Eng. Julian S. Bruno

REAL TIME DIGITAL SIGNAL PROCESSING

Fixed Point Algorithm

Bla bla bla

Eng. Julian S. Bruno

RMS Algorithm

RMS

$$RMS = \sqrt{\frac{\sum_{i=0}^{N-1} x_i^2}{N}}$$

RMS
BF53X Fixed Point Simulation

```
%% Gold Standard
rms_gs = sqrt(sum(sig.^2)./N)

%% Algorithm (Double)
exp = fix(-1*log10(max(sig))/log10(2));
aux = sig*(2^exp);
sum_cuadra = sum(aux.^2)./N;
rms_double = sqrt(sum_cuadra)/(2^exp);

%% Fixed-Point Variable Definitions
nt_1_15 = numerictype(1, 16, 15);
nt_1_31 = numerictype(1, 32, 31);
nt_9_31 = numerictype(1, 40, 31);

dsp_bf537 = fimath('RoundMode', 'Fix', ...
    'OverflowMode', 'Saturate', ...
    'ProductMode', 'SpecifyPrecision', ...
    'ProductWordLength', 32, ...
    'ProductFractionLength', 31, ...
    'SumMode', 'SpecifyPrecision', ...
    'SumWordLength', 40, ...
    'SumFractionLength', 31, ...
    'CastBeforeSum', false);

sig_fix = fi(sig,nt_1_15,dsp_bf537);
reg_fix = fi(0,nt_1_31,dsp_bf537);
acc_fix = fi(0,nt_9_40,dsp_bf537);
```

```
%% Algorithm (Fixed-Point)

% deteccion de exponente
exp = fix(-1*log10(double(max(sig_fix)))/log10(2));
% log2(N)
 inv_size = fix(-1*log10(N)/log10(2));

for i =1:length(sig)
    reg_fix.data = sig_fix.data(i);
    % escalo la señal de entrada
    reg_fix.bin = bin(bitshift(reg_fix,exp));
    % elevo al cuadrado
    reg_fix = reg_fix * reg_fix;
    % divido por N
    reg_fix.bin = bin(bitshift(reg_fix,inv_size));
    % acumulo
    acc_fix = acc_fix + reg_fix ;
end;

% trunco y redondeo
reg_fix = fi(acc_fix,nt_q31);

% sqrt
reg_fix.data = sqrt(reg_fix .data);

% rescalo
reg_fix.bin = bin(bitshift(reg_fix,-1*exp));
```

RMS
ASM Implementation

```
//************************************************************************
// Detect Block exponent
//************************************************************************
        R2.h = 0; R5.h = 0;
        R2.l = 0x7F; R5.l = 0;
        R4 = w[P0++](Z);
        LSETUP (.loop_be,.loop_be) LC0 = P5;
.loop_be:       R2.l = expadj (R4.l, R2.l) || R4 = w[P0++](Z);
        R2.l = expadj (R4.l, R2.l);                         //R2.L block adjusted exp

//************************************************************************
// Escalo la señal de entrada segun lo indicado en el block exponent
//************************************************************************
        //  Se está pisando el vector de samples con las samples escaladas!!!  Más didactico
        I0 = R0; P0 = R0;
        R4 = w[P0++](Z);
        LSETUP (.loop_ba_s,.loop_ba_e) LC0 = P5;
.loop_ba_s:     R3.h = ashift R4.l by R2.l;             // escalo
.loop_ba_e:     R4=W[P0++](Z) || W[I0++]=R3.h;          // guardo la muestra escalada y leo la próxima
        R3.h = ashift R4.l by R2.l;                     // escalo
        W[I0++] =  R3.h ;                               // guardo la muestra escalada
```

RMS
ASM Implementation

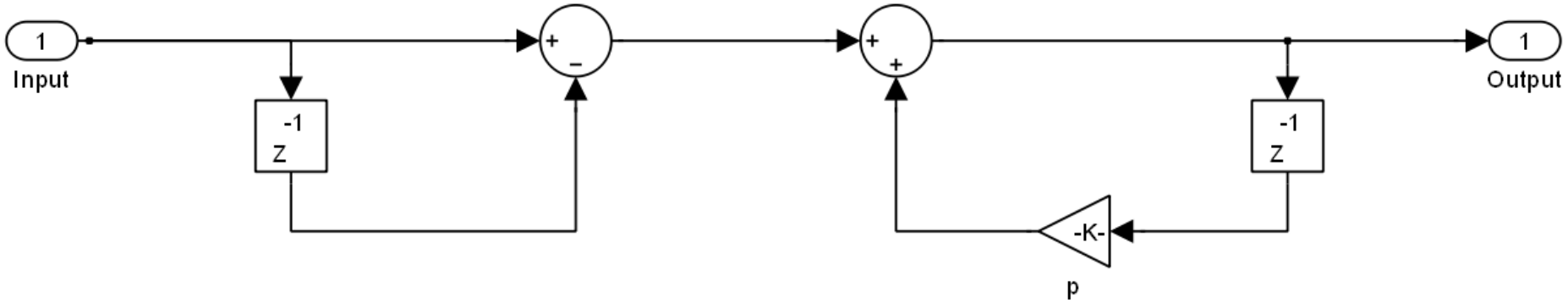
```
//*********************************************************************
// Calculo el log2(N) para poder hacer x/n como una rotación de bits
//*********************************************************************
        R3.h = 0;
        R3.l = SIGNBITS R1.l;
        R3 += -14;                                      // R3 = log2(N)
//*********************************************************************
// Calculo la suma de los cuadrados de los elementos divididos N
//*********************************************************************
        P0 = R0;
        A1 = A0 = 0   || R6 = W[P0++](Z);               // Prime loop by loading X[0]

        LSETUP (.loop_s,.loop_e) LC0 = P5;
.loop_s:    A1 = R6.l * R6.l || R6 = W[P0++](Z);        // A1 = x^2 , R6=x[i]
            A1 = ashift A1 by R3.l ;                    // A1 = (x^2)/N
.loop_e:    A0 += A1;                                   // A0 += (x^2)/N

        A1 = R6.l * R6.l;
        A1 = ashift A1 by R3.l ;
        A0 += A1;                                       // A0 = sumatoria de x^2/n (9.31)
//*********************************************************************
// Calculate SQRT ((sum of the squares)/N)
//*********************************************************************
        R0 = A0;                                        // sumatoria de x^2/n (1.31)
        CALL.X _sqrt_fr32;
```

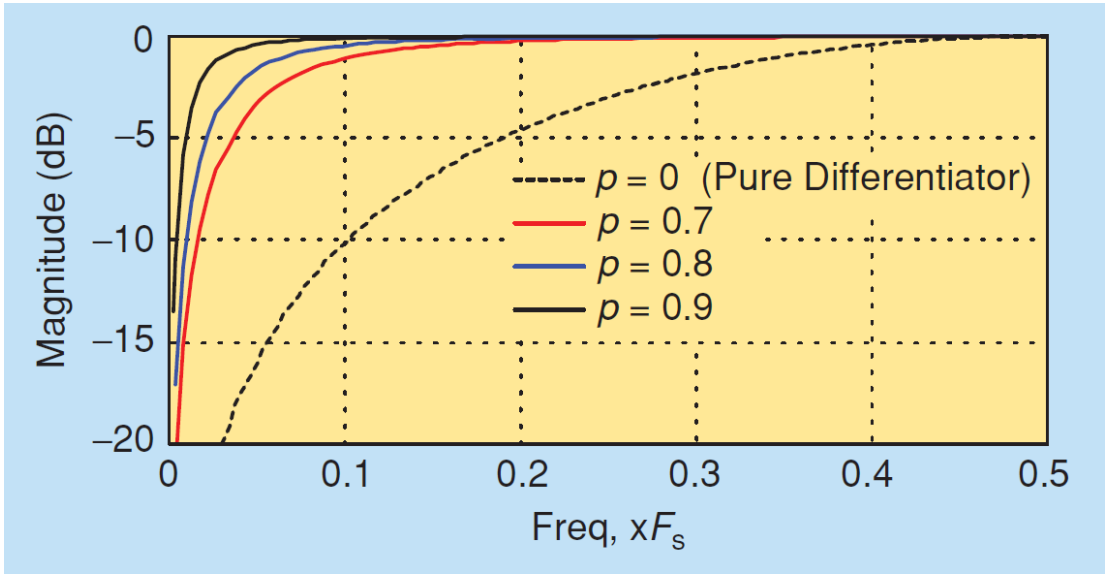
RMS
ASM Implementation

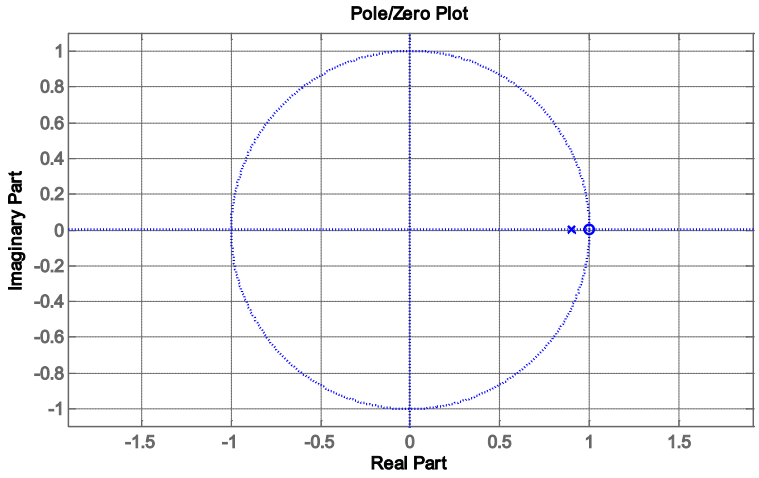
```
//*********************************************************************
// Corrijo el resultado debido al escalamiento de la señal de entrada
//*********************************************************************

        R4.l = 0;
        R3.l = R4.l - R2.l(NS);
        R4 = ashift R0 by R3.l ;                // apply any remaining scaling required
        R0 = R4;                                // R0 = rms(x)
```

DC Blocker

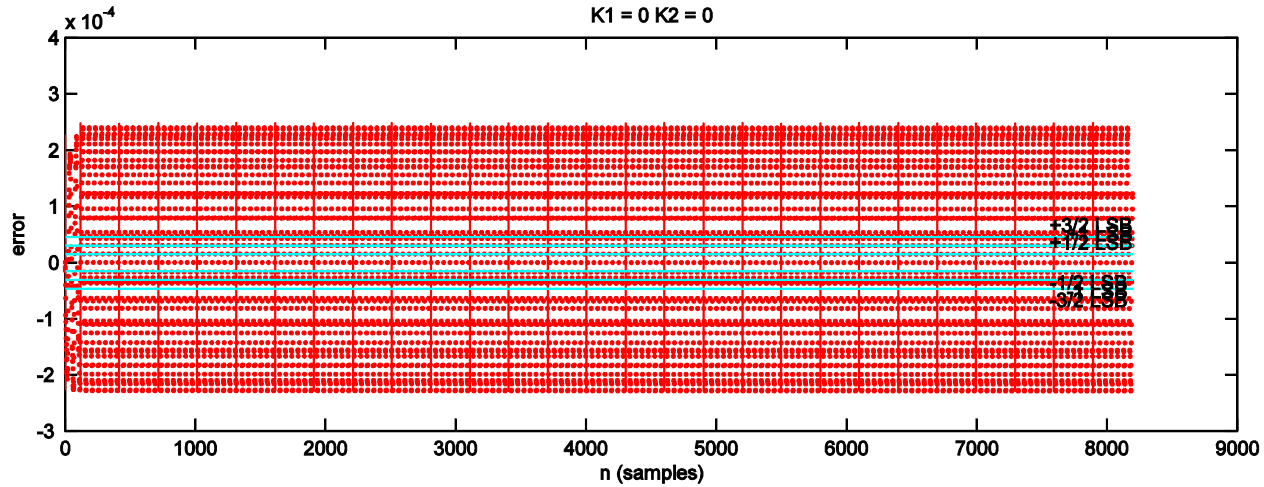
DC Blocker

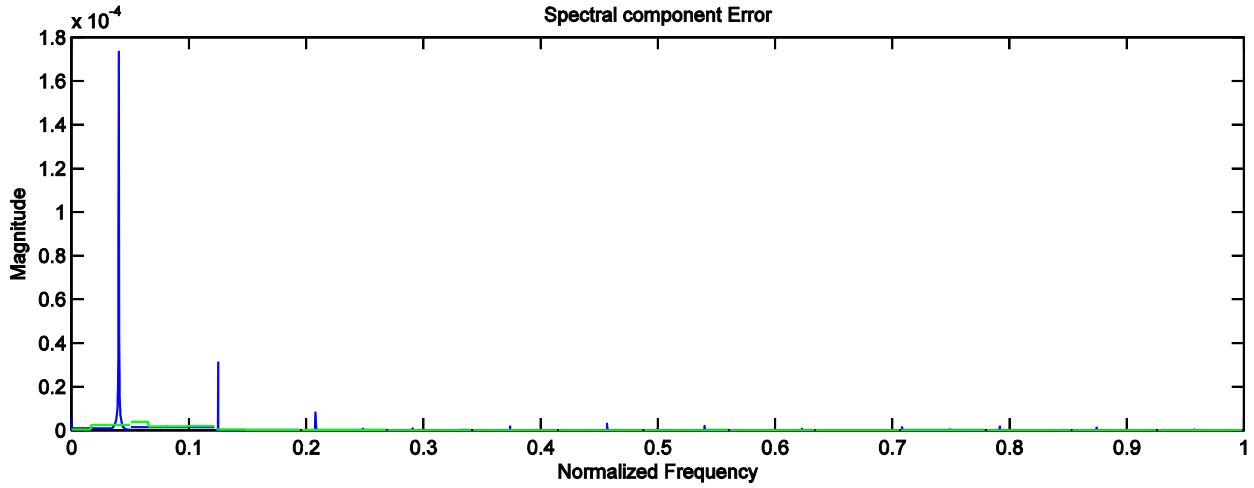

1
Input
-1
z
+
–
+
+
-1
z
1
Output
-K-
p


Magnitude (dB)
0
–5
–10
–15
–20
0
0.1
0.2
0.3
0.4
0.5
Freq, xF_s
p = 0 (Pure Differentiator)
p = 0.7
p = 0.8
p = 0.9

