A/D conversion as an error source. Fixed-point IIR filter design.

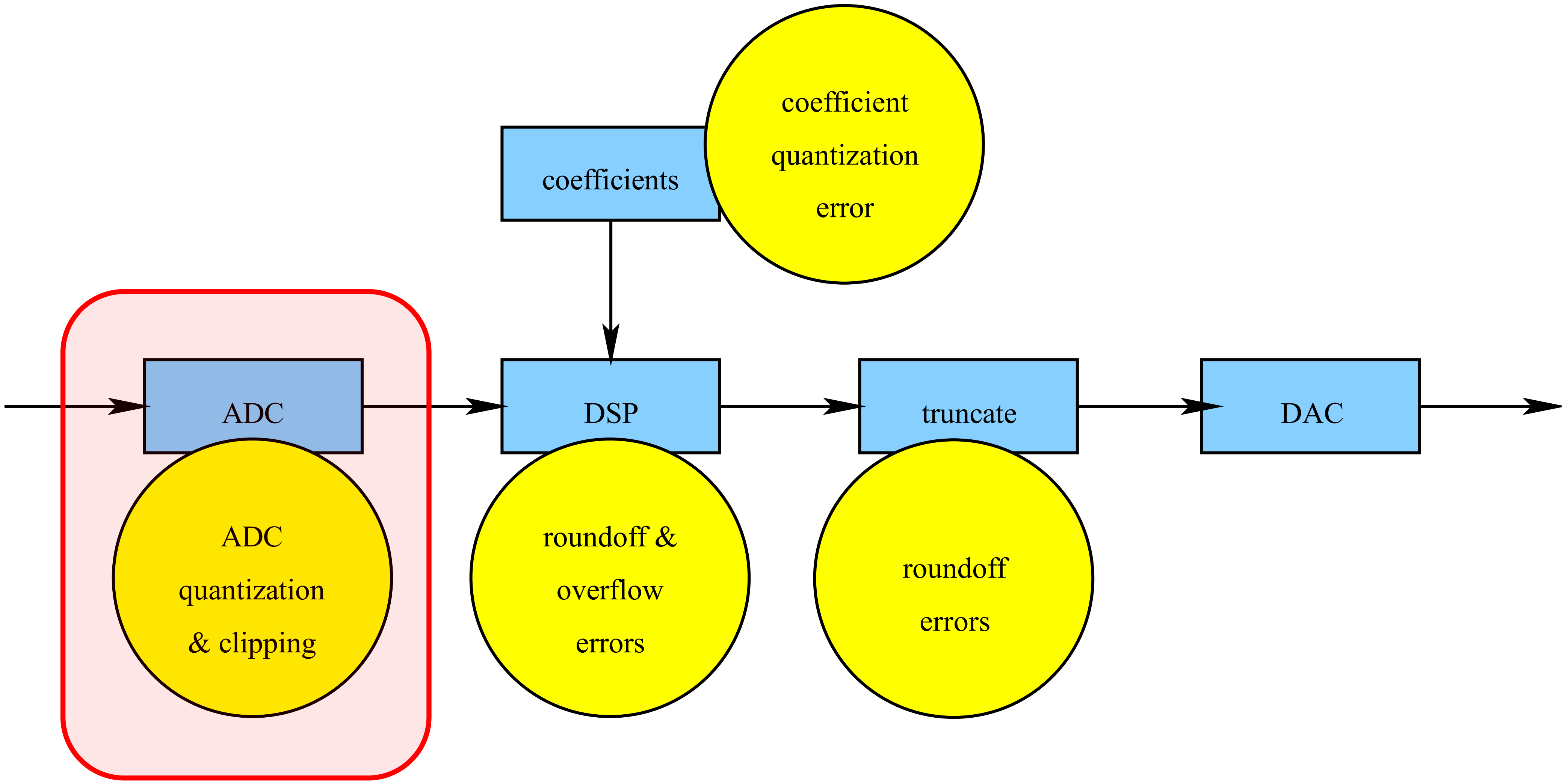
Signal Processing Systems Fall 2025

Lecture 5 (Monday 10.11.)

Outline

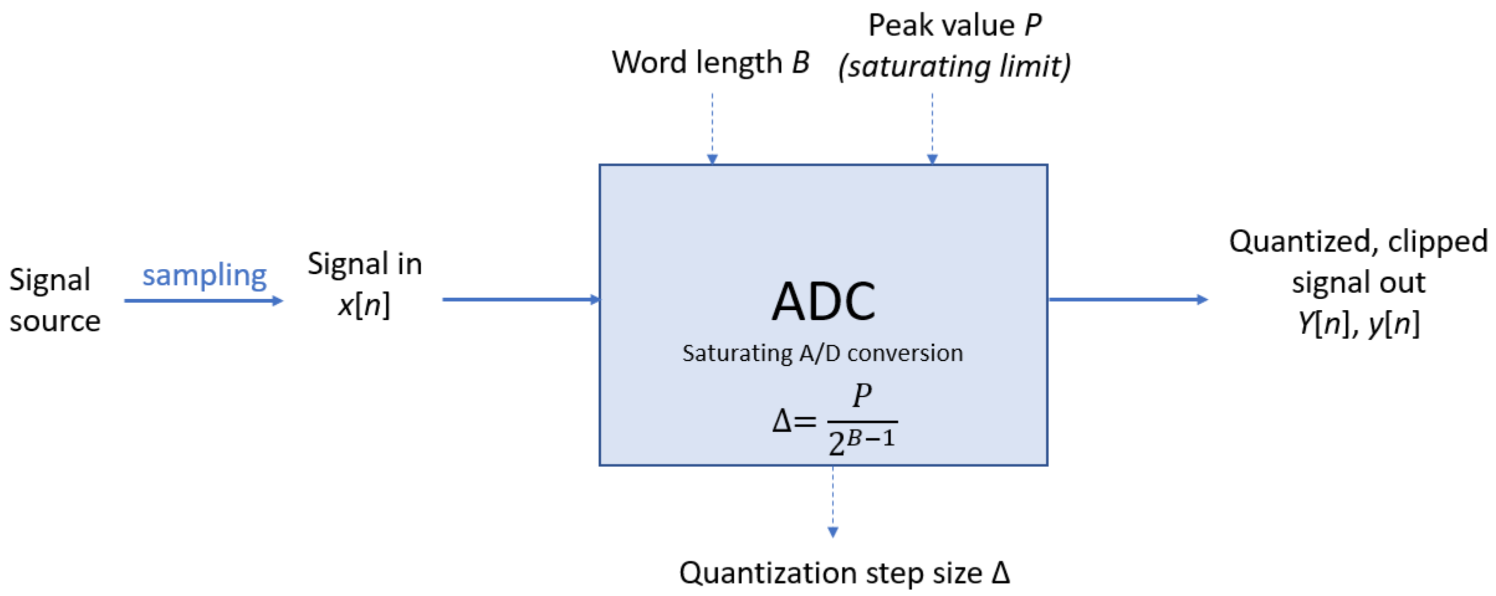
- A/D conversion as a noise source
  - Modelling ADC
  - Computing powers (using models / samples)
  - Crest factor, input noise floor, noise spectrum
- Fixed-point IIR filter design
  - Coefficient quantization & filter structure
  - Scoping of numerical ranges using Matlab’s Fixed-Point Toolset
  - Second-order section (SOS) design
    - Input scaling
  - Experiments (with relation to ADC noise floor)

1. A/D conversion as a noise source



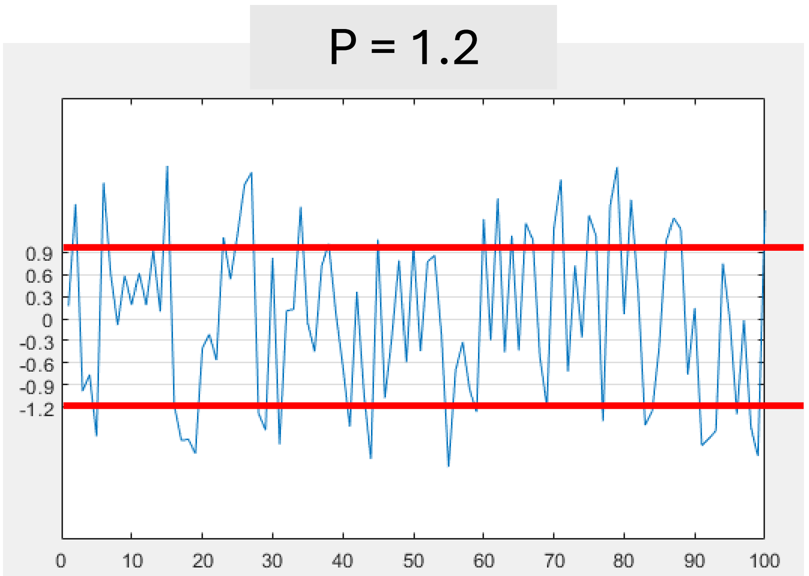
1.1. Model of A/D conversion

Saturating ADC:

