Ain Shams University

Faculty of Engineering

Postgraduate Program
(Master in Computer and Systems Engineering)

CSE620: Advanced Computer Architecture

Bonus Assignement

HDL Testbench

Register File

Elaborated By:

Eng. Youssef Nasser 2300315

Supervised by:

Prof. Mohamed Watheq Ali Kamel El-Kharashi

Contents

List of Figures

List of Tables

Register File Problem Statement

Using VHDL or Verilog model a register file that contains 2^n 32-bit registers. The register file has two read port and two write ports. Below is a symbol of this register file, showing all input and output signals with their bit-widths.

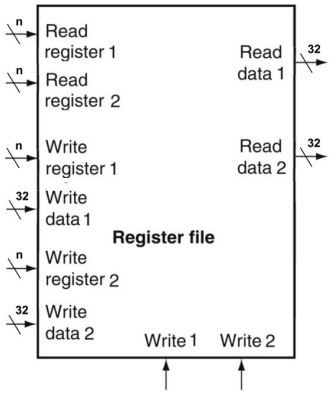

Figure 1 Dummy Register File

Circuit Diagram

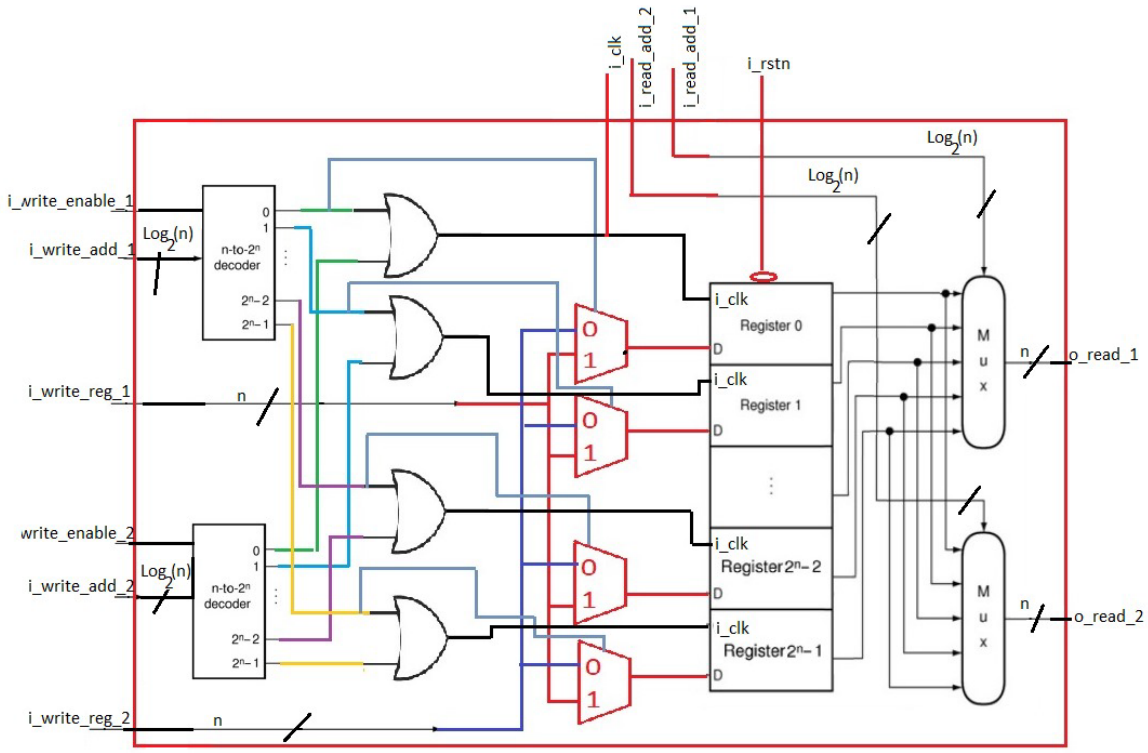

Figure 2 Circuit Diagram for Dual-Port Register File

If the two writing ports writes at the same address it will be written at port 1 by using Priority encoder.

