DSP Lab 7: IIR Filter Design

Nimal Sreekumar
EE23BTECH11044

I. Theory

The direct form-II realization of a second-order IIR filter is given by:

$$w(n) = x(n) - a_1 w(n-1) - a_2 w(n-2), \tag{1}$$

$$y(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2). \tag{2}$$

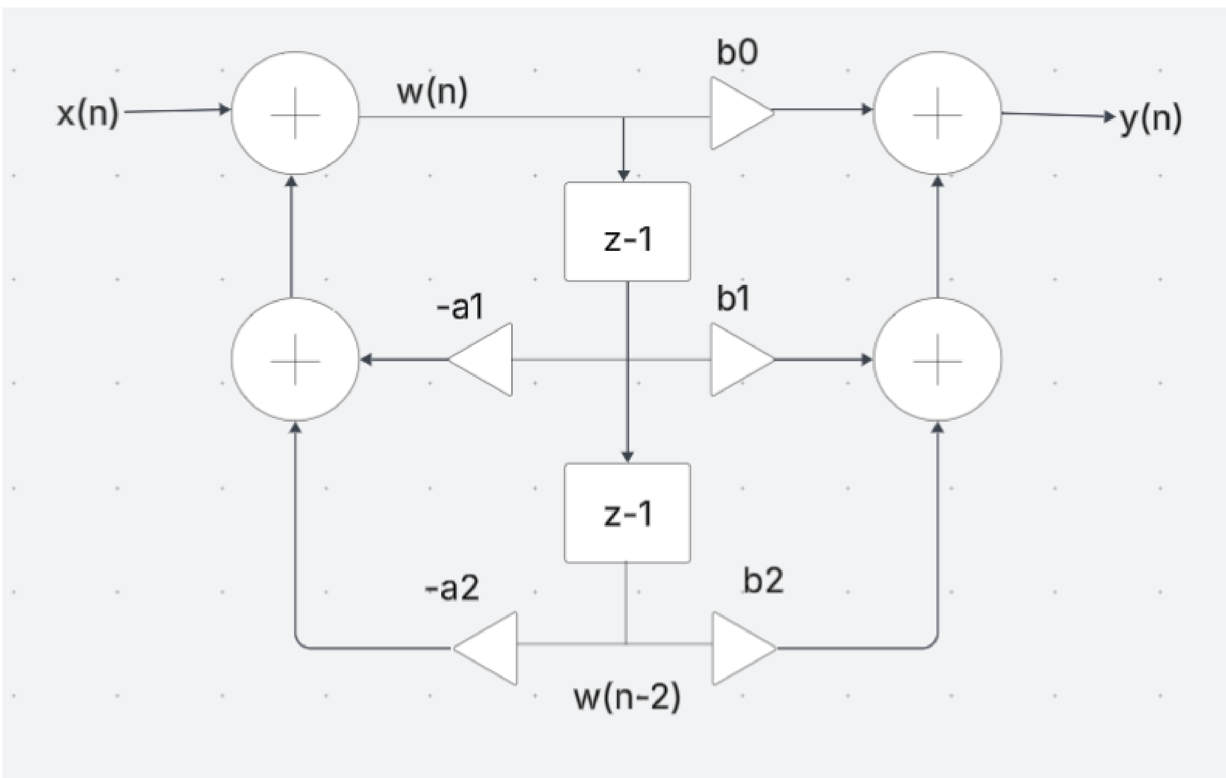

Fig. 1: Direct form-II structure of a second-order IIR filter.

II. MATLAB Implementation

A. Filter Design

Generate the IIR filter using MATLAB's `filterDesigner` with the following specifications:

B. Fixed-Point Implementation

```
clear; clc; close all;

% Load SOS matrix and gain
load('IIR.mat', 'SOS', 'G');
```

```
IIR filter Design

In MATLAB

1. Enter filterDesigner in MATLAB command window
2. It opens a filter designer window.
3. Select following options:
     a. Lowpass and Select IIR ->Butterworth
     b. FilterOrder = Minimum
     c. Density factor = 20
     d. Frequency spec :  Sampling freq = 48000, fstop1 = 900, fpass1 = 1000
```

Fig. 2: IIR filter design specifications.