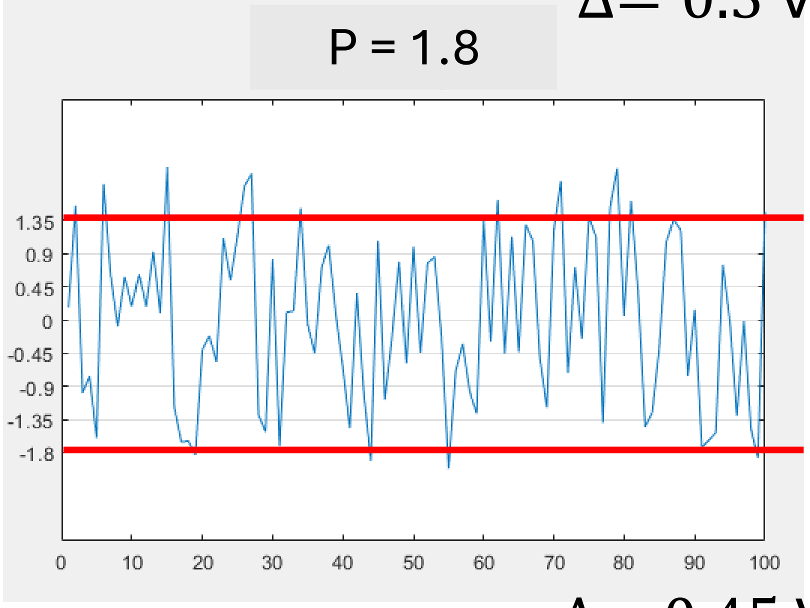


Larger P => less clipping, but larger Δ

B=3 bits



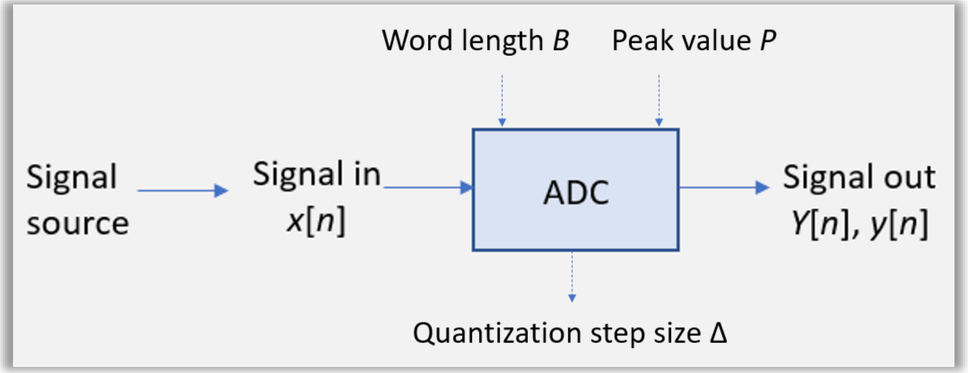
$\Delta = 0.3$ V



$\Delta = 0.45$ V

Model components

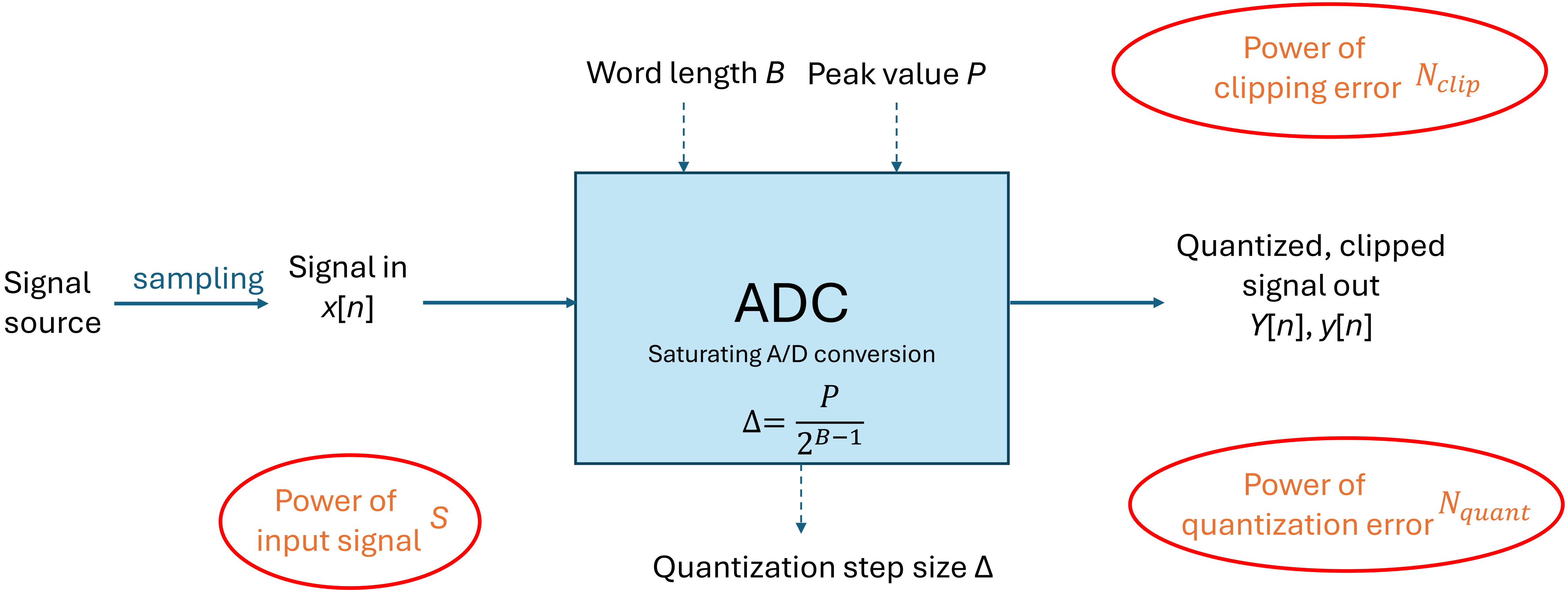
- Signal source
  - has certain characteristics, some quantities like power of the signal may be known (e.g. consider sine wave or Gaussian input)
- Input signal
  - sequence of samples, **x[n]**
- Output signal (quantized, clipped)
  - represented as two's complements integers **Y[n]** (***B*** bits)
  - values can be interpreted in terms of input signal levels, **y[n] = Δ Y[n]**, where Δ is the quantization step size
- Saturating ADC
  - peak absolute value ***P*** of input signal that clipping does not occur
  - quantization step size **Δ** can be calculated from ***P*** and ***B***



$$\Delta = \frac{P}{2^{B-1}}$$

Powers of interest

***Goal**: understand how signal/noise ratios change when B and P are adjusted*



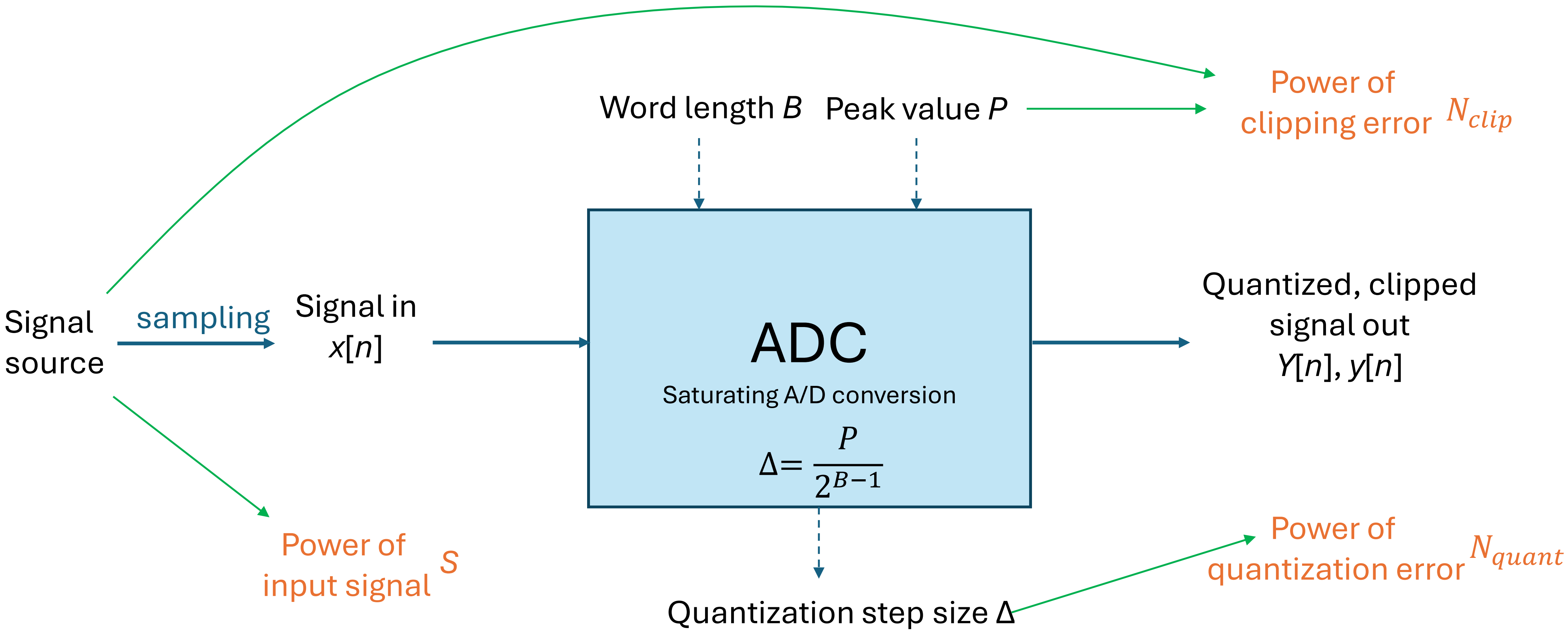
Computing powers

Two ways of evaluating signal powers in each case:
(1) model-based *analysis*
(2) sample-based

- Input signal power, S
  - Evaluated either as (1) signal source characterization or (2) from given input samples x[n]
- Quantization error power, N_{quant}
  - Evaluated either from (1) statistical characterization i.e. uniform distribution over [-Δ/2, +Δ/2] or (2) from given unclipped output samples y[n] and corresponding input x[n]
- Clipping error power, N_{clip}
  - Computed using (1) signal source characterization and saturating peak P or (2) from given clipped output samples y[n] and corresponding input x[n]

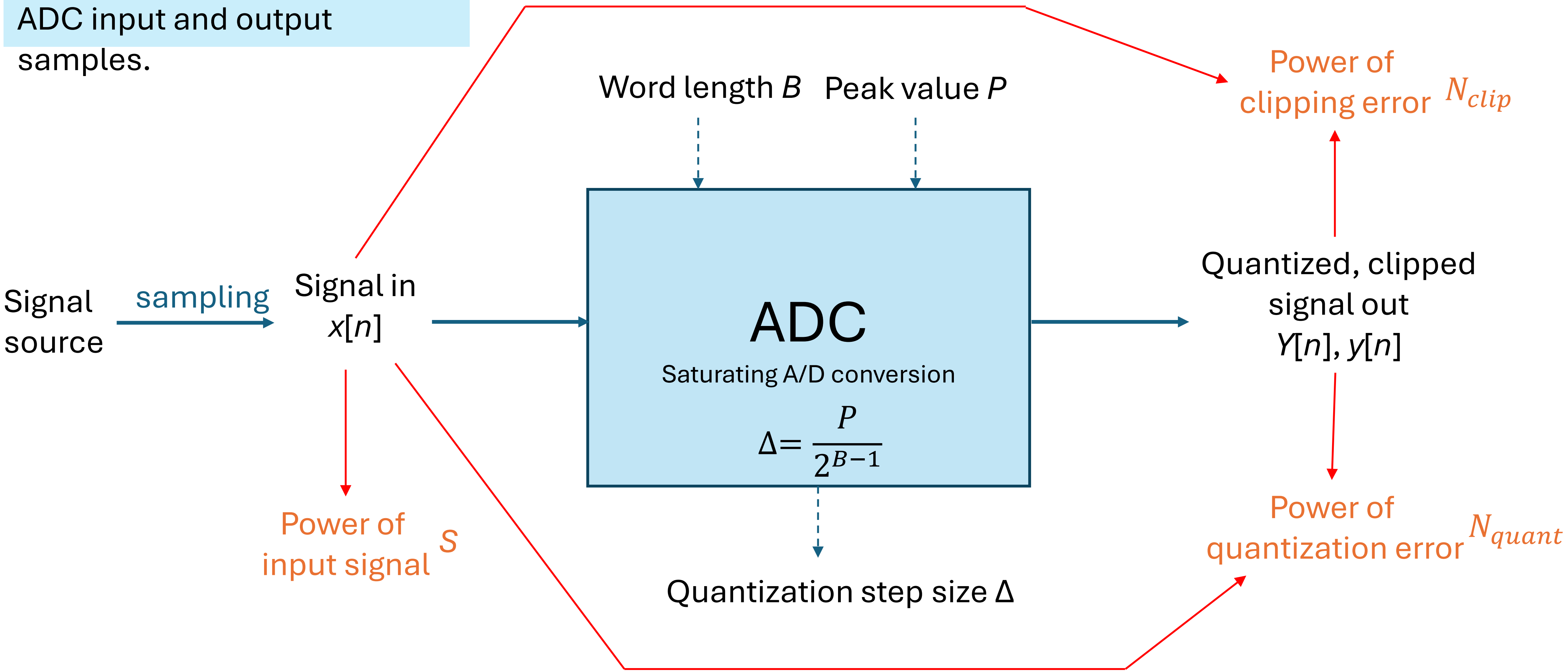
(1) Powers: model-based computation

Powers are computed from mathematical / statistical characterizations of the process.



(2) Powers: sample-based computation

Powers are computed from ADC input and output samples.



