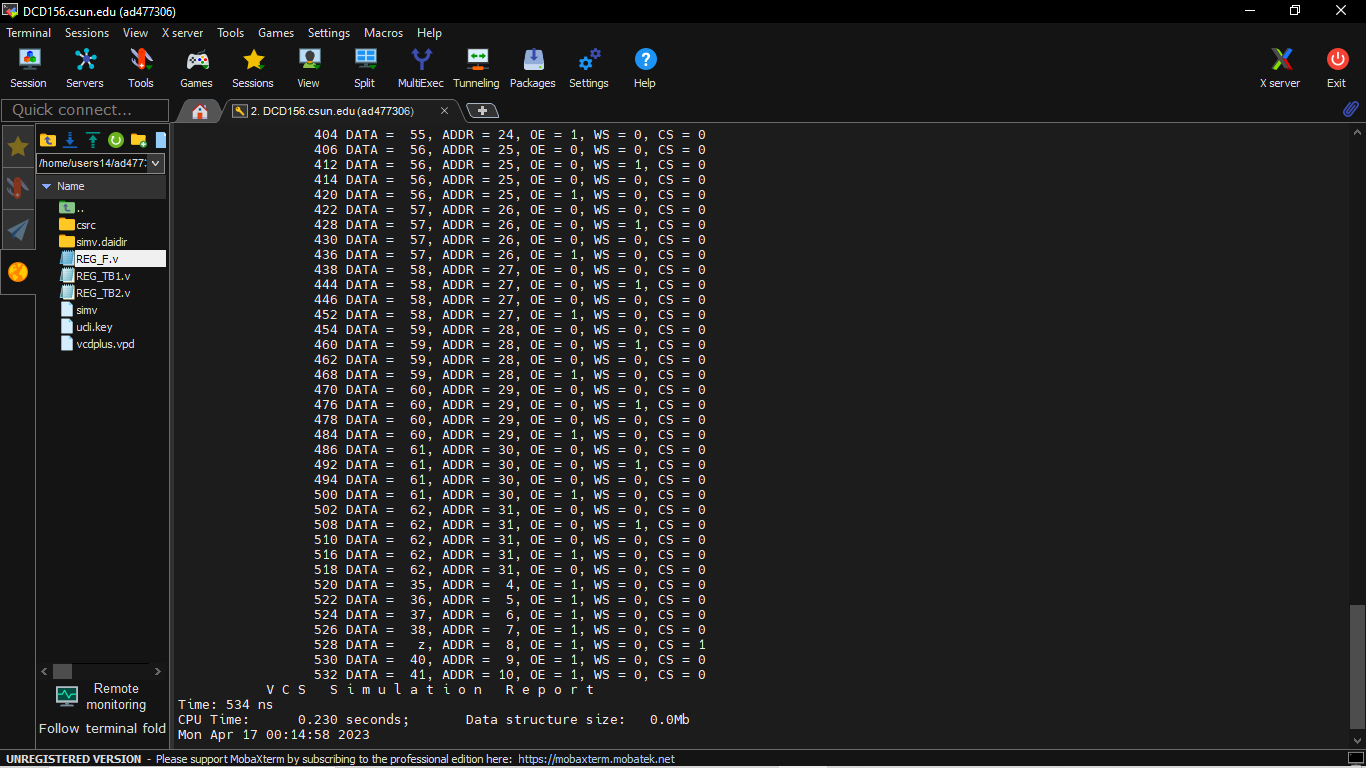
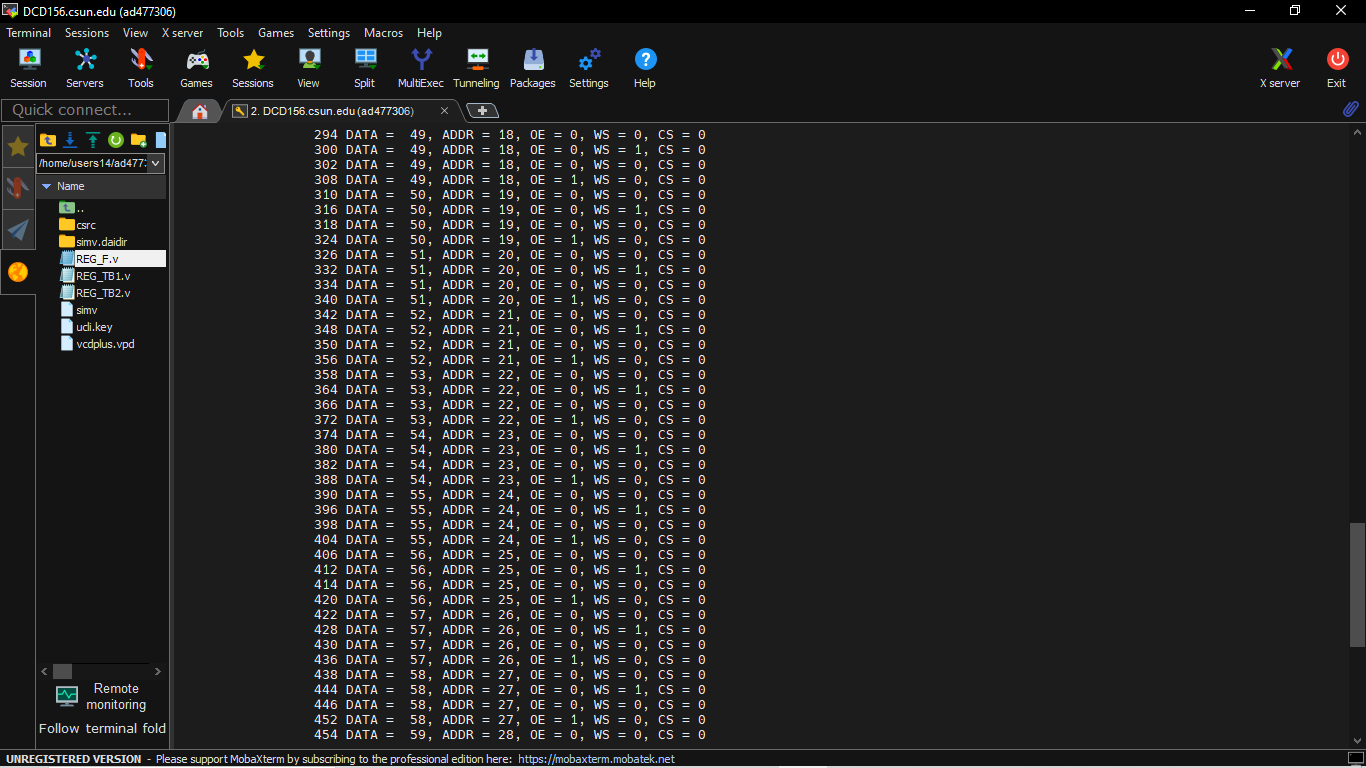