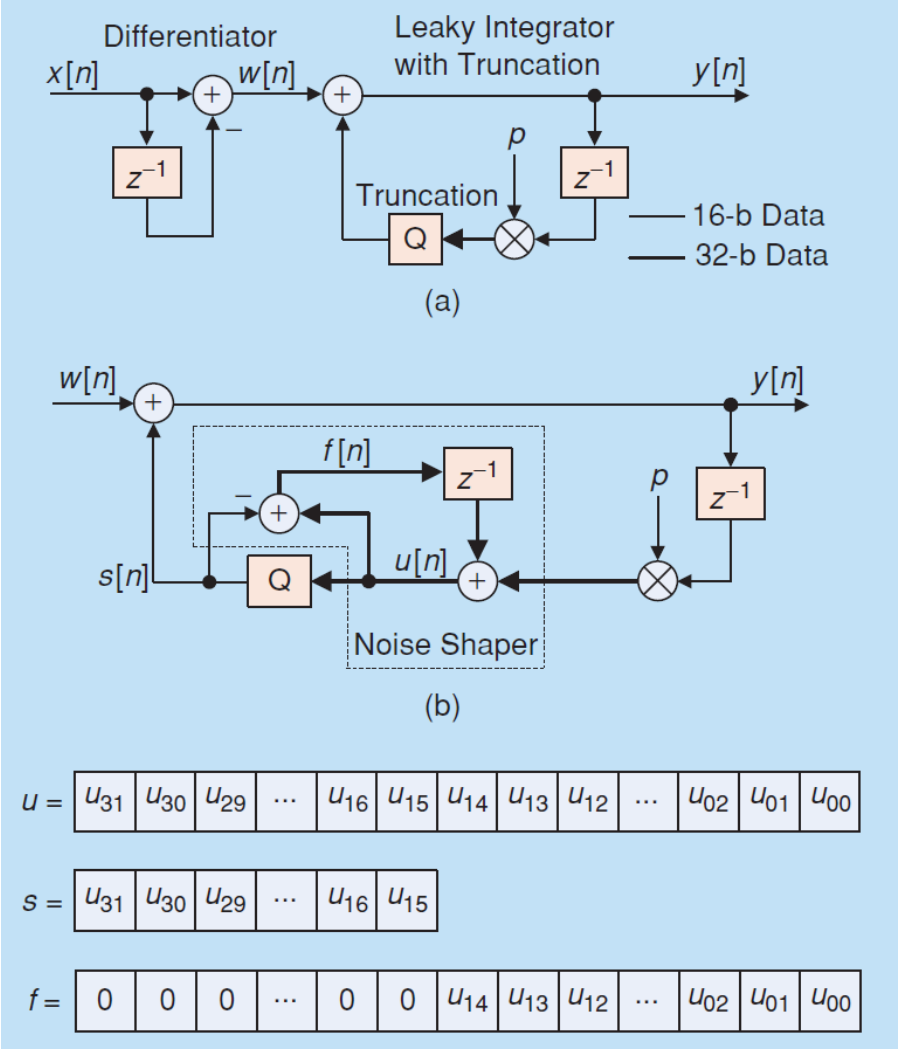

Pole/Zero Plot
Imaginary Part
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.5
-1
-0.5
0
0.5
1
1.5
Real Part

DC Blocker without Error Spectral Shaping (ESS)

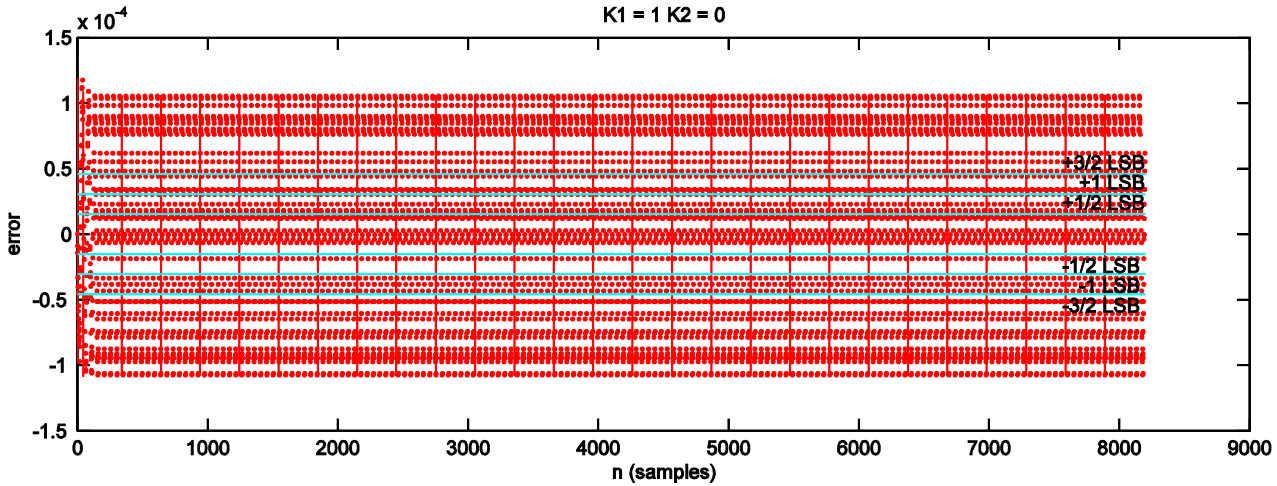


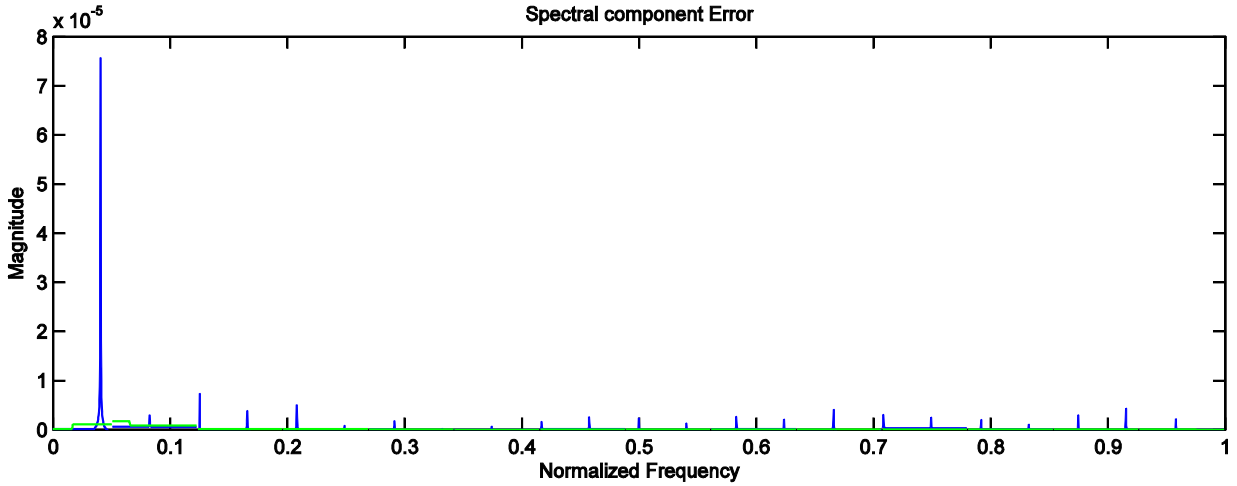


DC Blocker with Error Spectral Shaping (ESS)

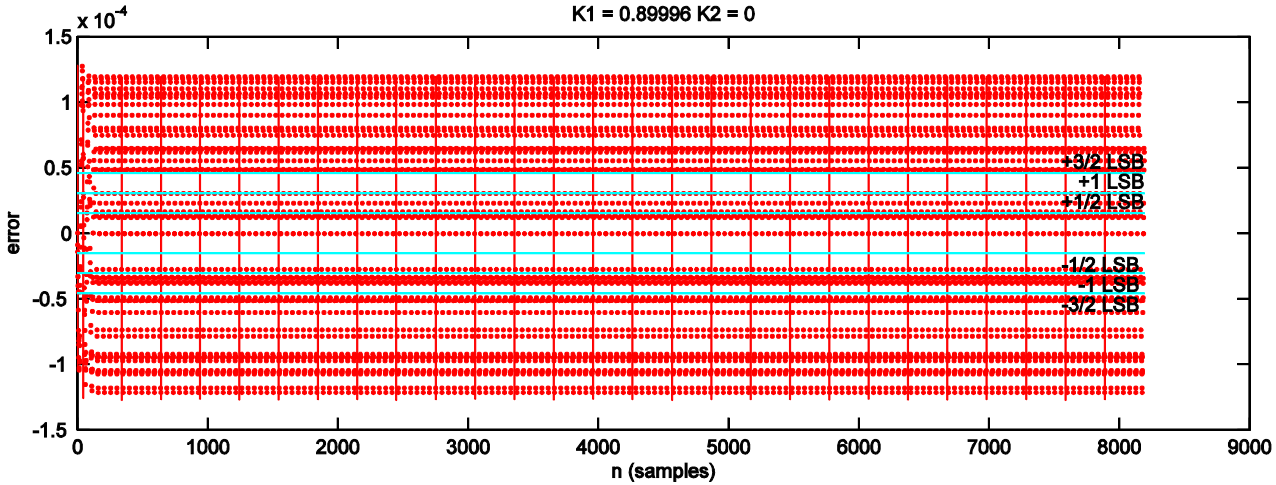


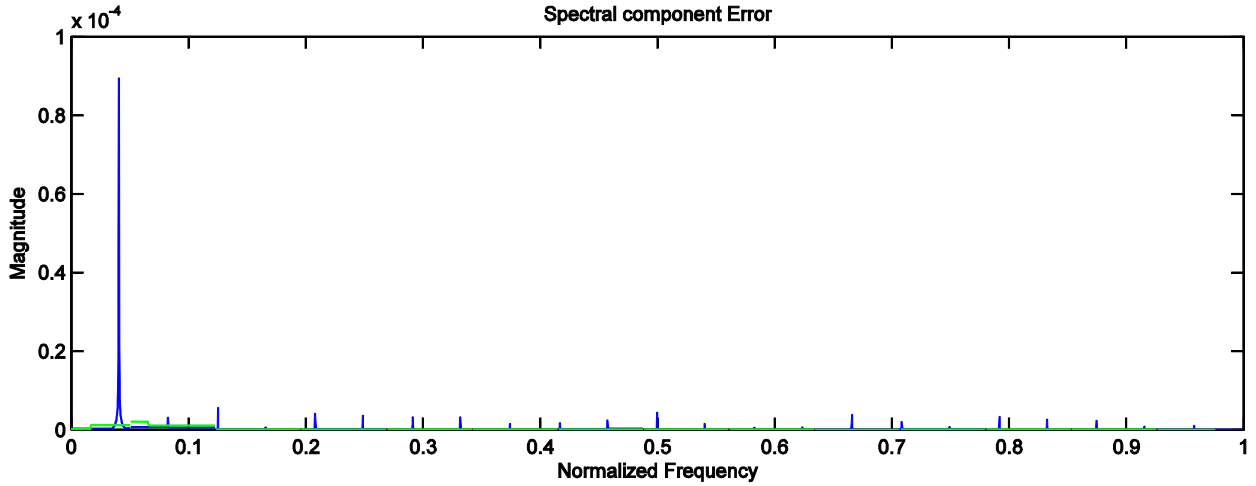
DC Blocker with Error Spectral Shaping (ESS)





DC Blocker with Error Spectral Shaping (ESS)

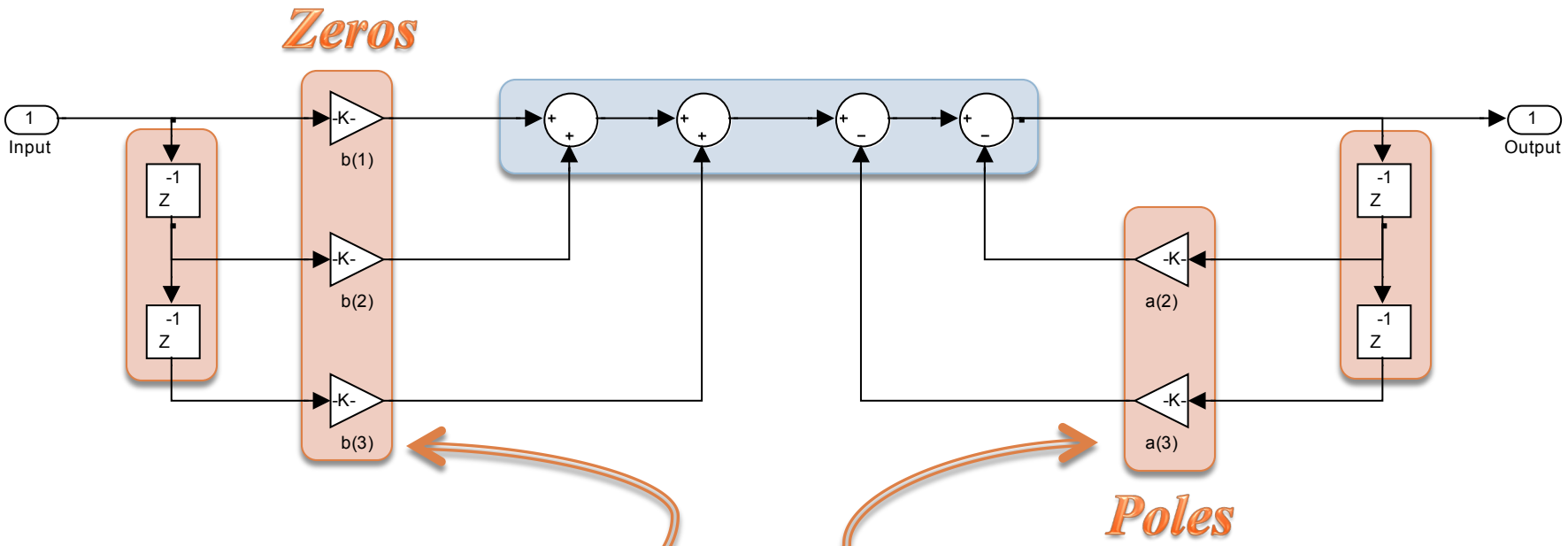




Second Order Section (SOS)

