

EE5900 Programming Assignment

Gautam Singh
CS21BTECH11018

CONTENTS

- 1 Solutions
- 2 Code

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.

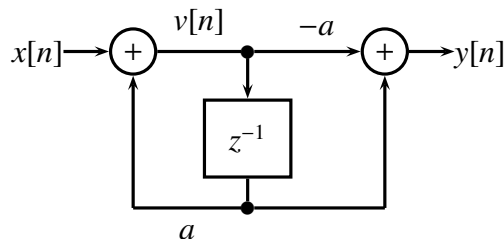


Fig. 1: First Order Direct Form II Allpass Filter.

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

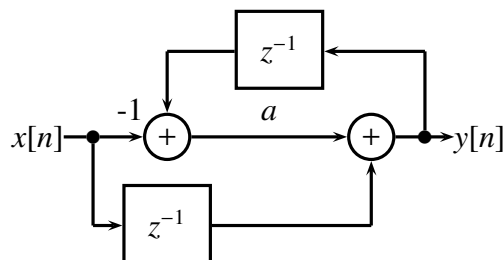


Fig. 2: Reduced First Order Allpass Filter.

- 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

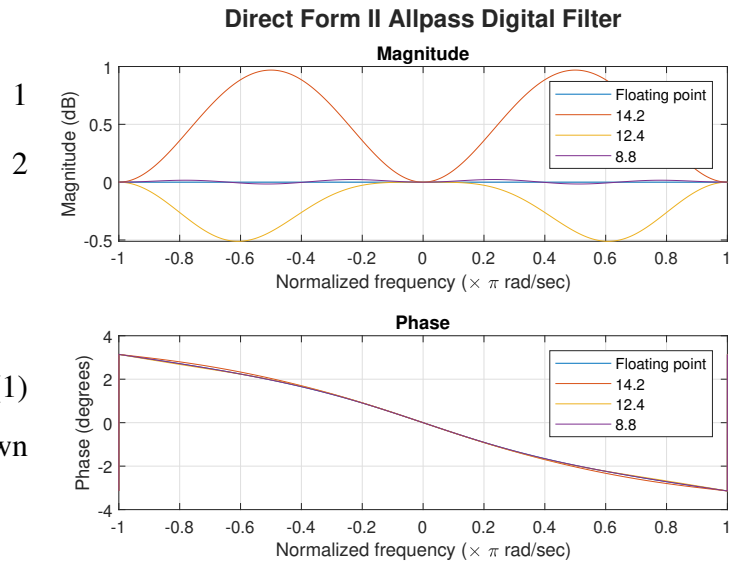


Fig. 3: Direct Form II Filter Response for Various Levels of Quantization.

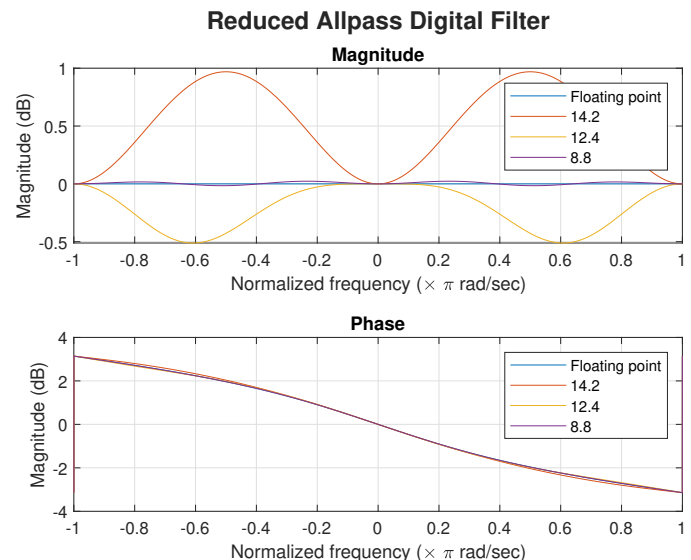


Fig. 4: Reduced Form Filter Response for Various Levels of Quantization.

for reduced multiplication implementation are shown in Figure 4.

