Computer Architecture

Lab # 01 : Register File Implementation

Section # 7

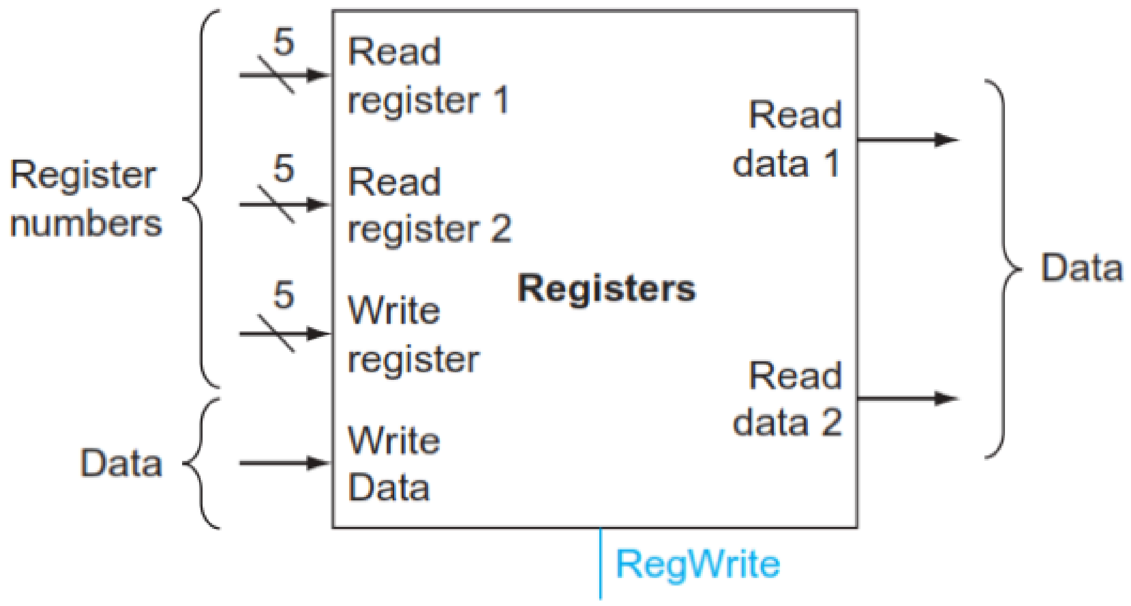
Mohamed Mohamed Hawas
ID : 18011595

Dr.Laila Abo-Hadid
TA.Monica Magdi

Register File Design

Introduction

- **Objective**: Implement a register file in VHDL consisting of 32 registers to facilitate read and write operations on data.



- **Input Signals**:
  - Three 5-bit register numbers: Two for specifying read registers and one for the write register.
  - 32-bit data input for write operation.
  - `RegWrite` control line to enable writing to the register file.
  - Clock signal for synchronization.
- **Output Signals**:
  - Two 32-bit data outputs corresponding to the read registers specified.
- **Operation**:
  - Write operations occur in the first half cycle of the clock signal.
  - Read operations occur in the second half cycle of the clock signal.

Entity Description

The entity `RegisterFile` defines a register file module. It has the following ports:

- `clk`: Clock input signal.
- `regWrite`: Write control signal.
- `writeRegNum`: Write register address.
- `readRegNum1`: Read register address 1.
- `readRegNum2`: Read register address 2.
- `dataIn`: Data input for write operation.
- `dataOut1`: Data output for read operation 1.

- `dataOut2`: Data output for read operation 2.

Architecture Description

The architecture `Behavioral` implements the register file functionality. It consists of a single process sensitive to the rising edge of the clock signal. The process handles both write and read operations based on the `regWrite` control signal and the clock edge.

Write Phase

- On a rising clock edge, if `regWrite` is asserted (`'1'`), the process writes the `dataIn` to the register specified by `writeRegNum`.

Read Phase

- On a falling clock edge, the process reads data from the registers specified by `readRegNum1` and `readRegNum2` and assigns them to `dataOut1` and `dataOut2` respectively.