RTL Code

```
/*********************************************************************************
 * Master Advanced Computer Architecture Course CSE620
 *
 * Module: Dual Port Register File
 *
 * Description:
 * This Verilog module implements a simple dual-port register file with
 * Asynchronous reset functionality. It allows simultaneous read and write
 * Operations on two ports. The register file is parameterized with data width
 * And address depth.
 *
 * Parameters:
 * - data_width: Width of each register in bits (default: 32)
 * - Address_depth: Number of registers in the file (default: 4)
 *
 * Ports:
 * - i_clk: Clock input for synchronous operations
 * - i_rstn: Active-low asynchronous reset input
 *
 * Write Port 1:
 * - i_write_enable_1: Write enable signal for port 1
 * - i_write_reg_1: Data input for write port 1
 * - i_write_add_1: Address input for write port 1
 *
 * Write Port 2:
 * - i_write_enable_2: Write enable signal for port 2
 * - i_write_reg_2: Data input for write port 2
 * - i_write_add_2: Address input for write port 2
 *
 * Read Port 1:
 * - i_read_add_1: Address input for read port 1
 * - o_read_1: Data output for read port 1
 *
 * Read Port 2:
 * - i_read_add_2: Address input for read port 2
 * - o_read_2: Data output for read port 2
 *
 * Memory Organization:
 * - The register file is implemented as an array named Dual_RF with data_width
 *   Bits for each register and 2^Address_depth registers.
 *
 * Reset Behavior:
 * - Upon assertion of the asynchronous reset (i_rstn), all registers are
 *   Cleared to 0.
 *
 * Write Operation:
 * - If both write ports are enabled simultaneously and have the same address,
 *   The data from write port 1 is written to the specified address.
 * - If only one write port is enabled, data is written to the specified address
 *   For that port.
 *
 * Read Operation:
 * - Data is read from the specified address for each read port independently.
 *
 * Student Name: Youssef Nasser
 * Student ID: 2300315
 * Date: 12/12/2023
 * Issued for: Master Final Project (Bonus)
 *********************************************************************************/
```

```
// Specify the timescale for simulation, with 1ns time units and 1ps
precision
`timescale 1ns/1ps
module regfile #(parameter data_width = 32 , Address_depth = 4) (
  input wire i_clk,     // Clock input
  input wire i_rstn,    // write port 1
  // Write port 1 inputs
  input wire i_write_enable_1,
  input wire [data_width-1:0] i_write_reg_1,
  input wire [Address_depth-1:0] i_write_add_1,
  //write port 2
  input wire i_write_enable_2,
  input wire  [data_width-1:0] i_write_reg_2,
  input wire  [Address_depth-1:0] i_write_add_2,
  // read port 1
  input wire  [Address_depth-1:0] i_read_add_1,
  output wire [data_width-1:0] o_read_1,
  // read port 2
  input wire  [Address_depth-1:0] i_read_add_2,
  output wire [data_width-1:0] o_read_2
);


  // Declare an array Dual_RF as a memory with specified data_width and
Address_depth
  reg [data_width-1:0] Dual_RF [0:((2**Address_depth)-1)]; // 0:15

  // Declare an integer variable i for loop indexing
  integer i;


  // Always block triggered by the positive edge of the clock or negative
edge of the asynchronous reset
  always@(posedge i_clk or negedge i_rstn) begin
    if (!i_rstn) begin
      // Reset condition: set all memory locations to 0
      for(i=0;i<(2**Address_depth);i=i+1)
        begin
          Dual_RF[i] <= 0;
        end
    end
```

```
else
      begin
      // Write Operation
      // handling the case when both write ports try to write to the same
register
        if(i_write_enable_1 && i_write_enable_2 && (i_write_add_1 ==
i_write_add_2) )
          begin
            Dual_RF[i_write_add_1] <= i_write_reg_1;
          end

        else
          begin
            if (i_write_enable_1)
              Dual_RF[i_write_add_1] <= i_write_reg_1;

            if (i_write_enable_2)
              Dual_RF[i_write_add_2] <= i_write_reg_2;

          end

      end

  end

  // Assign read outputs based on read addresses
  assign o_read_1 = Dual_RF[i_read_add_1];
  assign o_read_2 = Dual_RF[i_read_add_2];
endmodule
```

Test Strategy

Reset pin
i_rstn
Write Port 1 Pins
i_write_enable_1 , i_write_reg_1 , i_write_add_1

Write Port 2 Pins
i_write_enable_2 , i_write_reg_2 , i_write_add_2

Read Port 1 Pin
i_read_add_1

Read Port 2 Pin
i_read_add_2

First, we need to initialize inputs , then reset sequence and waiting a delta cycle to avoid metastability then running the following test vectors.

