

CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Department of Electrical and Computer Engineering

ECE 526L

LAB – 8: Register file modelling.

Written By: Raj Kumar

Introduction:

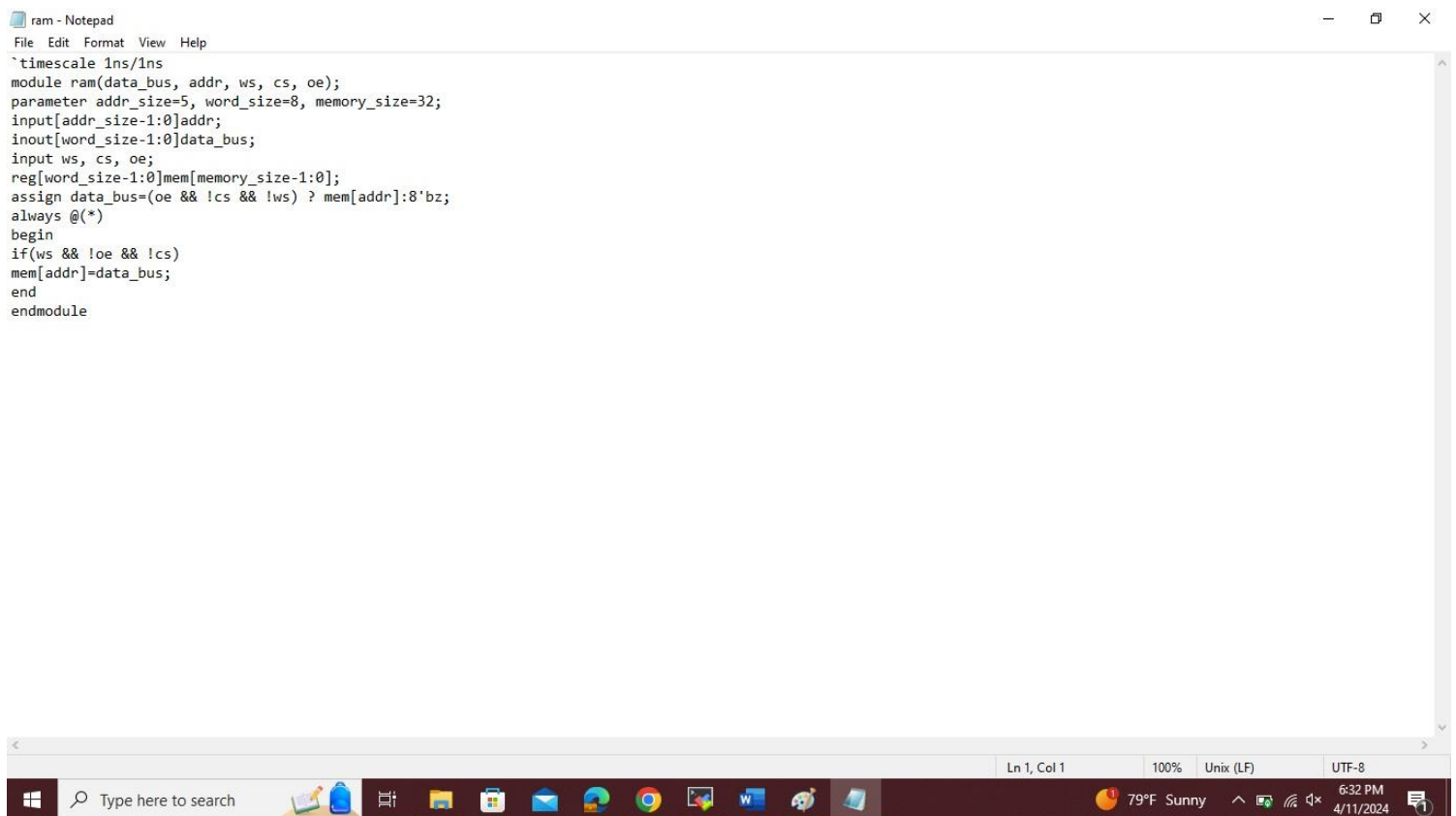
In this lab we will create a Register file model by using **Synopsys VCS** on **Linux OS environment** and different terminal commands.

Methodology:

It is being familiar with the **Linux** and **Synopsys VCS** that will navigate your system using terminal. Now to write a Verilog code for the module using a text editor. Use Linux based text editors and the file should have “.v” extension.