2) The given filter is

$$y[n] = 0.999y[n-1] + x[n]. \quad (2)$$

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

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

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

From (4), we see that

$$h[n] = (0.999)^n u[n]. \quad (5)$$

Using (5), we have

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

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

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

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

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.

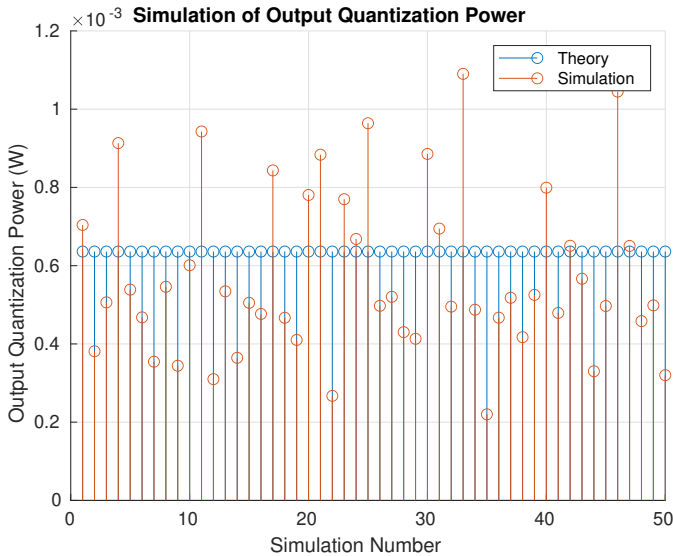


Fig. 5: Simulated Quantization Power.

3) The given impulse response is

$$h[n] = -.375\delta[n] + .75\delta[n-1] - .375\delta[n-2]. \quad (9)$$

a) Using (9), the difference equation is

$$y[n] = -.375x[n] + .75x[n-1] - .375x[n-2]. \quad (10)$$

Thus, we have

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

$$\Rightarrow y[n] \leq [.375 + .75 + .375] x_{max} \quad (12)$$

$$= 1.5x_{max} < 1 \quad (13)$$

$$(14)$$

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

b) We have,

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

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

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

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

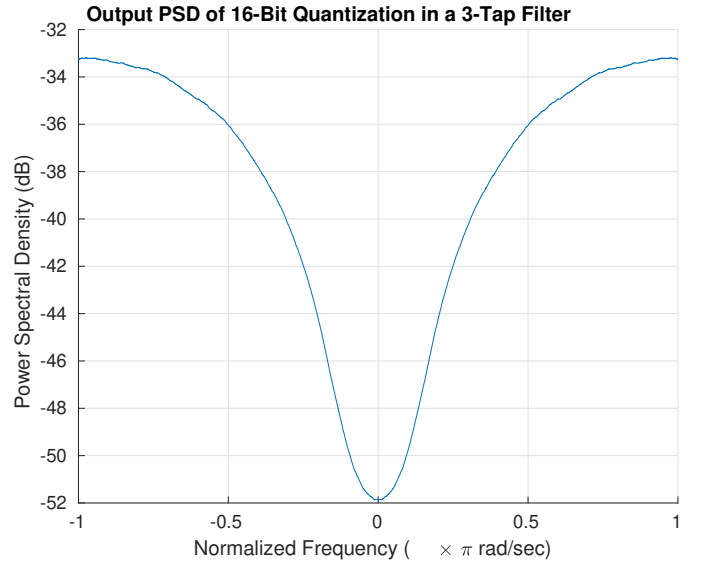


Fig. 6: Power Spectral Density of Quantization Noise.