Eng. Julian S. Bruno

SOS DFI – Study case

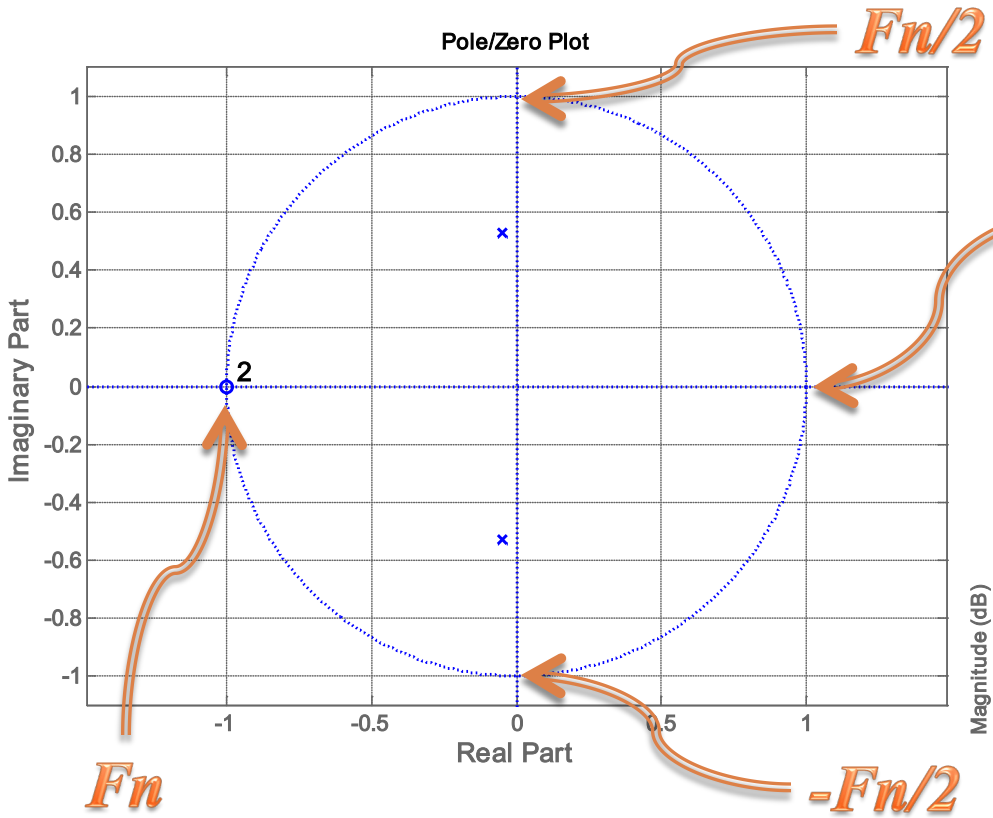


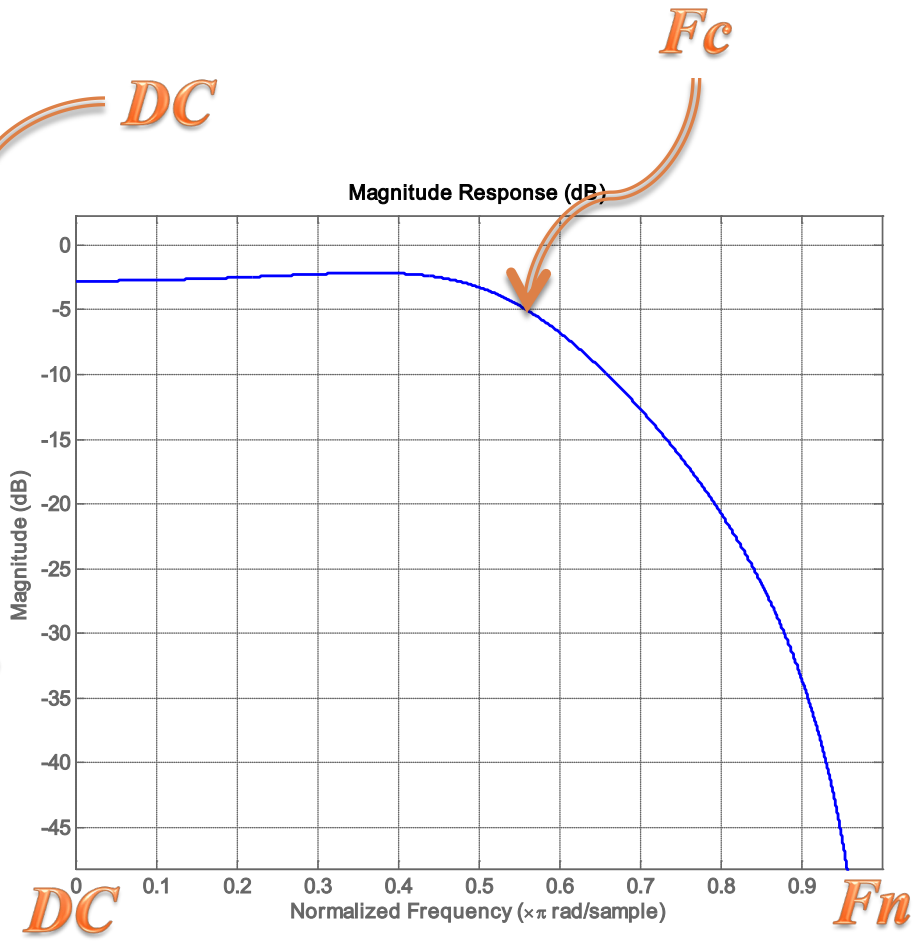
$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

$$Y(z) = X(z)(b_0 + b_1 z^{-1} + b_2 z^{-2}) - Y(z)(a_1 z^{-1} + a_2 z^{-2})$$

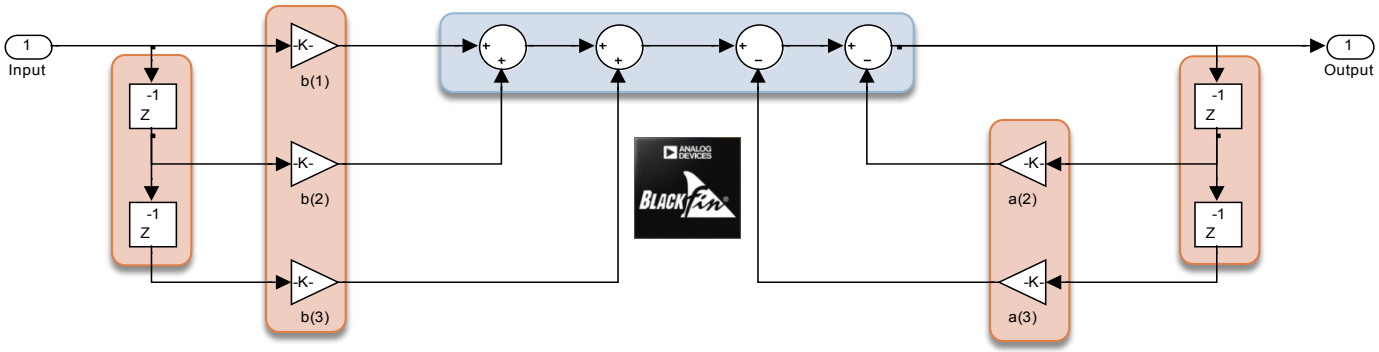
$$b_0 = 0.25$$
$$b_1 = 0.50 \quad a_1 = 0.09375$$
$$b_2 = 0.25 \quad a_2 = 0.28125$$

SOS DFI – Study case





SOS DFI – Study case
BF53X Floating Point Simulation



$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

$$b_0 = 0.25$$

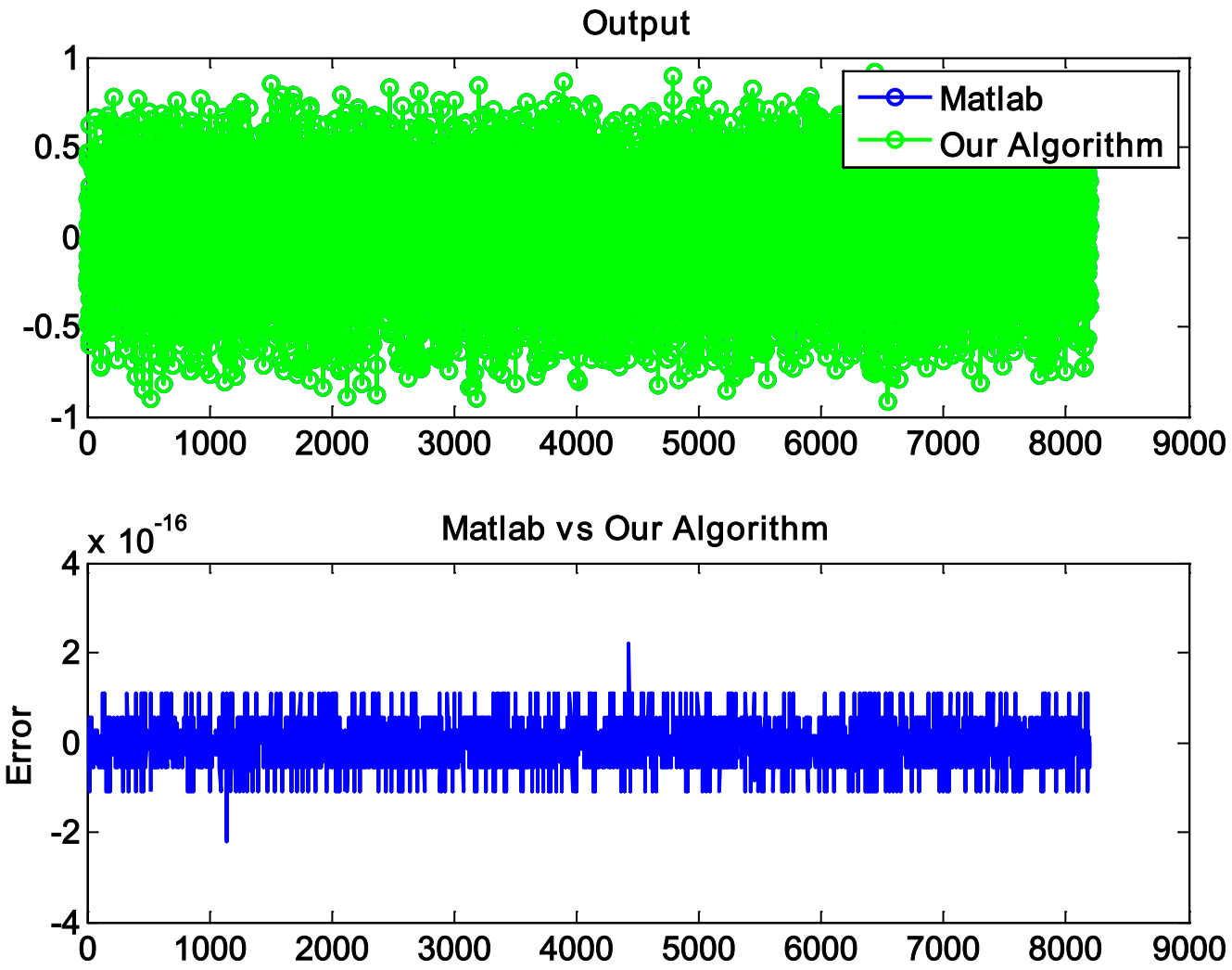
$$b_1 = 0.50 \quad a_1 = 0.09375$$

$$b_2 = 0.25 \quad a_2 = 0.28125$$

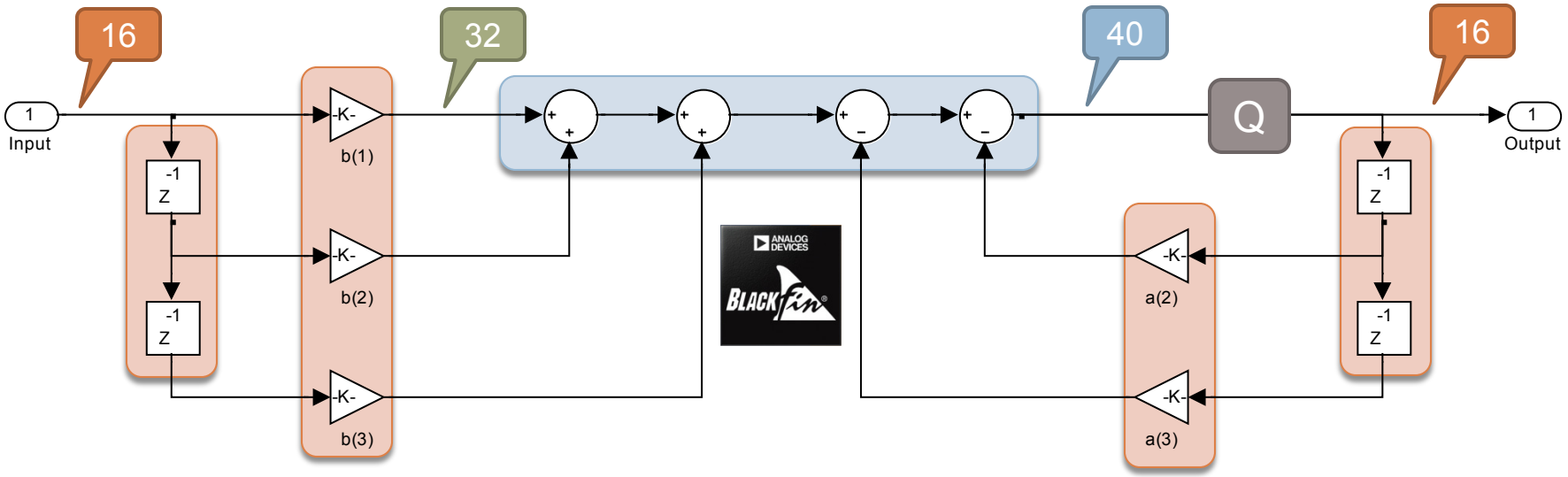
```
%% Gold Standard

Numerator   = [0.25 0.5 0.25];
Denominator = [1 0.09375 0.28125];
y_gs = filter(Numerator, Denominator, x);
```

```
% Algorithm (Double)

for k=1:N
    A0 = 0;
    R0 = b(1)*x(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y(k)=A0;

    xz(2) = xz(1);
    xz(1) = x(k);

    yz(2) = yz(1);
    yz(1) = y(k);
end
```