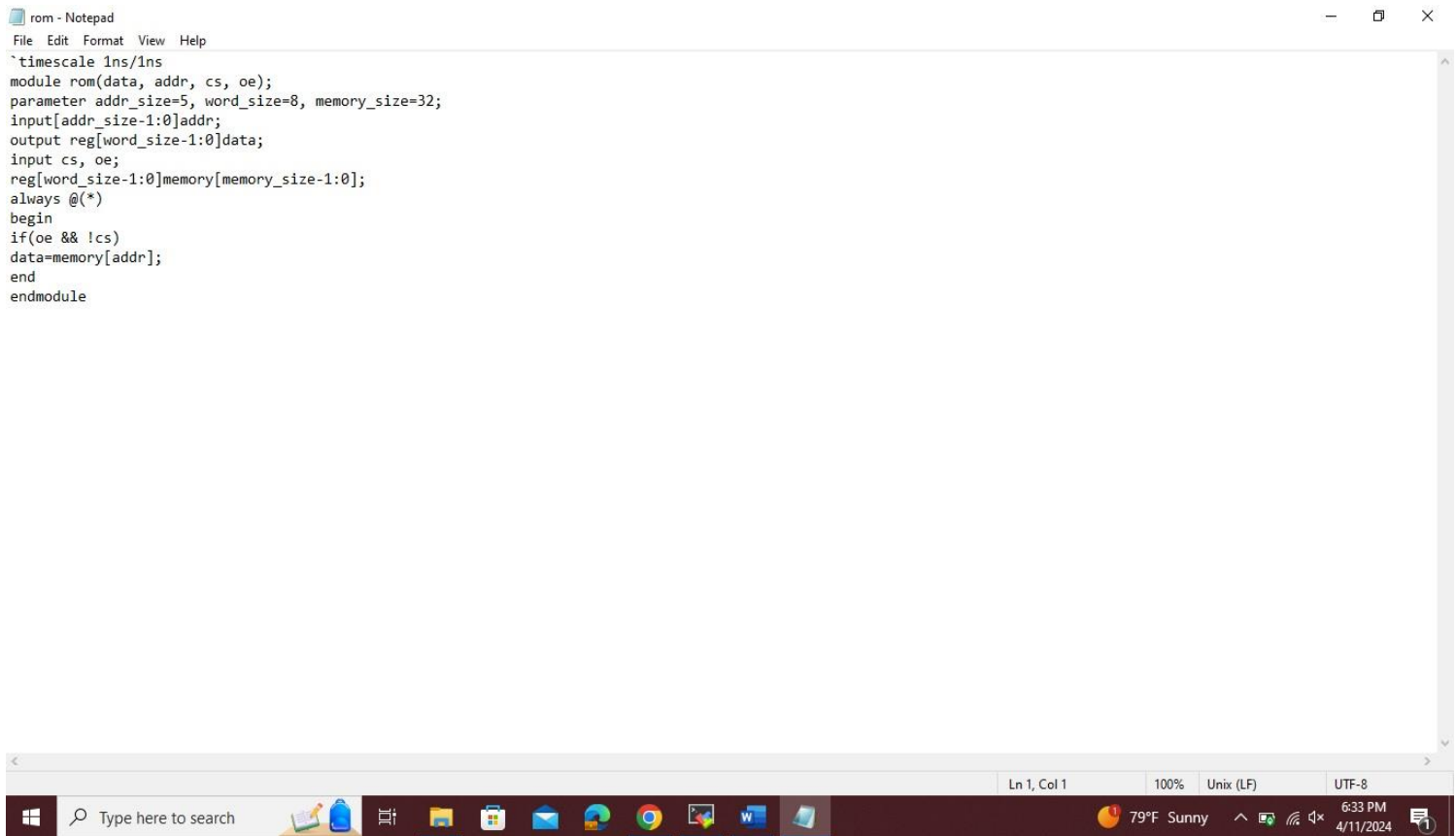
In this lab, we will create a register file that will be used as a **random-access memory**. Use parameters for both width and depth in our models and for the ram and create a Verilog model of a register file with the following specifications. Now, save as “**ram.v**”.

The module name and file name should be consistent.

A screenshot of a Windows Notepad application window titled 'ram - Notepad'. The window contains Verilog code for a RAM module. The code defines parameters for address size (5), word size (8), and memory size (32). It declares an input address, an input data bus, and control signals (ws, cs, oe). A memory array 'mem' is declared with dimensions [word_size-1:0] by [memory_size-1:0]. The code uses an 'assign' statement to connect the data bus to the memory output and an 'always' block to handle write operations when both 'ws' and 'oe' are active. The status bar at the bottom of the Notepad window shows 'Ln 1, Col 1', '100%', 'Unix (LF)', and 'UTF-8'. Below the Notepad window, a portion of the Windows taskbar is visible, showing the search bar, task view button, and several application icons. The system tray on the right shows the date and time as '6:32 PM 4/11/2024' and the weather as '79°F Sunny'.

This is the image of “**ram.v**”, store it in new file, also in folder “**Lab8**”.

We will create a second model that has no write capability to be used as **read only memory (ROM)**. Verilog system tasks **\$readmemb** and or **\$readmemh** will be used to set values in the module.