Example. Gaussian input

(1) **From model:**

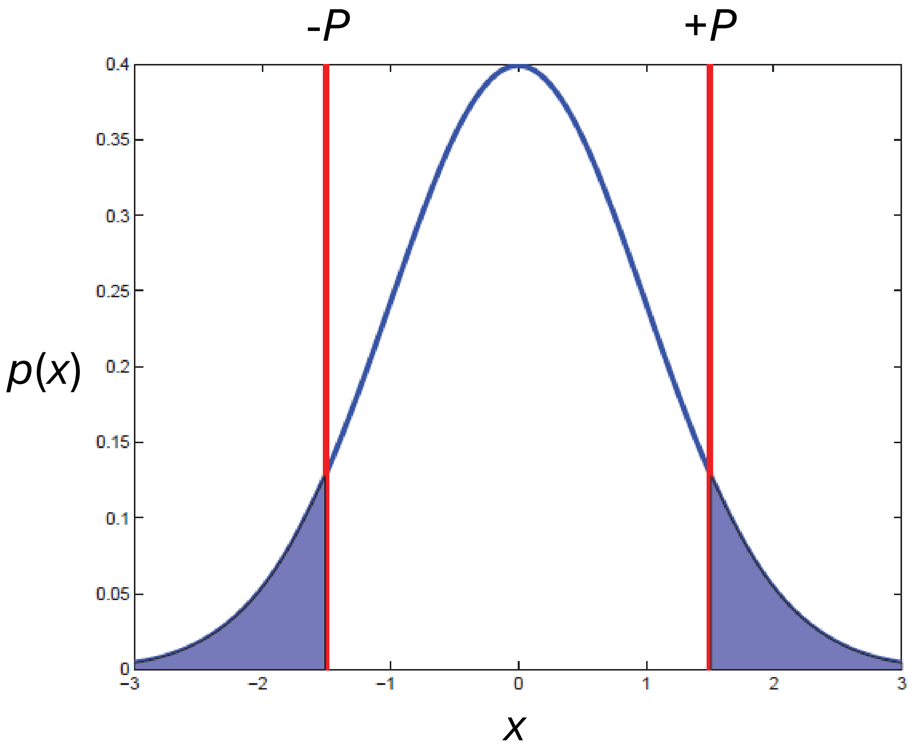
Signal source characterized by its probability density function (pdf)

Signal power $S = \sigma^2$ Variance

Clipping error power $N_{clip} = S - \int_{-P}^{+P} x^2 p(x) dx$

Quantization error power $N_{quant} = \frac{1}{\Delta}\int_{-\Delta/2}^{+\Delta/2} x^2 dx = \frac{\Delta^2}{12}$

$$\Delta = \frac{P}{2^{B-1}}$$



(2) **From samples:** $S = \frac{1}{N}\sum x[n]^2$ $N_{clip} = \frac{1}{N}\sum_{n \in clip} (y[n] - x[n])^2$ $N_{quant} = \frac{1}{N}\sum_{n \in noclip} (y[n] - x[n])^2$

1.2. Crest factor (CF)

- Property of the input signal, defined as

$$CF_{\mathrm{SIG}} = 10 \log_{10} \frac{Q^2}{S}$$

Q : peak value of the signal
(or value exceeded with some low probability)

S : input signal power = RMS^2 (RMS = root mean square)

- Quantifies how large deviations from RMS value there can be at the ADC input
- ADC configuration must tolerate those deviations
  - Adjust saturating limit P properly
  - Then, no clips or just few

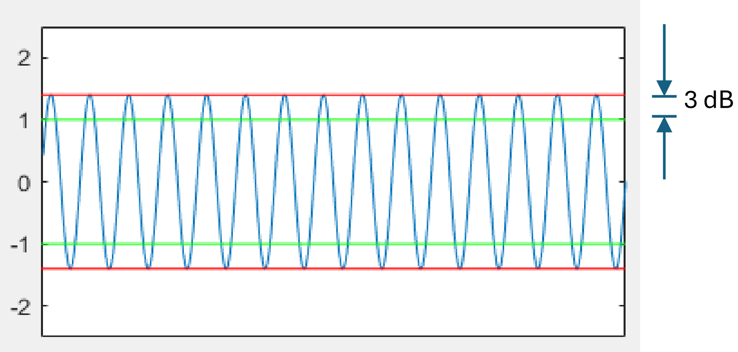
Crest factor examples

$$CF_{\mathrm{SIG}} = 10\ \log_{10}\frac{Q^2}{S}\ \text{[dB]}$$

Q : peak value of the signal
(or value exceeded with some low probability)

S : input signal power = RMS^2 (RMS = root mean square)

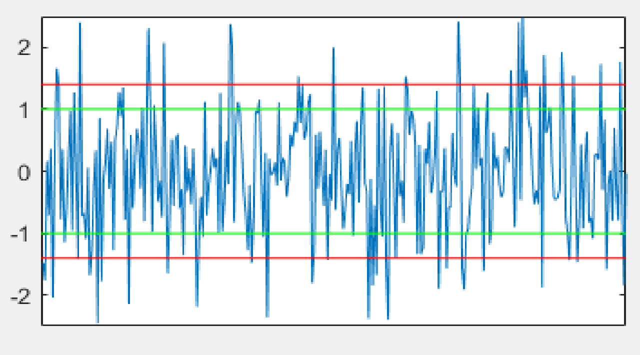
Sinusoid



$$\mathrm{RMS} = \frac{Q}{\sqrt{2}}$$

$$CF_{\mathrm{SIG}} = 10\log\frac{Q^2}{Q^2/2} = 10\log 2 = 3.01\ \mathrm{dB}$$

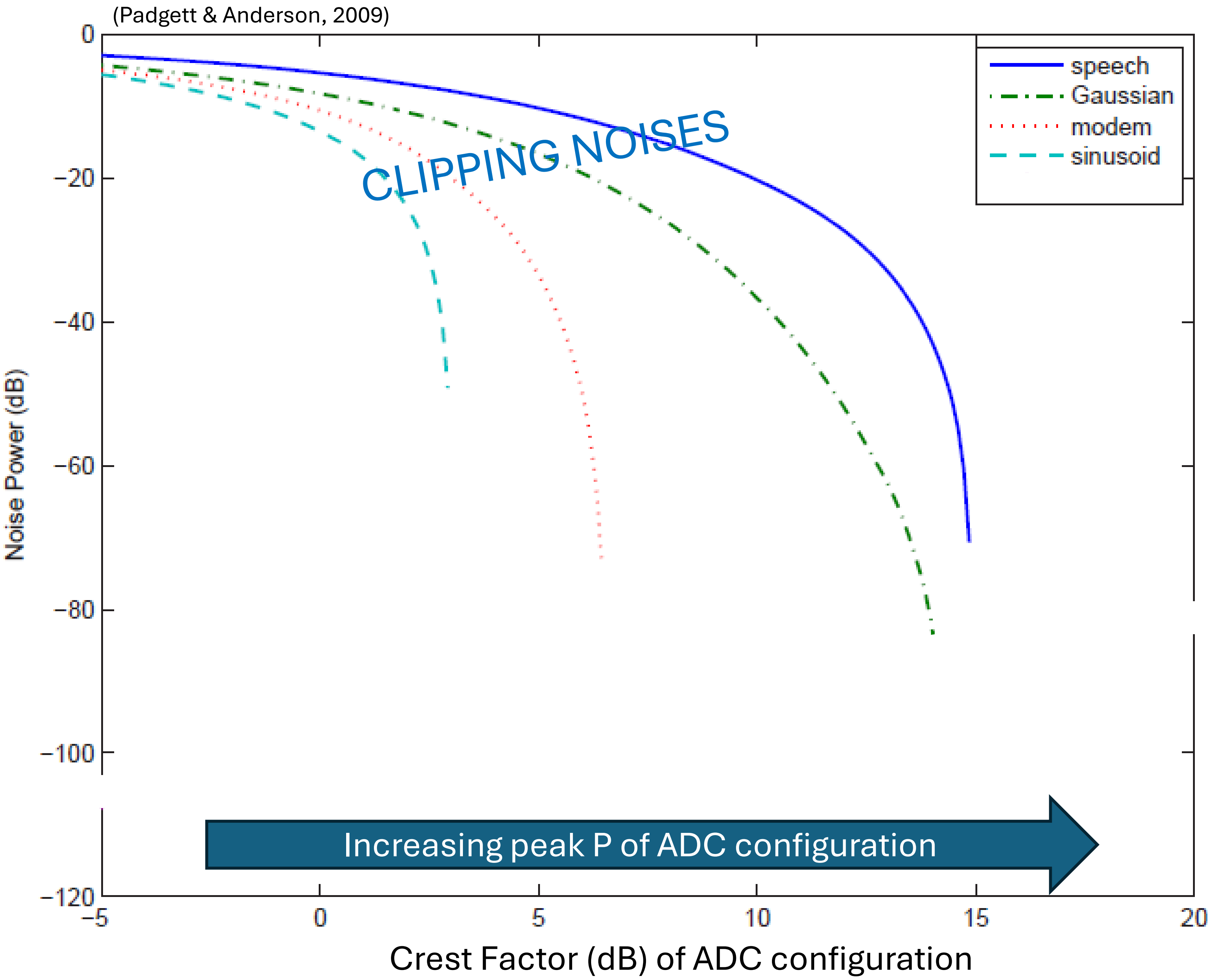
Gaussian signal



Event $|x[n]| > 3.29\sigma$ occurs with probability 0.001.

Set $Q = 3.29\sigma$ in the equation =>

$$CF_{\mathrm{SIG}} = 10\log\frac{(3.29\sigma)^2}{\sigma^2} = 20\log 3.29 = 10.34\ \mathrm{dB}$$

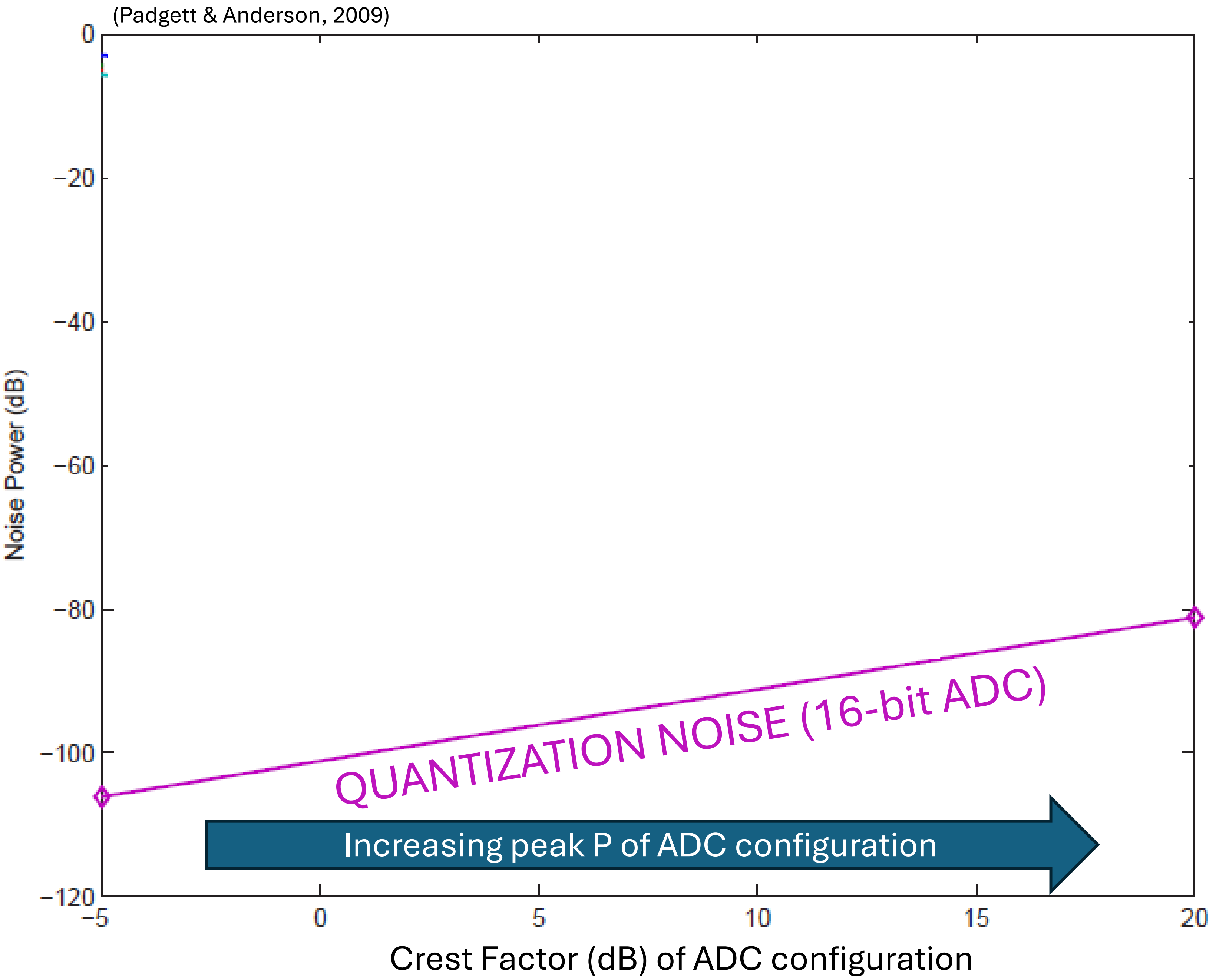


With increasing peak P level clipping noises drop.