SOS DFI – Study case
BF53X Floating Point Simulation



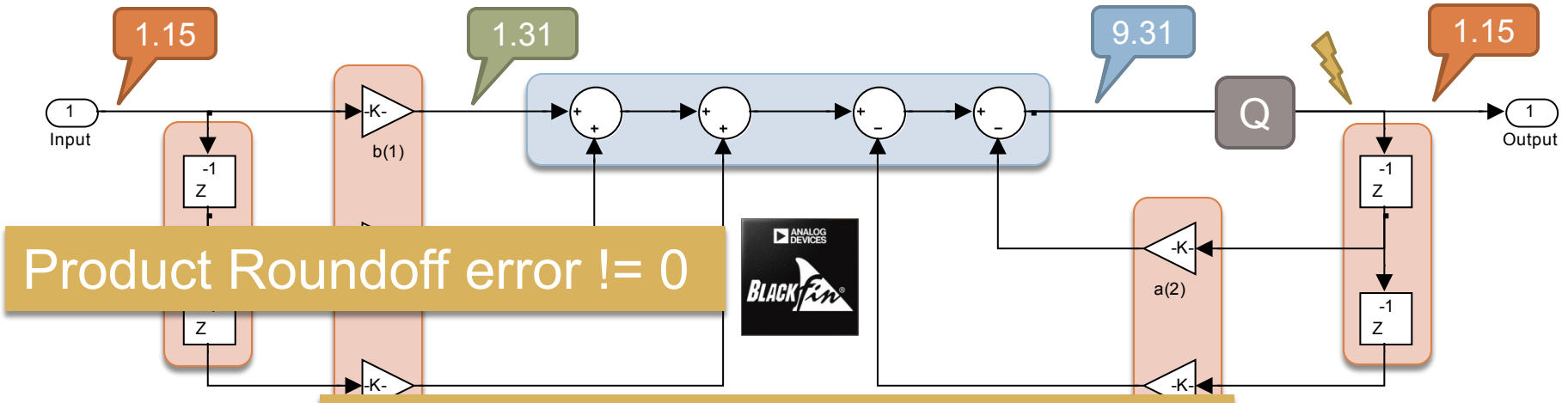
SOS DFI – Study case
BF53X Architecture



$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

$$b_0 = 0.25$$
$$b_1 = 0.50 \quad a_1 = 0.09375$$
$$b_2 = 0.25 \quad a_2 = 0.28125$$

SOS DFI – Study case
BF53X Fixed Point Simulation



$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

$$\hat{H}(z) = \frac{\hat{Y}(z)}{\hat{X}(z)} = \frac{\hat{b}_0 + \hat{b}_1 z^{-1} + \hat{b}_2 z^{-2}}{1 + \hat{a}_1 z^{-1} + \hat{a}_2 z^{-2}}$$

Coefficient quantization error = 0

$$b_0 = 0.25$$
$$b_1 = 0.50 \quad a_1 = 0.09375$$
$$b_2 = 0.25 \quad a_2 = 0.28125$$

1.15

$$\hat{b}_0 = 0.25$$
$$\hat{b}_1 = 0.50 \quad \hat{a}_1 = 0.09375$$
$$\hat{b}_2 = 0.25 \quad \hat{a}_2 = 0.28125$$

SOS DFI – Study case
BF53X Fixed Point Simulation

```
% Algorithm (Double)

for k=1:N
    A0 = 0;
    R0 = b(1)*x(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y(k)=A0;

    xz(2) = xz(1);
    xz(1) = x(k);

    yz(2) = yz(1);
    yz(1) = y(k);
end
```

```
% Define Numeric Types
nt_1_15 = numerictype(1, 16, 15);
nt_1_31 = numerictype(1, 32, 31);
nt_9_31 = numerictype(1, 40, 31);

% Define Fimath
dsp_bf537 = fimath('RoundMode', 'Fix', ...
    'OverflowMode', 'Saturate', ...
    'ProductMode', 'SpecifyPrecision', ...
    'ProductWordLength', 32, ...
    'ProductFractionLength', 31, ...
    'SumMode', 'SpecifyPrecision', ...
    'SumWordLength', 40, ...
    'SumFractionLength', 31, ...
    'CastBeforeSum', false);

%  fi(Data, NumericType, Fimath)
b = fi(b, nt_1_15, dsp_bf537);
a = fi(a, nt_1_15, dsp_bf537);

x_fix = fi(x, nt_1_15, dsp_bf537);
y_fix = fi(zeros(N,1), nt_1_15, dsp_bf537);
xz = fi([0;0], nt_1_15, dsp_bf537);
yz = fi([0;0], nt_1_15, dsp_bf537);

R0 = fi(0, nt_1_31, dsp_bf537);
A0 = fi(0, nt_9_31, dsp_bf537);
```

```
% Algorithm (Fixed-Point)

for k=1:N
    A0.data = 0;
    R0 = b(1)*x_fix(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y_fix(k)=fi(A0,nt_1_15); % redondeo y trunco

    xz(2) = xz(1);
    xz(1) = x_fix(k);

    yz(2) = yz(1);
    yz(1) = y_fix(k);
end
```

SOS DFI – Study case
BF53X Fixed Point Simulation

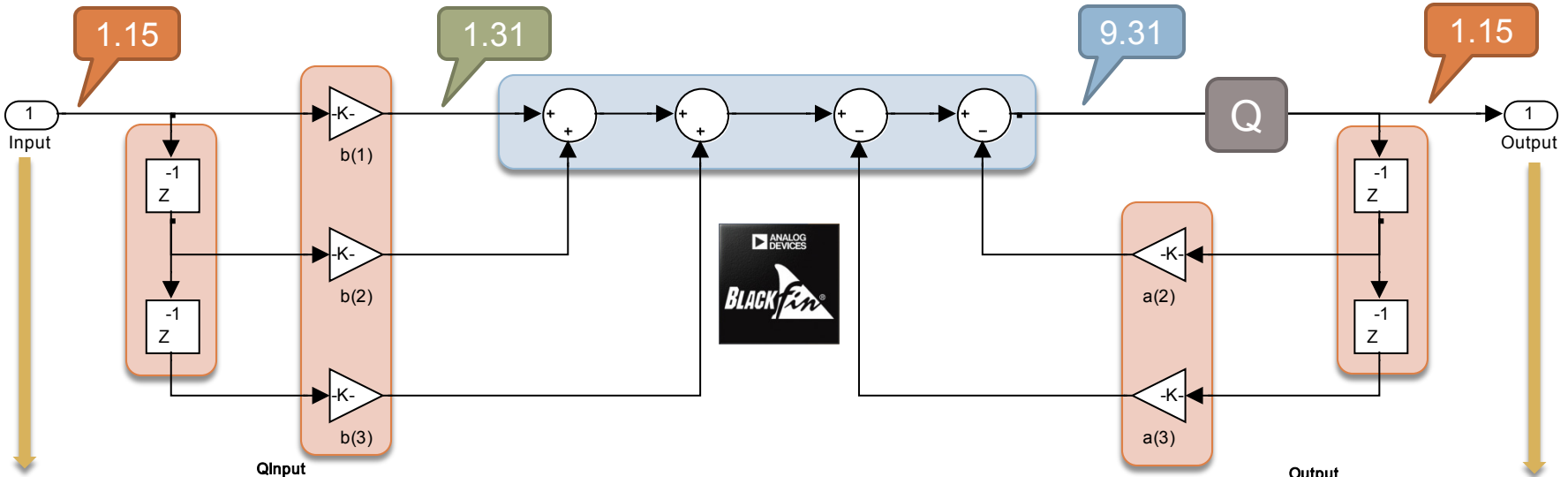
```
% Algorithm (Double)

for k=1:N
    A0 = 0;
    R0 = b(1)*x(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y(k)=A0;

    xz(2) = xz(1);
    xz(1) = x(k);

    yz(2) = yz(1);
    yz(1) = y(k);
end
```

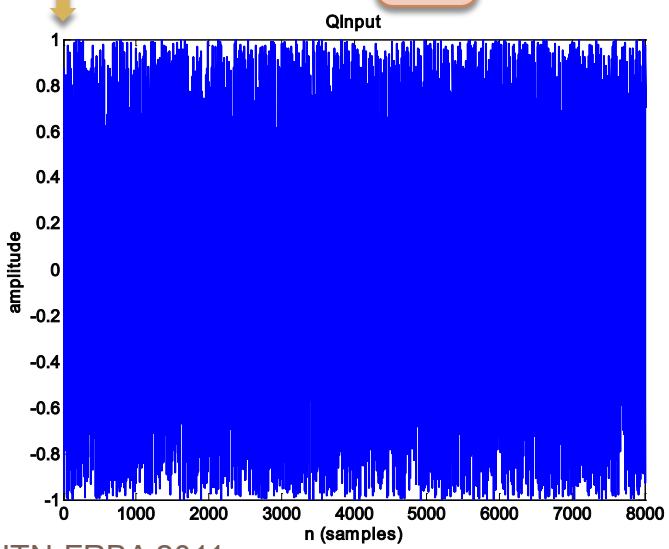
```
% Define Numeric Types
nt_1_15 = numerictype(1, 16, 15);
nt_1_31 = numerictype(1, 32, 31);
nt_9_31 = numerictype(1, 40, 31);

% Define Fimath
dsp_bf537 = fimath('RoundMode', 'Fix', ...
    'OverflowMode', 'Saturate', ...
    'ProductMode', 'SpecifyPrecision', ...
    'ProductWordLength', 32, ...
    'ProductFractionLength', 31, ...
    'SumMode', 'SpecifyPrecision', ...
    'SumWordLength', 40, ...
    'SumFractionLength', 31, ...
    'CastBeforeSum', false);

%  fi(Data, NumericType, Fimath)
b = fi(b, nt_1_15, dsp_bf537);
a = fi(a, nt_1_15, dsp_bf537);

x_fix = fi(x, nt_1_15, dsp_bf537);
y_fix = fi(zeros(N,1), nt_1_15, dsp_bf537);
xz = fi([0;0], nt_1_15, dsp_bf537);
yz = fi([0;0], nt_1_15, dsp_bf537);

R0 = fi(0, nt_1_31, dsp_bf537);
A0 = fi(0, nt_9_31, dsp_bf537);
```

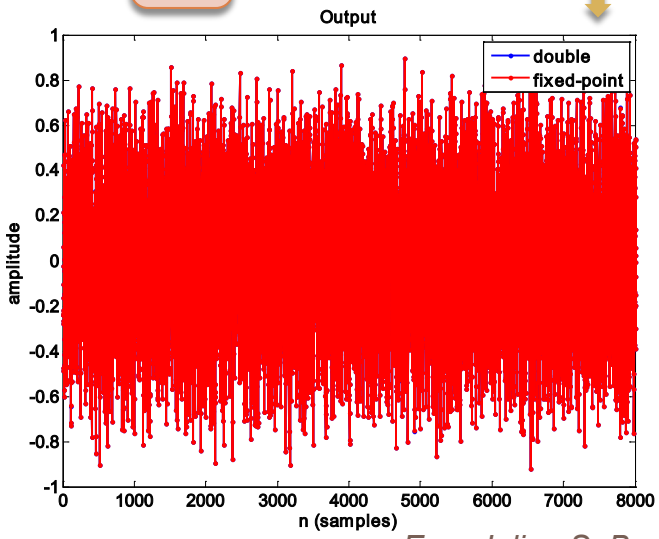
```
% Algorithm (Fixed-Point)

for k=1:N
    A0.data = 0;
    R0 = b(1)*x_fix(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y_fix(k)=fi(A0,nt_1_15); % redondeo y trunco

    xz(2) = xz(1);
    xz(1) = x_fix(k);

    yz(2) = yz(1);
    yz(1) = y_fix(k);
end
```