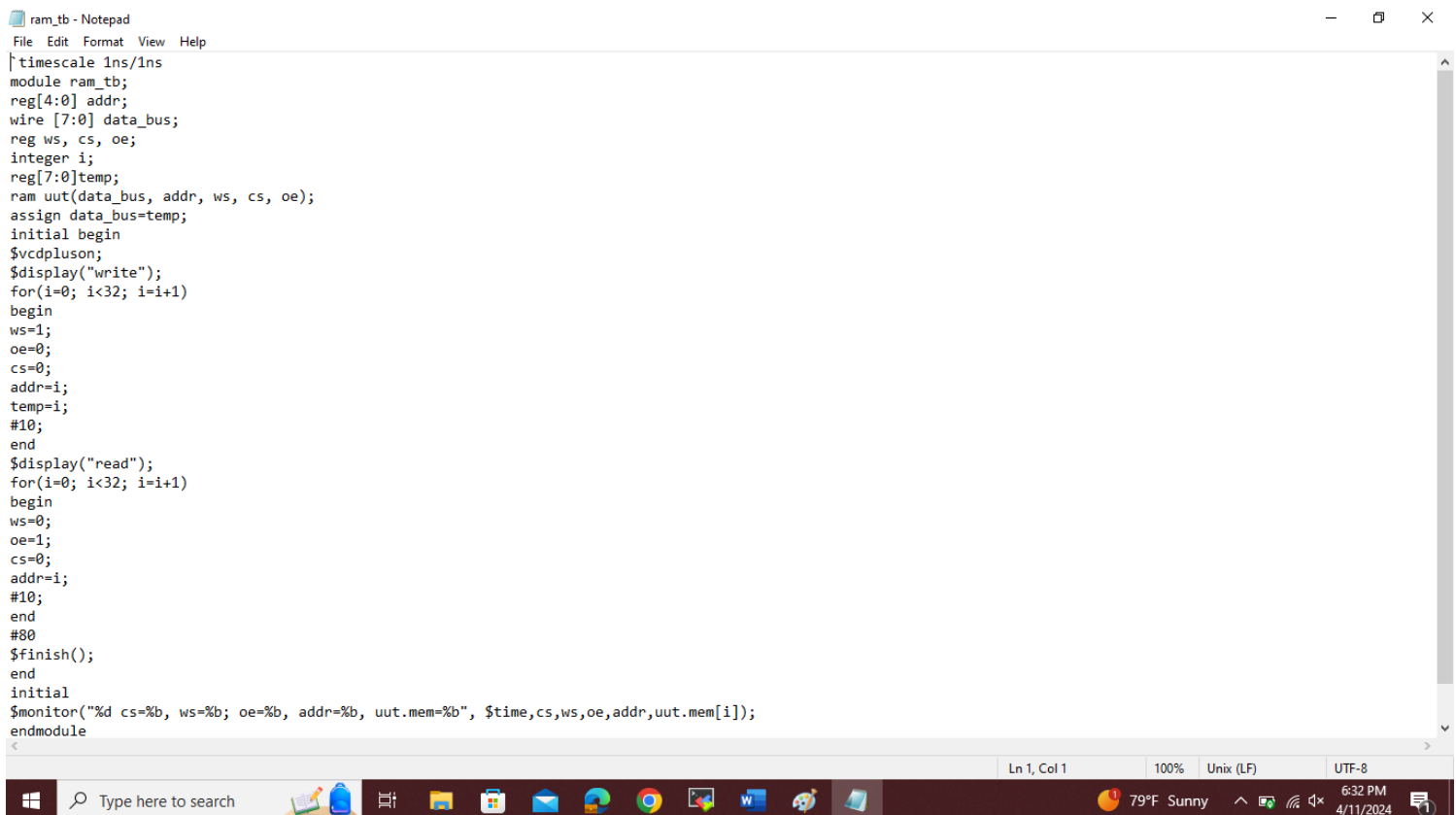


```
rom - Notepad
File Edit Format View Help
`timescale 1ns/1ns
module rom(data, addr, cs, oe);
parameter addr_size=5, word_size=8, memory_size=32;
input[addr_size-1:0]addr;
output reg[word_size-1:0]data;
input cs, oe;
reg[word_size-1:0]memory[memory_size-1:0];
always @(*)
begin
if(oe && !cs)
data=memory[addr];
end
endmodule
```

Ln 1, Col 1 100% Unix (LF) UTF-8 79°F Sunny 6:33 PM 4/11/2024

The above image is of “**rom.v**”.

We create a verilog testbench for the ram.v, it must do the following things as per the instructions and save it as “**ram_tb.v**”.



```
ram_tb - Notepad
File Edit Format View Help
`timescale 1ns/1ns
module ram_tb;
reg[4:0] addr;
wire [7:0] data_bus;
reg ws, cs, oe;
integer i;
reg[7:0]temp;
ram uut(data_bus, addr, ws, cs, oe);
assign data_bus=temp;
initial begin
$vcddpluson;
$display("write");
for(i=0; i<32; i=i+1)
begin
ws=1;
oe=0;
cs=0;
addr=i;
temp=i;
#10;
end
$display("read");
for(i=0; i<32; i=i+1)
begin
ws=0;
oe=1;
cs=0;
addr=i;
#10;
end
#80
$finish();
end
initial
$monitor("%d cs=%b, ws=%b, oe=%b, addr=%b, uut.mem=%b", $time,cs,ws,oe,addr,uut.mem[i]);
endmodule
```