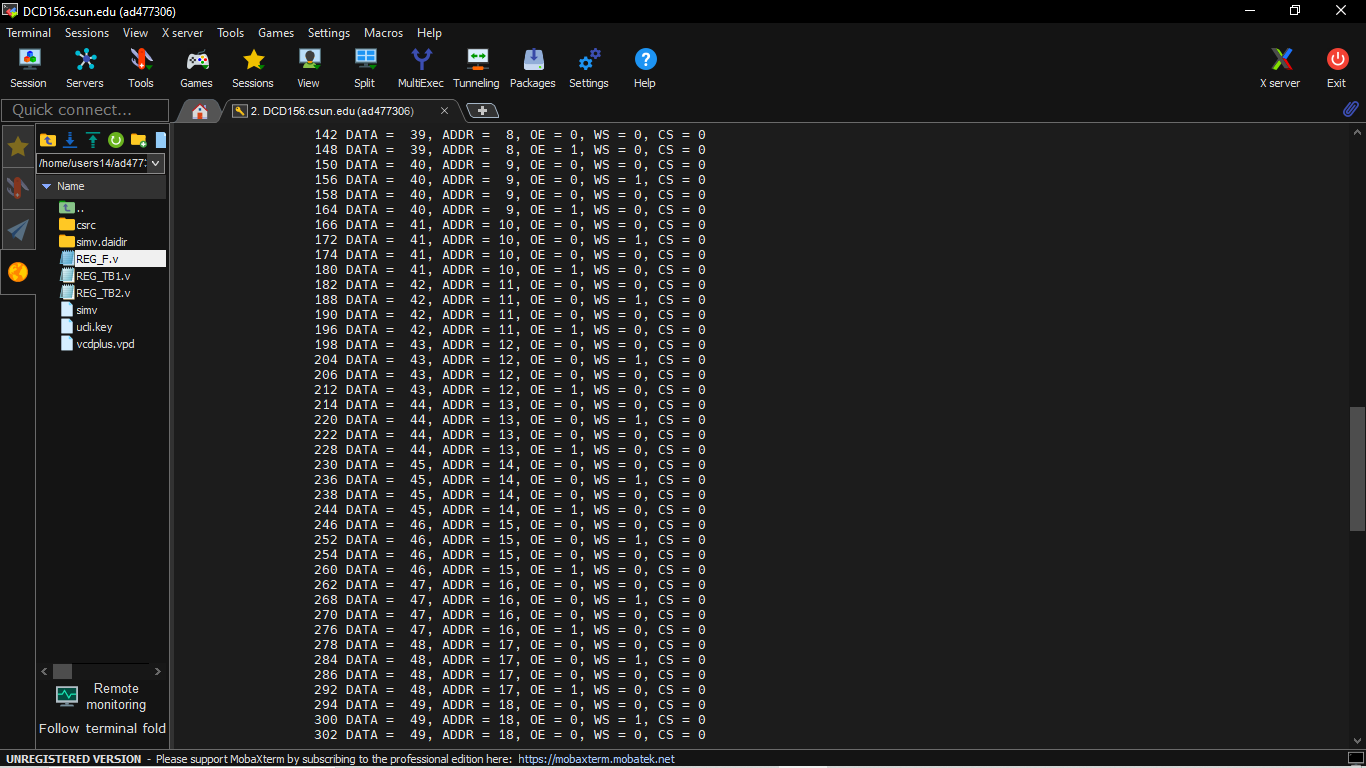
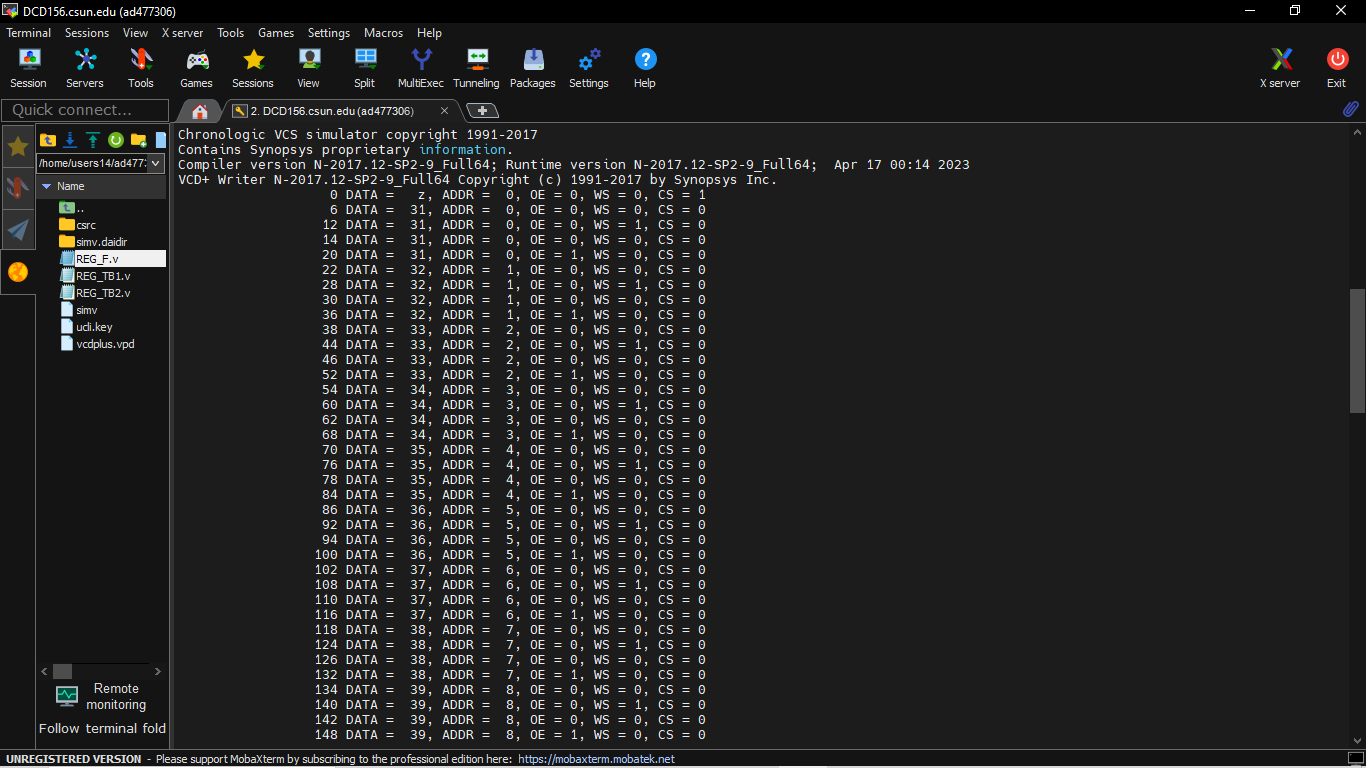
Using “vcs -debug -full64 REG\_F.v REG\_TB1.v” command I executed first testbench file.

