EE5900 Programming Assignment

Gautam Singh
CS21BTECH11018

Contents

1 Solutions

1) The given filter is

$$y[n] = ay[n-1] - ax[n] + x[n-1]. \quad (1)$$

a) The direct form II implementation is shown in Figure 1.

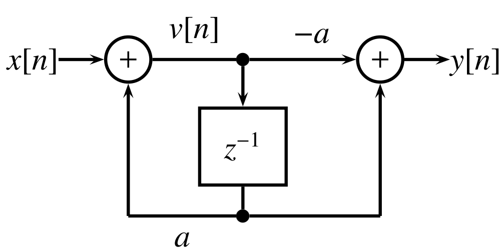

Fig. 1: First Order Direct Form II Allpass Filter.

b) The reduced multiplication implementation is shown in Figure 2.

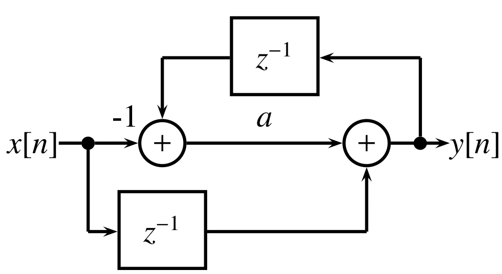

Fig. 2: Reduced First Order Allpass Filter.

c) Source Code 1 generates the outputs in C.

d) Source Code 2 plots the magnitude and phase response for various levels of quantization. The results for direct form II implementation are shown in Figure 3. The results for reduced multiplication implementation are shown in Figure 4.

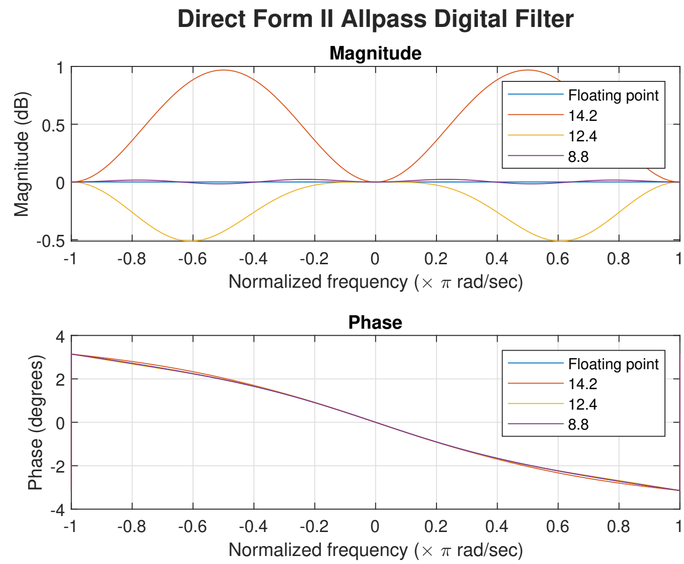

Fig. 3: Direct Form II Filter Response for Various Levels of Quantization.

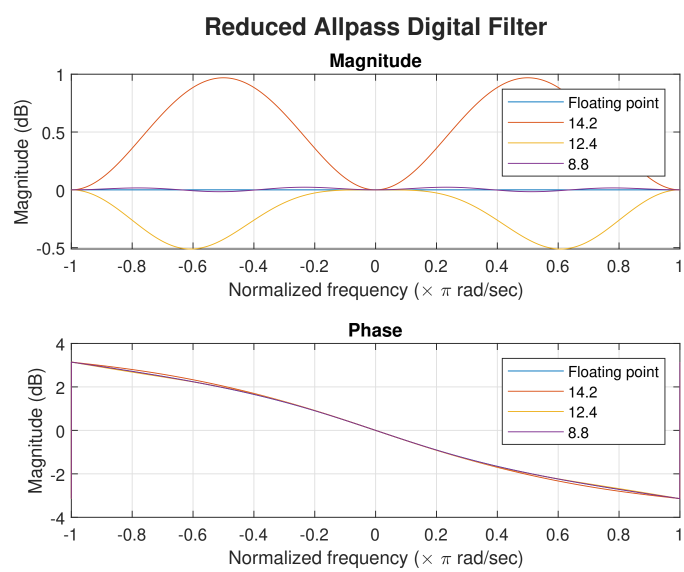

Fig. 4: Reduced Form Filter Response for Various Levels of Quantization.

