PROJECT: FIR Filter Design and Implementation

Mahendra Khinchi

FIR Filter Basics

An FIR (Finite Impulse Response) filter calculates the output as a weighted sum of current and past input samples. Mathematically, the output $y[n]$ is given by:

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k] \tag{1}$$

where:

- $y[n]$: Filtered output at sample n
- $x[n-k]$: Input sample delayed by k
- $h[k]$: Filter coefficients (impulse response)
- N: Number of filter coefficients (filter order + 1)

Filter Design in MATLAB

The filter is designed with the following specifications:

- Sampling Frequency: $F_s = 48000\,Hz$
- Stopband Frequencies:

$$f_{\text{stop1}} = 500\,Hz$$
$$f_{\text{stop2}} = 9000\,Hz$$

$$f_{\text{pass1}} = 1500\,Hz$$
$$f_{\text{pass2}} = 8000\,Hz$$

Normalized frequencies are calculated by dividing by the Nyquist frequency ($F_N = F_s/2 = 24000\,Hz$):

$$f_{\text{stop1(norm)}} = \frac{500}{24000} = 0.0208$$

$$f_{\text{pass1(norm)}} = \frac{1500}{24000} = 0.0625$$

$$f_{\text{pass2(norm)}} = \frac{8000}{24000} = 0.3333$$

$$f_{\text{stop2(norm)}} = \frac{9000}{24000} = 0.3750$$

Fixed-Point Conversion (Q(2,14))

In Q(2,14) format, numbers are represented as signed fixed-point values with:

- 2 integer bits (including sign bit)
- 14 fractional bits

Thus, the range of values is:

$$-2.0 \leq x < +1.9999...$$

To convert a floating-point coefficient to Q(2,14):

$$h[k]_{Q(2,14)} = \text{round}(h[k] \times 2^{14})$$

For example, for coefficient $h[k] = 0.1234$:

$$h[k]_{Q(2,14)} = \text{round}(0.1234 \times 2^{14}) = 2023$$

Overflow handling:

- If value exceeds +32767, wrap within signed 16-bit range.
- If value goes below -32768, wrap similarly.

Sinewave Generation

The equation for generating a sinewave is:

$$x(t) = A\,\sin(2\pi ft)$$

For a sinewave frequency f_i sampled at $F_s = 48000\,Hz$, the duration of one cycle is:

$$T_i = \frac{1}{f_i}$$

The total time for 5 cycles:

$$T_{\text{total}} = 5 \times T_i = \frac{5}{f_i}$$

Number of samples required:

$$N_{\text{samples}} = T_{\text{total}} \times F_s$$

Example calculation for $f_1 = 100\,Hz$:

$$T_1 = \frac{1}{100} = 0.01\,s, \quad T_{\text{total}} = \frac{5}{1}00 = 0.05\,s, \quad N_{\text{samples}} = 0.05{\times}48000 = 2400\,samples.$$

Repeat similarly for frequencies $f_2 = 2000\,Hz$, $f_3 = 6000\,Hz$, and $f_4 = 11000\,Hz$.

FIR Filtering (Convolution)

Filtering involves convolving the input signal with the filter coefficients:

$$y[n] = h[0]x[n] + h[1]x[n-1] + \cdots + h[N-1]x[n-(N-1)]$$

This can also be expressed as a dot product between the filter coefficients and a sliding window of input samples.

Verilog Implementation (MAC Operation)

Filtering in Verilog uses Multiply-and-Accumulate (MAC):

$$y[n] + = h[k] \times x[n-k]$$

Inputs $(x[n-k], h[k])$ are in Q(2,14). The output $y[n]$ is scaled back from Q(2,28) to Q(2,14).

Verification: MATLAB vs Verilog Output Comparison

To verify correctness, compare MATLAB and Verilog outputs for all input signals. Ensure both outputs match within acceptable error margins due to fixed-point rounding.

STEPS:

1 MATLAB CODE FOR FIR Filter Design and Export Coefficients:

```matlab
% FIR Filter Design
filterDesigner; % Open the filter designer GUI
```