It depends on the signal how much P must be increased.

Speech signal has heavier tails than Gaussian.

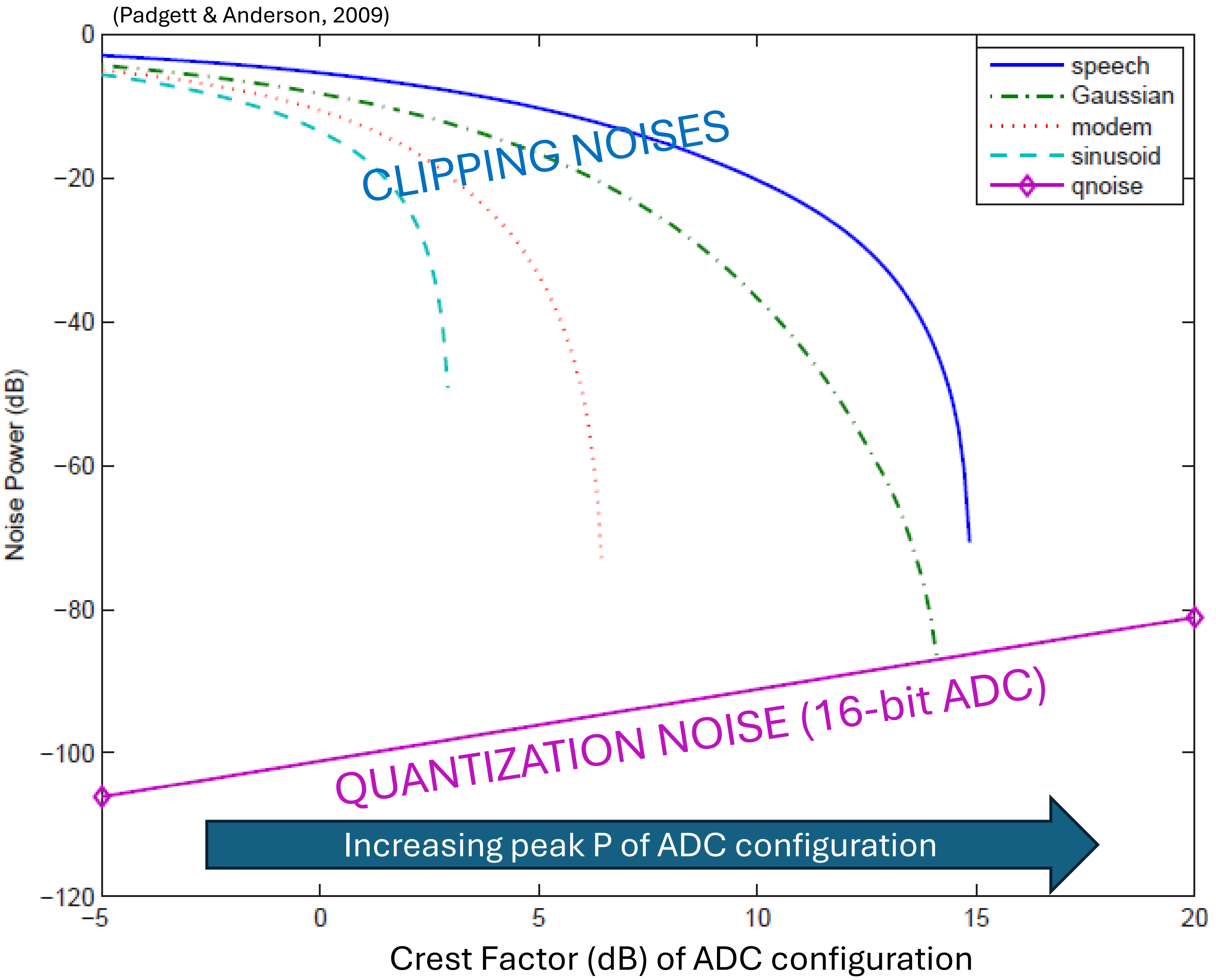
$$CF_{\mathrm{ADC}} = 10\ \log_{10}\frac{P^2}{S}\ [\mathrm{dB}]$$



With increasing peak P level quantization noise increases as quantization level difference increases.

$$\Delta = \frac{P}{2^{B-1}}$$

$$CF_{\mathrm{ADC}} = 10\ \log_{10}\frac{P^2}{S}\ [\mathrm{dB}]$$



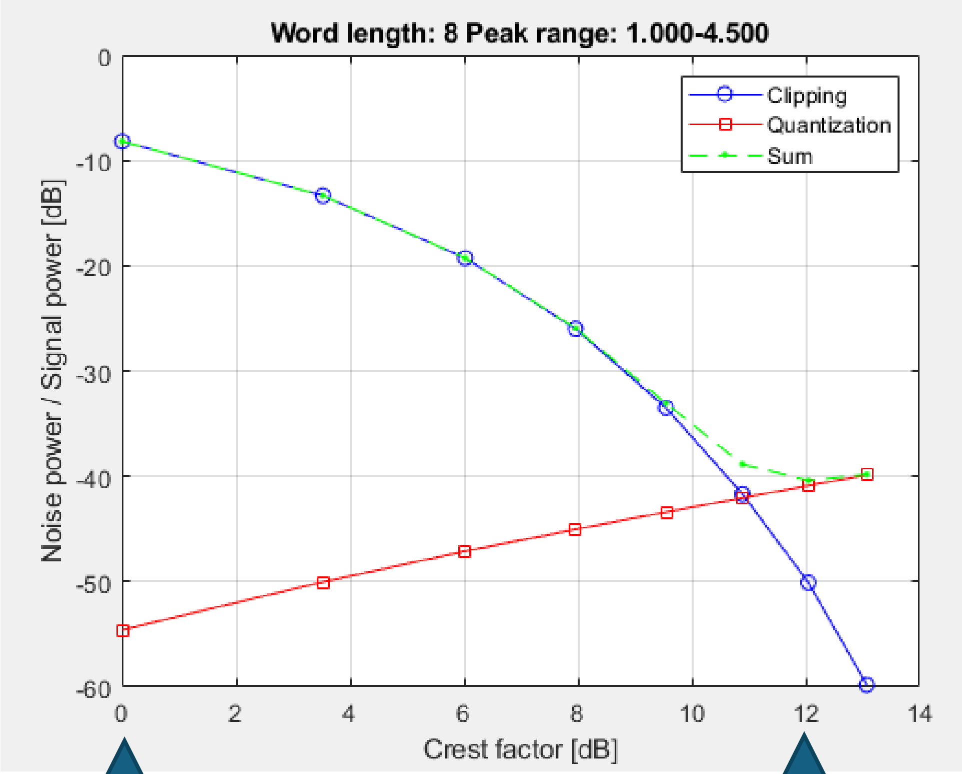
Considering ADC noise, optimum tuning of P is at the crossing point of these two noises.

$$CF_{\mathrm{ADC}} = 10\ \log_{10}\frac{P^2}{S}\ [\mathrm{dB}]$$

Matlab example

- Two Matlab files (given in Moodle):
  - `saturatingADC.m`
  - `crestFactorPlot.m`

```
% Gaussian signal with variance 1
x = randn(1,1000000);
Ps = 1.0:0.5:4.5; % ADC peaks P
B = 8; % ADC output word length
crestFactorPlot(x,Ps,B);
```



P = 1.0

CF of ADC configuration

P = 4.0

We see that clipping and quantization noise powers are similar when ADC is tuned for CF = 11 dB (ADC peak P = 3.5σ).

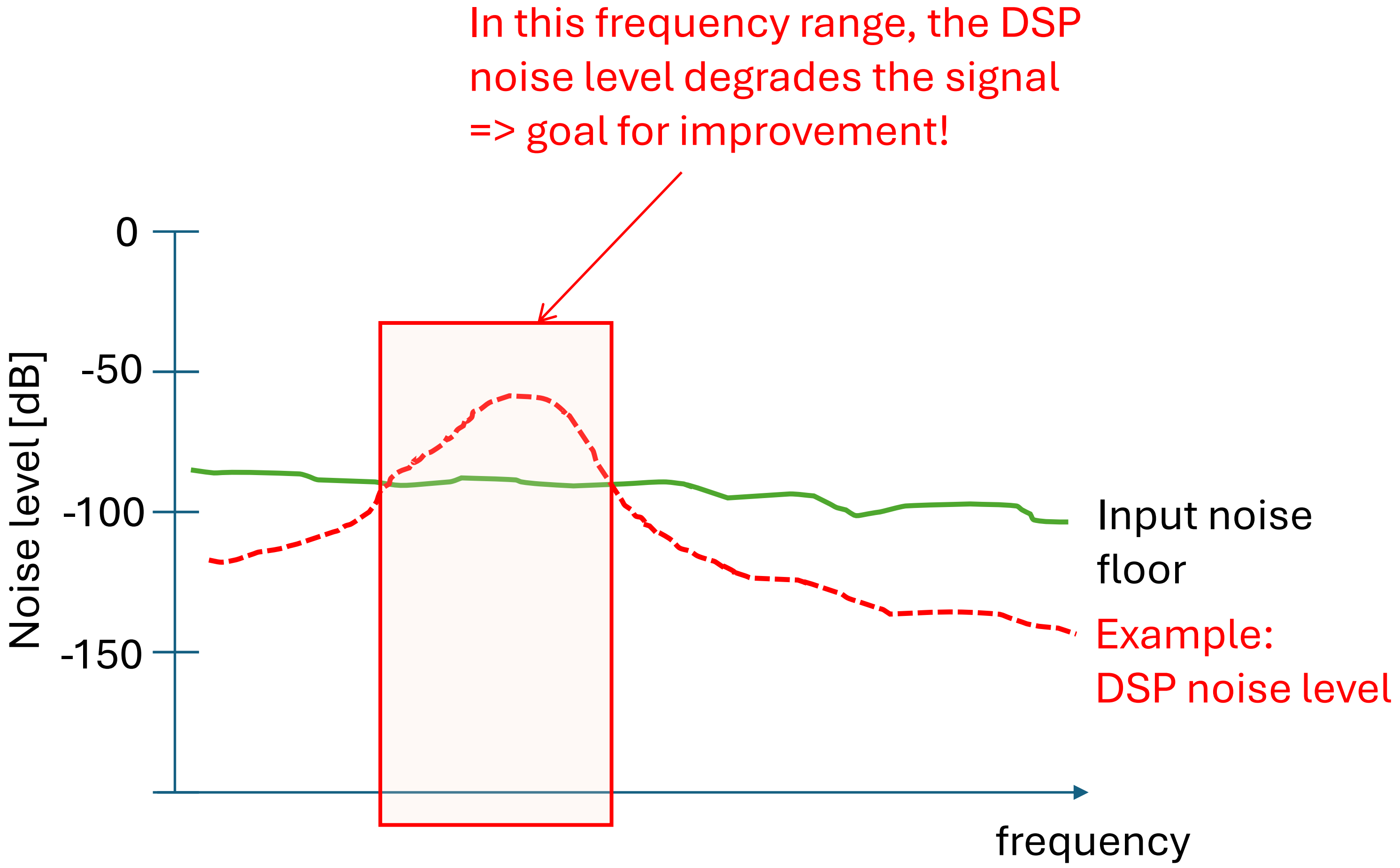
Perhaps this is a good choice for ADC peak value as quantization and clipping noises are in balance.