Again using “vcs -debug -full64 REG\_F.v REG\_TB2.v” command I executed second testbench file.

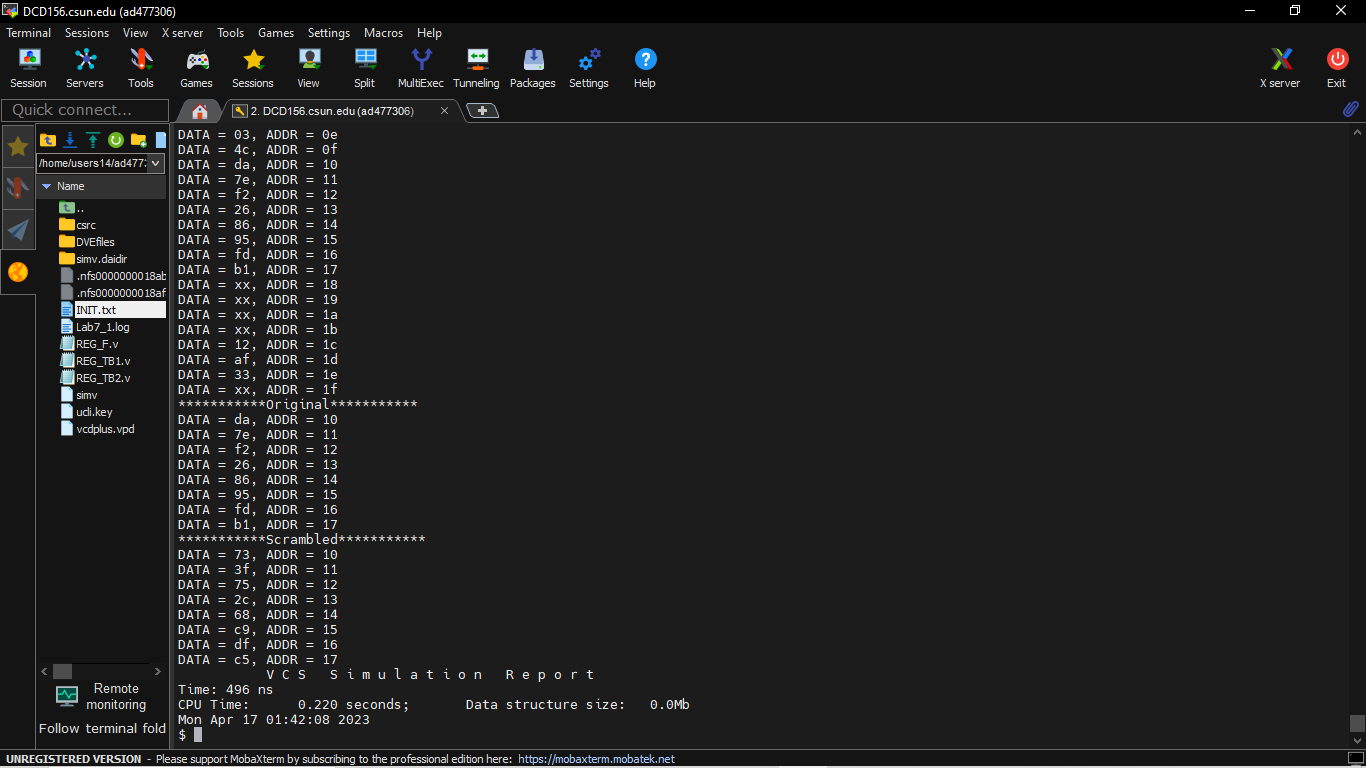
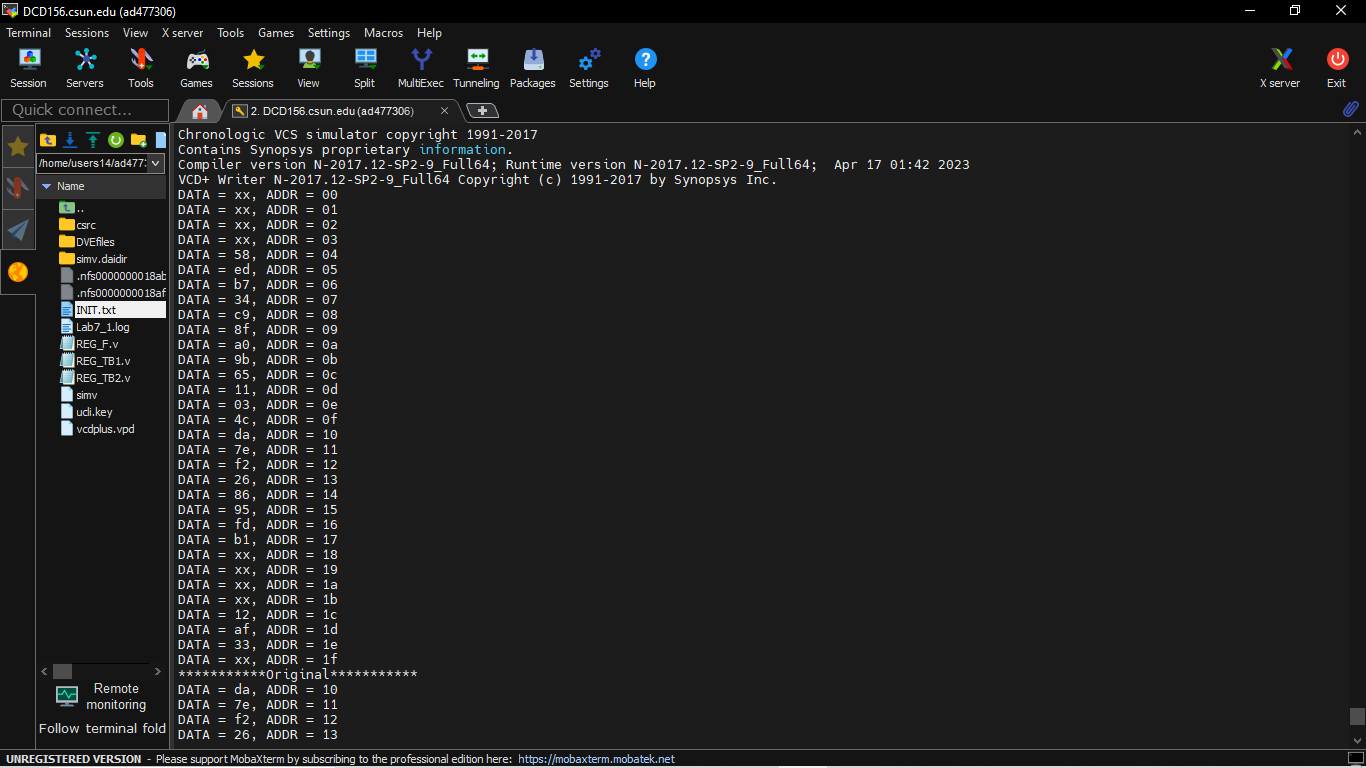
**d. Part 4: Simulation**