2 CODE

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

```

3 #define N (int)1e5
4
5 /**
6  * Quantize a number x given
7  *   ↪ precision prec
8  */
9 double quantize(double x, int prec) {
10     ↪
11     if (x != -1) {
12         long long X = x*(1LL<<prec);
13         double xq =
14             ↪ (X*1.0)/(1LL<<prec);
15         return xq;
16     }
17     return x;
18 }
19
20 /**
21  * Direct Form II Filter
22  */
23 double *ap_df2_filter(double a, int
24     ↪ prec) {
25     // Create input and output arrays
26     double x[N] = {0};
27     x[0] = 1;
28     double v[N] = {0};
29     v[0] = 1;
30     double *y = (double *)
31         ↪ malloc(N*sizeof(double));
32     double A = quantize(a, prec);
33     double Am = quantize(-a, prec);
34     y[0] = Am;
35     // Accumulate output
36     for (int i = 1; i < N; i++) {
37         v[i] = x[i] +
38             ↪ quantize(A*v[i-1], prec);
39         y[i] = quantize(Am*v[i],
40             ↪ prec) + v[i-1];
41     }
42     return y;
43 }
44
45 /**
46  * Reduced Form Filter
47  */
48 double *ap_red_filter(double a, int
49     ↪ prec) {
50     // Create input and output arrays
51     double x[N] = {0};
52     x[0] = 1;
53     double *y = (double *)
54         ↪ malloc(N*sizeof(double));
55     double A = quantize(a, prec);
56     double Am = quantize(-a, prec);
57     y[0] = Am;
58     // Accumulate output
59     for (int i = 1; i < N; i++) {
60         y[i] =
61             ↪ quantize(A*(y[i-1]-x[i]),
62             ↪ prec) + x[i-1];
63     }
64     return y;
65 }
66
67 int main() {
68     double a = 0.1;
69     // Quantized outputs for Direct
70     ↪ Form II filter
71     double *yd1 = ap_df2_filter(a,
72         ↪ -1), *yd2 = ap_df2_filter(a,
73         ↪ 2);
74     // Quantized outputs for Reduced
75     ↪ filter
76     double *yr1 = ap_red_filter(a,
77         ↪ -1), *yr2 = ap_red_filter(a,
78         ↪ 2);
79     // Compute differences
80     double ed[N], er[N];
81     for (int i = 0; i < N; i++) {
82         ed[i] = yd2[i] - yd1[i];
83         er[i] = yr2[i] - yr1[i];
84     }
85     // Write outputs to a file
86     FILE *dfile =
87         ↪ fopen("df2_error.txt", "w");
88     FILE *rfile =
89         ↪ fopen("red_error.txt", "w");
90     for (int i = 0; i < N; i++) {
91         fprintf(dfile, "%lf\n",
92             ↪ ed[i]);
93         fprintf(rfile, "%lf\n",
94             ↪ er[i]);
95     }
96     fclose(rfile);
97     fclose(dfile);
98     return 0;
99 }

```

Source Code 1: C Code for Question 1.

```

1  clc;
2  clear;
3  close all;
4
5  % Filter parameters
6  a = 0.199; %
   ↳ Parameter 'a' of the given allpass
   ↳ filter
7  N = 256; % Number
   ↳ of samples computed
8  w1 = [16]; % Word
   ↳ length for fixed point
   ↳ implementations
9  prec = [2 4 8]; %
   ↳ Precision for each word length
10 w = -pi:(2*pi/8192):pi; %
   ↳ Frequency range and step
11
12 % Array for legends
13 legend_arr = ["Floating point"];
14 for i = w1
15     for j = prec
16         str = append(num2str(i-j),
17                     ↳ ". ", num2str(j));
18         legend_arr = [legend_arr,
19                     ↳ str];
20     end
21 end
22
23 % For each implementation, we do the
24 ↳ following:
25 % 1. Plot response with floating
26 ↳ point precision
27 % 2. Plot response assuming
28 ↳ 12,14,16-bit word length and
29 ↳ precision of 2,4,8 bits for each
30 ↳ word length.
31
32 % Direct Form II
33 % -----
34
35 % Floating Point Implementation
36
37 % Obtain time impulse response h[n]
38 y1 = ap_df2_float(a, N);
39
40 % Obtain frequency response H(jw)
41 [h1, w1] = freqz(y1, 1, w);
42
43 % Plot magnitude and phase response
44 fig = figure(1);
45 t = tiledlayout(2,1);
46 nexttile(1);
47 plot(w1/pi, 20*log10(abs(h1)));
48 hold on;
49 nexttile(2);
50 plot(w1/pi, angle(h1));
51 hold on;
52
53 % Fixed Point Implementation
54
55 for i = w1
56     for j = prec
57         % Obtain hq[n]
58         y2 = ap_df2_fixed(a, i, j,
59                         ↳ N);
60         % Find frequency response of
61         ↳ quantized filter
62         [h2, w2] =
63             ↳ freqz(double(y2), 1, w);
64         % Plot magnitude and phase
65         ↳ response
66         nexttile(1);
67         plot(w2/pi,
68             ↳ 20*log10(abs(h2)));
69         hold on;
70         nexttile(2);
71         plot(w2/pi, angle(h2));
72         hold on;
73     end
74 end
75
76 % Further plotting commands
77
78 nexttile(1);
79 grid on;
80 xlabel('Normalized frequency (\times
81 ↳ \pi rad/sec)');
82 ylabel('Magnitude (dB)');
83 legend(legend_arr);
84 title('Magnitude');
85 nexttile(2);
86 grid on;
87 xlabel('Normalized frequency (\times
88 ↳ \pi rad/sec)');
89 ylabel('Magnitude (dB)');
90 legend(legend_arr);
91 title('Phase');

