INTERNSHIP TASK -2

RAM DESIGN

- **DEVELOP A SIMPLE SYNCHRONOUS RAM MODULE WITH READ AND WRITE OPERATIONS**.

Purpose of the RAM:

We want to build a **4-bit wide** RAM module with **16 locations** (addresses from `0` to `15`). It should perform **read** and **write** operations synchronously with a clock signal.

- **Inputs and Outputs**

1. **Inputs:**
   - **`clk`**: Clock signal to synchronize operations.
   - **`we`**: Write Enable signal (when `we = 1`, data is written; when `we = 0`, data is read).
   - **`address`**: A 4-bit address to specify which memory location to read or write.
   - **`data_in`**: A 4-bit data input to write into the memory (valid only when `we = 1`).
2. **Output:**
   - **`data_out`**: A 4-bit data output, which contains the data read from the memory.

- **Memory Structure**

We will create a 16-location memory, with each location being 4 bits wide. The memory will be represented as a **register array** in Verilog.

- Verilog Code for the RAM Module

```
    module simple_sync_ram (
  input wire clk,              // Clock signal (for synchronization)
  input wire we,               // Write enable: 1 for write, 0 for read
 input wire [3:0] address,     // 4-bit address (16 memory locations)
  input wire [3:0] data_in,    // 4-bit data input (for write operation)
  output reg [3:0] data_out    // 4-bit data output (for read operation)
);

  // Memory: 16 locations, each 4 bits wide
  reg [3:0] memory [15:0];

  // Always block triggered on the rising edge of the clock
  always @(posedge clk) begin
    if (we) begin
      // Write operation: Store data_in at the specified address
      memory[address] <= data_in;
    end else begin           // Read operation: Output data from the
specified address
      data_out <= memory[address];
    end
  end

endmodule
```

Explanation of the Code:

1. **Inputs:**
   - **`clk`**: This is the clock signal that synchronizes read/write operations.
   - **`we`**: The **write enable** signal. If `we = 1`, data is written to the memory; if `we = 0`, data is read from memory.
   - **`address`**: The address of the memory location (4-bit wide, so it can range from 0 to 15).

- **`data_in`**: The data input to be written to the memory (4-bit wide).

2. **Output:**
   - **`data_out`**: This is the data read from the memory (4-bit wide).
3. **Memory Array:**
   - `memory [15:0]` is a register array representing the memory. It contains 16 entries (from 0 to 15), each holding 4 bits of data.
4. **Always Block:**
   - The `always @(posedge clk)` block ensures that operations (write/read) are synchronized with the clock.
   - If **`we = 1`** (write enable), data is written into the memory at the specified address.
   - If **`we = 0`** (read operation), data from the specified memory address is read and sent to `data_out`.

Testbench to Verify the RAM Module

```
module testbench;

  // Declare signals
  reg clk;              // Clock signal
  reg we;               // Write Enable
  reg [3:0] address;    // 4-bit address
  reg [3:0] data_in;    // 4-bit data input
  wire [3:0] data_out;  // 4-bit data output
  // Instantiate the RAM module
  simple_sync_ram uut (
```

```
    .clk(clk),
     .we(we),
    .address(address),
    .data_in(data_in),
    .data_out(data_out)
);
// Generate the clock signal (toggles every 5 time units)
always begin
    #5 clk = ~clk;  // Toggle clock every 5 time units
end
// Initialize and perform tests
initial begin
    // Initialize signals
    clk = 0;
    we = 0;
    address = 4'b0000;   // Start with address 0
    data_in = 4'b0000;   // Start with data "0000"
    // Test 1: Write data to address 1
    #10;
    we = 1;              // Enable write operation
    address = 4'b0001;   // Address 1
    data_in = 4'b1010;   // Write data "1010" to address 1
    // Test 2: Read from address 1
    #10;
    we = 0;              // Disable write (read operation)
    address = 4'b0001;   // Read from address 1

    // Test 3: Write data to address 2
    #10;
    we = 1;              // Enable write operation
    address = 4'b0010;   // Address 2
    data_in = 4'b1100;   // Write data "1100" to address 2
```

```
        // Test 4: Read from address 2
        #10;
        we = 0;              // Disable write (read operation)
        address = 4'b0010;   // Read from address 2
        // End the simulation
        #10;
        $finish;
    end
    // Display the signals whenever they change
    initial begin
        $monitor("Time: %t | Address: %b | Data_in: %b | we: %b | Data_out: %b", $time, address, data_in, we, data_out);
    end
endmodule
```

Expected Output

After running the simulation, you should see an output like this:

```
Time: 10 | Address: 0001 | Data_in: 1010 | we: 1 | Data_out: x
Time: 20 | Address: 0001 | Data_in: 1010 | we: 0 | Data_out: 1010
Time: 30 | Address: 0010 | Data_in: 1100 | we: 1 | Data_out: 1010
Time: 40 | Address: 0010 | Data_in: 1100 | we: 0 | Data_out: 1100
```

At time 10: Data `1010` is written to address `1`.

- **At time 20**: Data is read from address `1`, and `data_out = 1010`.
- **At time 30**: Data `1100` is written to address `2`.
- **At time 40**: Data is read from address `2`, and `data_out = 1100`.