2) The given filter is

$$y[n] = 0.999y[n-1] + x[n]. \tag{2}$$

Taking the Z-transform on both sides of (2),

$$Y(z) = 0.999z^{-1}Y(z) + X(z) \tag{3}$$

$$\implies H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1-0.999z^{-1}}. \tag{4}$$

From (4), we see that

$$h[n] = (0.999)^n u[n]. \tag{5}$$

Using (5), we have

$$\sum_{n=-\infty}^{\infty} |h[n]|^2 = \sum_{n=0}^{\infty} (0.999)^{2n} \tag{6}$$

$$= \frac{1}{1-(0.999)^2} = 500.25 \tag{7}$$

Therefore, the total quantization output power is ($B = 8$ bits)

$$P_{e,f} = \frac{2^{-2B}}{12}\frac{1}{1-|a|^2} \text{ W} = 6.36 \times 10^{-4} \text{ W}. \tag{8}$$

Source Code 3 simulates the error signal and computes the output quantization power over a number of simulations. The results are shown in Figure 5.

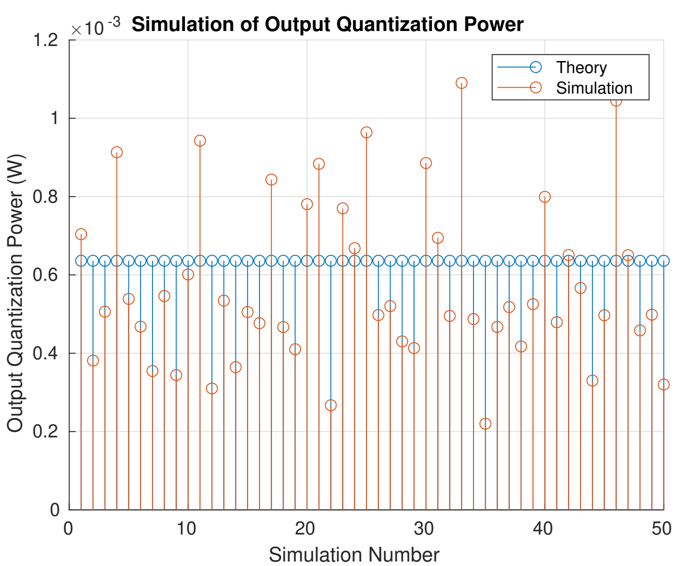

Fig. 5: Simulated Quantization Power.

3) The given impulse response is

$$h[n] = -.375\delta[n] + .75\delta[n-1] - .375\delta[n-2]. \tag{9}$$

a) Using (9), the difference equation is

$$y[n] = -.375x[n] + .75x[n-1] - .375x[n-2]. \tag{10}$$

Thus, we have

$$y[n] = |-.375x(n)| + |.75x(n-1)| + |-.375x(n-2)| \tag{11}$$

$$\implies y[n] \le [.375 + .75 + .375]\, x_{max} \tag{12}$$

$$= 1.5x_{max} < 1 \tag{13}$$

$$\tag{14}$$

Hence, $x_{max} = \frac{1}{1.5} = \frac{2}{3}$.

b) We have,

$$\sum_{n=-\infty}^{\infty} |h[n]|^2 = \sum_{n=0}^{2} |h[n]|^2 = 0.84375. \tag{15}$$

Therefore, the output power is (where $B = 16$ bits)

$$P_{e,f} = \frac{2^{-2B}}{12}\sum_{n=-\infty}^{\infty} |h[n]|^2 = 1.64 \times 10^{-11} \text{ W} \tag{16}$$

Source Code 4 plots the power spectral density shown in Figure 6.