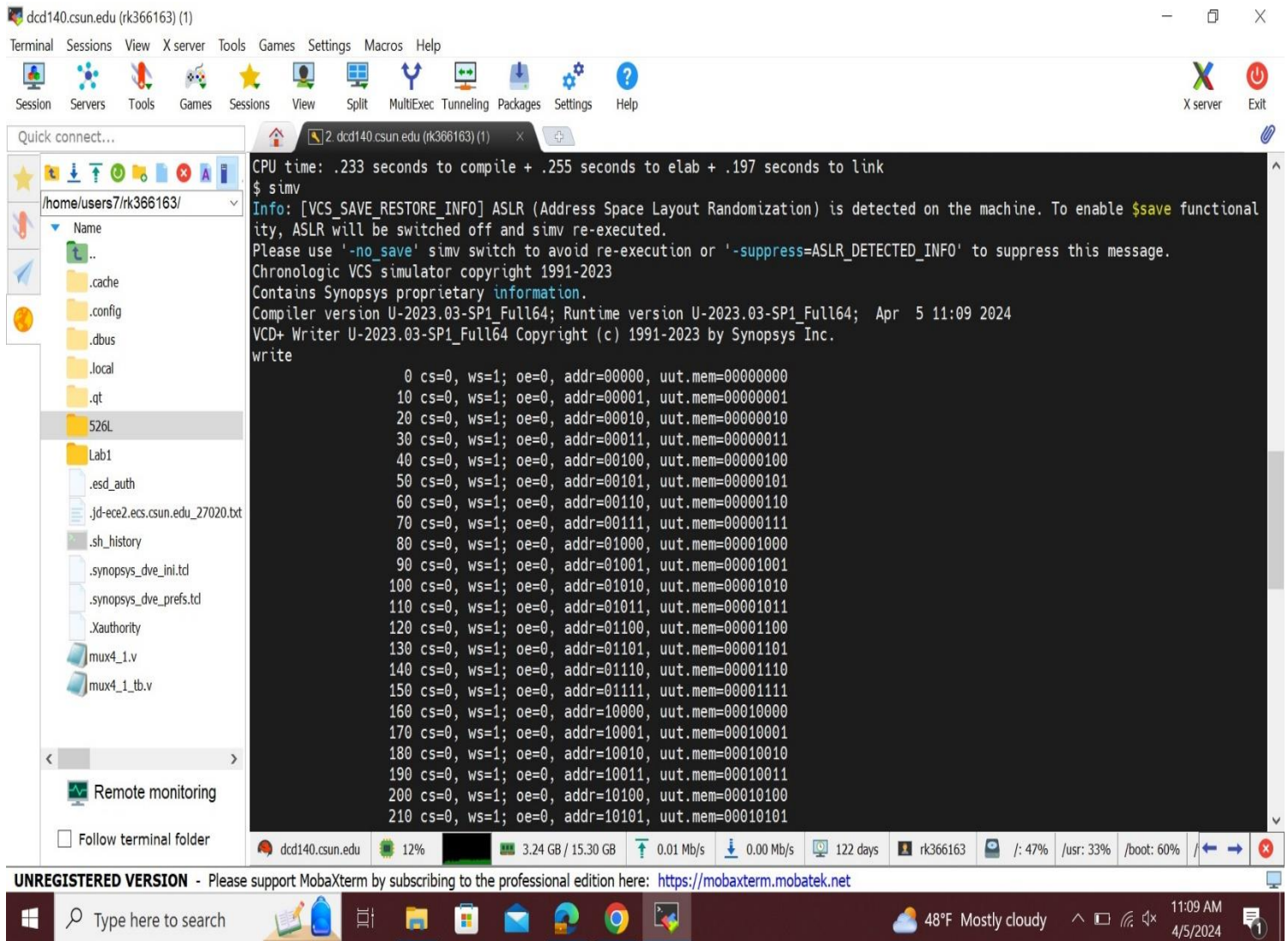
Ln 1, Col 1 100% Unix (LF) UTF-8 79°F Sunny 6:32 PM 4/11/2024

This above image is of “ram_tb.v”.

Then use the command as follows: “vcs -debug_access+all ram.v rom.v ram_tb.v ”.

If we don’t see any error messages if everything is type correctly then output should look like below figure.

simv:



```
dcd140.csun.edu (rk366163) (1)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
/home/users7/rk366163/
Name
..
.cache
.config
.dbus
.local
.qt
526L
Lab1
.esd_auth
.jd-ec2.ecs.csun.edu_27020.bt
.sh_history
.synopsys_dve_ini.td
.synopsys_dve_prefs.td
.Xauthority
mux4_1.v
mux4_1_tb.v
Remote monitoring
Follow terminal folder

CPU time: .233 seconds to compile + .255 seconds to elab + .197 seconds to link
$ simv
Info: [VCS_SAVE_RESTORE_INFO] ASLR (Address Space Layout Randomization) is detected on the machine. To enable $save functional
ity, ASLR will be switched off and simv re-executed.
Please use '-no_save' simv switch to avoid re-execution or '-suppress=ASLR_DETECTED_INFO' to suppress this message.
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version U-2023.03-SP1_Full64; Runtime version U-2023.03-SP1_Full64; Apr 5 11:09 2024
VCD+ Writer U-2023.03-SP1_Full64 Copyright (c) 1991-2023 by Synopsys Inc.
write
0 cs=0, ws=1; oe=0, addr=00000, uut.mem=00000000
10 cs=0, ws=1; oe=0, addr=00001, uut.mem=00000001
20 cs=0, ws=1; oe=0, addr=00010, uut.mem=00000010
30 cs=0, ws=1; oe=0, addr=00011, uut.mem=00000011
40 cs=0, ws=1; oe=0, addr=00100, uut.mem=00000100
50 cs=0, ws=1; oe=0, addr=00101, uut.mem=00000101
60 cs=0, ws=1; oe=0, addr=00110, uut.mem=00000110
70 cs=0, ws=1; oe=0, addr=00111, uut.mem=00000111
80 cs=0, ws=1; oe=0, addr=01000, uut.mem=00001000
90 cs=0, ws=1; oe=0, addr=01001, uut.mem=00001001
100 cs=0, ws=1; oe=0, addr=01010, uut.mem=00001010
110 cs=0, ws=1; oe=0, addr=01011, uut.mem=00001011
120 cs=0, ws=1; oe=0, addr=01100, uut.mem=00001100
130 cs=0, ws=1; oe=0, addr=01101, uut.mem=00001101
140 cs=0, ws=1; oe=0, addr=01110, uut.mem=00001110
150 cs=0, ws=1; oe=0, addr=01111, uut.mem=00001111
160 cs=0, ws=1; oe=0, addr=10000, uut.mem=00010000
170 cs=0, ws=1; oe=0, addr=10001, uut.mem=00010001
180 cs=0, ws=1; oe=0, addr=10010, uut.mem=00010010
190 cs=0, ws=1; oe=0, addr=10011, uut.mem=00010011
200 cs=0, ws=1; oe=0, addr=10100, uut.mem=00010100
210 cs=0, ws=1; oe=0, addr=10101, uut.mem=00010101
```