```

```

80 title(t, 'Direct Form II Allpass
    ↳ Digital Filter', 'fontweight',
    ↳ 'bold');
81 saveas(fig, [pwd, '/q1_fig1'],
    ↳ 'epsc');
82
83 % Reduced Multiplication
84 % -----
85
86 % Floating Point Implementation
87
88 % Obtain time impulse response h[n]
89 y1 = ap_red_float(a, N);
90
91 % Obtain frequency response H(jw)
92 [h1, w1] = freqz(y1, 1, w);
93
94 % Plot magnitude and phase response
95 fig = figure(2);
96 t = tiledlayout(2,1);
97 nexttile(1);
98 plot(w1/pi, 20*log10(abs(h1)));
99 hold on;
100 nexttile(2);
101 plot(w1/pi, angle(h1));
102 hold on;
103
104 % Fixed Point Implementation
105
106 for i = w1
107     for j = prec
108         % Obtain hq[n]
109         y2 = ap_red_fixed(a, i, j,
            ↳ N);
110         % Find frequency response of
            ↳ quantized filter
111         [h2, w2] =
            ↳ freqz(double(y2), 1, w);
112         % Plot magnitude and phase
            ↳ response
113         nexttile(1);
114         plot(w2/pi,
            ↳ 20*log10(abs(h2)));
115         hold on;
116         nexttile(2);
117         plot(w2/pi, angle(h2));
118         hold on;
119     end
120 end
121

```

```

122 % Further plotting commands
123
124 nexttile(1);
125 grid on;
126 xlabel('Normalized frequency (\times
    ↳ \pi rad/sec)');
127 ylabel('Magnitude (dB)');
128 legend(legend_arr);
129 title('Magnitude');
130 nexttile(2);
131 grid on;
132 xlabel('Normalized frequency (\times
    ↳ \pi rad/sec)');
133 ylabel('Magnitude (dB)');
134 legend(legend_arr);
135 title('Phase');
136
137 title(t, 'Reduced Allpass Digital
    ↳ Filter', 'fontweight', 'bold');
138 saveas(fig, [pwd, '/q1_fig2'],
    ↳ 'epsc');
139
140 % Floating point all pass filter,
    ↳ direct form II
141 function y = ap_df2_float(a, N)
142     x = zeros(1,N);
143     x(1) = 1;
144     y = zeros(1, N);
145     y(1) = -a;
146     v = zeros(1, N);
147     v(1) = 1;
148     for n = 2:N
149         v(n) = x(n) + a*v(n-1);
150         y(n) = -a*v(n) + v(n-1);
151     end
152 end
153
154 % Floating point all pass filter,
    ↳ reduced multiplications
155 function y = ap_red_float(a, N)
156     x = zeros(1,N);
157     x(1) = 1;
158     y = zeros(1, N);
159     y(1) = -a;
160     for n = 2:N
161         y(n) = a*(y(n-1)-x(n)) +
            ↳ x(n-1);
162     end
163 end
164

