INTERNSHIP TASK -4

DIGITAL FILTER DESIGN

- DESIGN AND SIMULATE A DIGITAL FIR (FINITE IMPULSE RESPONSE)FILTER USING VERILOG OR MATLAB.

Designing an FIR Filter

1. Filter Specifications

Before designing the filter, define the following parameters:

- **Filter Type**: Lowpass, Highpass, Bandpass, or Bandstop
- **Order**: Number of taps (coefficients)
- **Cutoff Frequency**: Determines the frequency range the filter will pass or attenuate
- **Sampling Rate**: The rate at which the signal is sampled

2. MATLAB: Designing and Generating Filter Coefficients

MATLAB provides several methods to design FIR filters:

Using `fir1` Function

The `fir1` function designs a linear-phase FIR filter using the window method.

n = 50; % Filter order

Wn = 0.4; % Normalized cutoff frequency (0 to 1, where 1 corresponds to Nyquist frequency)

b = fir1(n, Wn);

This designs a lowpass FIR filter with a cutoff frequency at 0.4 times the Nyquist frequency. The `fir1` function uses a Hamming window by

default, but you can specify other windows like Hanning, Blackman, or Kaiser.

Using `firpm` Function

For more control over the frequency response, use the Parks-McClellan algorithm with the `firpm` function.

F = [0 0.4 0.5 1]; % Frequency bands

A = [1 1 0 0]; % Desired amplitude in each band

b = firpm(50, F, A);

his designs a lowpass filter with a transition band between 0.4 and 0.5 times the Nyquist frequency.

Visualizing the Filter Response

Use `fvtool` to visualize the frequency response of the designed filter.

fvtool(b, 1);

3. Verilog: Implementing the FIR Filter

A simple Verilog implementation of a 3-tap FIR filter is as follows:

```
module fir_filter (
    input clk,
    input rst,
    input [15:0] x, // Input signal
    output [15:0] y // Output signal
);
    reg [15:0] delay1, delay2;
    wire [15:0] h0 = 16'h1, h1 = 16'h2, h2 = 16'h1; // Filter coefficients
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            delay1 <= 16'h0;
```

```
        delay2 <= 16'h0;
    end else begin
        delay1 <= x;
        delay2 <= delay1;
    end
  end
  assign y = h0 * x + h1 * delay1 + h2 * delay2;
endmodule
```

This module implements a 3-tap FIR filter with coefficients 1, 2, and 1. The input signal `x` is processed through two delay elements (`delay1` and `delay2`), and the output `y` is the weighted sum of the current and previous inputs.

4. **Simulation and Testing**

```
module tb_fir_filter;
  reg clk;
  reg rst;
  reg [15:0] x;
  wire [15:0] y;

  fir_filter uut (
    .clk(clk),
    .rst(rst),
    .x(x),
    .y(y)
  );

  initial begin
    clk = 0;
```

```
        rst = 1;
        x = 16'h0;
        #10 rst = 0;
        #10 x = 16'h1;
        #10 x = 16'h2;
        #10 x = 16'h3;
        #10 x = 16'h4;
        #10 x = 16'h5;
        #10 x = 16'h6;
        #10 x = 16'h7;
        #10 x = 16'h8;
        #10 x = 16'h9;
        #10 x = 16'hA;
        #10 x = 16'hB;
        #10 x = 16'hC;
        #10 x = 16'hD;
        #10 x = 16'hE;
        #10 x = 16'hF;
    end

    always #5 clk = ~clk;
endmodule
```

This testbench applies a sequence of input values to the FIR filter and observes the output. Use a simulation tool like ModelSim or Vivado to compile and simulate the Verilog code, observing the waveform outputs to verify correct operation.