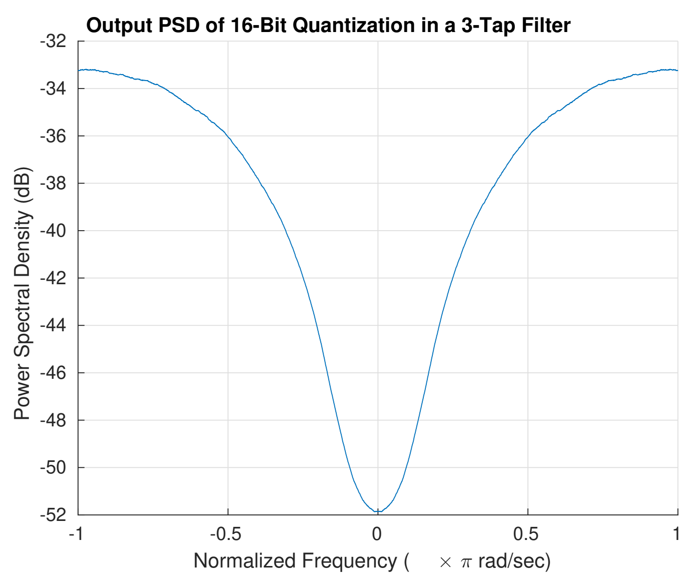

Fig. 6: Power Spectral Density of Quantization Noise.

2 Code

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#define N (int)1e5

/**
 * Quantize a number x given precision prec
 */
double quantize(double x, int prec) {
    if (x != -1) {
        long long X = x*(1LL<<prec);
        double xq = (X*1.0)/(1LL<<prec);
        return xq;
    }
    return x;
}

/**
 * Direct Form II Filter
 */
double *ap_df2_filter(double a, int prec) {
    // Create input and output arrays
    double x[N] = {0};
    x[0] = 1;
    double v[N] = {0};
    v[0] = 1;
    double *y = (double *) malloc(N*sizeof(double));
    double A = quantize(a, prec);
    double Am = quantize(-a, prec);
    y[0] = Am;
    // Accumulate output
    for (int i = 1; i < N; i++) {
        v[i] = x[i] + quantize(A*v[i-1], prec);
        y[i] = quantize(Am*v[i], prec) + v[i-1];
    }
    return y;
}

/**
 * Reduced Form Filter
 */
double *ap_red_filter(double a, int prec) {
    // Create input and output arrays
    double x[N] = {0};
    x[0] = 1;
    double *y = (double *) malloc(N*sizeof(double));
    double A = quantize(a, prec);
    double Am = quantize(-a, prec);
    y[0] = Am;
    // Accumulate output
    for (int i = 1; i < N; i++) {
        y[i] = quantize(A*(y[i-1]-x[i]), prec) + x[i-1];
    }
    return y;
}

int main() {
    double a = 0.1;
    // Quantized outputs for Direct Form II filter
    double *yd1 = ap_df2_filter(a, -1), *yd2 = ap_df2_filter(a, 2);
    // Quantized outputs for Reduced filter
    double *yr1 = ap_red_filter(a, -1), *yr2 = ap_red_filter(a, 2);
    // Compute differences
    double ed[N], er[N];
    for (int i = 0; i < N; i++) {
        ed[i] = yd2[i] - yd1[i];
        er[i] = yr2[i] - yr1[i];
    }
    // Write outputs to a file
    FILE *dfile = fopen("df2_error.txt", "w");
    FILE *rfile = fopen("red_error.txt", "w");
    for (int i = 0; i < N; i++) {
        fprintf(dfile, "%lf\n", ed[i]);
        fprintf(rfile, "%lf\n", er[i]);
    }
    fclose(rfile);
    fclose(dfile);
    return 0;
}
```

Source Code 1: C Code for Question 1.