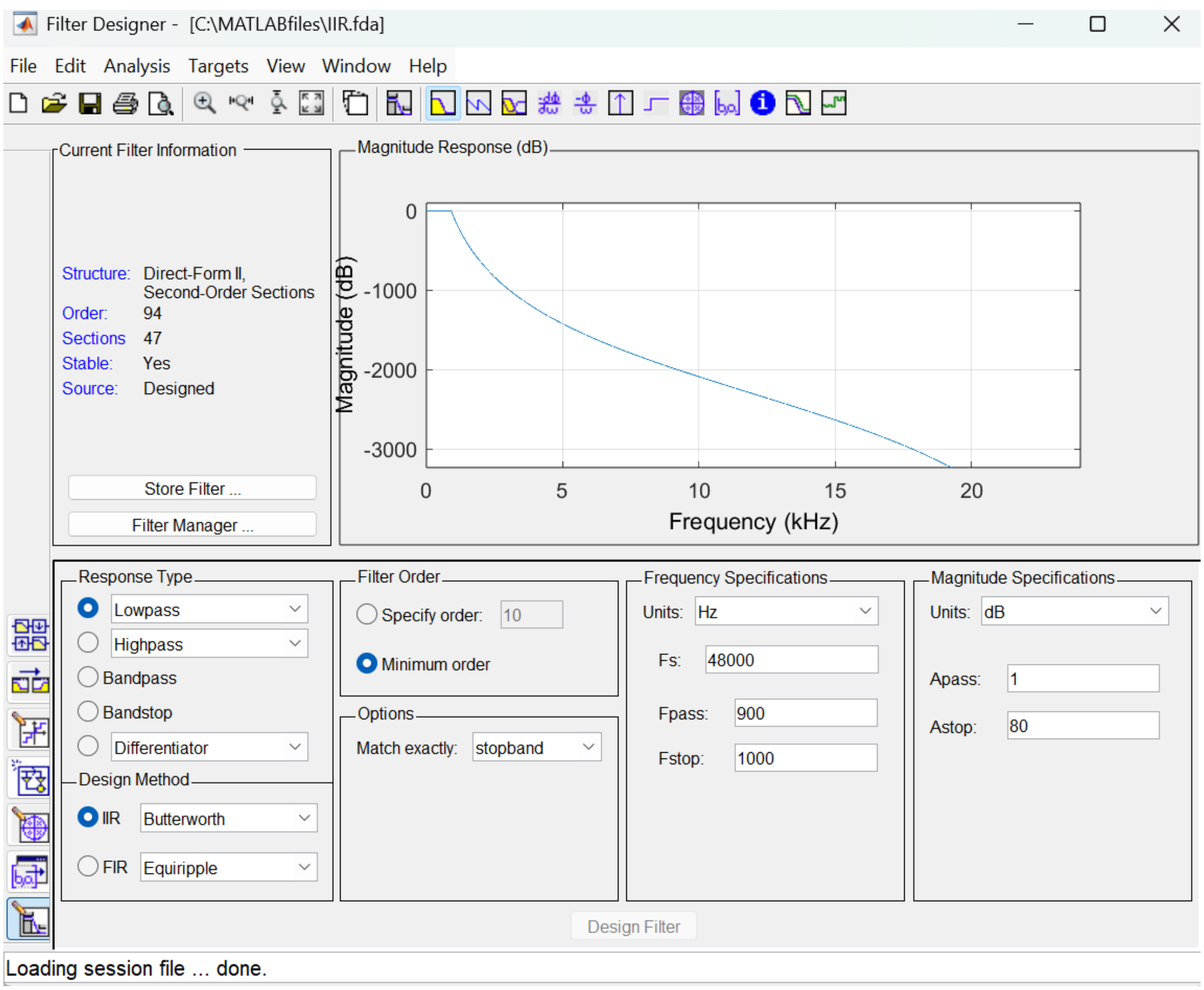

Fig. 3: IIR filter magnitude and phase response.

```
% Convert to fixed-point (Q2.14)
sos_fixed = fi(SOS, true, 16, 14);
G_fixed   = fi(G,   true, 16, 14);

% Save fixed-point coefficients
dlmwrite('SOS_q.txt', sos_fixed.bin, 'delimiter', '\n');
dlmwrite('G_q.txt',   G_fixed.bin,   'delimiter', '\n');

% Sampling parameters
Fs = 48000; duration = 1;
t = 0:1/Fs:(duration - 1/Fs);

% Generate sine waves at 100, 1000, and 2000 Hz
freqs = [100, 1000, 2000];
sine_fx = arrayfun(@(f) fi(sin(2*pi*f*t), true, 16, 14),
freqs, 'UniformOutput', false);

% Save sine waves
cellfun(@(s,f) dlmwrite(sprintf('sine%d_q.txt', f),
s.bin, 'delimiter', '\n'), sine_fx, num2cell(freqs));

% Plot original sine waves
figure;
for i = 1:3
    subplot(3,1,i);
    plot(t, double(sine_fx{i}));
    title(sprintf('%d Hz Sine Wave', freqs(i)));
    xlabel('Time (s)'); ylabel('Amplitude');
    grid on;
end

disp('Fixed-point conversion, file saving, and plotting completed!');
```