Workflow

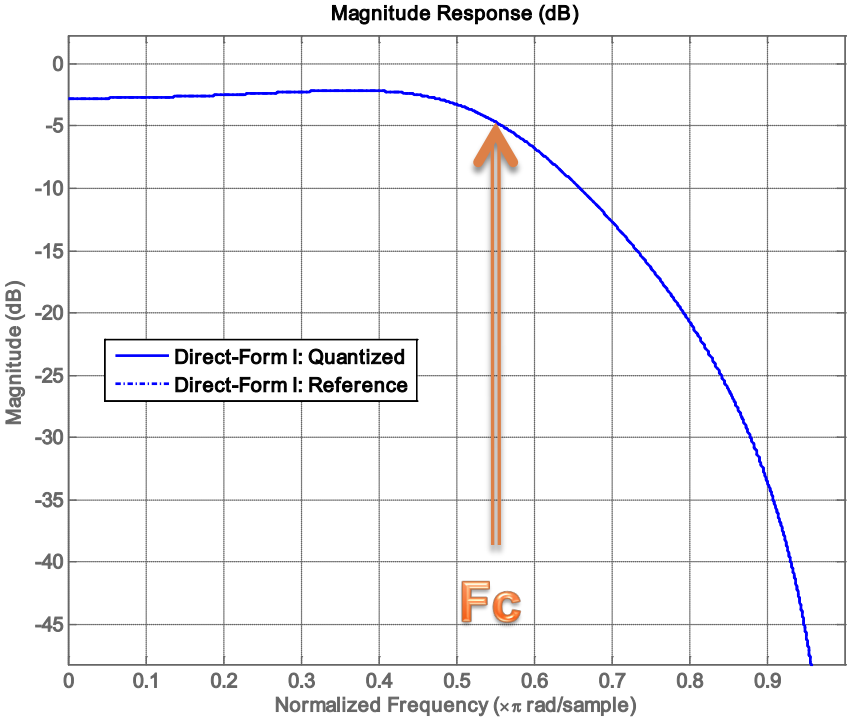
SOS DFI – Study case
BF53X Fixed Point Simulation

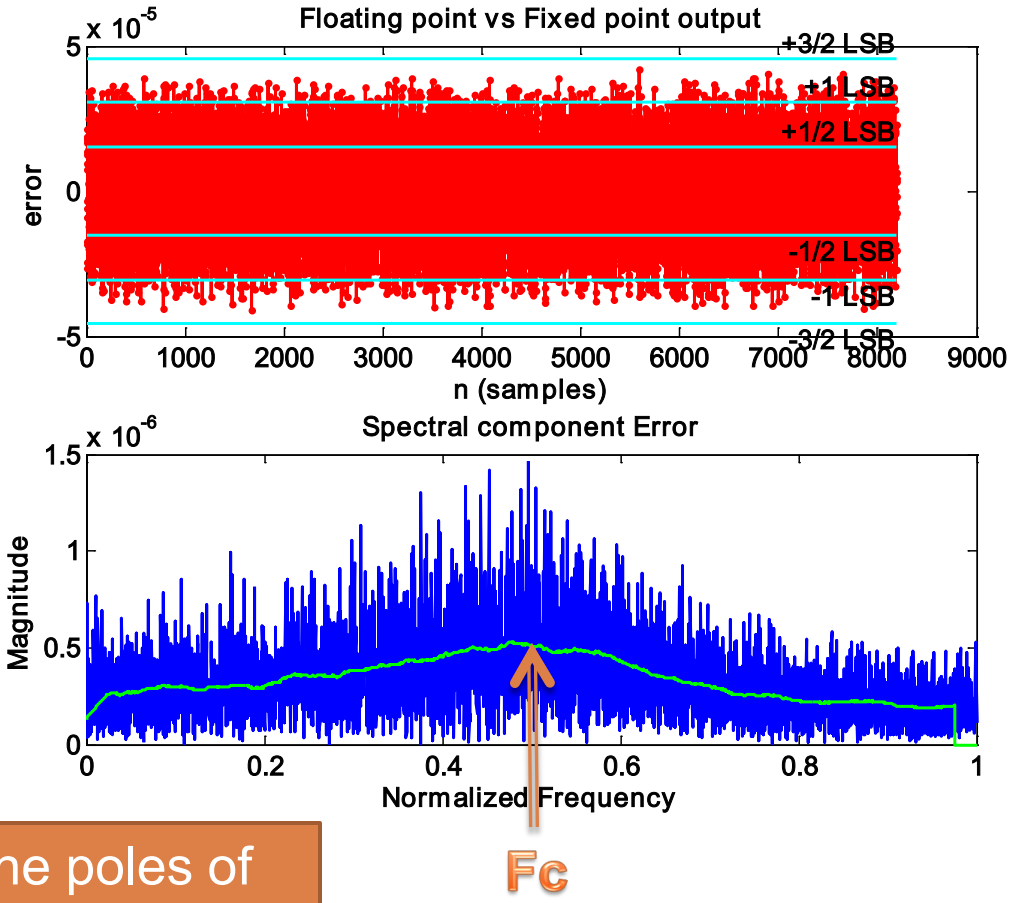






SOS DFI – Study case
BF53X Fixed Point Simulation





The error spectrum is amplified by the poles of the filter, $1 + a_1 z^{-1} + a_2 z^{-2}$, regardless of the characteristics of the filter.

SOS DFI – Case Study ASM implementation

Roundoff noise reduction schemes

- ***Product roundoff errors*** due to rounding or truncation of products may lead a ***noticeable distortion*** at the filter output with ***low level input*** signal and for ***high fidelity systems*** should be minimized
- A number of schemes have been devised for reducing or eliminating the effects of roundoff error in IIR filters.
- These schemes ***shape de noise spectrum*** in such way as to reduce or cancel their effect over certain bands of the filter.
- The schemes have been collectively called ***error spectral shaping (ESS)***.

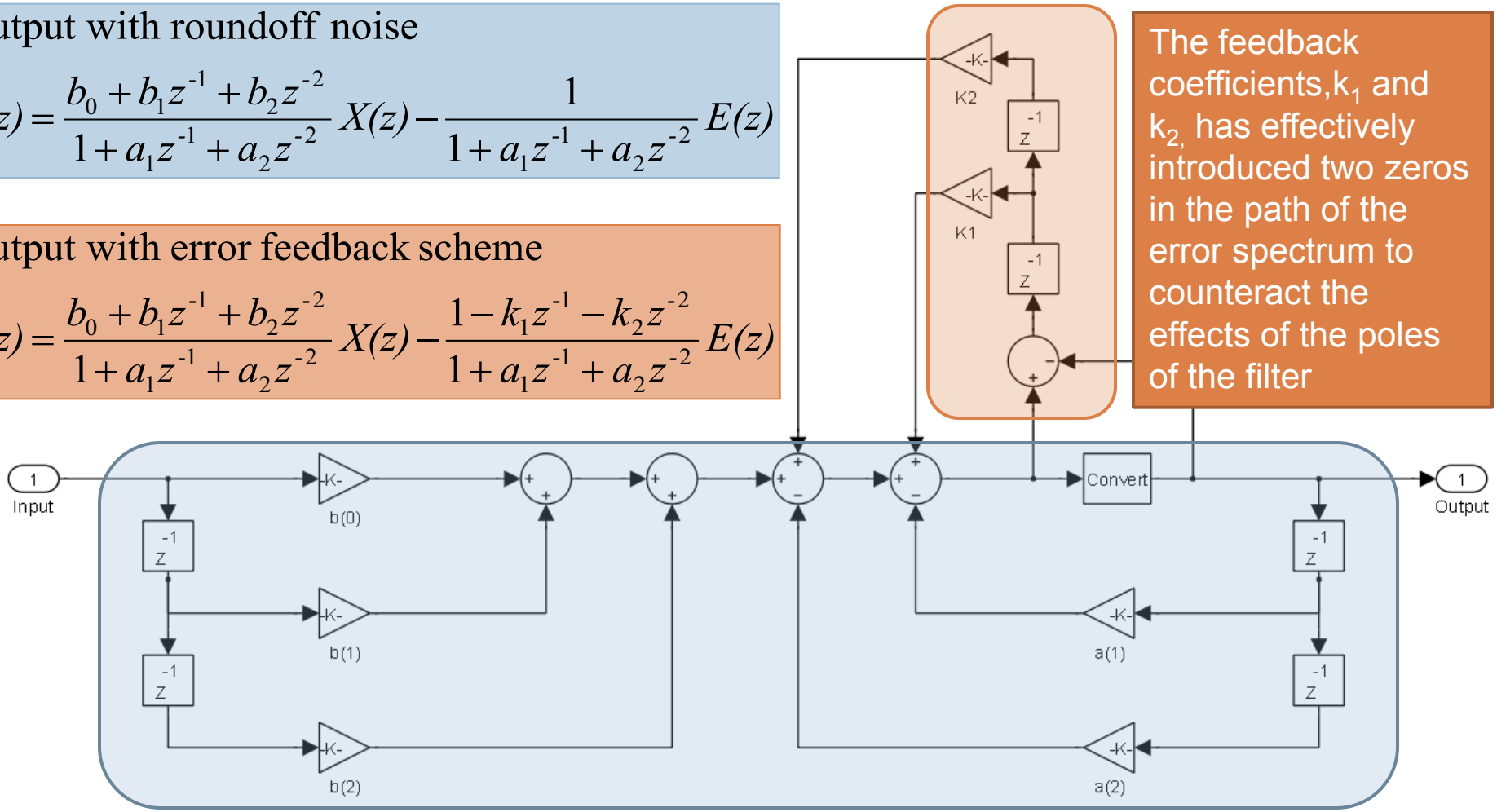
SOS DFI with Error Spectral Shaping (ESS)

Output with roundoff noise

$$Y(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} X(z) - \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} E(z)$$

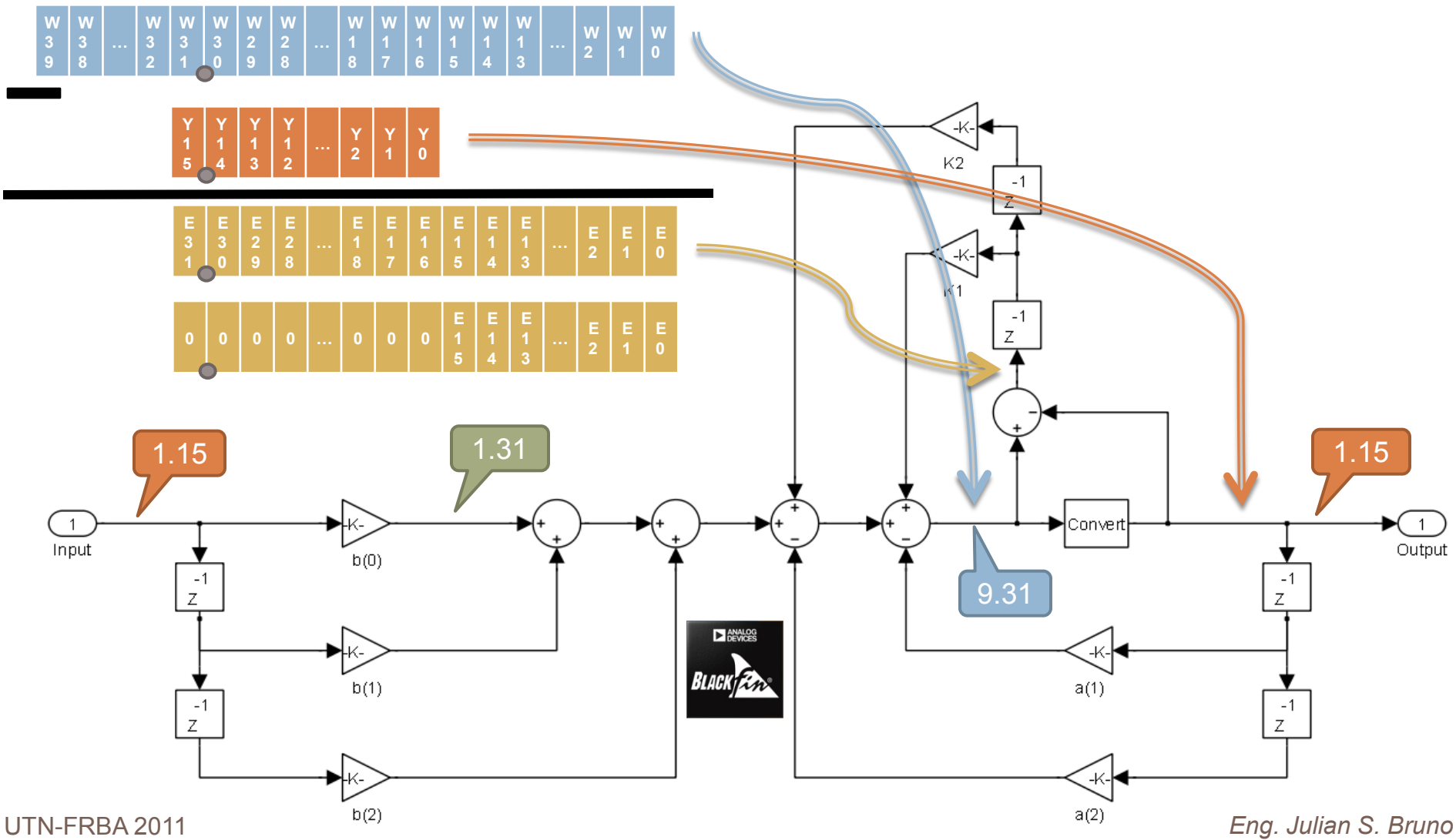
Output with error feedback scheme

$$Y(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} X(z) - \frac{1 - k_1 z^{-1} - k_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} E(z)$$

