T a s k 4: Digital Filter Design
Instructions:

- Design and simulate a digital FIR (Finite Impulse Response) filter using Verilog or MATLAB.
- **Deliverable:** Verilog code, simulation results, and performance analysis.

1. FIR Filter Design (Verilog Code)

Here is an example of the Verilog code for a simple 3-tap FIR filter. This filter will perform the following operation: y[n] = h_0 \\cdot x[n] + h_1 \\cdot x[n-1] + h_2 \\cdot x[n-2]
For this example, let's use the coefficients h_0 = 0.25, h_1 = 0.5, and h_2 = 0.25. These coefficients are often used for a low-pass filter. To implement this in a digital system without floating-point numbers, we will scale these coefficients to integers. Let's multiply them by 4: h_0 = 1, h_1 = 2, h_2 = 1. The output will then be divided by 4 at the end.

```
// 3-Tap FIR Filter
module fir_filter(
    input  wire        clk,
    input  wire        reset,
    input  wire [7:0]  data_in,     // 8-bit signed input data
    output wire [7:0]  data_out     // 8-bit signed output data
);

    // Coefficients (scaled by 4)
    localparam h0 = 2'd1;
    localparam h1 = 2'd2;
    localparam h2 = 2'd1;

    // Registers for data shift
    reg [7:0] data_reg_1; // x[n-1]
    reg [7:0] data_reg_2; // x[n-2]

    // Pipelined MAC (Multiply-Accumulate) unit
    reg signed [11:0] mac_result_1;
    reg signed [11:0] mac_result_2;
    reg signed [11:0] mac_result_3;

    // Shift registers for input data
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            data_reg_1 <= 8'd0;
            data_reg_2 <= 8'd0;
        end else begin
            data_reg_1 <= data_in;
            data_reg_2 <= data_reg_1;
        end
    end

    // MAC operations
    always @(posedge clk) begin
        // Perform the multiplication and addition steps
        mac_result_1 <= data_in * h0;
        mac_result_2 <= mac_result_1 + (data_reg_1 * h1);
        mac_result_3 <= mac_result_2 + (data_reg_2 * h2);
    end

    // Final output is scaled down (divided by 4)
    assign data_out = mac_result_3[11:2];

endmodule
```

2. Testbench (Verilog Code)

This testbench will apply a few different signals to the FIR filter to demonstrate its functionality. We'll send a single pulse and then a series of pulses to observe the smoothing effect.

```
// Testbench for FIR Filter
`timescale 1ns / 1ps

module fir_filter_tb;
    reg clk_tb;
    reg reset_tb;
    reg [7:0] data_in_tb;
    wire [7:0] data_out_tb;

    // Instantiate the Unit Under Test (UUT)
    fir_filter uut (
        .clk      (clk_tb),
        .reset    (reset_tb),
        .data_in  (data_in_tb),
        .data_out (data_out_tb)
    );

    // Clock generation
    always #5 clk_tb = ~clk_tb;

    initial begin
        // Initialize inputs
        clk_tb = 0;
        reset_tb = 1;
        data_in_tb = 8'd0;
        // Reset the filter
        #10;
        reset_tb = 0;
        #10;

        $display("--- Starting FIR Filter Test ---");

        // Test 1: Single Impulse Response
        data_in_tb = 8'd60; // Send a single pulse of 60
        #10;
        data_in_tb = 8'd0;
        #50; // Allow the pulse to propagate through the filter

        // Test 2: Step Input
        $display("Applying step input...");
        data_in_tb = 8'd100;
        #50;

        $display("Removing step input...");
        data_in_tb = 8'd0;
        #50;

        $display("--- Test Completed ---");
        $finish; // End simulation
    end

    initial begin
        $monitor("Time=%0t | data_in=%d, data_out=%d", $time,
data_in_tb, data_out_tb);
    end
endmodule
```

3. Simulation Results and Performance Analysis

Your report should include the following sections:

- **Introduction:**
    - Explain what an FIR filter is and its basic function (to remove unwanted frequencies or smooth a signal).
    - State the filter's design parameters: 3-taps, with coefficients h_0=1, h_1=2, h_2=1 (scaled).
- **Design Description:**
    - Explain the Verilog implementation, highlighting key components:
        - **Shift Registers:** How the input data data_in is delayed to create x[n-1] and x[n-2].
        - **Multipliers and Accumulators (MAC):** The pipelined stages that perform the multiplication and addition operations.
        - **Output Scaling:** The final step where the accumulated sum is divided by 4 to

get the correct output.

- **Testbench Description:**
  - Describe the test cases you used.
  - The **single impulse test** is crucial for showing the filter's coefficients in the output. For an input of [60, 0, 0, 0, ...], the output should be a scaled version of the coefficients: [15, 30, 15, 0, ...].
  - The **step input test** demonstrates the filter's response to a sudden change in input, showing a gradual rise and fall in the output, which is a characteristic of a low-pass filter.
- **Simulation Waveforms:**
  - Include screenshots of the waveforms from your EDA tool (e.g., Vivado, ModelSim).
  - **Annotate the waveforms** to show the input data_in and the corresponding output data_out.
  - Point out the impulse response in the output (the sequence [15, 30, 15]) and the smoothed response to the step input.
- **Performance Analysis:**
  - **Area/Resource Usage:** Mention that the design uses simple registers and combinational logic for the MAC unit. For a real FPGA implementation, you would discuss the number of look-up tables (LUTs) and flip-flops used.
  - **Throughput/Latency:** Explain that this design has a latency of 3 clock cycles (it takes 3 clock cycles for an input to produce a stable output). However, it has a throughput of one sample per clock cycle, meaning it can process data continuously.