Terminal window showing a file listing on the left and a hex dump output in the terminal. The terminal output shows memory addresses and values for various components.

```
200 cs=0, ws=1; oe=0, addr=10100, uut.mem=00010100
210 cs=0, ws=1; oe=0, addr=10101, uut.mem=00010101
220 cs=0, ws=1; oe=0, addr=10110, uut.mem=00010110
230 cs=0, ws=1; oe=0, addr=10111, uut.mem=00010111
240 cs=0, ws=1; oe=0, addr=11000, uut.mem=00011000
250 cs=0, ws=1; oe=0, addr=11001, uut.mem=00011001
260 cs=0, ws=1; oe=0, addr=11010, uut.mem=00011010
270 cs=0, ws=1; oe=0, addr=11011, uut.mem=00011011
280 cs=0, ws=1; oe=0, addr=11100, uut.mem=00011100
290 cs=0, ws=1; oe=0, addr=11101, uut.mem=00011101
300 cs=0, ws=1; oe=0, addr=11110, uut.mem=00011110
310 cs=0, ws=1; oe=0, addr=11111, uut.mem=00011111

read

320 cs=0, ws=0; oe=1, addr=00000, uut.mem=00000000
330 cs=0, ws=0; oe=1, addr=00001, uut.mem=00000001
340 cs=0, ws=0; oe=1, addr=00010, uut.mem=00000010
350 cs=0, ws=0; oe=1, addr=00011, uut.mem=00000011
360 cs=0, ws=0; oe=1, addr=00100, uut.mem=00000100
370 cs=0, ws=0; oe=1, addr=00101, uut.mem=00000101
380 cs=0, ws=0; oe=1, addr=00110, uut.mem=00000110
390 cs=0, ws=0; oe=1, addr=00111, uut.mem=00000111
400 cs=0, ws=0; oe=1, addr=01000, uut.mem=00001000
410 cs=0, ws=0; oe=1, addr=01001, uut.mem=00001001
420 cs=0, ws=0; oe=1, addr=01010, uut.mem=00001010
430 cs=0, ws=0; oe=1, addr=01011, uut.mem=00001011
440 cs=0, ws=0; oe=1, addr=01100, uut.mem=00001100
450 cs=0, ws=0; oe=1, addr=01101, uut.mem=00001101
460 cs=0, ws=0; oe=1, addr=01110, uut.mem=00001110
470 cs=0, ws=0; oe=1, addr=01111, uut.mem=00001111
480 cs=0, ws=0; oe=1, addr=10000, uut.mem=00010000
490 cs=0, ws=0; oe=1, addr=10001, uut.mem=00010001
500 cs=0, ws=0; oe=1, addr=10010, uut.mem=00010010
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

Terminal window showing a file listing on the left and a hex dump output in the terminal. The terminal output shows memory addresses and values for various components, followed by a simulation report.

```
400 cs=0, ws=0; oe=1, addr=01000, uut.mem=00001000
410 cs=0, ws=0; oe=1, addr=01001, uut.mem=00001001
420 cs=0, ws=0; oe=1, addr=01010, uut.mem=00001010
430 cs=0, ws=0; oe=1, addr=01011, uut.mem=00001011
440 cs=0, ws=0; oe=1, addr=01100, uut.mem=00001100
450 cs=0, ws=0; oe=1, addr=01101, uut.mem=00001101
460 cs=0, ws=0; oe=1, addr=01110, uut.mem=00001110
470 cs=0, ws=0; oe=1, addr=01111, uut.mem=00001111
480 cs=0, ws=0; oe=1, addr=10000, uut.mem=00010000
490 cs=0, ws=0; oe=1, addr=10001, uut.mem=00010001
500 cs=0, ws=0; oe=1, addr=10010, uut.mem=00010010
510 cs=0, ws=0; oe=1, addr=10011, uut.mem=00010011
520 cs=0, ws=0; oe=1, addr=10100, uut.mem=00010100
530 cs=0, ws=0; oe=1, addr=10101, uut.mem=00010101
540 cs=0, ws=0; oe=1, addr=10110, uut.mem=00010110
550 cs=0, ws=0; oe=1, addr=10111, uut.mem=00010111
560 cs=0, ws=0; oe=1, addr=11000, uut.mem=00011000
570 cs=0, ws=0; oe=1, addr=11001, uut.mem=00011001
580 cs=0, ws=0; oe=1, addr=11010, uut.mem=00011010
590 cs=0, ws=0; oe=1, addr=11011, uut.mem=00011011
600 cs=0, ws=0; oe=1, addr=11100, uut.mem=00011100
610 cs=0, ws=0; oe=1, addr=11101, uut.mem=00011101
620 cs=0, ws=0; oe=1, addr=11110, uut.mem=00011110
630 cs=0, ws=0; oe=1, addr=11111, uut.mem=00011111
640 cs=0, ws=0; oe=1, addr=11111, uut.mem=xxxxxxxx