After an execution of all modules, I have run the command “simv” for simulation.

**Simulation for Test Strategy 1:**



**Simulation for Test Strategy 2:**



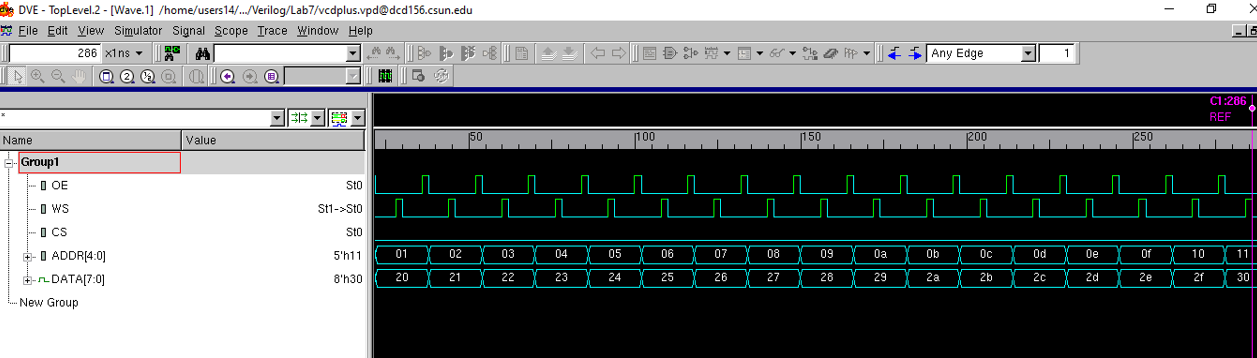
**e. Part 5: Creating Log File**

After running the simulation I created the log file using the “simv -l Lab6.log” command.

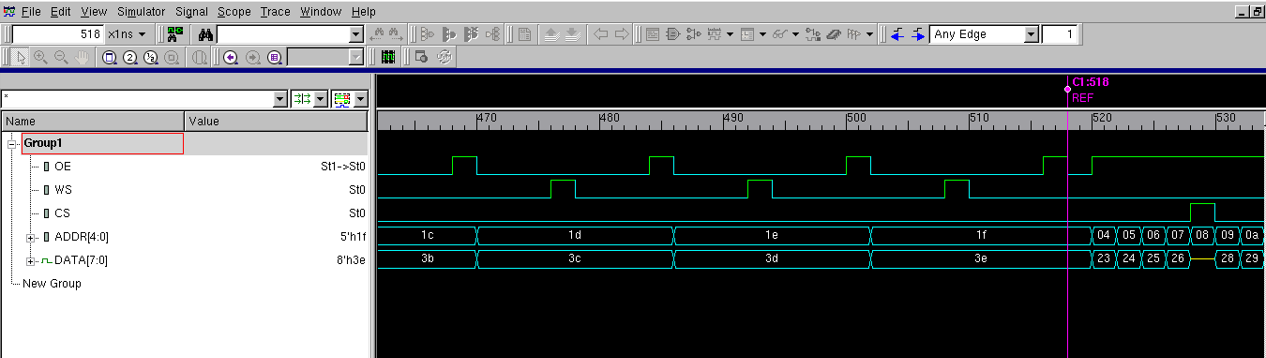
**f. Part 6: Seeing the waveform.**

After creating the log file I opened the DVE using “dve -full64 &” command to see the waveforms.