C. Generated Signals

Signals generated:

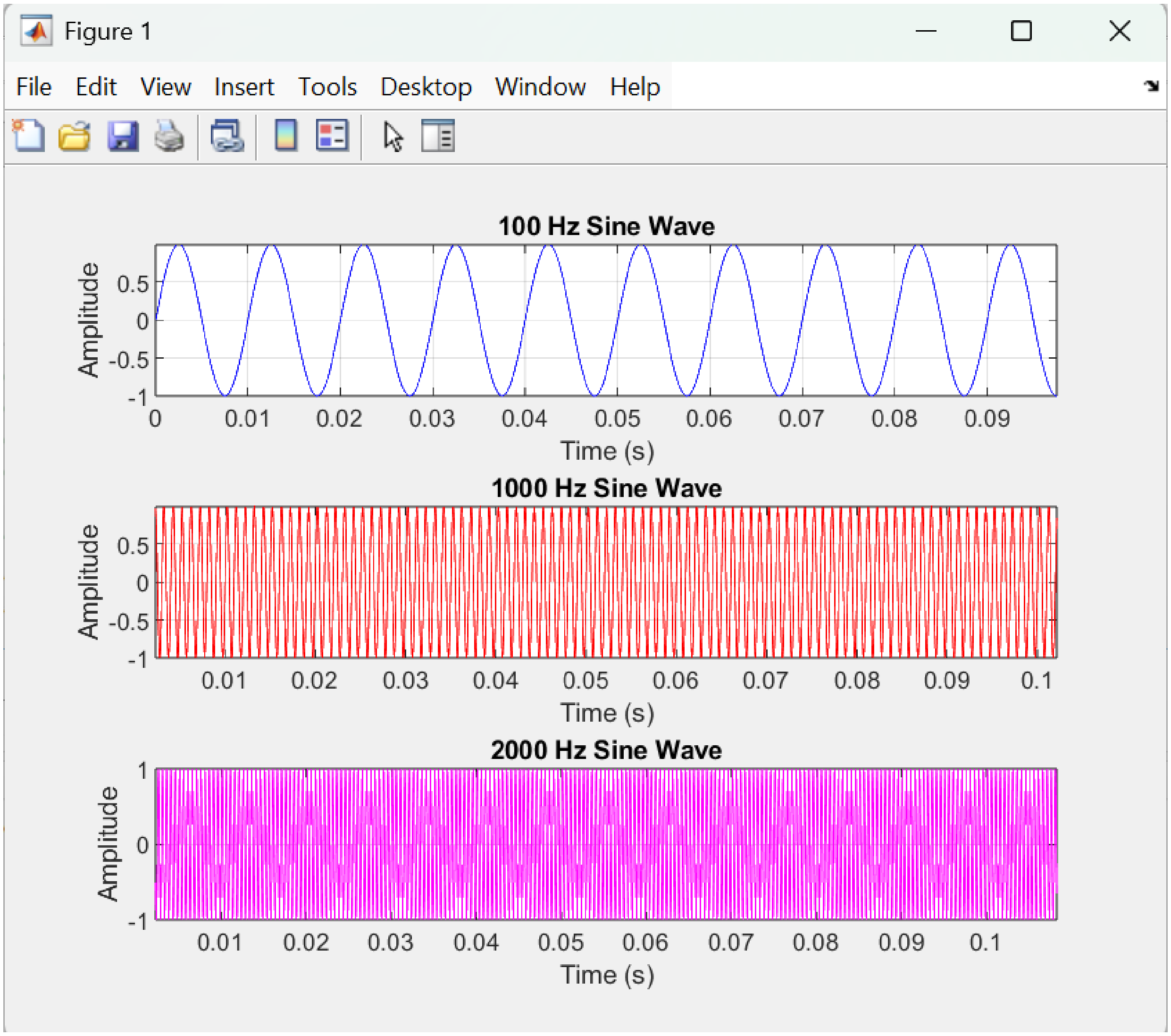

Fig. 4: Generated sine-wave signals.

These values are then stored as text files for use in Verilog code.