For initialization there is a task for it at the testbench code , reset sequence has a task too and the delta cycle is clock period / 10 .

Clock Frequency is 20 ns which is 1000/20 ➔ 50 MHz

Table 1 Test Strategy

Test ID	Test Feature	Inputs	Delay	Expected Output
1	Write and read interaction at port 1	Reset is 1 Write Port 1 is on Adds 0 , Data 10 Read address 1,2 are 0	44 ns	Read Port 1 -> 10 Read Port 1 -> 10
2	Write and read interaction at port 2	Reset is 1 Write Port 2 is on Adds 15 , Data 9 Read address 1,2 are 15	44 ns	Read Port 1 -> 9 Read Port 1 -> 9
3	Simultaneous Writes and Reads	Reset is 1 Write port 1 , 2 are on Adds1 is 4 and Adds 2 is 5 Data is 7 , 8 Read address 1,2 are 4 , 5	44 ns	Read Port 1 -> 7 Read Port 1 -> 8
4	Overlapping Writes and reads	Reset is 1 Write port 1 , 2 are on Adds1 and Adds 2 are 10 Data is 22 , 33 Read address 1,2 are 10	44 ns	Read Port 1 -> 22 Read Port 1 -> 22

Testbench

```
/**************************************************************************************************************
 * Testbench: regfile_tb
 *
 * Description:
 * This Verilog testbench is designed to verify the functionality of the
 * regfile module. It includes test cases to perform write and read operations
 * on the dual-port register file and checks the output against expected values.
 *
 * Parameters:
 * - clk_period: Clock period (default: 20 time units)
 * - data width: Width of each register in bits (default: 32)
 * - Address_depth: Number of registers in the file (default: 4)
 * - delta_cycle: Time delay for simulation purposes (default: clk_period/10)
 *
 * Testbench Ports:
 * - i_clk: Clock input for synchronous operations
 * - i_rstn: Active-low asynchronous reset input
 * - Various input and output ports for the regfile module
 *
 * Tasks:
 * - initialization: Sets initial values for testbench variables
 * - reset_sequence: Performs a reset sequence on the regfile module
 * - write_operation: Simulates a write operation on a specified port
 * - read_scoreboard: Checks the output of a read operation against expected data
 * - clear: Resets all testbench variables
 * - clock_cycle: Advances the simulation time by one clock cycle
 *
 * Simulation Flow:
 * 1. Initialization of variables
 * 2. Reset sequence to initialize the regfile module
 * 3. Test Case 1: Write data to port 1, perform read operations, and check results
 * 4. Test Case 2: Write data to port 2, perform read operations, and check results
 * 5. Test Case 3: Simultaneous writes to both ports, read operations, and checks
 * 6. Test Case 4: Overlapping writes on the same address, read operations, and checks
 * 7. End of simulations after a certain number of clock cycles
 *
 * Student Name: Youssef Nasser
 * Student ID: 2300315
 * Date: 12/12/2023

**************************************************************************************************************/

// Begin regfile_tb module
module regfile_tb;

// Parameters
localparam clk_period    = 20;
localparam data_width    = 32;
localparam Address_depth = 4;
localparam delta_cycle = (clk_period/10);

// TB ports
reg i_clk;
reg i_rstn;
// Write port 1
reg i_write_enable_1;
reg [data_width-1:0] i_write_reg_1;
reg [Address_depth-1:0] i_write_add_1;
// Write port 2
reg i_write_enable_2;
reg [data_width-1:0] i_write_reg_2;
reg [Address_depth-1:0] i_write_add_2;
// Read port 1
reg [Address_depth-1:0] i_read_add_1;
wire [data_width-1:0] o_read_1;
// Read port 2
reg [Address_depth-1:0] i_read_add_2;
wire [data_width-1:0] o_read_2;
```

```
/*********************************************************
 * Instantiate regfile module
 *********************************************************/
regfile #(data_width, Address_depth) CUT (
  .i_clk(i_clk),
  .i_rstn(i_rstn),
  // Write port 1
  .i_write_enable_1(i_write_enable_1),
  .i_write_reg_1(i_write_reg_1),
  .i_write_add_1(i_write_add_1),
  // Write port 2
  .i_write_enable_2(i_write_enable_2),
  .i_write_reg_2(i_write_reg_2),
  .i_write_add_2(i_write_add_2),
  // Read port 1
  .i_read_add_1(i_read_add_1),
  .o_read_1(o_read_1),
  // Read port 2
  .i_read_add_2(i_read_add_2),
  .o_read_2(o_read_2)
);

/*********************************************************
 * Task to initialize testbench variables
 *********************************************************/
task initialization;
begin
  $display("Initialization start at time %0t", $time);
  i_write_enable_1 = 0;