1.3. Input noise floor

The noise induced by A/D conversion and inherent noise in the analog input set up a <u>noise floor</u> for processing.

For the signal processing output, <u>we cannot get above</u> this level.

So, this input noise floor can set up a goal for DSP design: **keep the noise at the level of input noise**!



(Ifeachor & Jervis, 2002, p. 817)

1.4. Spectrum of the quantization noise

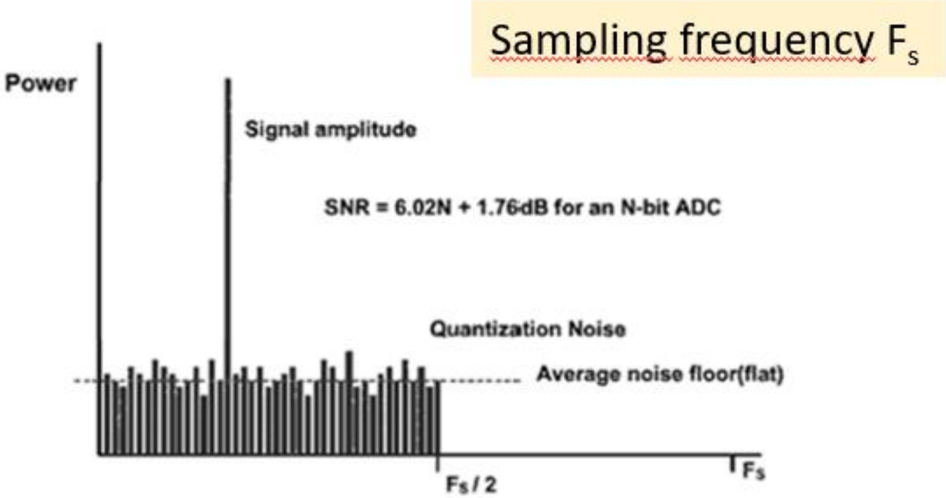
- Spectrum of the ADC quantization noise is approximately flat
- The power spectral density (PSD) for sample rate F_s is

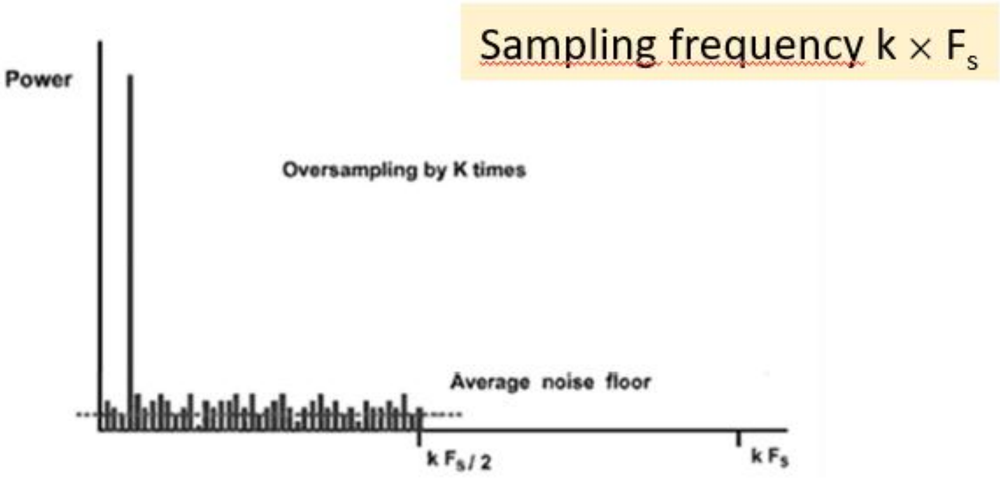
$$S(f) = \frac{N_{quant}}{F_s}$$

where

$$N_{quant} = \frac{1}{\Delta}\int_{-\Delta/2}^{+\Delta/2} x^2 dx = \frac{\Delta^2}{12}$$

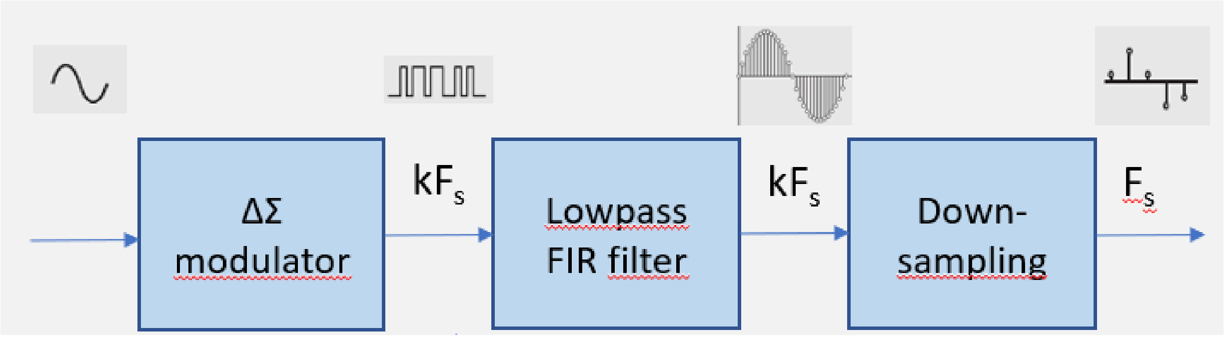
- **Note:** If sampling rate is increased, noise floor becomes lower



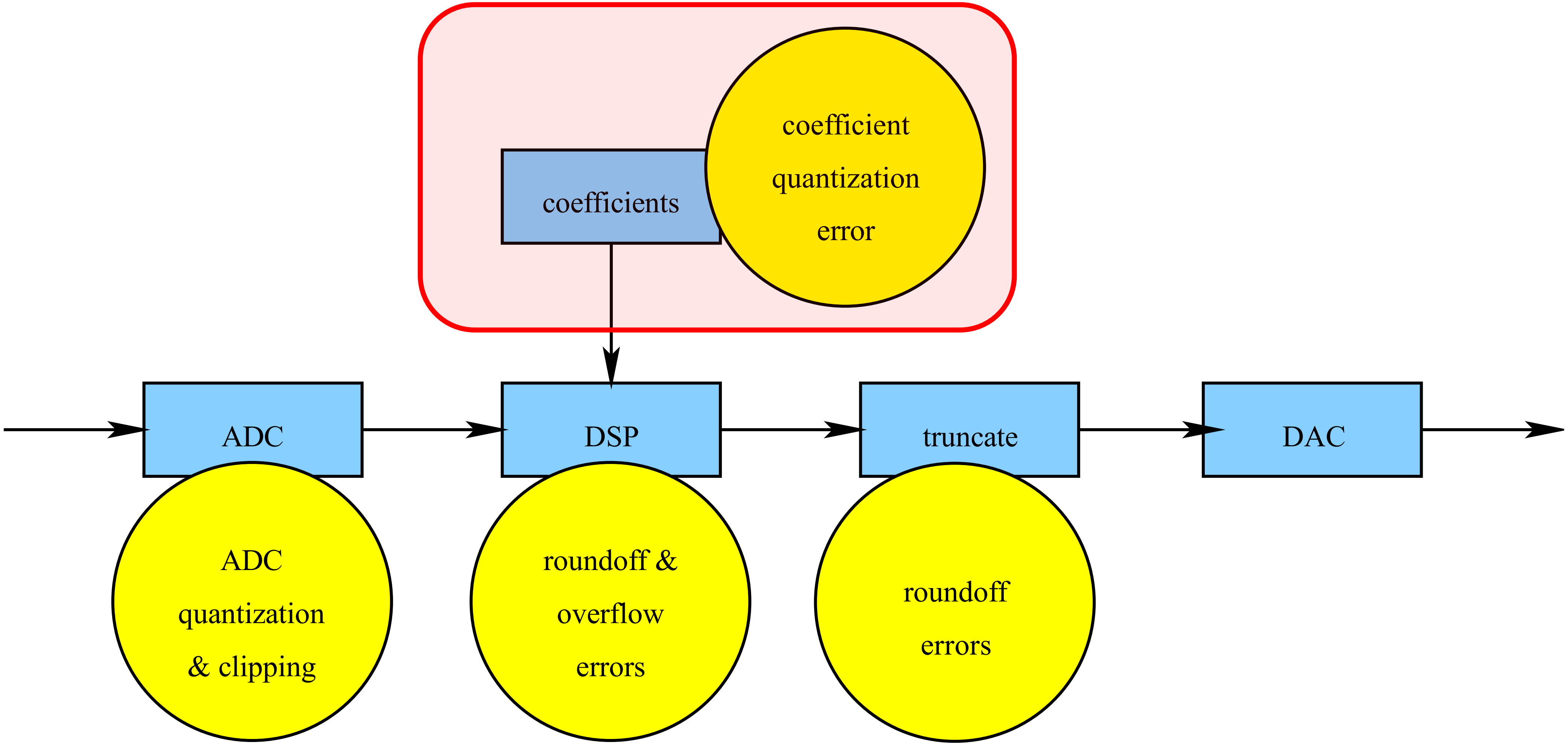


Oversampling ADC

- Exploit this notion about sample rate and noise floor
- Oversampling ADC is a combination of
  - High-frequency sampling
  - A/D conversion
  - Decimation to lower the sample rate
- Extreme: One can even use one-bit ADC with very high sampling rate
  - ΔΣ modulator
- Result: ADC quantization noise reduced
- Common technique in audio signal processing
- A multirate technique: more on this later



2. IIR filter coefficient quantization

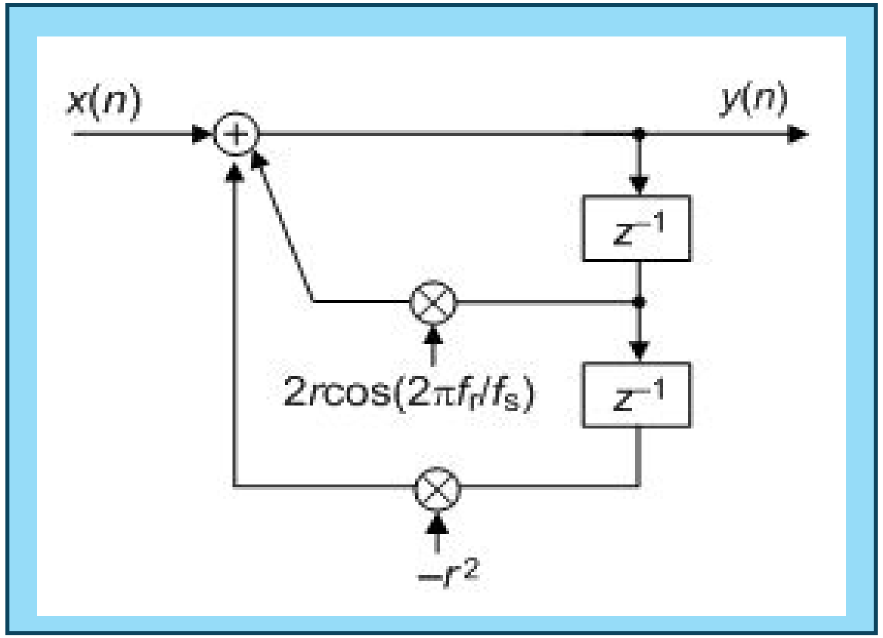


Problem

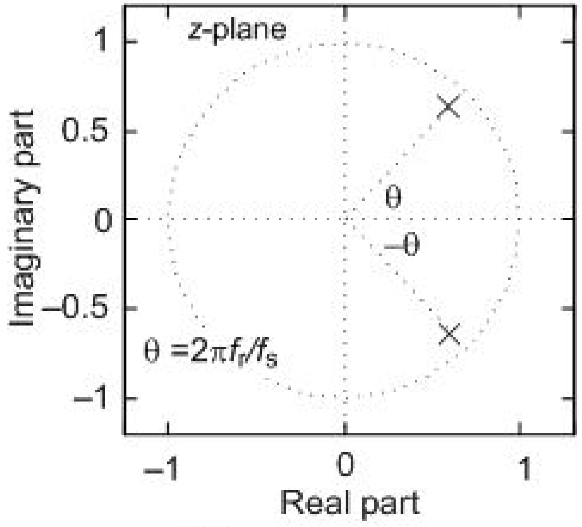
- When we quantize IIR filter coefficients
  - What happens to the response of the filter?
  - Is the filter still stable?
- How to find optimal solutions?
  - Not just coefficient quantization, also choice of implementation structure important
- Considering two cases
  - 2nd order IIR resonator
  - High-order IIR filter