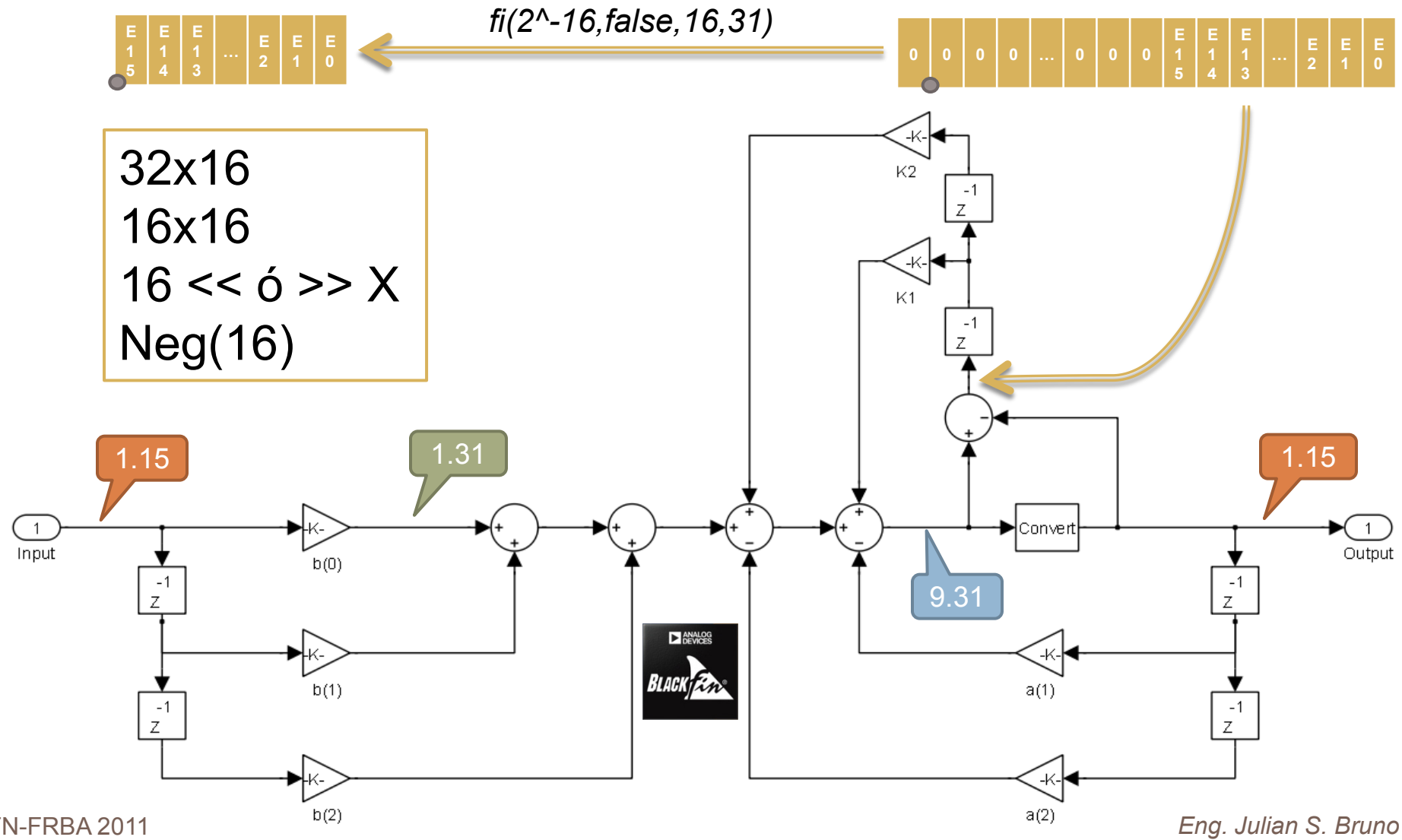


The feedback coefficients, k_1 and k_2, has effectively introduced two zeros in the path of the error spectrum to counteract the effects of the poles of the filter

SOS DFI with ESS – Case Study BF53X Architecture

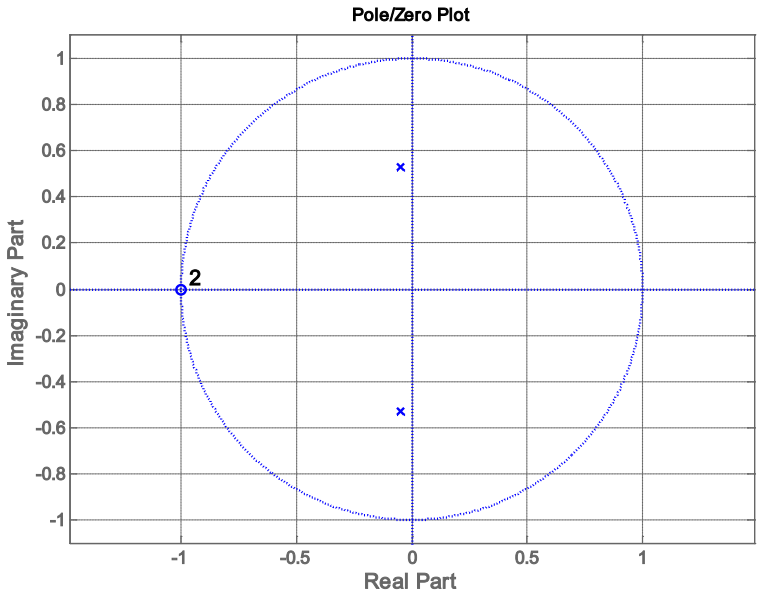


SOS DFI with ESS – Case Study BF53X Architecture



SOS DFI with ESS – Study case

K1	K2	Locations of zeros
0	1	A pair of zeros at 0° and 180° (i.e. at dc and Fs/2)
0	-1	A pair of complex conjugate zeros at ±90° (i.e. at ±Fs/4)
1	0	A single zero at 0° (i.e. at dc)
1	-1	A pair of complex conjugate zeros at ±60° (i.e. at ±Fs/6)
2	-1	A double zero at 0° (i.e. at dc)
-2	-1	A single zero at 180° (i.e. at Fs/2)
-1	-1	A pair of complex conjugate zeros at ±120° (i.e. at ±Fs/3)
-1	0	A single zero at 180° (i.e. at Fs/2)

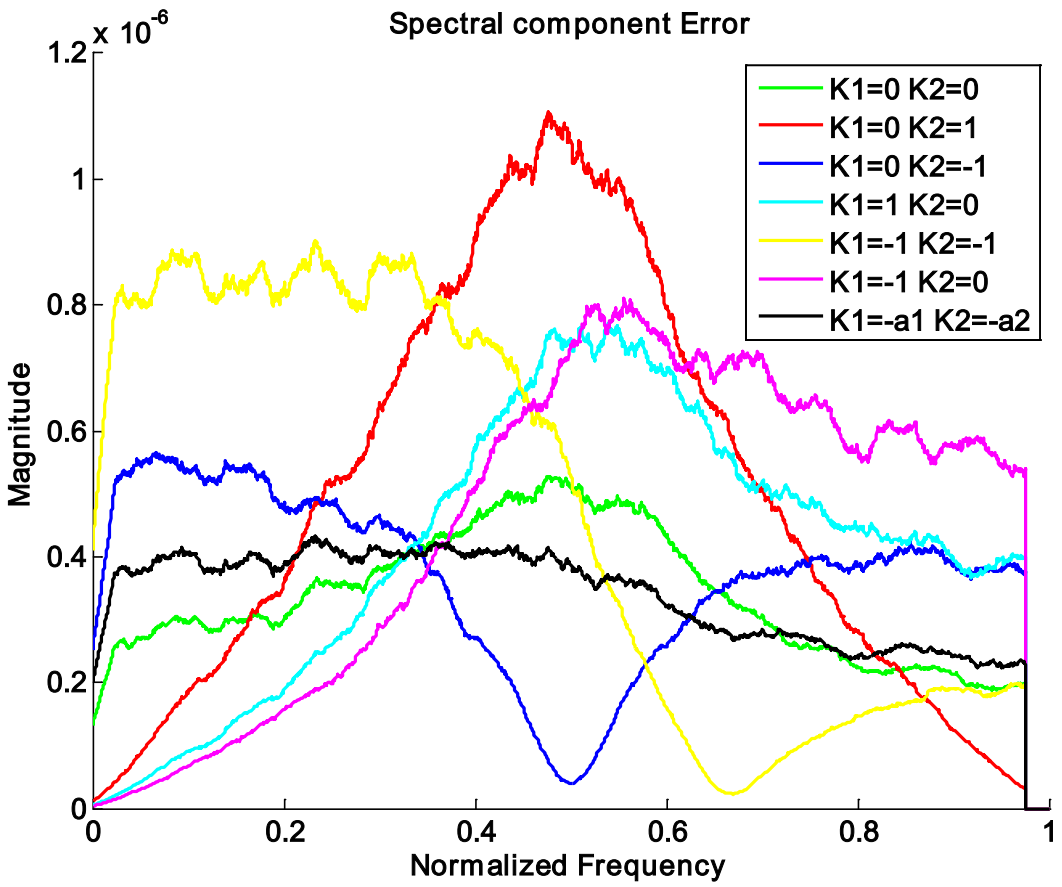


Output with error feedback scheme

$$Y(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} X(z) - \frac{1 - k_1 z^{-1} - k_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} E(z)$$

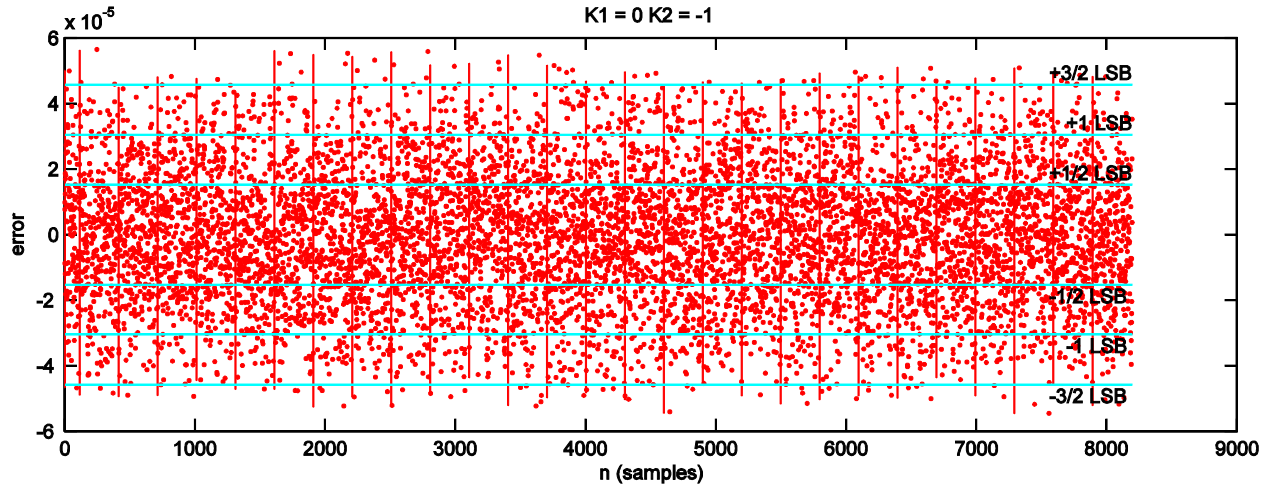
SOS DFI with ESS – Study case Fixed Point Simulation

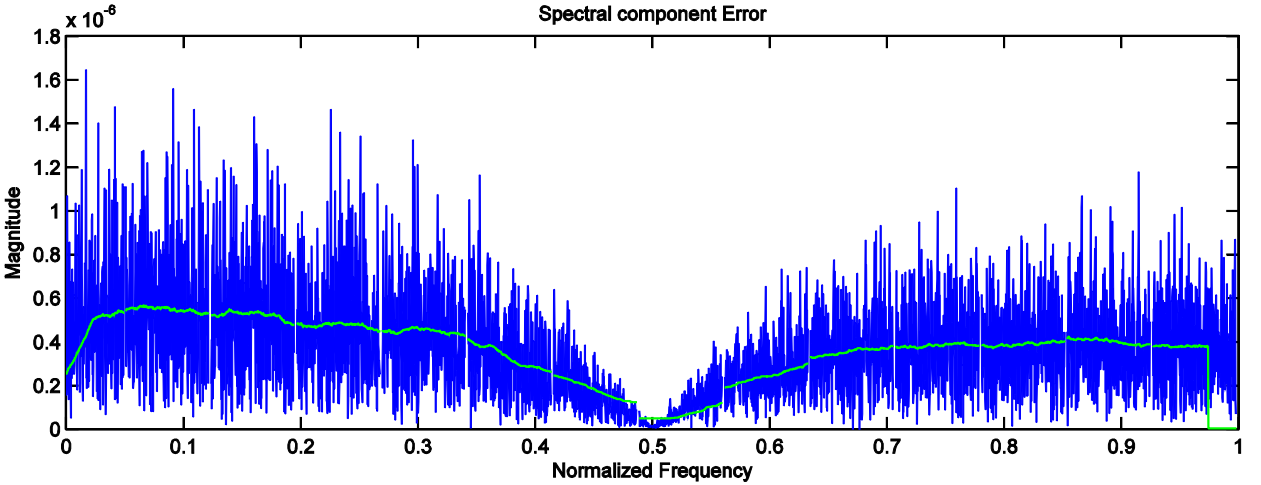
```
%% Algorithm (Fixed-Point)
for k=1:N
    A0.data =0;
    R0 = b(1)*x_fix(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    R1 =  fi(K1*ez.data(1),nt_1_31);
    R2 =  fi(K2*ez.data(2),nt_1_31);

    A0 = A0 + R1;
    A0 = A0 + R2;

    y_fix(k)=fi(A0,nt_1_15); % redondeo y trunco

    xz(2) = xz(1);
    xz(1) = x_fix(k);

    yz(2) = yz(1);
    yz(1) = y_fix(k);

    ez(2) = ez(1);
    ez(1) = fi(A0-y_fix(k), nt_1_31);
end
```