  i_write_reg_1 = 0;
  i_write_add_1 = 0;
  i_write_enable_2 = 0;
  i_write_reg_2 = 0;
  i_write_add_2 = 0;
  i_read_add_1 = 0;
  i_read_add_2 = 0;
end
endtask
```

```
/*********************************************
 * Task to perform a reset sequence
 *********************************************/
task reset_sequence;
begin
  i_rstn = 0;
  $display("Reset is active low, reset now is %0d at time %0t", i_rstn,
$time);
  #(clk_period*10);
  i_rstn = 1;
end
endtask

/***********************************************************
 * Task to simulate a write operation on a specified port
 ***********************************************************/
task write_operation;
input reg [1:0] ID;
input reg [Address_depth-1:0] adds;
input reg [data_width-1:0] data;
begin
  $display("Write operation from port %0d at time %0t", ID, $time);
  $display("Address is %0d, data in is %0d", adds, data);
  case (ID)
    1: begin
      i_write_enable_1 = 1;
      i_write_reg_1 = data;
      i_write_add_1 = adds;
    end
    2: begin
      i_write_enable_2 = 1;
      i_write_reg_2 = data;
      i_write_add_2 = adds;
    end
    default: begin
      $display("Invalid ID");
    end
  endcase
end
endtask
```

```
/*************************************************************************
 * Task to check the output of a read operation against expected data
 *************************************************************************/
task read_scoreboard;
input reg [1:0] ID;
input reg [Address_depth-1:0] adds;
input reg [data_width-1:0] expected_data;
begin
  $display("Read check operation from port %0d at time %0t", ID, $time);
  $display("Address is %0d, expected data is %0d", adds, expected_data);
  case (ID)
    1: begin
      i_read_add_1 = adds;
      #delta_cycle;
      if (o_read_1 == expected_data) begin
        $display("Read Test passed");
      end else begin
        $display("Read test failed");
      end
    end
    2: begin
      i_read_add_2 = adds;
      #delta_cycle;
      if (o_read_2 == expected_data) begin
        $display("Read Test passed");
      end else begin
        $display("Read test failed");
      end
    end
    default: begin
      $display("Invalid ID");
    end
  endcase
end
endtask

/*********************************************************
 * Task to clear all testbench variables
 *********************************************************/
task clear;
begin
  i_write_enable_1 = 0;
  i_write_reg_1 = 0;
  i_write_add_1 = 0;
  i_write_enable_2 = 0;
  i_write_reg_2 = 0;
  i_write_add_2 = 0;
  i_read_add_1 = 0;
  i_read_add_2 = 0;
end
endtask
```

```
/*************************************************************
 * Task to advance simulation time by one clock cycle
 *************************************************************/
task clock_cycle;
begin
  #clk_period;
end
endtask

/*************************************************************
 * Initial block for clock generation
 *************************************************************/
initial begin
  i_clk = 0;
  forever begin
    #(clk_period/2) i_clk = ~i_clk;
  end
end
/*************************************************************
 * Initial block for simulation setup
 *************************************************************/
initial begin
  // Dumping waveform to a VCD file
  $dumpfile("regfile.vcd");
  $dumpvars(0, regfile_tb);

// Dumping FSDB File (Fast Signal Database) needed for Verdi

  $fsdbDumpfile("tb.fsdb");
  $fsdbDumpvars;

  $display("Welcome to testbench");

  // Execute test cases
  initialization;
  reset_sequence;
  #delta_cycle;

  // Test Case 1
  $display("Test Case 1");
  write_operation(1, 0, 10);
  clock_cycle;
  clear;
  read_scoreboard(1, 0, 10);
  read_scoreboard(2, 0, 10);
  clock_cycle;

  // Test Case 2
  $display("Test Case 2");
  write_operation(2, 15, 9);
  clock_cycle;
  clear;
  read_scoreboard(1, 15, 9);
  read_scoreboard(2, 15, 9);
  clock_cycle;
```

```
  // Test Case 3
  $display("Test Case 3");
  write_operation(1, 4, 7);
  write_operation(2, 5, 8);
  clock_cycle;
  clear;
  read_scoreboard(1, 4, 7);
  read_scoreboard(2, 5, 8);
  clock_cycle;

  // Test Case 4
  $display("Test Case 4");
  write_operation(1, 10, 22);
  write_operation(2, 10, 33);
  clock_cycle;
  clear;
  read_scoreboard(1, 10, 22);
  read_scoreboard(2, 10, 22);

  // End of simulations
  #(clk_period*100);
  $display("End of Simulations");
  $finish;
end
endmodule
```