```matlab
clc;
clear;
close all;

% Filter parameters
a = 0.199;                    % Parameter 'a' of the given allpass filter
N = 256;                      % Number of samples computed
wl = [16];                    % Word length for fixed point implementations
prec = [2 4 8];               % Precision for each word length
w = -pi:(2*pi/8192):pi;       % Frequency range and step

% Array for legends
legend_arr = ["Floating point"];
for i = wl
    for j = prec
        str = append(num2str(i-j), "." ,num2str(j));
        legend_arr = [legend_arr, str];
    end
end

% For each implementation, we do the following:
% 1. Plot response with floating point precision
% 2. Plot response assuming 12,14,16-bit word length and
%   precision of 2,4,8 bits for each word length.

% Direct Form II
% --------------

% Floating Point Implementation

% Obtain time impulse response h[n]
y1 = ap_df2_float(a, N);

% Obtain frequency response H(jw)
[h1, w1] = freqz(y1, 1, w);

% Plot magnitude and phase response
fig = figure(1);
t = tiledlayout(2,1);
nexttile(1);
plot(w1/pi, 20*log10(abs(h1)));
hold on;
nexttile(2);
plot(w1/pi, angle(h1));
hold on;

% Fixed Point Implementation

for i = wl
    for j = prec
        % Obtain hq[n]
        y2 = ap_df2_fixed(a, i, j, N);
        % Find frequency response of quantized filter
        [h2, w2] = freqz(double(y2),1,w);
        % Plot magnitude and phase response
        nexttile(1);
        plot(w2/pi, 20*log10(abs(h2)));
        hold on;
        nexttile(2);
        plot(w2/pi, angle(h2));
        hold on;
    end
end

% Further plotting commands

nexttile(1);
grid on;
xlabel('Normalized frequency (\times \pi rad/sec)');
ylabel('Magnitude (dB)');
legend(legend_arr);
title('Magnitude');
nexttile(2);
grid on;
xlabel('Normalized frequency (\times \pi rad/sec)');
ylabel('Magnitude (dB)');
legend(legend_arr);
title('Phase');
```

```
title(t, 'Direct Form II Allpass Digital Filter', 'fontweight', 'bold');
saveas(fig, [pwd,'/q1_fig1'], 'epsc');

% Reduced Multiplication
% ----------------------

% Floating Point Implementation

% Obtain time impulse response h[n]
y1 = ap_red_float(a, N);

% Obtain frequency response H(jw)
[h1, w1] = freqz(y1, 1, w);

% Plot magnitude and phase response
fig = figure(2);
t = tiledlayout(2,1);
nexttile(1);
plot(w1/pi, 20*log10(abs(h1)));
hold on;
nexttile(2);
plot(w1/pi, angle(h1));
hold on;

% Fixed Point Implementation

for i = wl
    for j = prec
        % Obtain hq[n]
        y2 = ap_red_fixed(a, i, j, N);
        % Find frequency response of quantized filter
        [h2, w2] = freqz(double(y2),1,w);
        % Plot magnitude and phase response
        nexttile(1);
        plot(w2/pi, 20*log10(abs(h2)));
        hold on;
        nexttile(2);
        plot(w2/pi, angle(h2));
        hold on;
    end
end

% Further plotting commands

nexttile(1);
grid on;
xlabel('Normalized frequency (\times \pi rad/sec)');
ylabel('Magnitude (dB)');
legend(legend_arr);
title('Magnitude');
nexttile(2);
grid on;
xlabel('Normalized frequency (\times \pi rad/sec)');
ylabel('Magnitude (dB)');
legend(legend_arr);
title('Phase');

title(t, 'Reduced Allpass Digital Filter', 'fontweight', 'bold');
saveas(fig, [pwd,'/q1_fig2'], 'epsc');