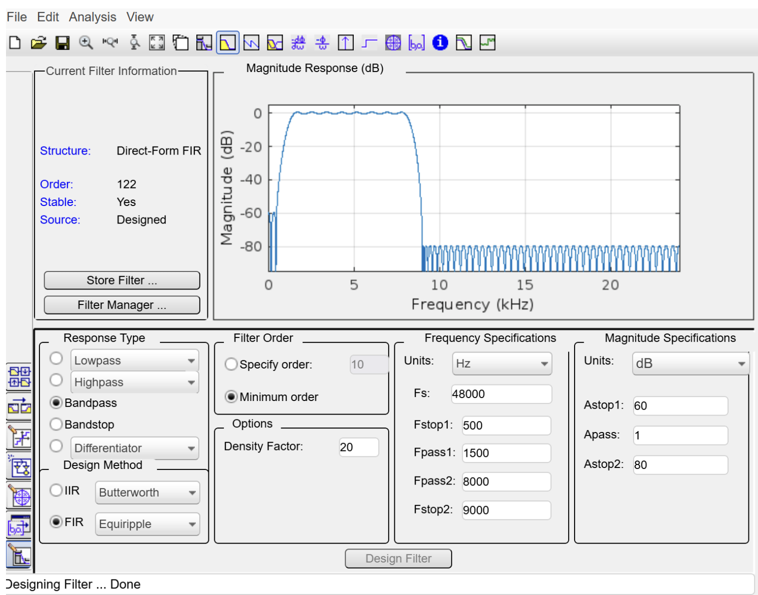

Figure 1: Response of BANDPASS FILTER

2 MATLAB CODE FOR Convert Coefficients to Fixed-Point Q(2,14):

```matlab
% Load filter coefficients (replace 'Num' with your variable name)
load('filter_coefficients.mat'); % Load exported coefficients
coeff_fixed = round(Num * 2^14); % Convert to Q(2,14) format
coeff_fixed(coeff_fixed >= 2^15) = coeff_fixed(coeff_fixed >= 2^15)
     - 2^16; % Handle overflow

% Save coefficients to a file
fileID = fopen('filter_coefficients_q214.txt', 'w');
fprintf(fileID, '%d\n', coeff_fixed);
fclose(fileID);
```

3 MATLAB CODE FOR Generate Input Signals:

```
fs = 48000; % Sampling frequency
t = 0:1/fs:5*(1/100); % Time vector for 5 cycles of the lowest frequency

% Generate sine waves
f1 = 100; f2 = 2000; f3 = 6000; f4 = 11000;
x1 = sin(2*pi*f1*t);
x2 = sin(2*pi*f2*t);
x3 = sin(2*pi*f3*t);
x4 = sin(2*pi*f4*t);

% Normalize and convert to Q(2,14)
signals = {x1, x2, x3, x4};
for i = 1:4
    signal_fixed{i} = round(signals{i} * (2^14)); % Q(2,14)
    signal_fixed{i}(signal_fixed{i} >= 2^15) = signal_fixed{i}(signal_fixed{i} >= 2^15) - 2^16; % Handle overflow

    % Save each signal to a file
    filename = sprintf('input_signal_%d_q214.txt', i);
    fileID = fopen(filename, 'w');
    fprintf(fileID, '%d\n', signal_fixed{i});
    fclose(fileID);
end
```

4 Apply FIR Filter in MATLAB:

```
% Apply FIR filter to each signal
outputs = cell(1,4);
for i = 1:4
    outputs{i} = filter(Num, 1, signals{i}); % Apply filter in floating-point
end

% Plot outputs for visualization
figure;
for i = 1:4
    subplot(4,1,i);
    plot(outputs{i});
    title(['Filtered Output for Signal ', num2str(i)]);
end

% Save outputs in Q(2,14)
for i = 1:4
    output_fixed{i} = round(outputs{i} * (2^14)); % Q(2,14)
    output_fixed{i}(output_fixed{i} >= 2^15) = output_fixed{i}(output_fixed{i} >= 2^15) - 2^16; % Handle overflow

    filename = sprintf('output_signal_%d_q214.txt', i);
    fileID = fopen(filename, 'w');
```

```
    fprintf(fileID, '%d\n', output_fixed{i});
    fclose(fileID);
end
```

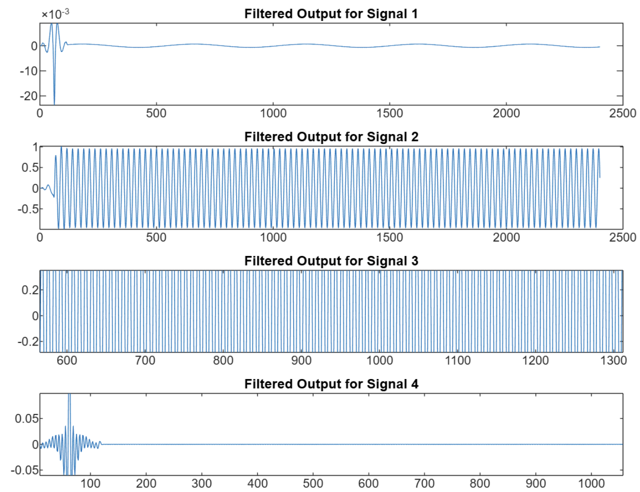

Figure 2:

5 For 5 Cycles:

```
% Sampling frequency
fs = 48000; % Sampling frequency in Hz

% Frequencies of the sinewaves
frequencies = [100, 2000, 6000, 11000]; % Frequencies in Hz

% Initialize cell array to store signals
signals = cell(1, length(frequencies));

% Generate sinewaves for each frequency
for i = 1:length(frequencies)
    f = frequencies(i); % Current frequency
    t = 0:1/fs:(5*(1/f)); % Time vector for 5 cycles of the current
        frequency
    signals{i} = sin(2*pi*f*t); % Generate sinewave

    % Save each signal to a file in Q(2,14) format
    signal_fixed = round(signals{i} * (2^14)); % Convert to Q(2,14)
    signal_fixed(signal_fixed >= 2^15) = signal_fixed(signal_fixed
        >= 2^15) - 2^16; % Handle overflow
```

```

    filename = sprintf('input_signal_%dHz_q214.txt', f);
    fileID = fopen(filename, 'w');
    fprintf(fileID, '%d\n', signal_fixed);
    fclose(fileID);

    % Plot the sinewave
    figure;
    plot(t, signals{i});
    title(['Sinewave for ', num2str(f), ' Hz (5 cycles)']);
    xlabel('Time (s)');
    ylabel('Amplitude');
end
```

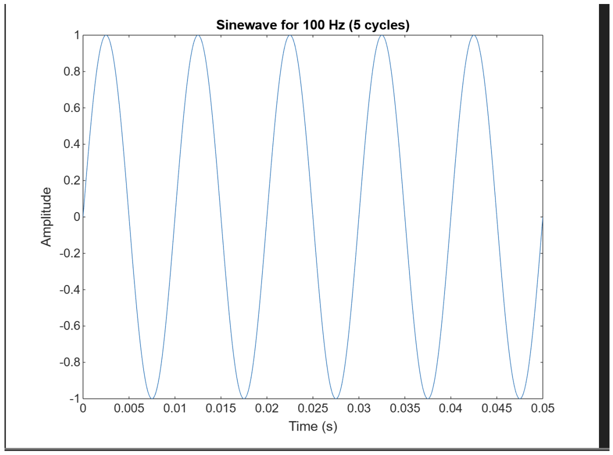

Figure 3:

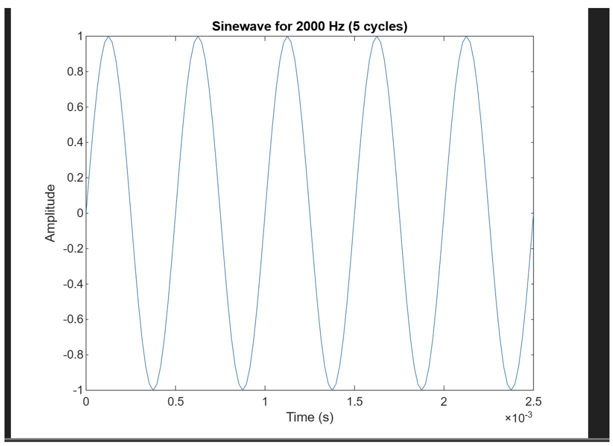

Figure 4:

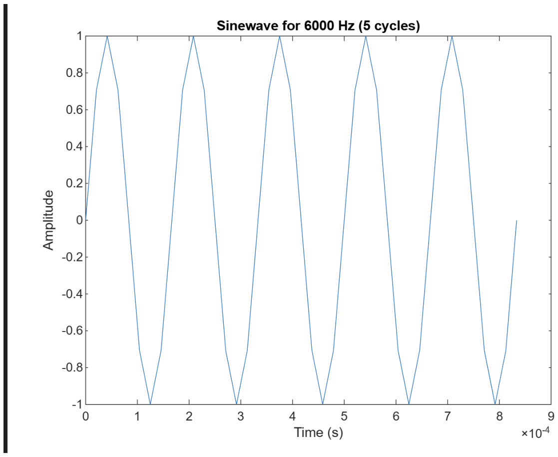

Figure 5:

6 Verilog Impletation:

```
module fir_filter(
    input clk,
    input reset,
    input signed [15:0] x_in, // Input signal in Q(2,14)
    output reg signed [31:0] y_out // Filtered output in Q(2,14)
);

parameter N = 20; // Number of filter coefficients (taps)

// Registers for coefficients and shift registers
reg signed [15:0] coeffs[N-1:0]; // Filter coefficients in Q(2,14)
reg signed [15:0] shift_reg[N-1:0]; // Shift register for input samples

integer i;

// Initialize coefficients from file
initial begin
    $readmemb("filter_coefficients_q214.txt", coeffs); // Load coefficients from file
end

// FIR filter implementation
always @(posedge clk or posedge reset) begin
    if (reset) begin
        y_out <= 32'd0;
        for (i = 0; i < N; i = i + 1) begin
            shift_reg[i] <= 16'd0; // Clear shift registers on reset
        end
    end else begin
```