Implementation Details

- The register file is implemented as an array of 32-bit vectors (`RegisterArray`).
- The write operation occurs on the rising edge of the clock, while the read operation occurs on the falling edge.

Test Bench for Register File

Test Case 1: Write to Register 1 and 4

- **Description**: Writes data to registers 1 and 4.
- **Write Operations**:
  - Write to register 1: Data `00000001` is written.
  - Write to register 4: Data `00000002` is written.

Test Case 2: Read from Registers 1 and 4

- **Description**: Reads data from registers 1 and 4.
- **Read Operations**:
  - Read from register 1: Data output from register 1.
  - Read from register 4: Data output from register 4.

Test Case 3: Write and Read in Same Register at Same Clock Cycle

- **Description**: Writes and reads data from register 1 in the same clock cycle.
- **Write Operation**:
  - Write to register 1: Data `00000003` is written.
- **Read Operations**:
  - Read from register 1: Data output from register 1.

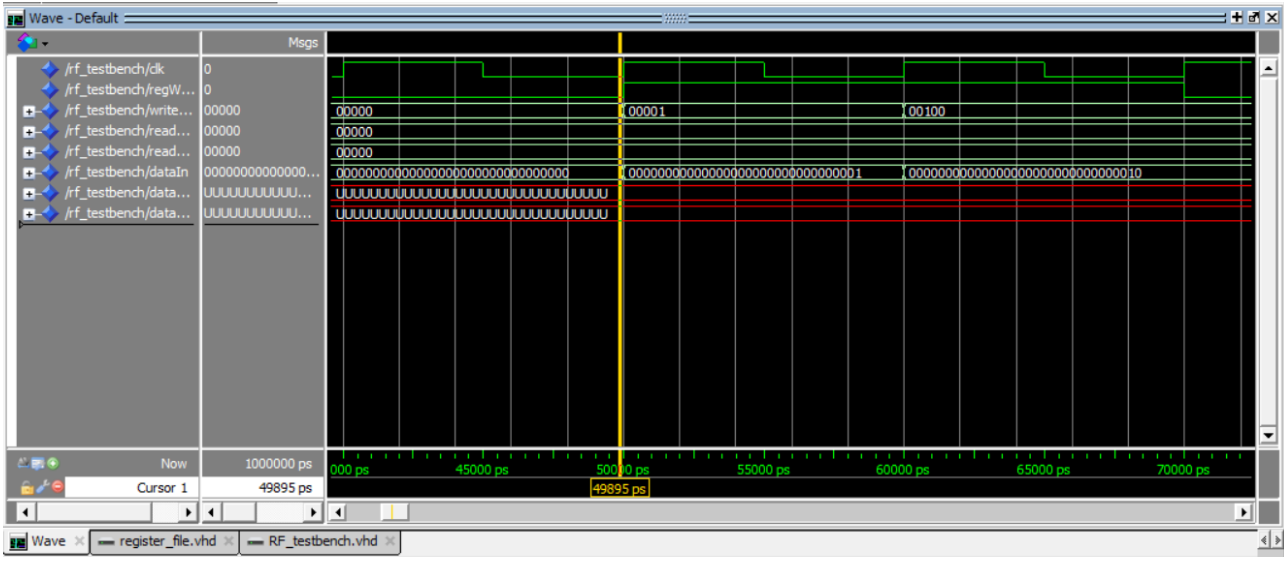
Test Case 4: Write when `regWrite = 0`

- **Description**: Attempts to write data to register when `regWrite` is low.
- **Write Operation**:
  - No write operation should occur as `regWrite` is low.

