Digital System Design using HDL Lab Report

Experiment 8a

32-bit Register file

Submitted by

Madhu Krishnan A P

M.Tech VLSI and Embedded Systems

Department of Electronics

Cochin University of Science and Technology

Contents

1 Description

The register file is a 32-element memory structure, where each element is 32 bits wide, representing registers x0 through x31. Register x0 is hardwired to 0 and cannot be modified. The register file includes two read ports and one write port. The read ports are addressed by 5-bit inputs, A1 and A2, allowing access to the 32-bit data stored in the respective registers, which are output through RD1 and RD2. The write port, controlled by a write enable signal (WE3), allows data from the input WD3 to be written into the register addressed by A3 on the rising edge of the clock. This configuration supports simultaneous dual reads and a single write operation, which is commonly used in processor architectures for efficient data handling.

2 Block diagram

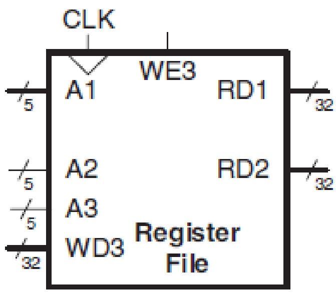

Figure 1: 32-bit Register file

3 Verilog description

```verilog
module RegisterFile (
    input logic clk,
    input logic WE3,
    input logic [4:0] A1, A2, A3,
    input logic [31:0] WD3,
    output logic [31:0] RD1, RD2
);

    // Define a 32x32 register file
    logic [31:0] regfile [31:0];

    // Initialize register x0 to always be 0
    initial begin
        regfile[0] = 32'b0;
    end

    // Read operations (combinational logic)
    assign RD1 = regfile[A1];
    assign RD2 = regfile[A2];

    // Write operation (sequential logic)
    always_ff @(posedge clk) begin
        if (WE3 && A3 != 5'b0) begin
            regfile[A3] <= WD3; // Write data to register A3
        end
    end

```

```
endmodule
```

4 Testbench

```
module RegisterFile_tb;

    // Testbench signals
    logic clk;
    logic WE3;
    logic [4:0] A1, A2, A3;
    logic [31:0] WD3;
    logic [31:0] RD1, RD2;

    // Instantiate the RegisterFile module
    RegisterFile dut (
        .clk(clk),
        .WE3(WE3),
        .A1(A1),
        .A2(A2),
        .A3(A3),
        .WD3(WD3),
        .RD1(RD1),
        .RD2(RD2)
    );

    // Clock generation
    initial clk = 0;
    always #5 clk = ~clk; // 10 time unit clock period

    // Test procedure
    initial begin
        // Initialize inputs
        WE3 = 0;
        A1 = 0;
        A2 = 0;
        A3 = 0;
        WD3 = 0;

        // Write data into 5 specific registers (1 to 5)
        for (int i = 1; i <= 5; i++) begin
            @(posedge clk);
            WE3 = 1;
            A3 = i;              // Target register address
            WD3 = i * 32'h1111; // Example data: multiples of 0x1111
        end
        @(posedge clk);
        WE3 = 0;

        // Read back the data using both read ports
        for (int i = 1; i <= 5; i++) begin
            @(posedge clk);
            A1 = i; // Address for read port 1
            A2 = i; // Address for read port 2
            @(posedge clk);
        end

        $finish;
    end
endmodule
```

The testbench for the `RegisterFile` module is designed to validate the functionality of a 32-element, 32-bit register file with two read ports and one write port. The testbench description is as follows.

4.1 Signal Initialization

- The testbench defines signals for the clock (`clk`), write enable (`WE3`), read/write addresses (`A1`, `A2`, `A3`), and data inputs/outputs (`WD3`, `RD1`, `RD2`).
- These signals are connected to the corresponding ports of the `RegisterFile` module instantiated as `dut`.

4.2 Clock Generation

- A simple clock signal is generated with a period of 10 time units, toggling every 5 time units.

4.3 Write Operations

- A loop writes data into 5 consecutive registers (addresses 1 through 5) using the write port.
- The data written is calculated as $i \times 32'$`h1111`, where i is the current loop index, ensuring each register gets a unique value.
- During this phase, the write enable (`WE3`) is asserted, and the `A3` signal specifies the target register address.

4.4 Read Operations

- After writing, another loop reads the data back from the same 5 registers using both read ports (`A1` and `A2`).
- The register addresses are sequentially assigned to both `A1` and `A2` to confirm that the data was correctly stored.

4.5 Test Completion

- After performing the write and read operations, the testbench ends with a `$finish` command.

5 Simulation result

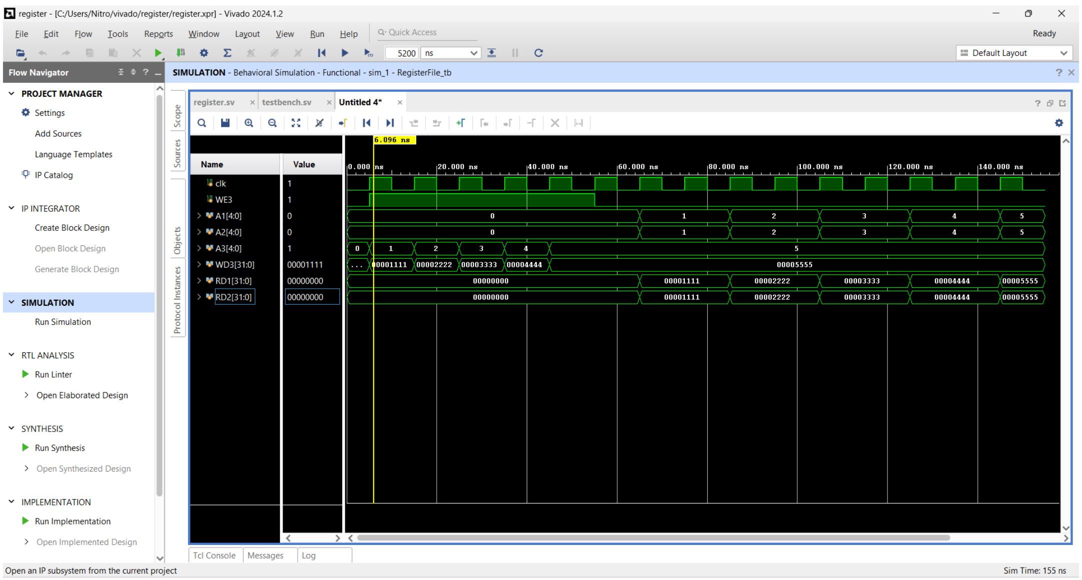

Figure 2: Register file: Testbench simulation