```

```

165 % Fixed point all pass filter, direct
    ↪ form II
166 function y = ap_df2_fixed(a, wl,
    ↪ prec, N)
167     x = zeros(1,N);
168     x(1) = 1;
169     y = fi(zeros(1, N), 1, wl, prec);
170     y(1) = -a;
171     v = fi(zeros(1, N), 1, wl, prec);
172     v(1) = 1;
173     for n = 2:N
174         v(n) = x(n) + a*v(n-1);
175         y(n) = -a*v(n) + v(n-1);
176     end
177 end
178
179 % Fixed point all pass filter,
    ↪ reduced multiplications
180 function y = ap_red_fixed(a, wl,
    ↪ prec, N)
181     x = zeros(1,N);
182     x(1) = 1;
183     y = fi(zeros(1, N), 1, wl, prec);
184     y(1) = -a;
185     for n = 2:N
186         y(n) = a*(y(n-1)-x(n)) +
            ↪ x(n-1);
187     end
188 end

```

Source Code 2: MATLAB Code for Question 1.

```

1 clc;
2 clear;
3 close all;
4
5 % Filter parameters
6 a = 0.999; %
    ↪ Parameter 'a' of the given filter
7 prec = 8; %
    ↪ Precision of fractional part
8 N = 1e4; % Number
    ↪ of samples
9 M = 50; % Number
    ↪ of simulations
10
11 % Transfer function of filter B/A
12 B = 1;
13 A = [1 -a];
14

```

```

15 % Quantization step
16 delta = 2^(-prec);
17
18 ype = zeros(1, M);
19 for i = 1:M
20     % Generate a random error signal
21     xe = rand(1,N)*delta - delta/2;
22     % Filter signal according to
        ↪ transfer function
23     ye = filter(B,A,xe);
24     % Calculate output power
25     ype(i) = sumsqr(ye)/length(ye);
26 end
27
28 % Expected power
29 qp = (delta^2/12)*(1/(1-abs(a)^2));
30
31 % Plot simulated power against
    ↪ expected value
32 fig = figure;
33 hold on;
34 stem(1:1:M, qp*ones(1,M));
35 stem(1:1:M, ype);
36 xlabel('Simulation Number');
37 ylabel('Output Quantization Power
    ↪ (W)');
38 title('Simulation of Output
    ↪ Quantization Power');
39 legend('Theory', 'Simulation');
40 grid on;
41 saveas(fig, [pwd, '/q2_fig1'],
    ↪ 'epsc');

```

Source Code 3: MATLAB Code for Question 2.

```

1 clc;
2 clear;
3 close all;
4
5 % Filter parameters
6 h = [-0.375, 0.75, -0.375]; %
    ↪ Transfer function of filter
7 prec = 16; %
    ↪ Precision of fractional part
8 N = 1e4; % Number
    ↪ of samples
9
10 % Quantization step
11 delta = 2^(-prec);
12

```

```

13 % Generate a random error signal
14 xe = rand(1,N)*delta - delta/2;
15 % Filter signal according to transfer
    ↪ function
16 ye = filter(h,1,xe);
17
18 % Find frequency response of ye
19 w = -pi:2*pi/8192:pi;
20 [yp,wp] = freqz(ye,1,w);
21
22 % Smoothen the data for plotting
23 ypsd = smoothdata(10*log10(abs(yp)));
24
25 % Plot power spectral density in
    ↪ principal frequency range
26 fig = figure;
27 hold on;
28 plot(wp/pi, ypsd);
29 xlabel('Normalized Frequency (\times
    ↪ \pi rad/sec)');
30 ylabel('Power Spectral Density
    ↪ (dB)');
31 title('Output PSD of 16-Bit
    ↪ Quantization in a 3-Tap Filter');
32 grid on;
33 saveas(fig, [pwd, '/q3_fig1'],
    ↪ 'epsc');

```

Source Code 4: MATLAB Code for Question 3.