Case 1: 2nd order IIR resonator

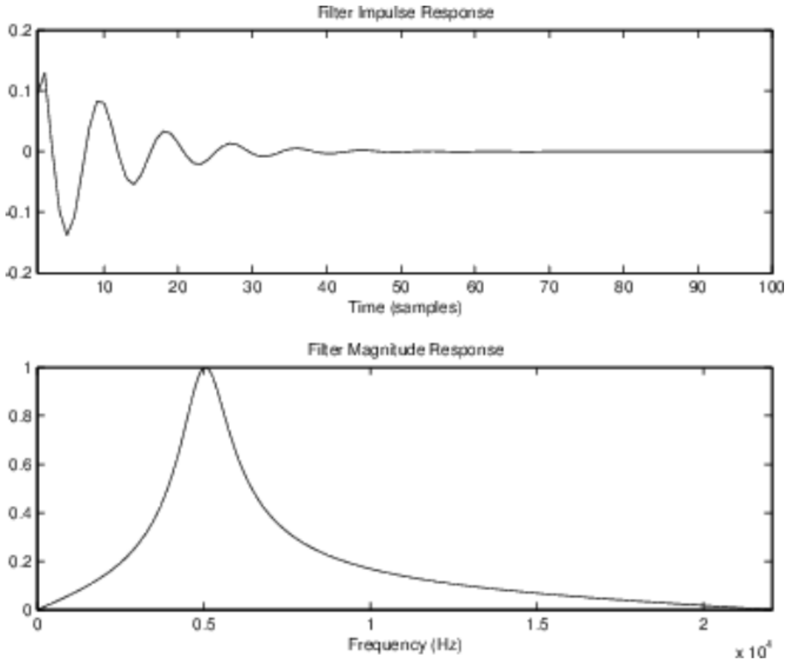
- <u>Resonator</u>: impulse response of the filter is a damped sinusoid with frequency f_r and damping factor r
- Is a band-pass filter



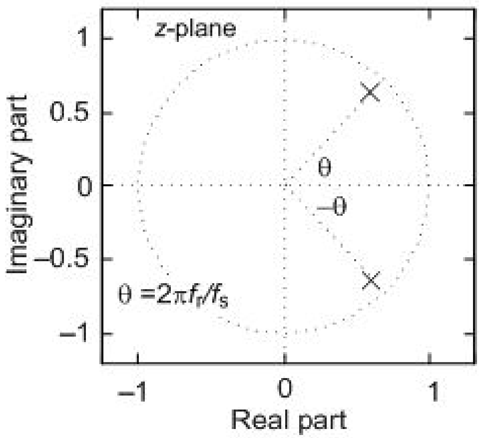
Poles



Response



2nd order IIR resonator



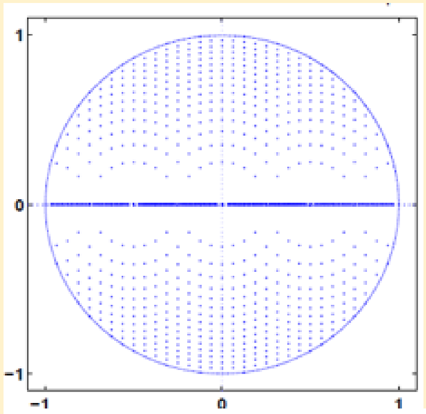
- The transfer function of the filter is of form

$$H(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

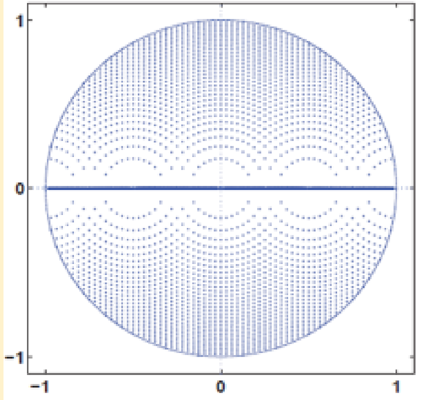
- The filter has complex-conjugate roots (poles) at $z = re^{\pm j\theta}$, where $\theta = 2\pi f_r / f_s$
- In direct fixed-point implementation, we need to quantize $a_1 = 2r\cos\theta$ and $a_2 = -r^2$
  - But: not full control of pole positions
  - Observable sparsities in pole position plots
  - Especially low-pass (f_r close to 0) and high-pass (f_r close to $f_s/2$) filters are harder to realize - longer word lengths needed

Available pole positions

5-bit coefficients



6-bit coefficients

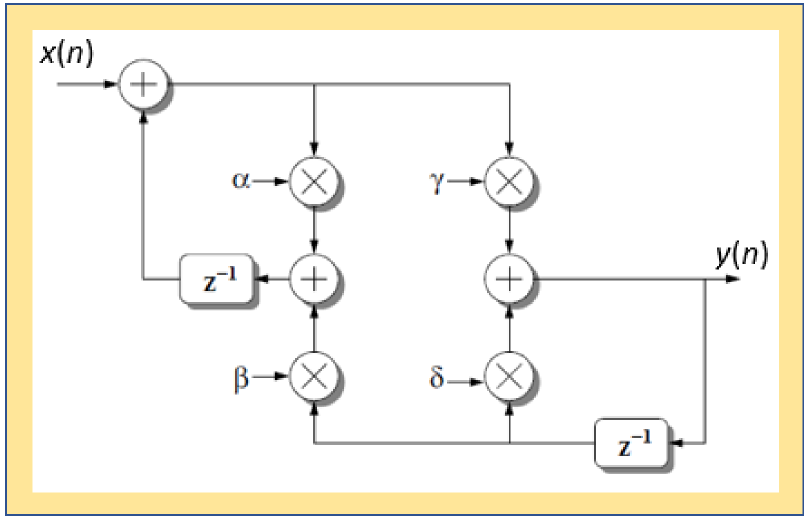


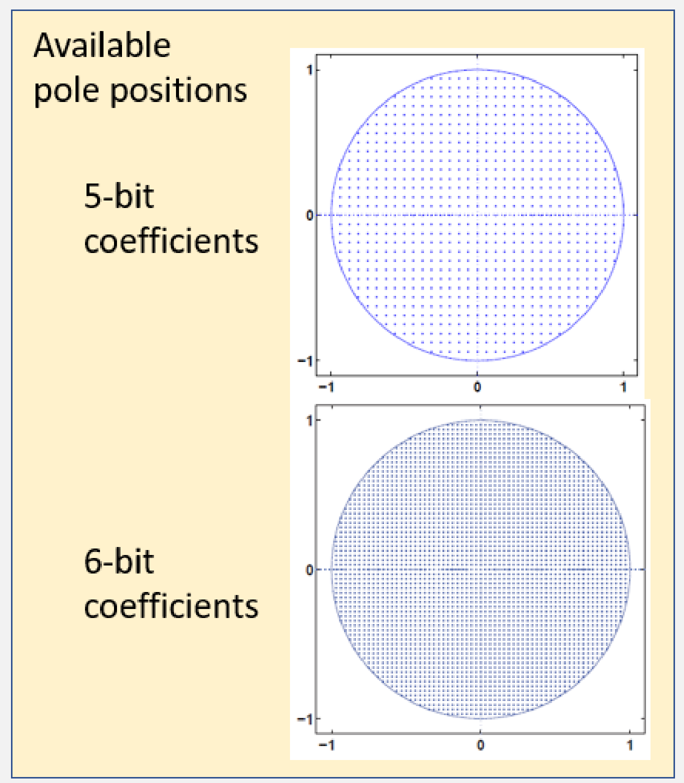
2nd order coupled form IIR

- Provides a solution to resonator implementation problems
  - note: four multiplications instead of two
- The transfer function is

$$H(z) = \frac{\gamma}{1-(\alpha+\delta)z^{-1}-(\beta\gamma-\alpha\delta)z^{-2}}$$

- Setting $\alpha = \delta = r\cos\theta$ and $\beta = -\gamma = r\sin\theta$, we get resonator's transfer function (up to scaling γ)
- Quantization of coefficients (α, β, γ, δ) leads to **rectangular grid** of pole positions





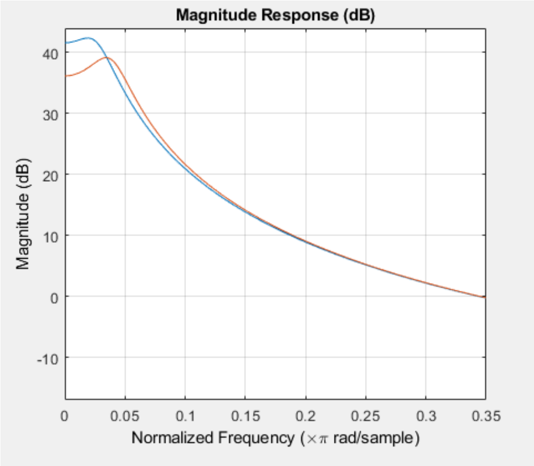
Discussed in https://www.dsprelated.com/showarticle/183.php

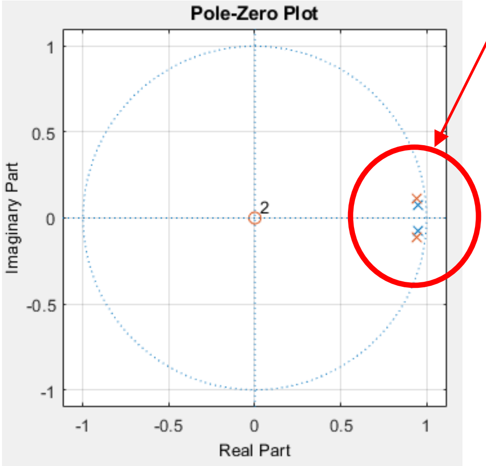
Example. Resonator for $f_r = f_s/80, r = 0.95$