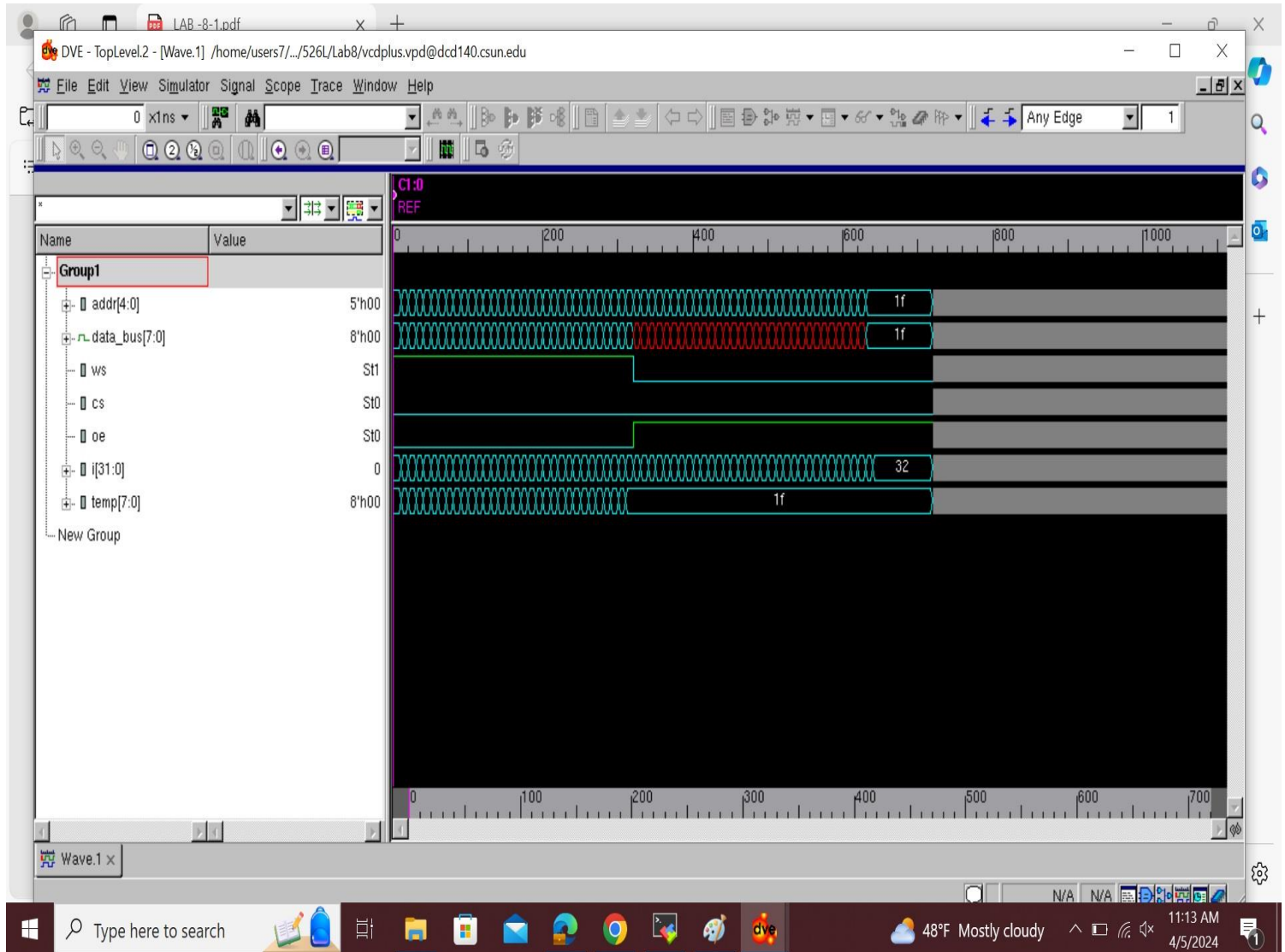
$finish called from file "ram_tb.v", line 32.
$finish at simulation time 720
VCS Simulation Report
Time: 720 ns
CPU Time: 0.210 seconds; Data structure size: 0.0Mb
Fri Apr 5 11:09:04 2024
$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

