Alexandria University  
Faculty of Engineering  
Computer Architecture  
Spring 2025

**Register File Implementation Report**

Waleed Emad Yahyaa20012197

Mohamed Atef Yousef20011630

1. **Introduction:**

In this lab, we designed and implemented a 32-register file using VHDL. The register file allows read and write operations based on a control signal and a clock cycle mechanism. The design ensures that data is written in the first half of the clock cycle and read in the second half.

1. **Design Specifications:**

**2.1. Inputs**

* Three Register Numbers: Two for reading and one for writing, each consisting of 5 bits.
* Write Data: A 32-bit input value to be stored in the selected register.
* RegWrite Control Line: Enables the write operation when set to '1'.
* Clock Signal: Controls read and write operations, where writing occurs in the first half cycle and reading in the second half.
  1. **Outputs**
* Two Read Data Outputs: Corresponding to the values stored in the registers indexed by the two read addresses.

1. **VHDL Implementation:**

The register file was implemented using an array of 32 registers, each 32 bits wide. The design adheres to the following logic:

* Writing occurs when RegWrite = '1' and the clock is in its falling edge.
* Reading occurs asynchronously and outputs the stored values based on the given read addresses.

**VHDL Code**

The complete VHDL implementation is included in the Appendix.

1. **Testbench and Simulation:**

To verify the correctness of our design, a testbench was developed covering multiple test scenarios, including:

* Writing and reading from registers.
* Writing to register zero to confirm that it remains zero.
* Performing a read and write operation on the same register in the same clock cycle.
* Disabling the write operation and ensuring no unintended changes occur.

**Simulation Results**

Below are screenshots showing the behavior of the register file during multiple clock cycles. The results confirm that the implementation meets the expected functionality.

صورة تحتوي على نص, لقطة شاشة, برمجيات, برامج الوسائط المتعددة

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.

1. **Tools and Environment:**

Vivado was used for compilation and simulation.

1. **Submission Details:**

The report includes:

* The full VHDL implementation of the register file.
* A detailed testbench covering multiple scenarios.
* Screenshots demonstrating simulation results.
* A link to the complete project repository: <https://github.com/moateff/Computer-Architecture-labs>

1. **Conclusion:**

The register file was successfully implemented and tested. The design meets all requirements, ensuring correct read/write operations and handling edge cases effectively.

**Appendix: VHDL Code**

*(Include VHDL code for the register file and testbench here.)*

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity REG\_FILE is

    Port (

        clk       : in  STD\_LOGIC;                      -- Clock signal

        wr\_en     : in  STD\_LOGIC;                      -- Write enable signal

        wr\_addr   : in  STD\_LOGIC\_VECTOR (4 downto 0);  -- Write address (5-bit)

        wr\_data   : in  STD\_LOGIC\_VECTOR (31 downto 0); -- Write data (32-bit)

        rd\_addr1  : in  STD\_LOGIC\_VECTOR (4 downto 0);  -- Read address 1 (5-bit)

        rd\_addr2  : in  STD\_LOGIC\_VECTOR (4 downto 0);  -- Read address 2 (5-bit)

        rd\_data1  : out STD\_LOGIC\_VECTOR (31 downto 0); -- Read data 1 (32-bit)

        rd\_data2  : out STD\_LOGIC\_VECTOR (31 downto 0)  -- Read data 2 (32-bit)

    );

end REG\_FILE;

architecture Behavioral of REG\_FILE is

    type reg\_array is array (0 to 31) of STD\_LOGIC\_VECTOR(31 downto 0);

    signal reg\_memory : reg\_array := (others => (others => '0')); -- Initialize to zeros

begin

    -- Read logic: Outputs data from register memory

    rd\_data1 <= reg\_memory(to\_integer(unsigned(rd\_addr1))) when rd\_addr1 /= "00000" else (others => '0');

    rd\_data2 <= reg\_memory(to\_integer(unsigned(rd\_addr2))) when rd\_addr2 /= "00000" else (others => '0');