Samples of Run

Test Case 1: Write to Register 1 and 4

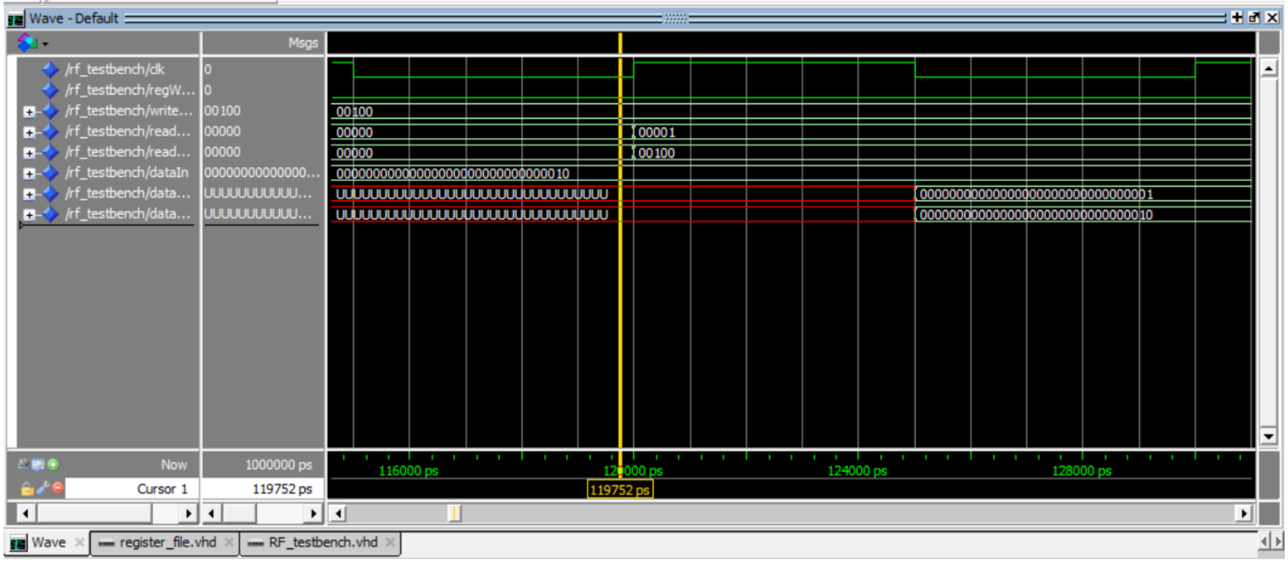
- **Expected Result**: Data `00000001` should be written to register 1, and data `00000002` should be written to register 4.



- **Comment**: This test case verifies the ability of the register file to correctly write data to specified registers.

Test Case 2: Read from Registers 1 and 4

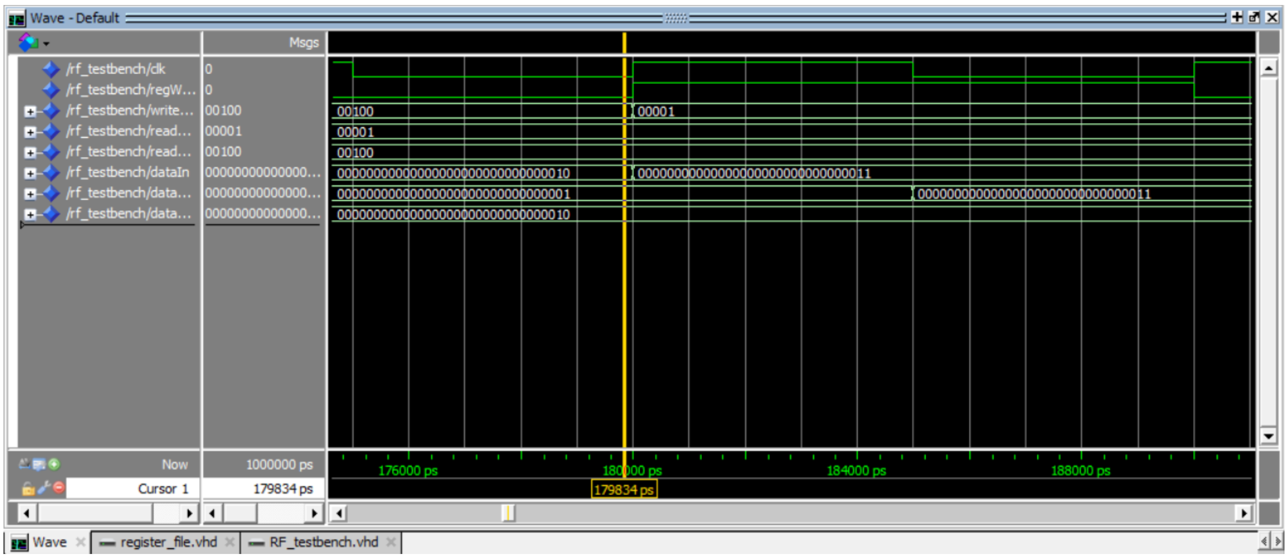
- **Expected Result**: Data stored in registers 1 and 4 should be read and output in the second half cycle.



