TASK-4

Designing and simulating a Digital FIR (Finite Impulse Response) Filter involves creating a system that processes a finite sequence of input samples using a set of coefficients. The system performs a convolution of the input signal with a set of filter coefficients, producing an output signal.

We can design an FIR filter in two ways:

1. **Using Verilog** (hardware description language for FPGA or ASIC design)
2. **Using MATLAB** (for simulation and analysis)

For the purpose of this explanation, we'll focus on both approaches with a simple example of a low-pass FIR filter, which is one of the most common types of FIR filters.

**1. FIR Filter Design Overview**

An FIR filter operates by calculating the weighted sum of a set of previous input values. The general equation for an FIR filter with N taps (coefficients) is:

y[n]=∑k=0N−1h[k]⋅x[n−k]y[n] = \sum\_{k=0}^{N-1} h[k] \cdot x[n-k]y[n]=k=0∑N−1​h[k]⋅x[n−k]

Where:

* y[n]y[n]y[n] is the output at the current sample.
* h[k]h[k]h[k] are the filter coefficients (taps).
* x[n−k]x[n-k]x[n−k] are the input signal samples at previous times.
* NNN is the number of filter taps (the length of the impulse response).

For simplicity, we’ll design a low-pass filter, which allows low-frequency signals to pass while attenuating higher-frequency signals.

**2. FIR Filter Design Using MATLAB**

MATLAB is a great tool for designing and simulating FIR filters. Here's how to design and simulate a basic FIR filter in MATLAB.

**Step 1: Define the Filter Specifications**

We’ll define a simple low-pass filter with the following parameters:

* **Sampling rate** = 1 kHz (just for example)
* **Cutoff frequency** = 100 Hz
* **Filter order** = 20 (number of taps)

**Step 2: Design the Filter**

MATLAB has built-in functions like fir1() that can help you design FIR filters. For example:

matlab

Copy

% Define the filter specifications

Fs = 1000; % Sampling rate in Hz

Fc = 100; % Cutoff frequency in Hz

N = 20; % Filter order (number of taps)

% Normalize the cutoff frequency

Wn = Fc / (Fs / 2); % Normalized cutoff frequency (0 to 1)

% Design the FIR filter using the 'fir1' function (low-pass filter)

h = fir1(N, Wn);

% Display the filter coefficients

disp('FIR Filter Coefficients:');

disp(h);

% Plot the frequency response of the filter

freqz(h, 1, 512, Fs);

**Step 3: Simulate the Filter Response**

You can simulate the filter's response to an input signal. Let's create a signal that is a combination of a low-frequency and a high-frequency component:

matlab

Copy

% Generate a test signal (combination of low and high-frequency sine waves)

t = 0:1/Fs:1; % Time vector for 1 second

x = sin(2 \* pi \* 50 \* t) + sin(2 \* pi \* 200 \* t); % Signal with 50Hz and 200Hz components

% Apply the FIR filter to the input signal

y = filter(h, 1, x);

% Plot the original and filtered signals

subplot(2, 1, 1);

plot(t, x);

title('Original Signal');

xlabel('Time (seconds)');

ylabel('Amplitude');

subplot(2, 1, 2);

plot(t, y);

title('Filtered Signal (Low-pass FIR Filter)');

xlabel('Time (seconds)');

ylabel('Amplitude');

**Explanation of the MATLAB Code:**

* **fir1()**: This function designs a low-pass FIR filter with specified order and normalized cutoff frequency.
* **filter()**: This function applies the filter to the input signal x and produces the output signal y.
* **freqz()**: This function plots the frequency response of the filter, showing how it attenuates or passes different frequencies.

**Step 4: Visualizing the Results**

When you run the above code, MATLAB will generate plots of:

* The **original signal** (which includes both low and high-frequency components).
* The **filtered signal**, which will show the effect of the FIR filter (attenuating the high-frequency component above 100 Hz).

**3. FIR Filter Design Using Verilog**

Now, let’s consider designing the same FIR filter using Verilog. We will describe a basic FIR filter that takes input samples and produces an output sample based on the filter's coefficients.