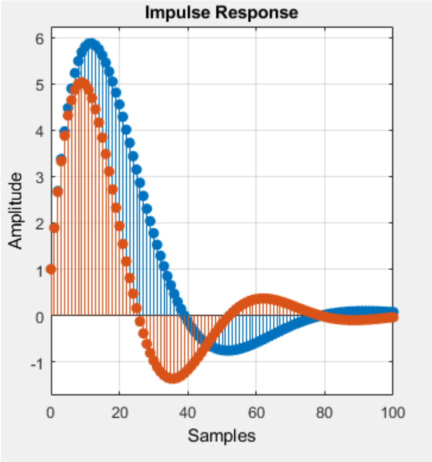
$\theta = \mathbf{0.025}\pi.$

difference

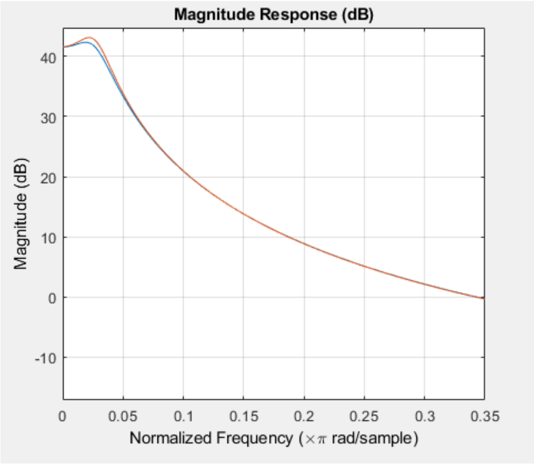
Direct form fixed-point
s8.6 format

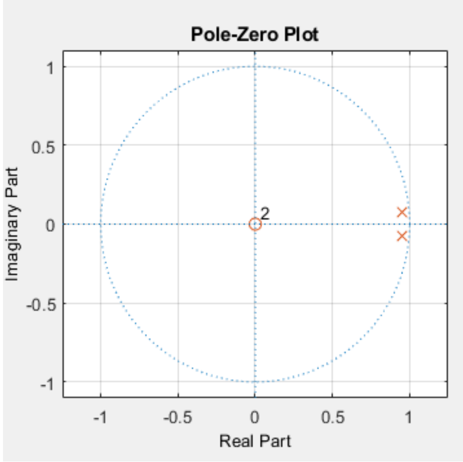


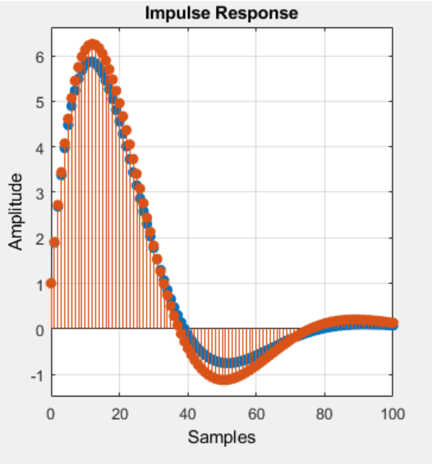




Coupled form fixed-point
s8.6 format





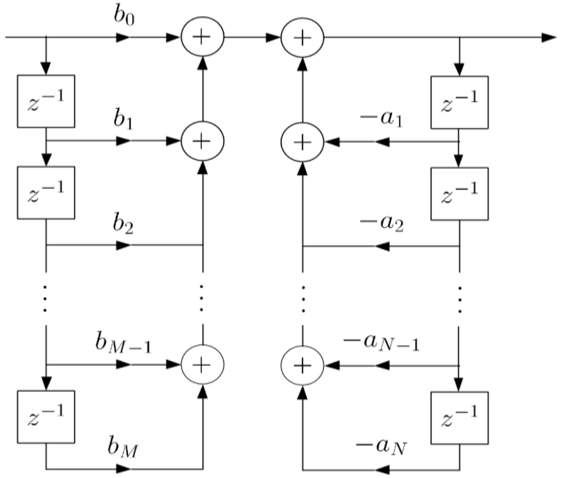


floating-point filter

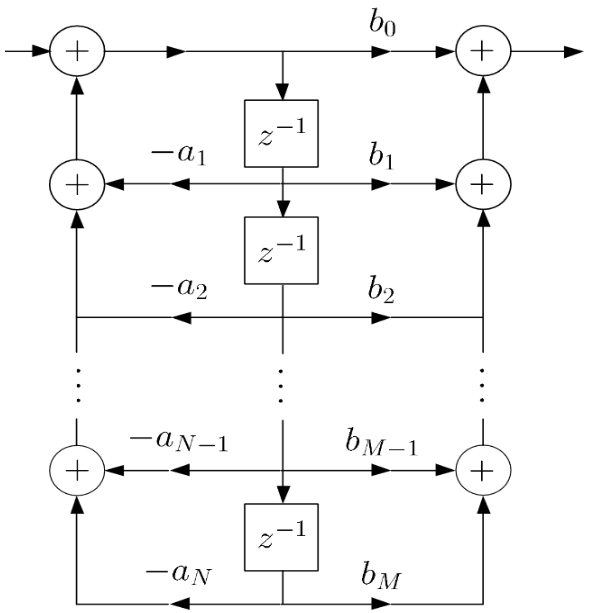
fixed-point filter

Case 2: high-order IIR filters

- Direct form implementations are problematic. Reason:
  - Typically, we want to use the same fixed-point format for all numerator coefficients and another one for all denominator coefficients
  - However, the **magnitude of the coefficients may vary a lot**, and then the precision of some coefficients can become too low



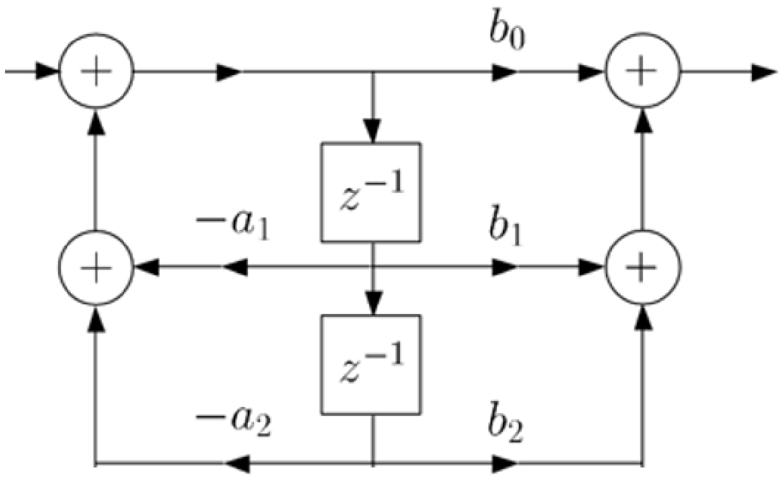
Direct form 1



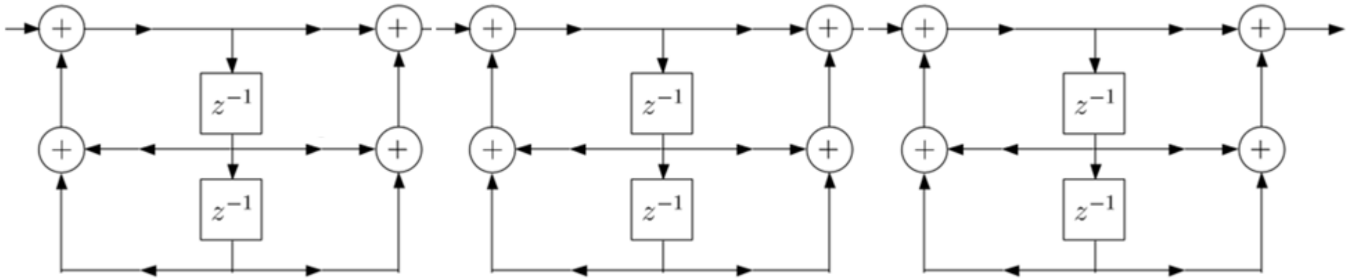
Direct form 2 (canonic)

Case 2: high-order IIR filters

- The solution is to consider cascaded second-order section (SOS) implementations
  - Coefficients of each section have relatively small range
  - Note: not first-order sections as we want to avoid complex arithmetic – 2nd order SOS has real coefficients



SOS

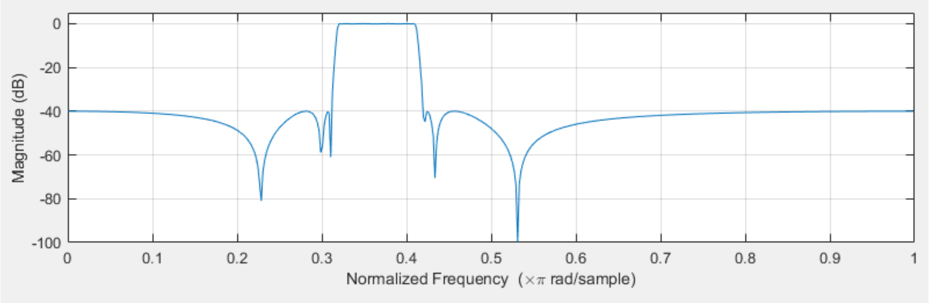


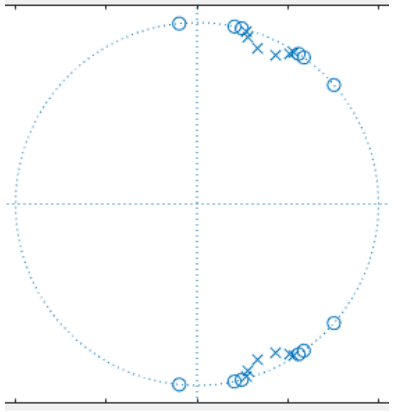
SOS cascade

Example. elliptic bandpass filter (order 12)

Specification
- passband cut-offs 0.32, 0.41 (x $F_s/2$)
- passband ripple 0.1 dB
- stopband attenuation 40 dB

Property: fast transition in gain



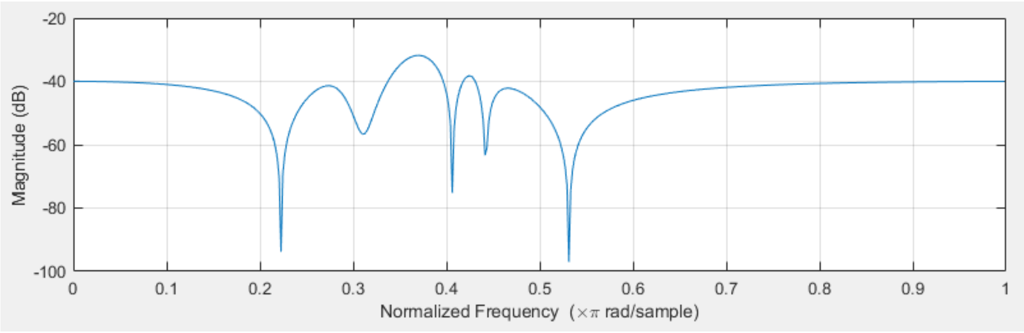


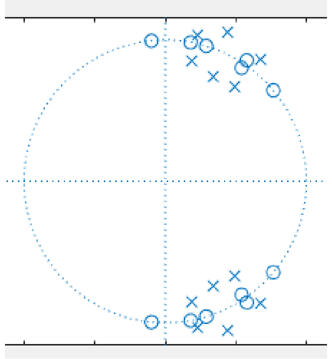
Designed using matlab: `[b,a] = ellip(6,0.1,40,[0.32,0.41])`

numerator *b* [0…12]	denominator *a* [0…12]
0.010402294826284	1.000000000000000
-0.046981738177133	- 4.736891331870121
0.140584418862936	14.770074008149525
-0.291587543984709	-31.339008666596676
0.483154936264528	52.290097782458098
-0.638421915199626	-68.277884768136573
0.705123004845763	73.209425975402013
-0.638421915199625	-63.111029848304227
0.483154936264528	44.673344222037109
-0.291587543984708	-24.741797451330484
0.140584418862936	10.774944914870538
-0.046981738177132	- 3.191627411828423
0.010402294826284	0.622743578239246

Large variation in magnitude

Trying 16-bit fixed-point quantization, one format for *a* and one for *b*, that is, s16.15 for *b and* s16.8 for *a*





No passband, not even stable!