% Floating point all pass filter, direct form II
function y = ap_df2_float(a, N)
    x = zeros(1,N);
    x(1) = 1;
    y = zeros(1, N);
    y(1) = -a;
    v = zeros(1, N);
    v(1) = 1;
    for n = 2:N
        v(n) = x(n) + a*v(n-1);
        y(n) = -a*v(n) + v(n-1);
    end
end

% Floating point all pass filter, reduced multiplications
function y = ap_red_float(a, N)
    x = zeros(1,N);
    x(1) = 1;
    y = zeros(1, N);
    y(1) = -a;
    for n = 2:N
        y(n) = a*(y(n-1)-x(n)) + x(n-1);
    end
end
```

```
% Fixed point all pass filter, direct form II
function y = ap_df2_fixed(a, wl, prec, N)
    x = zeros(1,N);
    x(1) = 1;
    y = fi(zeros(1, N), 1, wl, prec);
    y(1) = -a;
    v = fi(zeros(1, N), 1, wl, prec);
    v(1) = 1;
    for n = 2:N
        v(n) = x(n) + a*v(n-1);
        y(n) = -a*v(n) + v(n-1);
    end
end

% Fixed point all pass filter, reduced multiplications
function y = ap_red_fixed(a, wl, prec, N)
    x = zeros(1,N);
    x(1) = 1;
    y = fi(zeros(1, N), 1, wl, prec);
    y(1) = -a;
    for n = 2:N
        y(n) = a*(y(n-1)-x(n)) + x(n-1);
    end
end
```

Source Code 2: MATLAB Code for Question 1.

```
clc;
clear;
close all;

% Filter parameters
a = 0.999;                    % Parameter 'a' of the given filter
prec = 8;                     % Precision of fractional part
N = 1e4;                      % Number of samples
M = 50;                       % Number of simulations

% Transfer function of filter B/A
B = 1;
A = [1 -a];

% Quantization step
delta = 2^(-prec);

ype = zeros(1, M);
for i = 1:M
    % Generate a random error signal
    xe = rand(1,N)*delta - delta/2;
    % Filter signal according to transfer function
    ye = filter(B,A,xe);
    % Calculate output power
    ype(i) = sumsqr(ye)/length(ye);
end

% Expected power
qp = (delta^2/12)*(1/(1-abs(a)^2));

% Plot simulated power against expected value
fig = figure;
hold on;
stem(1:1:M, qp*ones(1,M));
stem(1:1:M, ype);
xlabel('Simulation Number');
ylabel('Output Quantization Power (W)');
title('Simulation of Output Quantization Power');
legend('Theory', 'Simulation');
grid on;
saveas(fig, [pwd,'/q2_fig1'], 'epsc');
```

Source Code 3: MATLAB Code for Question 2.

```
clc;
clear;
close all;

% Filter parameters
h = [-0.375, 0.75, -0.375]; % Transfer function of filter
prec = 16;                   % Precision of fractional part
N = 1e4;                     % Number of samples

% Quantization step
delta = 2^(-prec);

```

```
% Generate a random error signal
xe = rand(1,N)*delta - delta/2;
% Filter signal according to transfer function
ye = filter(h,1,xe);

% Find frequency response of ye
w = -pi:2*pi/8192:pi;
[yp,wp] = freqz(ye,1,w);

% Smoothen the data for plotting
ypsd = smoothdata(10*log10(abs(yp)));

% Plot power spectral density in principal frequency range
fig = figure;
hold on;
plot(wp/pi, ypsd);
xlabel('Normalized Frequency (\times \pi rad/sec)');
ylabel('Power Spectral Density (dB)');
title('Output PSD of 16-Bit Quantization in a 3-Tap Filter');
grid on;
saveas(fig, [pwd,'/q3_fig1'], 'epsc');
```

Source Code 4: MATLAB Code for Question 3.