- **Comment**: This test case ensures that the register file can properly read data from specified registers in the second half cycle.

Test Case 3: Write and Read in Same Register at Same Clock Cycle

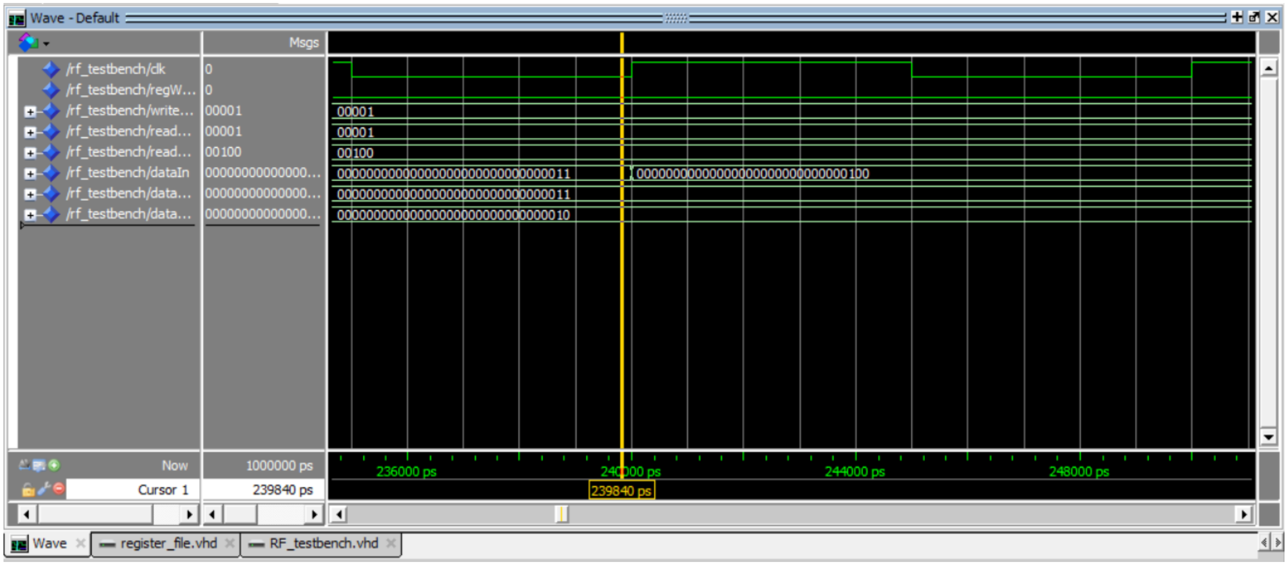
- **Expected Result**: Data `00000003` should be written to register 1, and then read from register 1 in the same clock cycle.



- **Comment**: This test case validates whether the register file can handle simultaneous write and read operations on the same register within a single clock cycle.

Test Case 4: Write when `regWrite = 0`

- **Expected Result**: No data should be written to any register since `regWrite` is low.



- **Comment**: This test case checks if the register file correctly ignores write operations when the `regWrite` control line is de-asserted.