SOS DFI with ESS – Study case Fixed Point Simulation

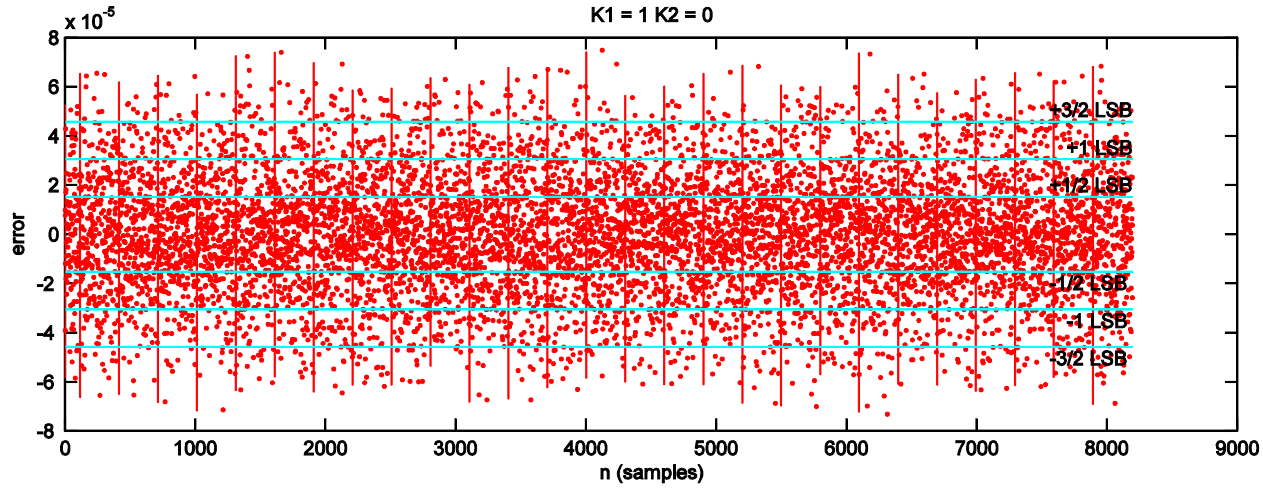
A pair of complex conjugate zeros at ±90° (i.e. at ±Fs/4)

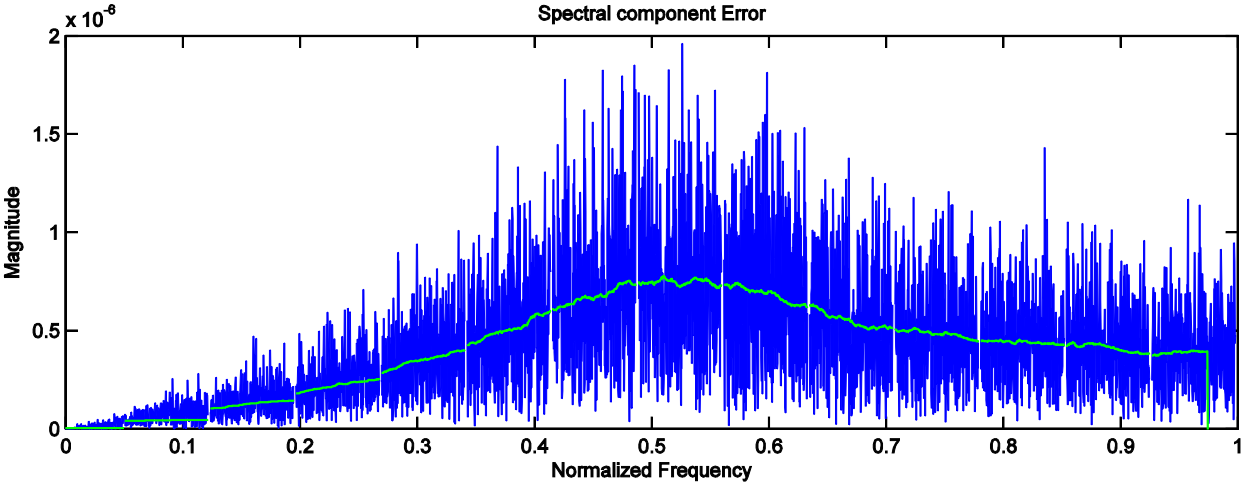




SOS DFI with ESS – Study case Fixed Point Simulation

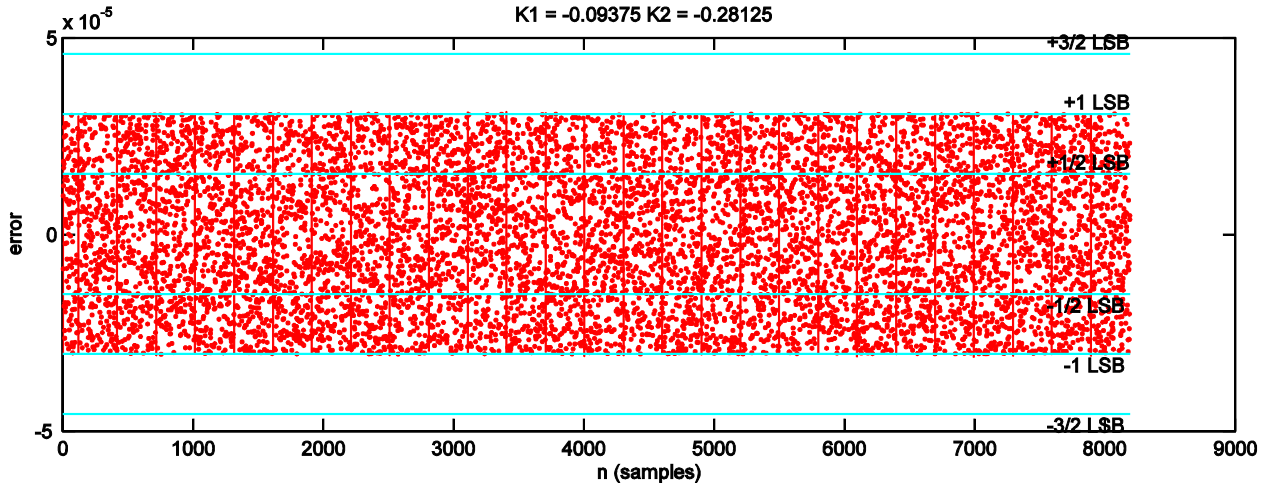
A single zero at 0° (i.e. at dc)

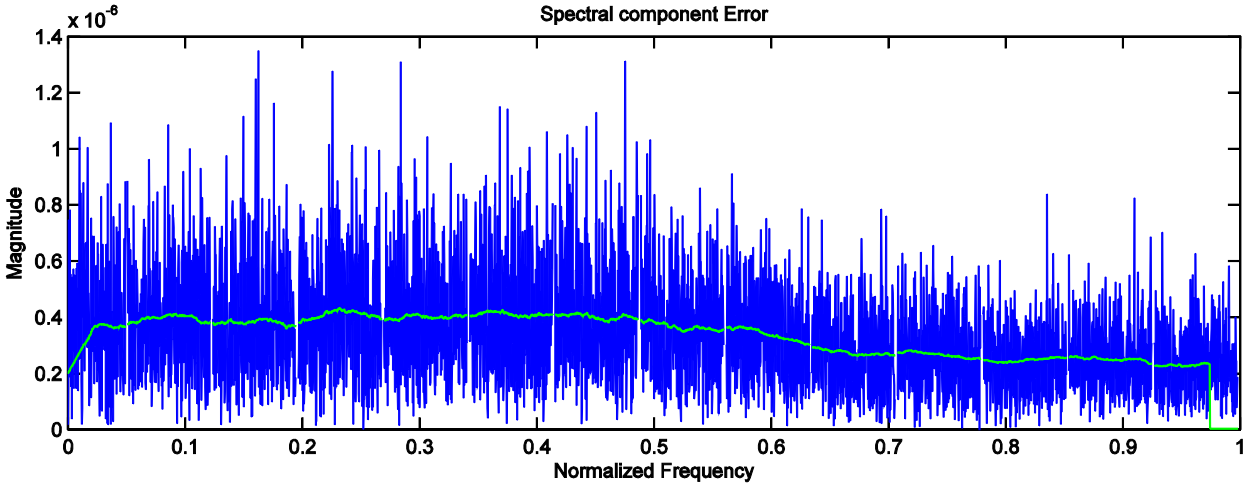




SOS DFI with ESS – Study case Fixed Point Simulation

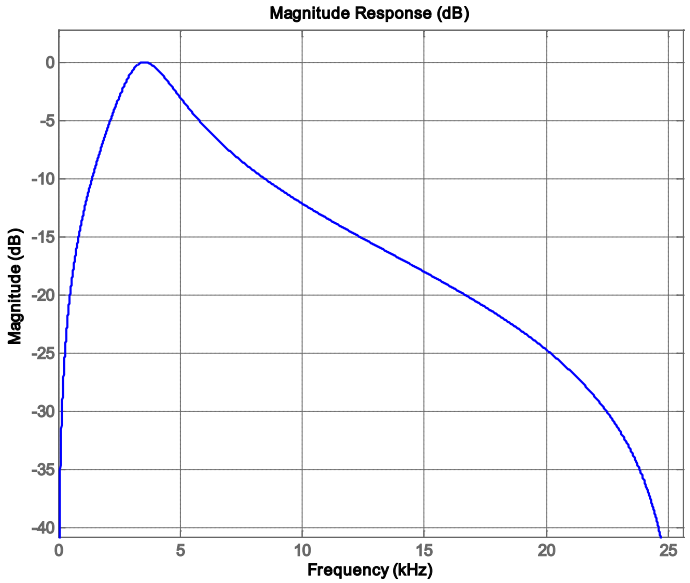
A pair of complex conjugate zeros at ±Fc

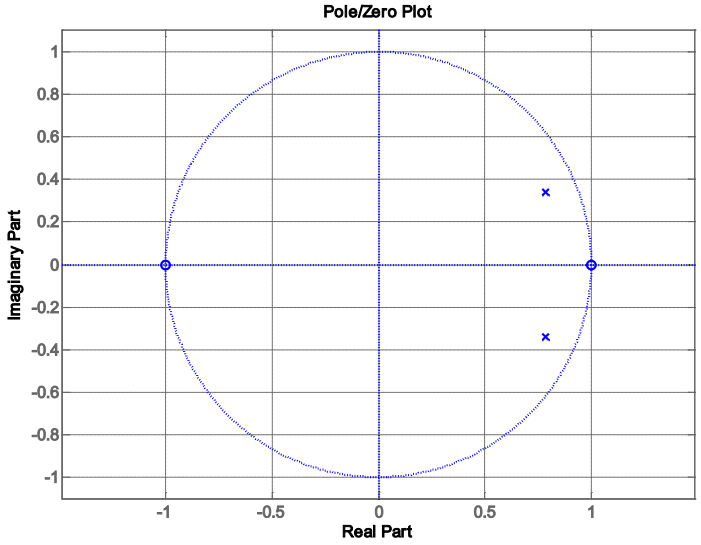




SOS DFI with ESS ASM implementation

SOS DFI with ESS – Study case





Band Pass Filter
2nd order Butterworth
2500Hz – 5000Hz
Fs: 51200Hz

$$H(z)=\frac{Y(z)}{X(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}$$

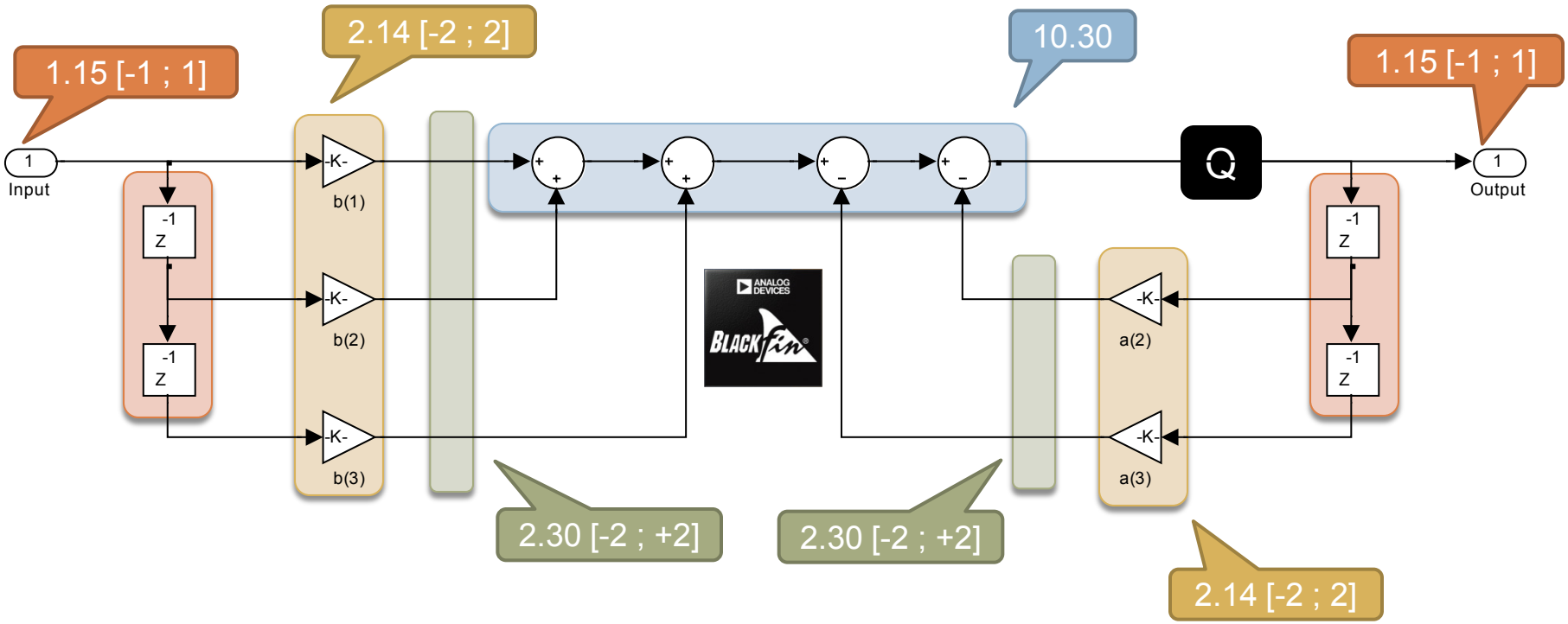
2.14

$$\hat{H}(z)=\frac{\hat{Y}(z)}{\hat{X}(z)}=\frac{\hat{b}_0+\hat{b}_1z^{-1}+\hat{b}_2z^{-2}}{1+\hat{a}_1z^{-1}+\hat{a}_2z^{-2}}$$

$$b_0=0.1339111328125$$
$$b_1=0.00000000000000 \quad a_1=-1.5704345703125$$
$$b_2=-0.1339111328125 \quad a_2=0.7321777343750$$

$$\hat{b}_0=0.1339111328125$$
$$\hat{b}_1=0.00000000000000 \quad \hat{a}_1=-1.5704345703125$$
$$\hat{b}_2=-0.1339111328125 \quad \hat{a}_2=0.7321777343750$$