**Test Strategy 1:**

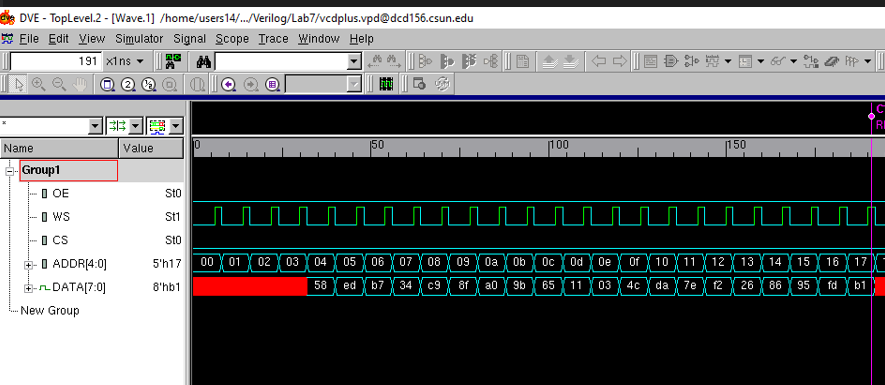
****

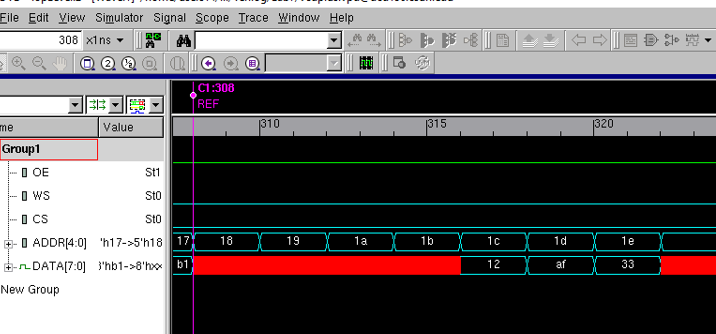
First, the register file was written to with unique values for each address using the first testbench. As shown above, the address starts at 00 (in hex), while the values assigned start at 31 (or 1F in hex) and incremented by 1 each time. Then each address was read from individually after writing to. Note that the data line only started storing upon the enabling of the chip select.

****

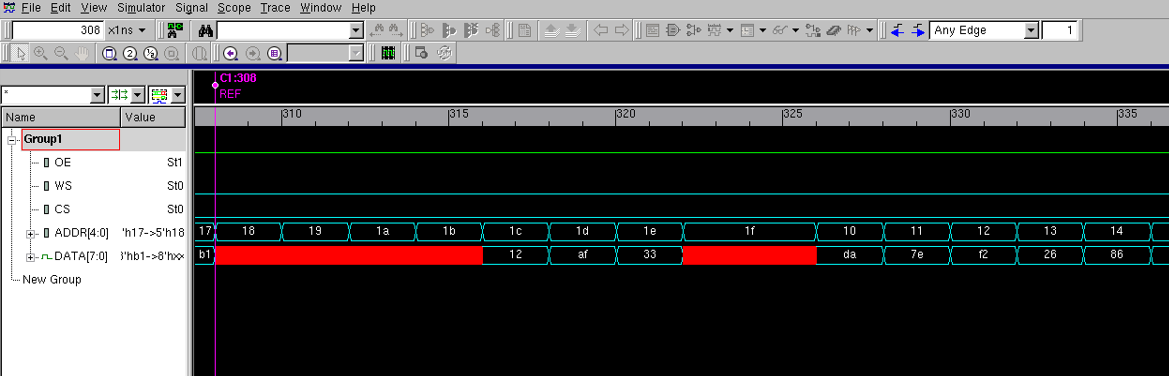
As shown above, the register file’s address ends at 31 (or 1F hex), corresponding to the default width of the register of 32. Each address was filled with unique values.