Linux Environment

rtl.f has the following two lines

./regfile.v

./regfile_tb.v

Make directory that has the following files:

Makefile

regfile_tb.v

regfile.v

rtl.f

make comp

Makefile

```
#-------------------------------------------------------------------------
# comp: Target that invokes the clean, vcs, and verdi targets in sequence.
#-------------------------------------------------------------------------
----------------------------
comp  : clean vcs verdi
#-------------------------------------------------------------------------
# vcs: Target to compile the Verilog source files using Synopsys VCS
#simulator.
#      Options:
#        -f rtl.f           : Specifies the file containing the list of #
                              Verilog source files.
#        -cm line+fsm+tgl+branch+cond : Enable code coverage metrics (line,
FSM, toggle, branch, condition).
#        -timescale=1ns/1ps : Sets the timescale for simulation.
#        +vcs+flush+all     : Enable flushing of files after simulation.
#        -full64            : Enables 64-bit compilation.
#        -R                 : Runs the simulation.
#        +vc                : Enables Verilog Compiler mode.
#        +v2k               : Enables Verilog-2001 features.
#        -fsdb              : Generates FSDB (Full Signal Database) file for debugging.
#        -debug_all         : Enables debugging for all modules.
#        -l run.log         : Redirects log output to run.log.
#-------------------------------------------------------------------------
vcs   :
    vcs -f rtl.f -cm line+fsm+tgl+branch+cond -timescale=1ns/1ps
+vcs+flush+all -full64  -R  +vc  +v2k  -fsdb -debug_all -l run.log
#-------------------------------------------------------------------------
# verdi: Target to launch the Synopsys Verdi debugger with the specified
options.
#        Options:
#          -f rtl.f : Specifies the file containing the list of Verilog
source files.
#          -ssf tb.fsdb : Specifies the FSDB file to load into Verdi.
#          &        : Runs Verdi in the background.
#-------------------------------------------------------------------------
verdi  :
    verdi -f rtl.f -ssf  tb.fsdb &
#-------------------------------------------------------------------------
----------------------------
# clean: Target to remove generated files and clean the working directory.
#-------------------------------------------------------------------------
----------------------------
clean  :
     rm  -rf  *~  core  csrc  simv* group_2*  vc_hdrs.h  ucli.key  urg*
*.log  novas.* *.fsdb* verdiLog  64* DVEfiles *.vpd group_2.daidir
group_2.vdb wave.vcd
#-------------------------------------------------------------------------
```

Simulation Results

```
Verdi>run
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
Welcome to testbench
Initialization start at time 0
Reset is active low, reset now is 0 at time 0
Test Case 1
Write operation from port 1 at time 202000
Address is 0, data in is 10
Read check operation from port 1 at time 222000
Address is 0, expected data is 10
Read Test passed
Read check operation from port 2 at time 224000
Address is 0, expected data is 10
Read Test passed
Test Case 2
Write operation from port 2 at time 246000
Address is 15, data in is 9
Read check operation from port 1 at time 266000
Address is 15, expected data is 9
Read Test passed
Read check operation from port 2 at time 268000
Address is 15, expected data is 9
Read Test passed
Test Case 3
Write operation from port 1 at time 290000
Address is 4, data in is 7
Write operation from port 2 at time 290000
Address is 5, data in is 8
Read check operation from port 1 at time 310000
Address is 4, expected data is 7
Read Test passed
Read check operation from port 2 at time 312000
Address is 5, expected data is 8
Read Test passed
Test Case 4
Write operation from port 1 at time 334000
Address is 10, data in is 22
Write operation from port 2 at time 334000
Address is 10, data in is 33
Read check operation from port 1 at time 354000
Address is 10, expected data is 22
Read Test passed
Read check operation from port 2 at time 356000
Address is 10, expected data is 22
Read Test passed
End of Simulations
$finish called from file "./regfile_tb.v", line 278.
$finish at simulation time              2358000
Simulation complete, time is 2358000 ps.
```