SOS DFI – Study case
BF53X Architecture



SOS DFI – Study case
BF53X Fixed Point Simulation

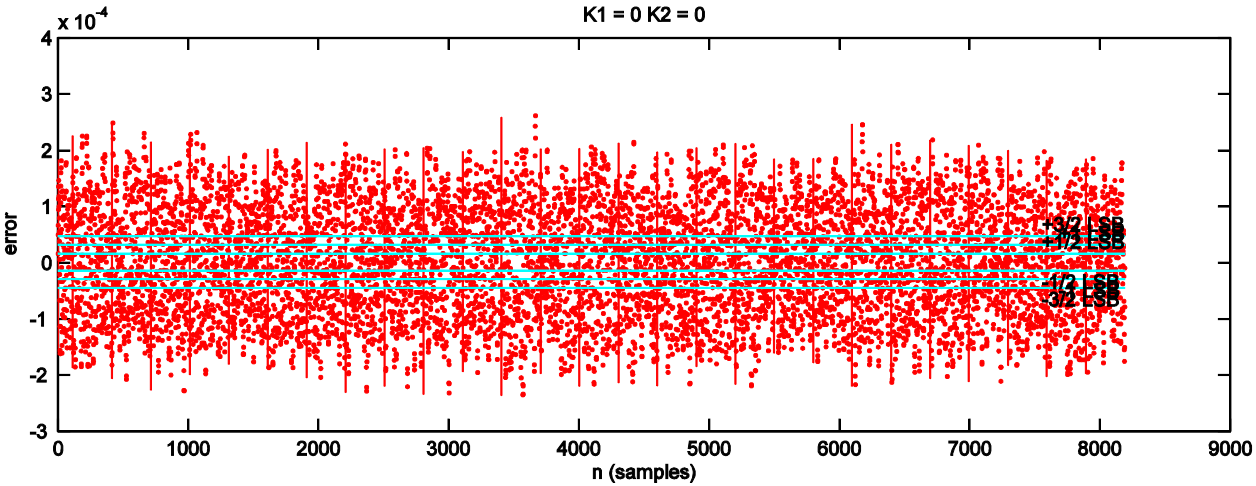
```
% Algorithm (Double)

for k=1:N
    A0 = 0;
    R0 = b(1)*x(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y(k)=A0;

    xz(2) = xz(1);
    xz(1) = x(k);

    yz(2) = yz(1);
    yz(1) = y(k);
end
```

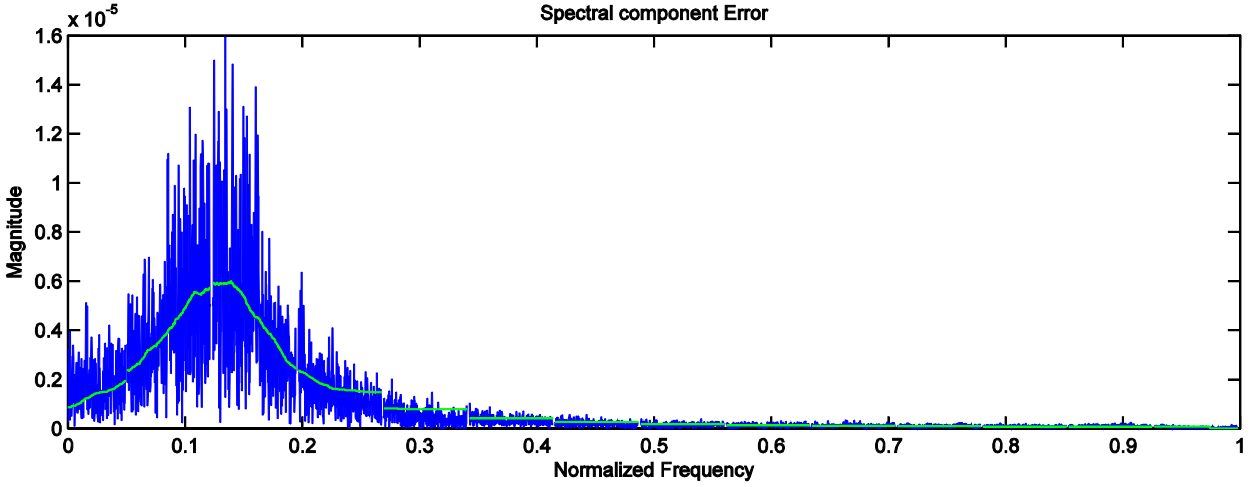
```
% Define Numeric Types
nt_1_15 = numerictype(1, 16, 15);
nt_2_30 = numerictype(1, 32, 30);
nt_10_30 = numerictype(1, 40, 30);
nt_2_14 = numerictype(1, 16, 14);

% Define Fimath
dsp_bf537 = fimath('RoundMode', 'Fix', ...
    'OverflowMode', 'Saturate', ...
    'ProductMode', 'SpecifyPrecision', ...
    'ProductWordLength', 32, ...
    'ProductFractionLength', 30, ...
    'SumMode', 'SpecifyPrecision', ...
    'SumWordLength', 40, ...
    'SumFractionLength', 30, ...
    'CastBeforeSum', false);

%  fi(Data, NumericType, Fimath)
b = fi(b, nt_2_14, dsp_bf537);
a = fi(a, nt_2_14, dsp_bf537);

x_fix = fi(x, nt_1_15, dsp_bf537);
y_fix = fi(zeros(N,1), nt_1_15, dsp_bf537);
xz = fi([0;0], nt_1_15, dsp_bf537);
yz = fi([0;0], nt_1_15, dsp_bf537);

R0 = fi(0, nt_2_30, dsp_bf537);
A0 = fi(0, nt_10_30, dsp_bf537);
```

```
% Algorithm (Fixed-Point)

for k=1:N
    A0.data = 0;
    R0 = b(1)*x_fix(k);
    A0 = A0 + R0;
    R0 = b(2) * xz(1);
    A0 = A0 + R0;
    R0 = b(3) * xz(2);
    A0 = A0 + R0;

    R0 = a(2)*yz(1);
    A0 = A0 - R0;
    R0 = a(3)*yz(2);
    A0 = A0 - R0;

    y_fix(k)=fi(A0,nt_1_15); % redondeo y trunco

    xz(2) = xz(1);
    xz(1) = x_fix(k);

    yz(2) = yz(1);
    yz(1) = y_fix(k);
end
```

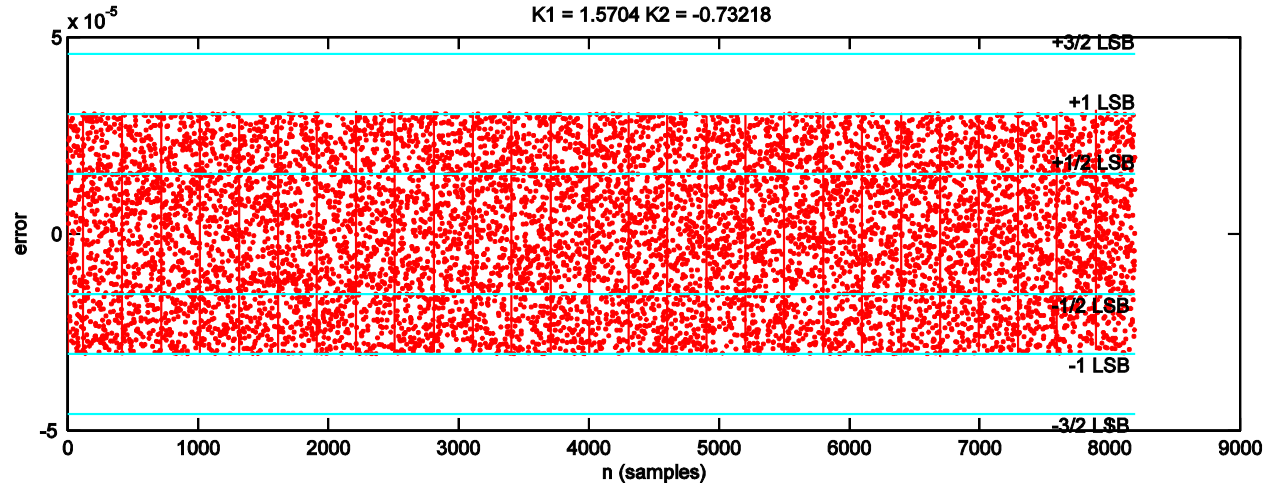
SOS DFI – Case Study ASM implementation

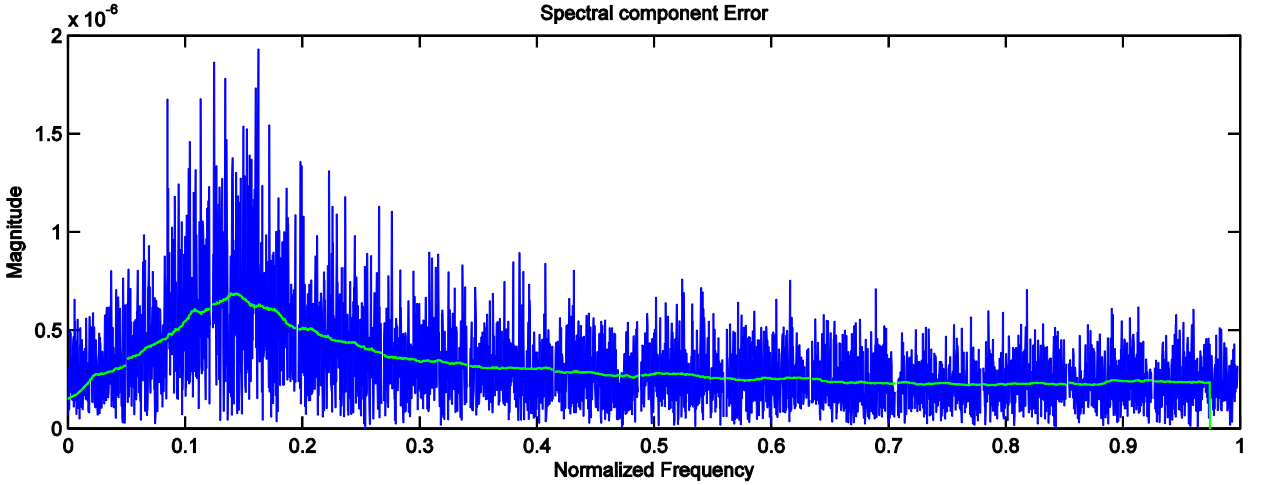
SOS DFI without ESS – Study case Fixed Point Simulation





SOS DFI with ESS – Study case Fixed Point Simulation

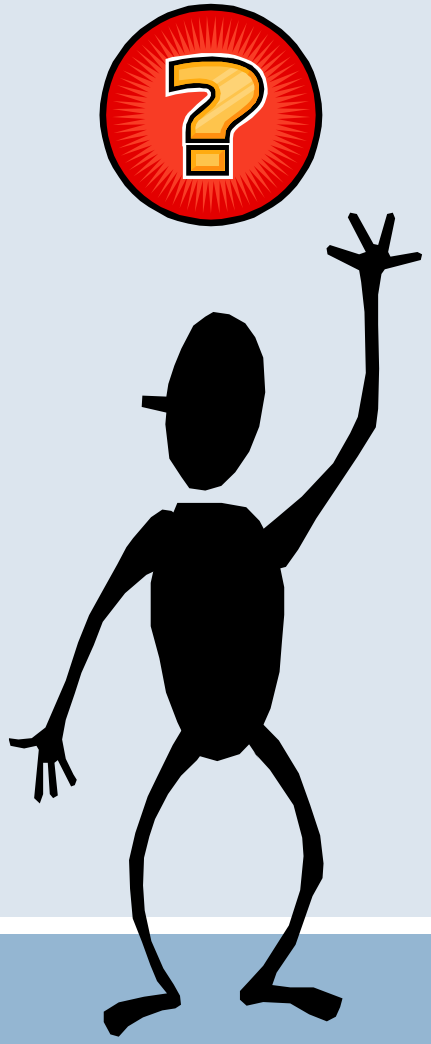




Recommended bibliography

- B Widrow and I Kollár .Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications. Cambridge University Press, Cambridge, UK, 2008.
  - Ch 16: Roundoff Noise in IIR Digital Filters
- EC Ifeachor, BW Jervis. Digital Signal Processing. A Practical approach. Second Edition. Prentice Hall. 2002
  - Ch13: Analysis of wordlength effects in fixed point DSP systems
- Jon Dattorro. The Implementation of Recursive Digital Filters for High- Fidelity Audio, J. Audio Eng. Soc., Vol. 36, No. 11, November 1988.
- R Yates and R Lyons. DC Blocker Algorithms[DSP Tips & Tricks]. IEEE Signal Processing Magazine, vol. 25, issue 2, pp. 132-134

- NOTE: Many images used in this presentation were extracted from the recommended bibliography.



Questions?

Thank you!