    -- Write logic: Updates register memory on rising edge of clk

    process (clk)

    begin

        if rising\_edge(clk) then

            if wr\_en = '1' then

                reg\_memory(to\_integer(unsigned(wr\_addr))) <= wr\_data;

            end if;

        end if;

    end process;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use std.textio.all;

entity REG\_FILE\_TB is

end REG\_FILE\_TB;

architecture testbench of REG\_FILE\_TB is

    -- DUT signals

    signal clk      : std\_logic := '0';

    signal wr\_en    : std\_logic := '0';

    signal wr\_addr  : std\_logic\_vector(4 downto 0) := (others => '0');

    signal wr\_data  : std\_logic\_vector(31 downto 0) := (others => '0');

    signal rd\_addr1 : std\_logic\_vector(4 downto 0) := (others => '0');

    signal rd\_addr2 : std\_logic\_vector(4 downto 0) := (others => '0');

    signal rd\_data1 : std\_logic\_vector(31 downto 0);

    signal rd\_data2 : std\_logic\_vector(31 downto 0);

    constant CLK\_PERIOD : time := 10 ns;

begin

    -- Instantiate the Register File DUT

    DUT: entity work.REG\_FILE

        port map (

            clk      => clk,

            wr\_en    => wr\_en,

            wr\_addr  => wr\_addr,

            wr\_data  => wr\_data,

            rd\_addr1 => rd\_addr1,

            rd\_addr2 => rd\_addr2,

            rd\_data1 => rd\_data1,

            rd\_data2 => rd\_data2

        );

    -- Clock process

    clk\_process: process

    begin

        while now < 500 ns loop  -- Limit simulation time

            clk <= '0';

            wait for CLK\_PERIOD / 2;

            clk <= '1';

            wait for CLK\_PERIOD / 2;

        end loop;

        wait;

    end process;

    -- Stimulus process

    stimulus\_process: process

    begin

        report "Starting REG\_FILE Testbench...";

        -- Test Case 1: Write and Read from Register

        wr\_en   <= '1';

        wr\_addr <= "00001";

        wr\_data <= x"DEADBEEF";

        wait for CLK\_PERIOD / 2;

        wait for CLK\_PERIOD;

        wr\_en   <= '0';

        -- Read back the value

        rd\_addr1 <= "00001";

        wait for CLK\_PERIOD;

        assert rd\_data1 = x"DEADBEEF"

            report "Error: Incorrect data read from register!" severity error;

        -- Test Case 2: Writing to Register 0 (Should Remain Zero)

        wr\_en   <= '1';

        wr\_addr <= "00000";

        wr\_data <= x"FFFFFFFF";

        wait for CLK\_PERIOD;

        wr\_en   <= '0';

        rd\_addr1 <= "00000";

        wait for CLK\_PERIOD;

        assert rd\_data1 = x"00000000"

            report "Error: Register 0 should always be zero!" severity error;

        -- Test Case 3: Multiple Writes and Reads

        wr\_en   <= '1';

        wr\_addr <= "00010";

        wr\_data <= x"12345678";

        wait for CLK\_PERIOD;

        wr\_addr <= "00011";

        wr\_data <= x"ABCDEF12";

        wait for CLK\_PERIOD;

        wr\_en   <= '0';

        rd\_addr1 <= "00010";

        rd\_addr2 <= "00011";

        wait for CLK\_PERIOD;

        assert rd\_data1 = x"12345678"

            report "Error: Incorrect data read from register 2!" severity error;

        assert rd\_data2 = x"ABCDEF12"

            report "Error: Incorrect data read from register 3!" severity error;

        -- Test Case 4: Write and Read in the Same Clock Cycle

        wr\_en   <= '1';

        wr\_addr <= "00100";

        wr\_data <= x"A5A5A5A5";

        rd\_addr1 <= "00100";

        wait for CLK\_PERIOD;

        wr\_en   <= '0';

        assert rd\_data1 = x"A5A5A5A5"

            report "Error: Failed to read and write in the same cycle!" severity error;

        -- Test Case 5: RegWrite Disabled (wr\_en = 0)

        wr\_en   <= '0';

        wr\_addr <= "00101";

        wr\_data <= x"BEEFCAFE";

        wait for CLK\_PERIOD;

        rd\_addr1 <= "00101";

        wait for CLK\_PERIOD;

        assert rd\_data1 = x"00000000"

            report "Error: Data should not be written when wr\_en is '0'!" severity error;

        report "Testbench Completed Successfully." severity note;

        wait;

    end process;

end testbench;