III. Without pipeline

Verilog code takes the in txt files of SOS and G and uses direct form 2 to filter. CODE:

```
`timescale 1ns/1ps

module df2_biquad #(
  parameter integer N = 16,
  parameter integer F = 14
)(
  input                      clk,
  input                      rst,
  input   signed [N-1:0]     x_in,
  output reg signed [N-1:0] y_out,
  input   signed [N-1:0]     b0, b1, b2,
```

```
  input  signed [N-1:0]   a1, a2
);

  // state registers
  reg signed [N-1:0] w1, w2;
  // full width multipliers
  wire signed [2*N-1:0] m0 = x_in * b0;
  wire signed [2*N-1:0] m1 = x_in * b1;
  wire signed [2*N-1:0] m2 = x_in * b2;
  wire signed [2*N-1:0] m3 = w1   * a1;
  wire signed [2*N-1:0] m4 = w2   * a2;

  always @(posedge clk) begin
    if (rst) begin
      y_out <= 0;
      w1    <= 0;
      w2    <= 0;
    end else begin
      // DF II transposed update:
      y_out <= (m0 >>> F) + w1;
      w1    <= (m1 >>> F) - (m3 >>> F) + w2;
      w2    <= (m2 >>> F) - (m4 >>> F);
    end
  end

endmodule
```

TESTBENCH:

```
`timescale 1ns/1ps

module tb_IIR;
  // parameters
  localparam integer N         = 16;
  localparam integer F         = 14;
  localparam integer N_SOS     = 3;
  localparam integer N_TAPS    = 5;        // b0,b1,b2,a1,a2
  localparam integer N_SAMPLES = 48000;

  // flat memories for coeffs, gain, and samples
  reg signed [N-1:0] coeff_flat [0:N_SOS*N_TAPS-1];
  reg signed [N-1:0] G_mem      [0:0];
  reg signed [N-1:0] sample_mem [0:N_SAMPLES-1];

  // clock, reset, sample in/out
  reg                clk, rst;
  reg signed [N-1:0] sample_in;
  wire signed [N-1:0] y0, y1, y2;
  wire signed [2*N-1:0] y_mul;
  wire signed [N-1:0]   y_out;
```

```
integer i, fid;

// apply overall gain G_mem[0]
assign y_mul = y2 * G_mem[0];
assign y_out = y_mul >>> F;

// instantiate 3 cascaded biquads
df2_biquad #(.N(N), .F(F)) biq0 (
  .clk(clk), .rst(rst), .x_in(sample_in), .y_out(y0),
  .b0(coeff_flat[0]),  .b1(coeff_flat[1]),  .b2(coeff_flat[2]),
  .a1(coeff_flat[3]),  .a2(coeff_flat[4])
);
df2_biquad #(.N(N), .F(F)) biq1 (
  .clk(clk), .rst(rst), .x_in(y0),       .y_out(y1),
  .b0(coeff_flat[5]),  .b1(coeff_flat[6]),  .b2(coeff_flat[7]),
  .a1(coeff_flat[8]),  .a2(coeff_flat[9])
);
df2_biquad #(.N(N), .F(F)) biq2 (
  .clk(clk), .rst(rst), .x_in(y1),       .y_out(y2),
  .b0(coeff_flat[10]), .b1(coeff_flat[11]), .b2(coeff_flat[12]),
  .a1(coeff_flat[13]), .a2(coeff_flat[14])
);

// clock gen: 100 MHz
initial clk = 0;
always #5 clk = ~clk;

// stimulus
initial begin
  // read 15 lines of SOS_q.txt into coeff_flat[0..14]
  $readmemb("SOS_q.txt", coeff_flat);
  // read 1 line of G_q.txt into G_mem[0]
  $readmemb("G_q.txt",  G_mem);

  // ---- run 100 Hz  signal ----
  $readmemb("sine100_q.txt", sample_mem);
  rst = 1; #20; rst = 0;
  fid = $fopen("y100_out.txt","w");
  for (i = 0; i < N_SAMPLES; i = i + 1) begin
    sample_in = sample_mem[i];
    @(posedge clk);
    $fwrite(fid, "%b\n", y_out);
  end
  $fclose(fid);

  // ---- run 1000 Hz  signal ----
  $readmemb("sine1000_q.txt", sample_mem);
  rst = 1; #20; rst = 0;
  fid = $fopen("y1000_out.txt","w");
```

```
    for (i = 0; i < N_SAMPLES; i = i + 1) begin
      sample_in = sample_mem[i];
      @(posedge clk);
      $fwrite(fid, "%b\n", y_out);
    end
    $fclose(fid);

    // ---- run 2000 Hz  signal ----
    $readmemb("sine2000_q.txt", sample_mem);
    rst = 1; #20; rst = 0;
    fid = $fopen("y2000_out.txt","w");
    for (i = 0; i < N_SAMPLES; i = i + 1) begin
      sample_in = sample_mem[i];
      @(posedge clk);
      $fwrite(fid, "%b\n", y_out);
    end
    $fclose(fid);

    $display("All done.");
    $finish;
  end
endmodule
```