Figure 3 Verdi Console

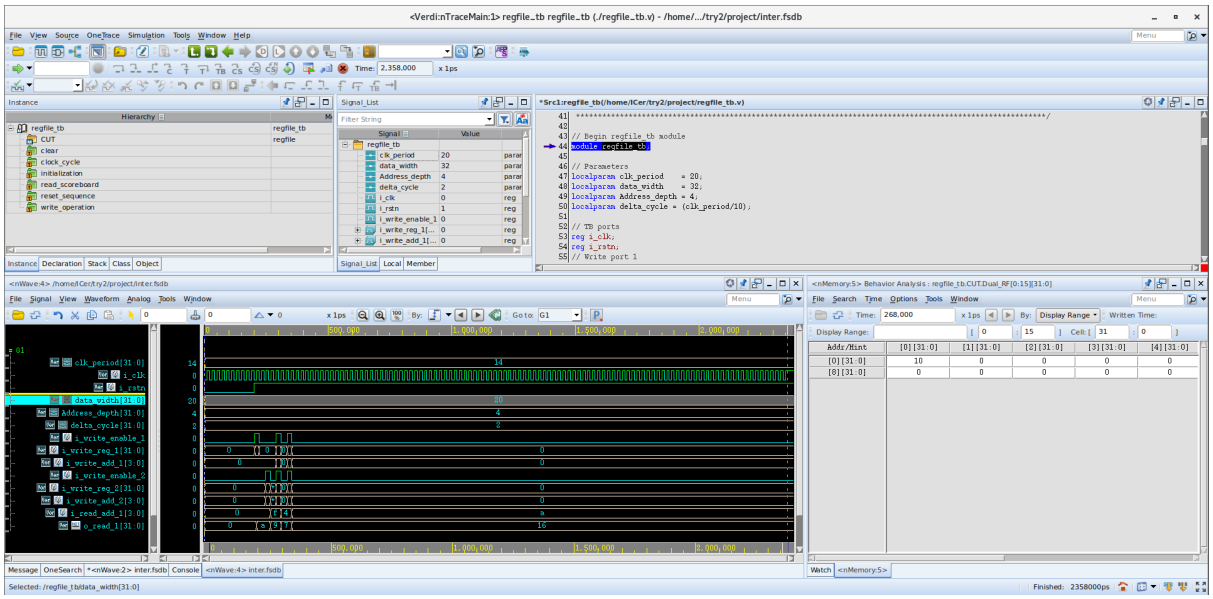

Figure 4 Verdi Environment

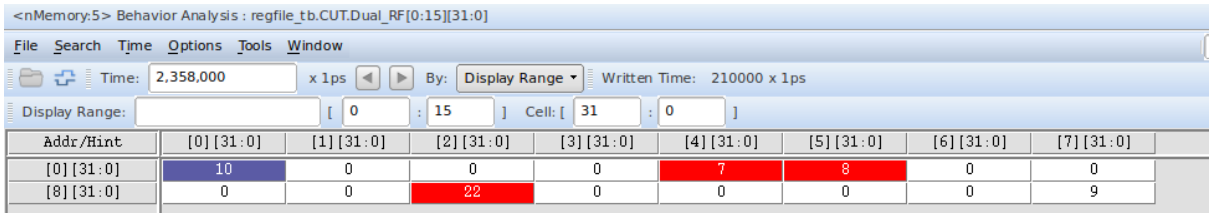

Figure 5 Verdi Content Memory

Test Case 1 data at address 0 is 10.

Test Case 2 data at address 15 is 9

Test Case 3 data at address 4 is 7, data at address 5 is 8

Test Case 4 data at address 10 is 22 how ever both 2 ports are writing at the same address by the priority of port1

Simulation Tools

1. **gedit**:

Text editor for writing code. It is a lightweight and simple editor commonly used on Linux systems.

2. GNU Make:

The native implementation of the Make utility on Linux. It interprets Makefiles to manage the build process of software projects.

3. **VCS**:

Synopsys VCS (Verilog Compiler Simulator) is a high-performance simulator for digital designs. It's commonly used in the hardware verification domain.

4. **Verdi**:

Synopsys Verdi is a debug and visualization tool for hardware designs. It helps in the verification and debugging process, providing features like waveform viewing, transaction-level debugging, and more.