**Step 1: FIR Filter Module in Verilog**

verilog

Copy

module fir\_filter #(

parameter N = 20, // Number of taps (filter order)

parameter WIDTH = 16 // Bit-width for the data

)(

input clk, // Clock signal

input rst, // Reset signal

input signed [WIDTH-1:0] x, // Input sample

output reg signed [WIDTH-1:0] y // Output sample

);

// Filter coefficients (example, could be any set of taps)

reg signed [WIDTH-1:0] h [0:N-1]; // Filter coefficients (taps)

// Shift registers to store past input samples

reg signed [WIDTH-1:0] x\_reg [0:N-1];

integer i;

// Initialize coefficients (for simplicity, we assume some example coefficients)

initial begin

h[0] = 16'b0000000000001000; // Example coefficient values

h[1] = 16'b0000000000001100;

// Add more coefficients...

end

always @(posedge clk or posedge rst) begin

if (rst) begin

// Reset the shift registers

for (i = 0; i < N; i = i + 1) begin

x\_reg[i] <= 0;

end

y <= 0;

end else begin

// Shift the input samples

for (i = N-1; i > 0; i = i - 1) begin

x\_reg[i] <= x\_reg[i-1];

end

x\_reg[0] <= x; // New input sample

// Compute the output

y <= 0;

for (i = 0; i < N; i = i + 1) begin

y <= y + x\_reg[i] \* h[i]; // Convolution (sum of products)

end

end

end

endmodule

**Explanation of the Verilog Code:**

* **Parameters**: We define the filter order N (number of taps) and the bit width of the data (WIDTH).
* **Filter Coefficients**: The filter coefficients h are stored in a register array. In practice, these coefficients can be generated based on the design (e.g., by using a tool like MATLAB to compute them).
* **Shift Registers**: The input samples are stored in an array of registers x\_reg for the convolution process.
* **Convolution**: The output y is computed by performing a sum of the products of the filter coefficients and the corresponding delayed input samples (a simple implementation of the FIR filter’s convolution operation).

**4. Simulation of the Verilog Design**

To simulate the Verilog design, we will write a testbench. The testbench will provide clock signals, reset, and input samples for the FIR filter.

**Testbench:**

verilog

Copy

module tb\_fir\_filter;

reg clk;

reg rst;

reg signed [15:0] x; // Input sample

wire signed [15:0] y; // Output sample

// Instantiate the FIR filter

fir\_filter #(

.N(20),

.WIDTH(16)

) uut (

.clk(clk),

.rst(rst),

.x(x),

.y(y)

);

// Clock generation

always begin

#5 clk = ~clk; // Generate a clock signal with a period of 10 time units

end

// Test sequence

initial begin

// Initialize signals

clk = 0;

rst = 0;

x = 0;

// Apply reset

rst = 1;

#10;

rst = 0;

// Apply input samples (a simple step signal)

x = 16'b0000000000000001; // First sample

#10;

x = 16'b0000000000000010; // Second sample

#10;

x = 16'b0000000000000100; // Third sample

#10;

x = 16'b0000000000001000; // Fourth sample

#10;

// Finish simulation

$finish;

end

// Monitor output

initial begin

$monitor("At time %t, x = %d, y = %d", $time, x, y);

end

endmodule

**Explanation of the Testbench Code:**

* **Clock Generation**: The clock signal is toggled every 5 time units to create a clock period of 10 time units.
* **Test Sequence**: We apply a sequence of input values (x) and observe how the output (y) changes.
* **Monitor**: The $monitor statement prints the values of x and y during the simulation.

**5. Conclusion**

In this explanation, we demonstrated how to design and simulate an FIR filter both in **MATLAB** and **Verilog**. MATLAB is a powerful tool for designing and analyzing filters, while Verilog allows you to implement and simulate hardware descriptions for digital filters. The MATLAB approach is typically used for simulation and analysis, while Verilog is more suitable for hardware implementation on FPGA or ASIC platforms.

Would you like further clarification on any part of the design or simulation?