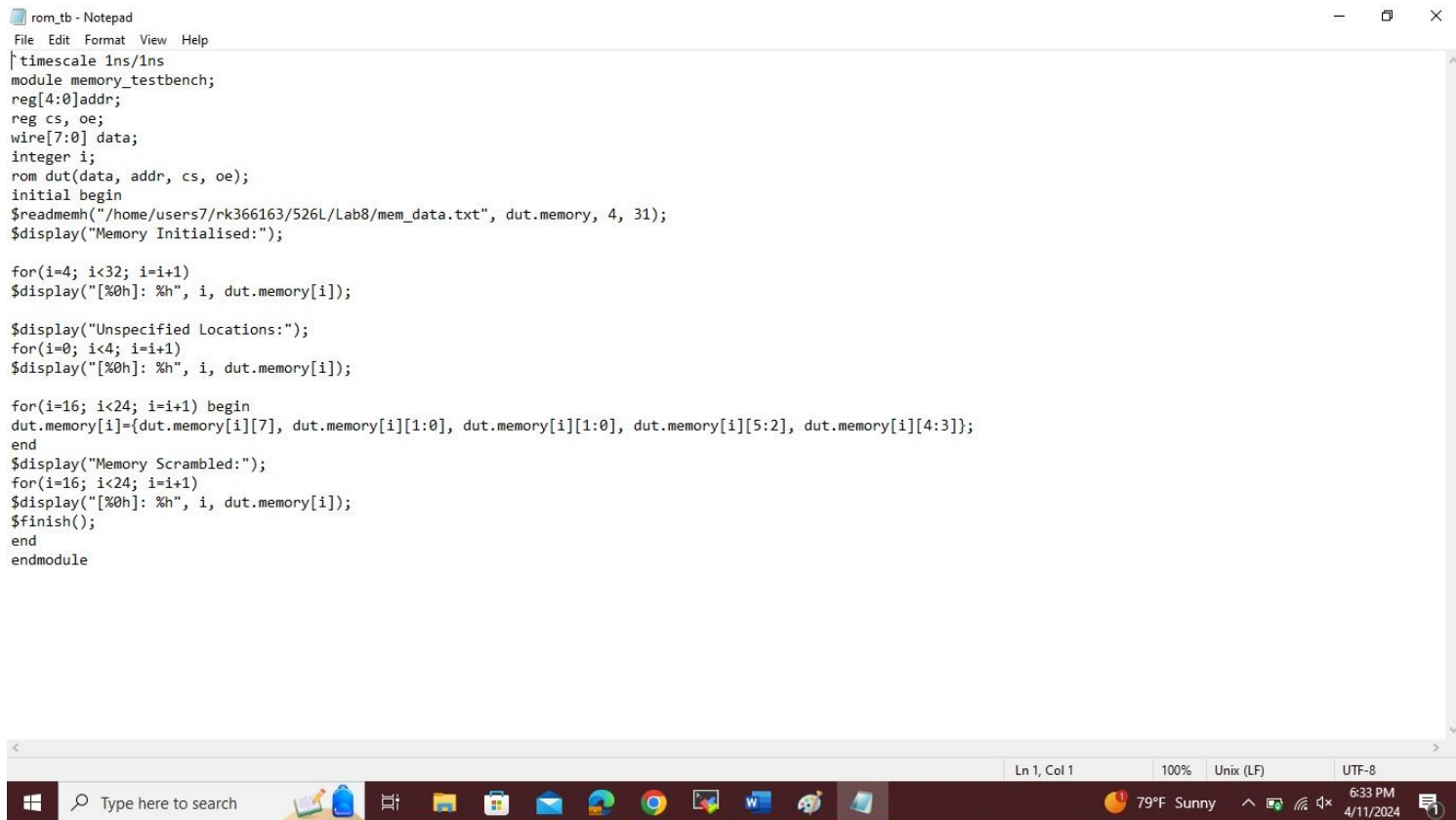
To see the wave form first type “dve” command line then we will get the blank simulation window then go to file and open database then choose **vcdplus.vpd** and open the file.

Then finally select all the signals with the mouse, then right click and select to “add to wave->new wave view”.

Then we will get the waveform as shown in below image;



Now we should create a testbench for rom based on the instructions given and save it as “rom_tb.v”.



```
rom_tb - Notepad
File Edit Format View Help
|timescale 1ns/1ns
module memory_testbench;
reg[4:0]addr;
reg cs, oe;
wire[7:0] data;
integer i;
rom dut(data, addr, cs, oe);
initial begin
$readmemh("/home/users7/rk366163/526L/Lab8/mem_data.txt", dut.memory, 4, 31);
$display("Memory Initialised:");

for(i=4; i<32; i=i+1)
$display("[%0h]: %h", i, dut.memory[i]);

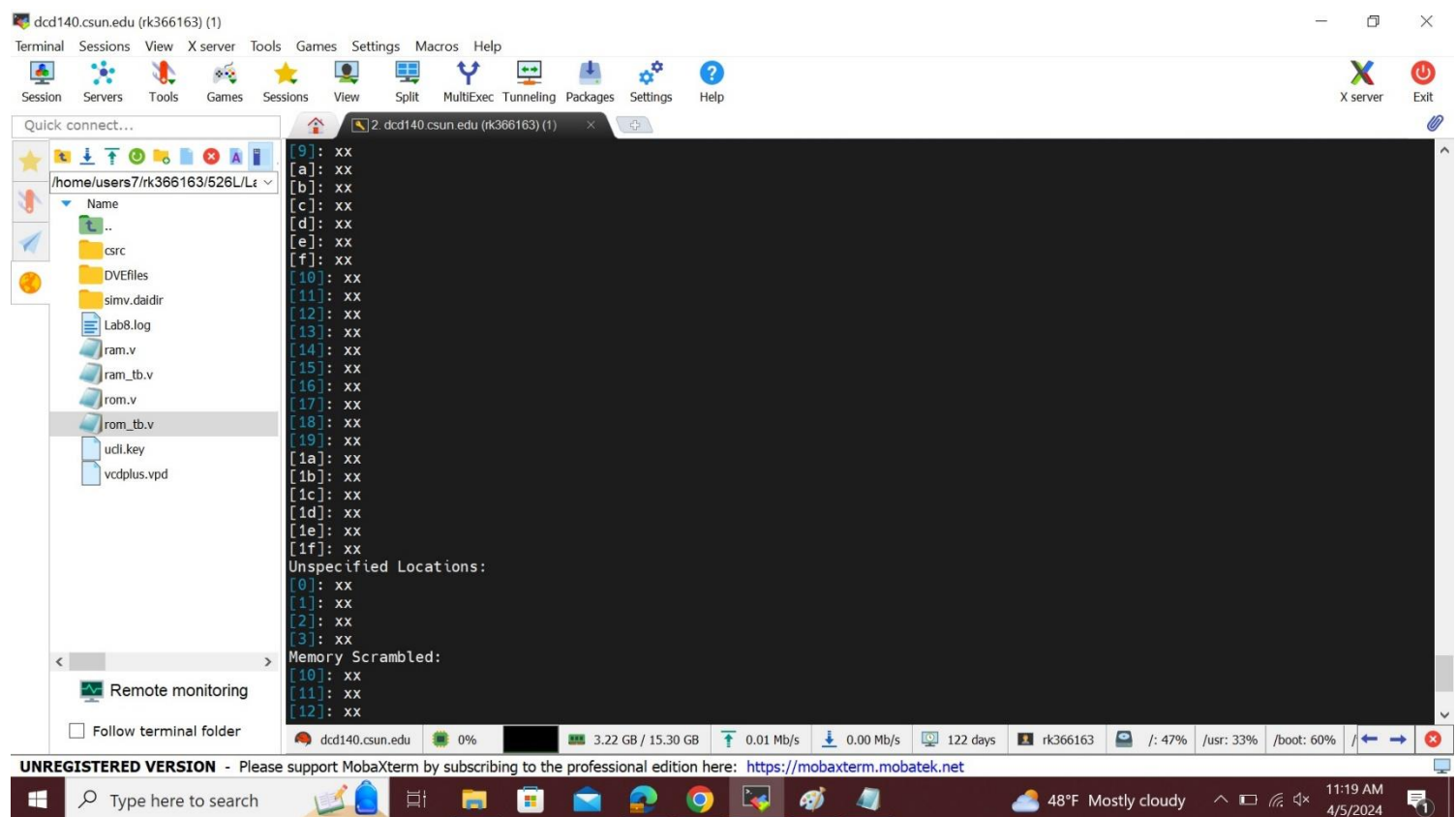
$display("Unspecified Locations:");
for(i=0; i<4; i=i+1)
$display("[%0h]: %h", i, dut.memory[i]);

for(i=16; i<24; i=i+1) begin
dut.memory[i]={dut.memory[i][1:0], dut.memory[i][1:0], dut.memory[i][1:0], dut.memory[i][5:2], dut.memory[i][4:3]};
end
$display("Memory Scrambled:");
for(i=16; i<24; i=i+1)
$display("[%0h]: %h", i, dut.memory[i]);
$finish();
end
endmodule
```