Code

1. Register File Design (RegisterFile.vhd)

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity RegisterFile is
    Port (
        clk : in STD_LOGIC;                              -- Clock input
        regWrite : in STD_LOGIC;                         -- Write control signal
        writeRegNum : in unsigned(4 downto 0);      -- Write register address
        readRegNum1 : in unsigned(4 downto 0);      -- Read register address 1
        readRegNum2 : in unsigned(4 downto 0);      -- Read register address 2
        dataIn : in std_logic_vector(31 downto 0); -- Data input for write
        dataOut1 : out std_logic_vector(31 downto 0); -- Data output for read 1
        dataOut2 : out std_logic_vector(31 downto 0)  -- Data output for read 2
    );
end RegisterFile;

architecture Behavioral of RegisterFile is
    type RegisterArray is array (31 downto 0) of std_logic_vector(31 downto 0);
    signal registers : RegisterArray;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            -- Write phase
            if regWrite = '1' then
                registers(to_integer(writeRegNum)) <= dataIn;
            end if;

            -- Read phase
            dataOut1 <= registers(to_integer(readRegNum1));
            dataOut2 <= registers(to_integer(readRegNum2));
        end if;
    end process;
end Behavioral;
```

2. Testbench (RF_testbench.vhd)

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity RF_testbench is
end RF_testbench;
```

```
architecture Behavioral of RF_testbench is
    constant CLK_PERIOD : time := 10 ns; -- Define clock period
    signal clk : std_logic := '1'; -- Clock signal
    signal regWrite : std_logic := '0'; -- Register write control
    signal writeRegNum : unsigned(4 downto 0) := (others => '0'); -- Write
register number
    signal readRegNum1 : unsigned(4 downto 0) := (others => '0'); -- Read register
number 1
    signal readRegNum2 : unsigned(4 downto 0) := (others => '0'); -- Read register
number 2
    signal dataIn : std_logic_vector(31 downto 0) := (others => '0'); -- Data to
be written
    signal dataOut1 : std_logic_vector(31 downto 0); -- Data output from register
1
    signal dataOut2 : std_logic_vector(31 downto 0); -- Data output from register
2
begin
    -- Instantiate the RegisterFile module
    uut: entity work.RegisterFile
        port map (
            clk => clk,
            regWrite => regWrite,
            writeRegNum => writeRegNum,
            readRegNum1 => readRegNum1,
            readRegNum2 => readRegNum2,
            dataIn => dataIn,
            dataOut1 => dataOut1,
            dataOut2 => dataOut2
        );

    -- Clock process
    clk_process: process
    begin
        while now < 1000 ns loop
            clk <= '1';
            wait for CLK_PERIOD / 2;
            clk <= '0';
            wait for CLK_PERIOD / 2;
        end loop;
        wait;
    end process clk_process;

    -- Test process
    test_process: process
    begin
        -- Test case 1: Write to register 5 and read from registers 0 and 5
        wait for 50 ns;
        regWrite <= '1';
        writeRegNum <= to_unsigned(1, 5);
        dataIn <= x"12345078";
        wait for CLK_PERIOD;
        writeRegNum <= to_unsigned(4, 5);
        dataIn <= x"13546478";
        wait for CLK_PERIOD;

        -- Test case 2: Read from the same register
        wait for 50 ns;
        readRegNum1 <= to_unsigned(1, 5);
        readRegNum2 <= to_unsigned(4, 5);
        wait for CLK_PERIOD;

        -- Test case 3: Write and read in same register at same clock cycle
        wait for 50 ns;
        regWrite <= '1';
        writeRegNum <= to_unsigned(1, 5);
        dataIn <= x"65893147";
        wait for CLK_PERIOD / 2; -- Introduce a slight delay
        readRegNum1 <= to_unsigned(1, 5);
        readRegNum2 <= to_unsigned(4, 5);
        wait for CLK_PERIOD / 2; -- Continue with the clock cycle
        regWrite <= '0';

        wait;
    end process test_process;
end Behavioral;
```

Link

Link To Repo