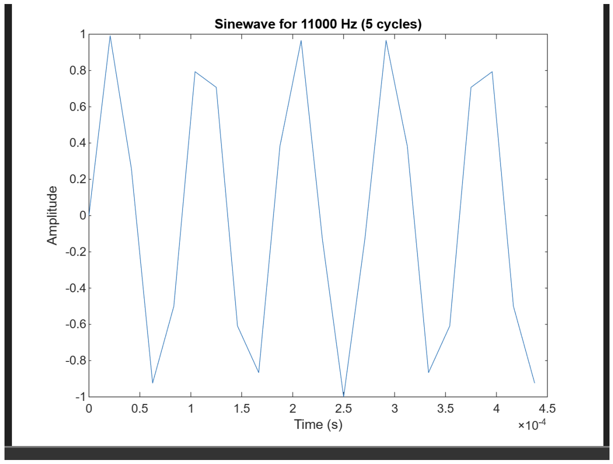

Figure 6:

```
        // Shift input samples into the shift register
        for (i = N-1; i > 0; i = i - 1) begin
            shift_reg[i] <= shift_reg[i-1];
        end
        shift_reg[0] <= x_in;

        // Multiply-Accumulate (MAC) operation to compute filter
            output
        y_out <= 32'd0; // Clear output accumulator
        for (i = 0; i < N; i = i + 1) begin
            y_out <= y_out + shift_reg[i] * coeffs[i];
        end
    end
end

endmodule
```

7 Test Bench:

```
    module tb_fir_filter;

// Inputs and outputs for the DUT (Device Under Test)
reg clk;
reg reset;
reg signed [15:0] x_in;
wire signed [31:0] y_out;

// Instantiate the FIR filter module
fir_filter uut (
    .clk(clk),
    .reset(reset),
```

```
    .x_in(x_in),
    .y_out(y_out)
);

// File handling variables
integer infile, outfile;
integer scan;
reg signed [15:0] input_sample;

// Clock generation (50 MHz clock)
always #10 clk = ~clk;

initial begin
    // Initialize clock and reset signals
    clk = 0;
    reset = 1;

    #20 reset = 0; // Deassert reset after some time

    // Process each input signal file one by one
    process_signal("input_signal_100Hz_q214.txt", "output_signal_100Hz_verilog_q214.txt");
    process_signal("input_signal_2000Hz_q214.txt", "output_signal_2000Hz_verilog_q214.txt");
    process_signal("input_signal_6000Hz_q214.txt", "output_signal_6000Hz_verilog_q214.txt");
    process_signal("input_signal_11000Hz_q214.txt", "output_signal_11000Hz_verilog_q214.txt");

    $stop; // Stop simulation after processing all signals
end

// Task to process a single signal file through the FIR filter
task process_signal(input string infile_name, output string outfile_name);
begin
    infile = $fopen(infile_name, "r"); // Open input signal file for reading
    outfile = $fopen(outfile_name, "w"); // Open output file for writing

    if (infile == 0 || outfile == 0) begin
        $display("Error opening file!");
        $stop;
    end

    while (!$feof(infile)) begin
        scan = $fscanf(infile, "%d\n", input_sample); // Read input sample from file

        @(posedge clk); // Wait for positive clock edge
        x_in <= input_sample;

        @(posedge clk); // Wait for another clock edge to capture output sample

```

```
        $fwrite(outfile, "%d\n", y_out >>> 14); // Write filtered output to file (scaled back to Q(2,14))
    end

    $fclose(infile); // Close input file after reading all samples
    $fclose(outfile); // Close output file after writing all samples

    $display("Processed %s -> %s", infile_name, outfile_name);
end
endtask

endmodule
```

8 Verification:

```
    matlab_output = load('output_signal_100Hz_q214.txt');
verilog_output = load('output_signal_100Hz_verilog_q214.txt');

figure;
plot(matlab_output, 'b'); hold on;
plot(verilog_output, 'r--');
legend('MATLAB␣Output', 'Verilog␣Output');
title('Comparison␣of␣MATLAB␣and␣Verilog␣Outputs␣(100␣Hz)');
xlabel('Samples'); ylabel('Amplitude');
```

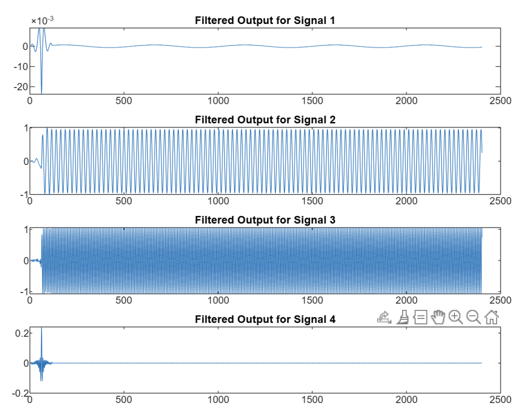

Figure 7: From Both