Ln 1, Col 1 100% Unix (LF) UTF-8 79°F Sunny 6:33 PM 4/11/2024

Then use the command as follows: **“vcs -debug_access+all ram.v rom.v rom_tb.v”**.

If we don't see any error messages if everything is type correctly then output should look like below image;



dcd140.csun.edu (rk366163) (1)

Terminal Sessions View X server Tools Games Settings Macros Help

Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help

Quick connect... /home/users7/rk366163/526L/Lt

Name

- ...
- csrc
- DVEfiles
- simv.daldir
- Lab8.log
- ram.v
- ram_tb.v
- rom.v
- rom_tb.v
- ucl.key
- vcdplus.vpd

Remote monitoring

Follow terminal folder

```
[9]: xx
[a]: xx
[b]: xx
[c]: xx
[d]: xx
[e]: xx
[f]: xx
[10]: xx
[11]: xx
[12]: xx
[13]: xx
[14]: xx
[15]: xx
[16]: xx
[17]: xx
[18]: xx
[19]: xx
[1a]: xx
[1b]: xx
[1c]: xx
[1d]: xx
[1e]: xx
[1f]: xx
Unspecified Locations:
[0]: xx
[1]: xx
[2]: xx
[3]: xx
Memory Scrambled:
[10]: xx
[11]: xx
[12]: xx
```

dcd140.csun.edu 0% 3.22 GB / 15.30 GB 0.01 Mb/s 0.00 Mb/s 122 days rk366163 /: 47% /usr: 33% /boot: 60%

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

48°F Mostly cloudy 11:19 AM 4/5/2024

Analysis of result:

The answer explains the design and implementation of a Verilog register file that serves as a random-access memory (RAM). It has a five-bit address bus, a bidirectional data bus, active low chip-select (cs), active high output-enable (oe), and a write strobe (ws) for writing data to memory. In order to verify enabled and disabled states, read and write to every memory address, show individual and block reads, and output a walking one's pattern to test the independence of each output bit, a separate Verilog testbench is constructed. Using the \$readmemh system task, a different testbench initializes the memory with values, shows that initialization was successful, jumbles the bytes in the designated memory locations, and prints the contents of every memory location.

Conclusion:

In conclusion, the Verilog modeling of a register file using Synopsys VCS on the Linux operating system has provided valuable insights into digital design and simulation methodologies. Through this project, we gained a deeper understanding of the hierarchical structure of a register file, its functionality in data storage and retrieval, and the implementation of such a design in a hardware description language.

The process involved writing Verilog code to define the register file's behavior and interfacing it with testbenches to verify its functionality. Synopsys VCS served as a reliable tool for compiling and simulating the Verilog code, allowing us to perform extensive testing and validation.

Throughout the project, we encountered challenges such as ensuring proper data integrity, managing clock cycles for synchronous operations, and optimizing the design for performance and resource utilization. By addressing these challenges, we were able to develop a robust and efficient register file model.

This experience has not only enhanced our Verilog programming skills but also provided practical exposure to industry-standard tools like Synopsys VCS. It underscores the importance of rigorous testing and verification in digital design projects, paving the way for future advancements in hardware design and simulation.

Overall, the successful completion of this lab report signifies a significant milestone in our understanding of register file modeling and its application in digital systems design.

Lab Questions:

1. How many edge constructs do you use in your models? why that is best answer for this design?

Ans. The number of edge constructions (such as {always @(posedge clk)} or comparable ones) in your design is determined by the particular needs of your design. Since this is a basic testbench without a timed process, we are not using any in this example. The quantity and kind of edge constructs in a real-world situation rely on things like clock domains, the necessity for synchronization, and the kind of signal processing (combinational or sequential).

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, neither have I allowed nor I will let anyone to copy my work.

Name(printed): Raj Kumar

Name(signed): Raj Kumar

Date 04/11/2024