**Test Strategy 2:**

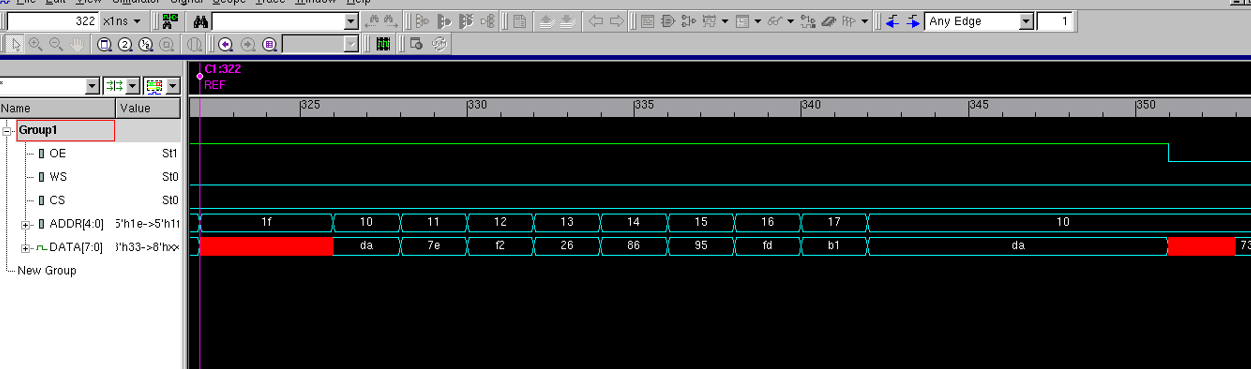
****

****

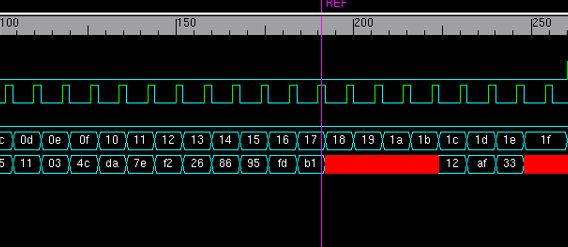
Shown above is the waveform that resulted from applying the second test bench. First the memory in the register model was initialized with the contents of a separate file (“INIT.txt”). All memory locations were accessed, but those without a value as specified in the data file, or have unspecified locations, remained uninitialized. This can be seen in the waveform above as the x construct, i.e. addresses 18 to 1b are x or uninitialized. Shown below is the period that output enable was triggered to high. The values stored were confirmed at this point as the data line is fed with these values during the read cycle.

****

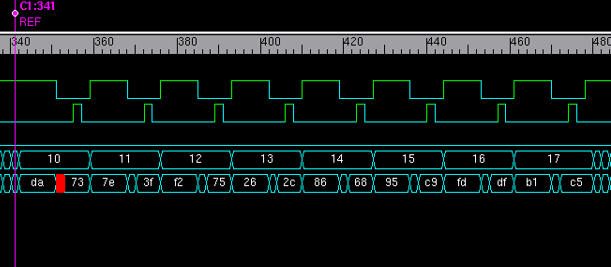
The next functionality demonstrated by the test bench involves scrambling part of the data in the memory. First, an address was read from memory during the read cycle. As the cycle was turned off, the value was scrambled according to instructions. (Below)

****

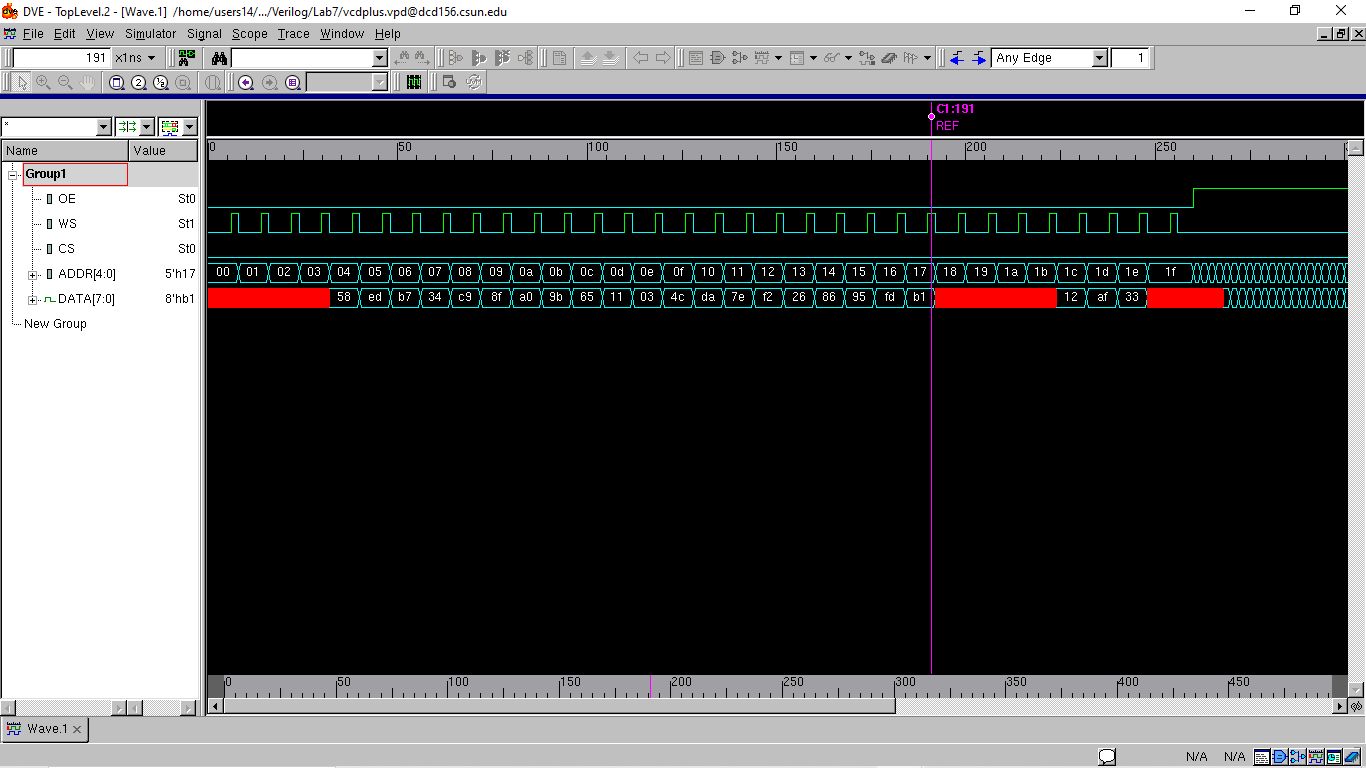
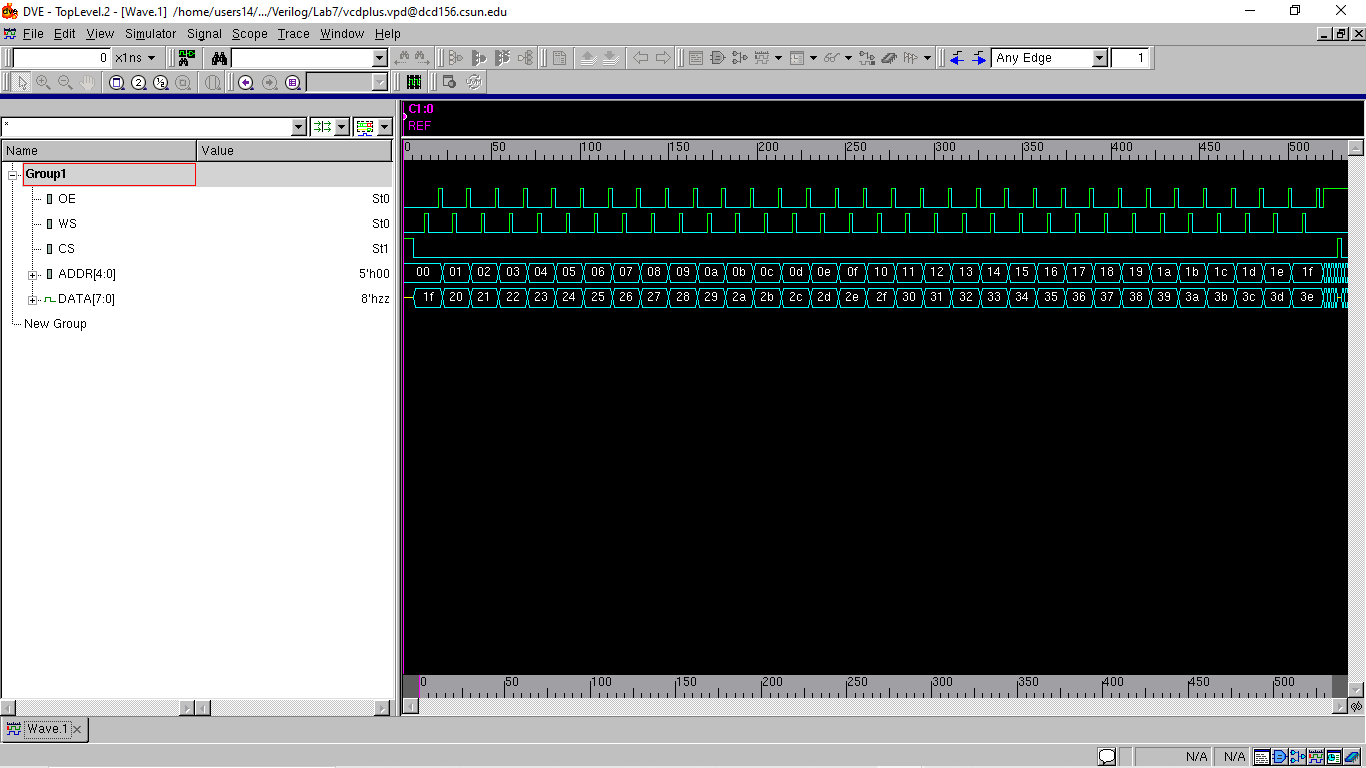
**Before Scrambling**

****

**After Scrambling**

****

**Waveforms:**



Lab report Extra Credit question: How many edge constructs do you use in your models?

Ans -

I used on edge construct in this model which is “posedge WS”.

**Conclusion:**

The register file model was successfully tested of its storing capabilities. The bidirectional data line also functioned as intended. All memory locations were successfully written to and read from, while also confirming the overriding functionality of the chip select line (memory enable). Individual and block read capabilities were also tested and confirmed. The read and write functions were confirmed of their functionality further using the second test bench and scrambling a segment of the memory. In this experiment, only one edge construct was used in the register file module.

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, nor have I allowed or will I allow anyone to copy my work.

Name (printed) Avinash Damse

Name(signed) Date : 16-April-2023