Solution: Cascaded SOS implementation

Matlab:

```
[b,a] = ellip(6,0.1,40,[0.32,0.41])
[SOS,G] = tf2sos(b,a)
```

denominator

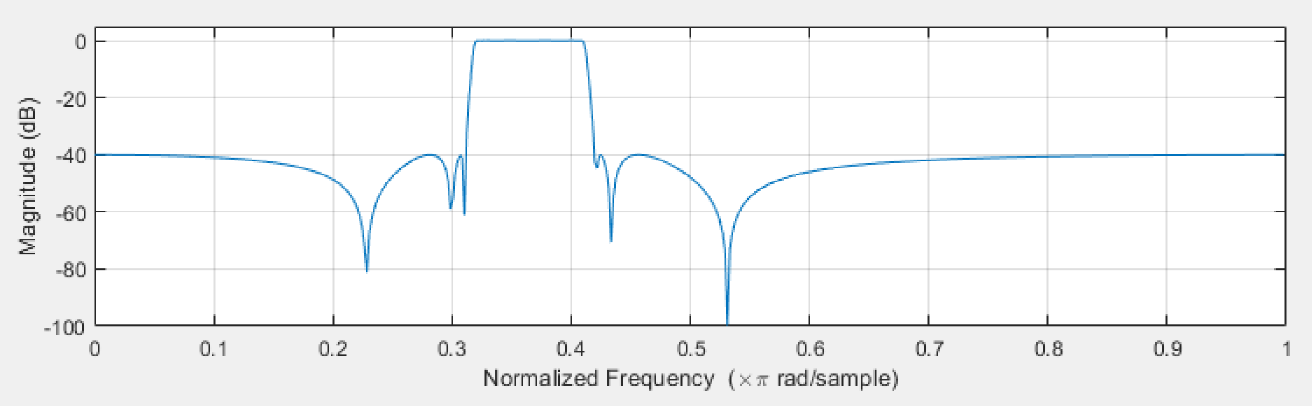
6 SOS sections

b[0]	*b*[1]	*b*[2]	*a*[0]	*a*[1]	*a*[2]
1	0.195430137972181	1.000000000000039	1	-0.664513985463847	0.852588727845048
1	-1.508636017182940	0.999999999999994	1	-0.873355320444148	0.860012128003494
1	-0.412640684596677	0.999999999998724	1	-0.564681165798558	0.932247834363709
1	-1.177232208175213	1.000000000000585	1	-1.019134004433838	0.939497814911048
1	-0.492675670575205	1.000000000001395	1	-0.543644167472686	0.983728242794488
1	-1.120723792458221	0.999999999999252	1	-1.071562688257045	0.985740948417509

Gain G: 0.010402294826284

Magnitudes are of the same order
(but note: separated gain G)

Trying 16-bit fixed-point quantization, one format for *a* and *b*, s16.14. Gain quantized separately as s16.21



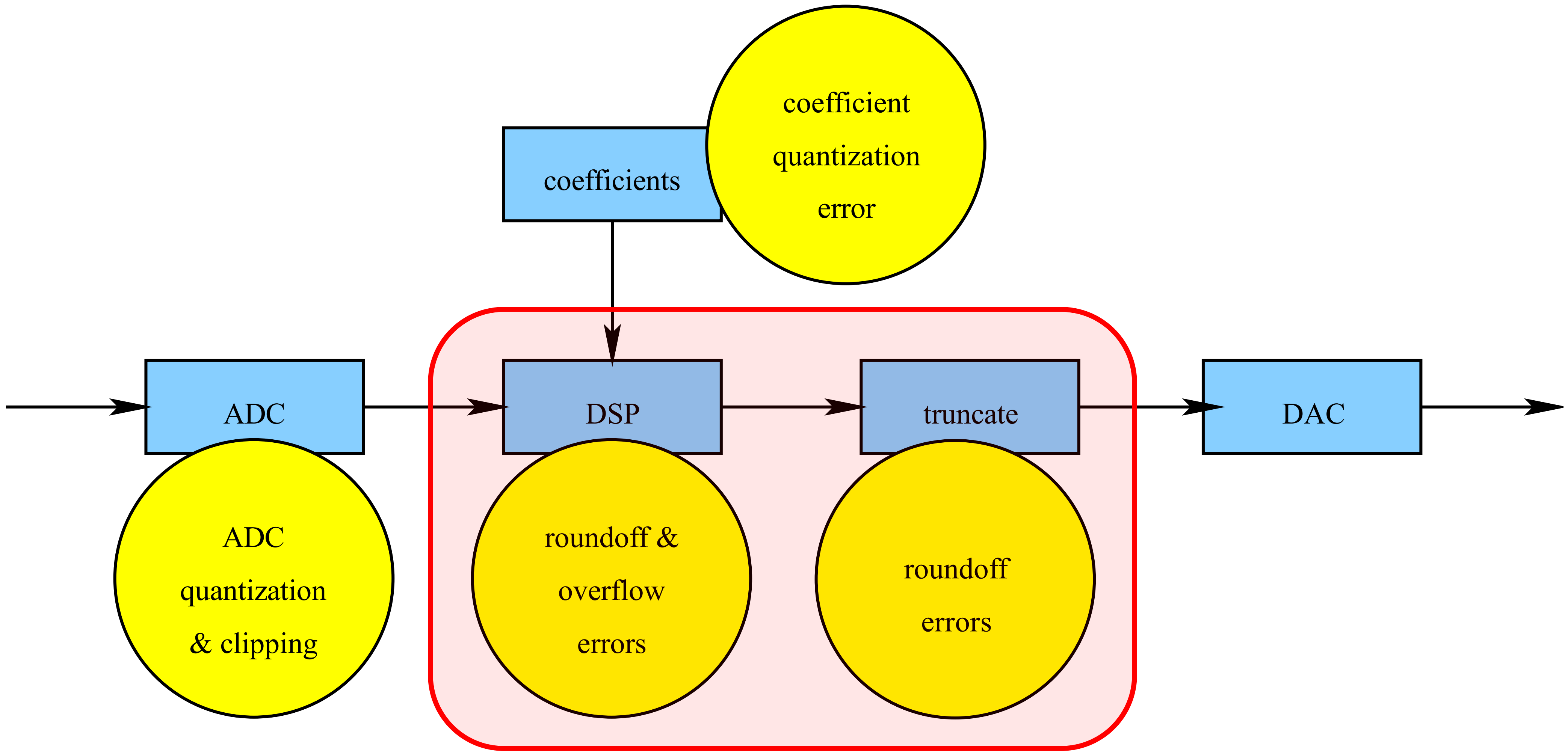
Very close to the ideal floating-point response ☺

Lesson

- To deal with coefficient quantization effects, fixed-point IIR filter design typically requires **special flow of computations**
  - It's not just coefficient quantization
  - Problem is due to limited dynamic range of fixed-point numbers

- Another example of having special structures for fixed-point computation
  - recall matrix inversion and use of decomposition techniques (see Lecture 3)

3. Quantized IIR filter: signal processing error



In the following

Implementing fixed-point IIR filter simulation takes effort.

To grasp some ideas, let us consider implementation of
second-order-section (SOS) IIR filter on a DSP platform.

- Word length tunings in the platform possible.
- Learning how **input scaling** is used to deal with the overflow problems.
- Then, we consider **choice of fixed-point types** for different points on the data path.
- Based on this, **Matlab code for simulating** the system written.
- Finally, some simulation **experiments** to study effects

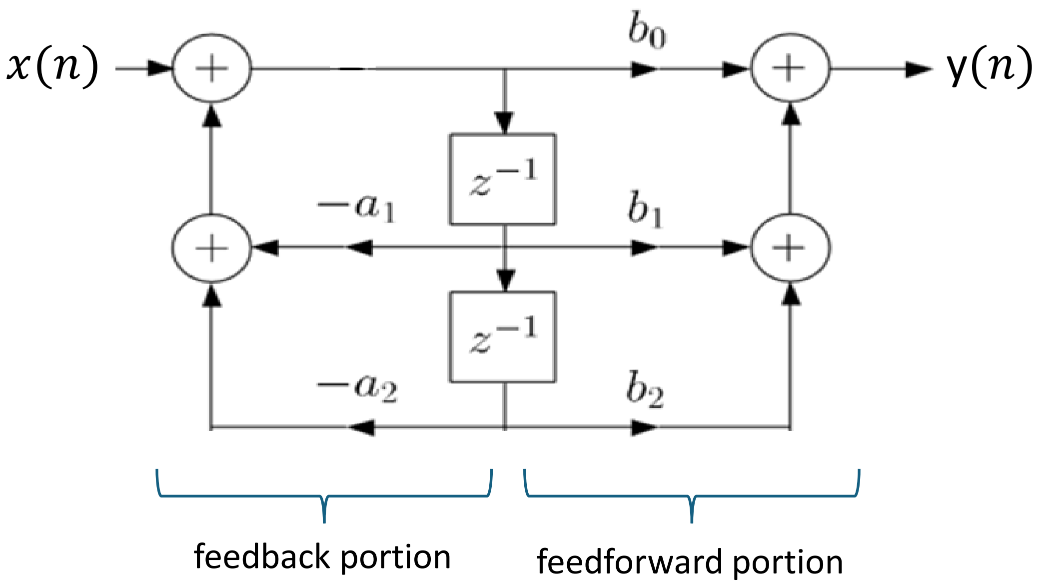
3.1. Problem: Implementing a second-order IIR filter

SOS is very useful and common component in fixed-point IIR filter implementations.

SOS transfer function:

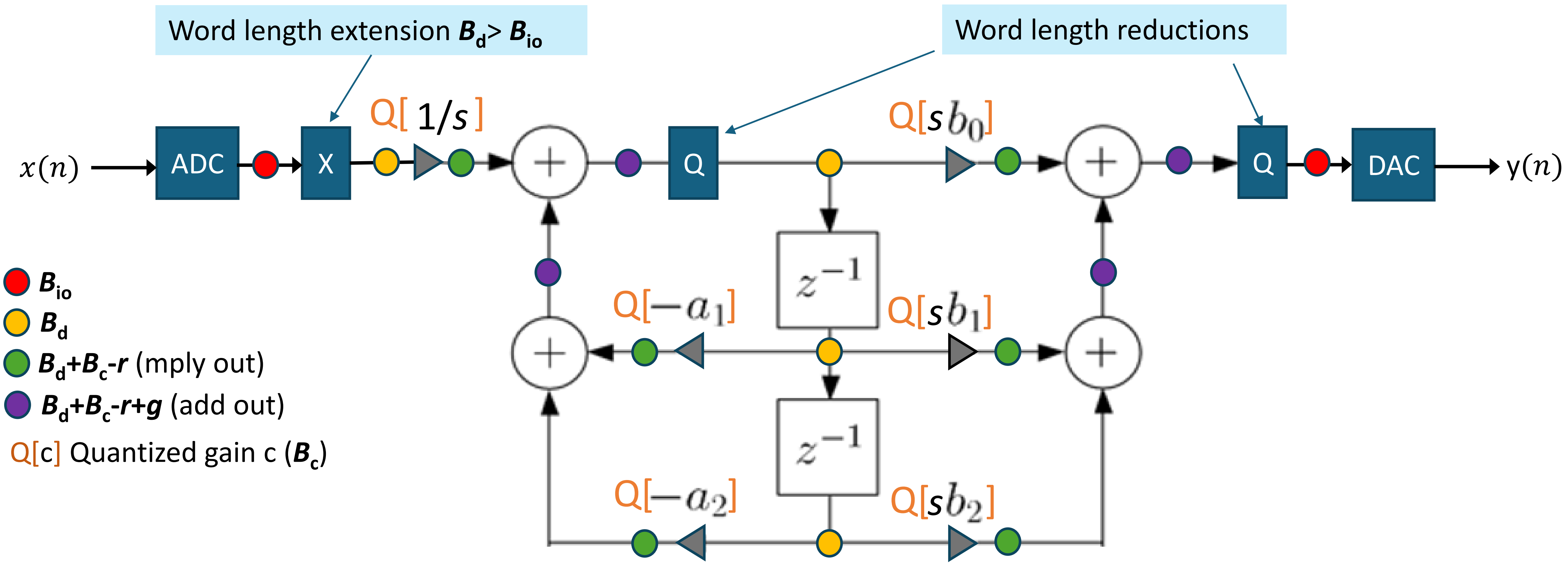
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Canonic direct form II –based implementation

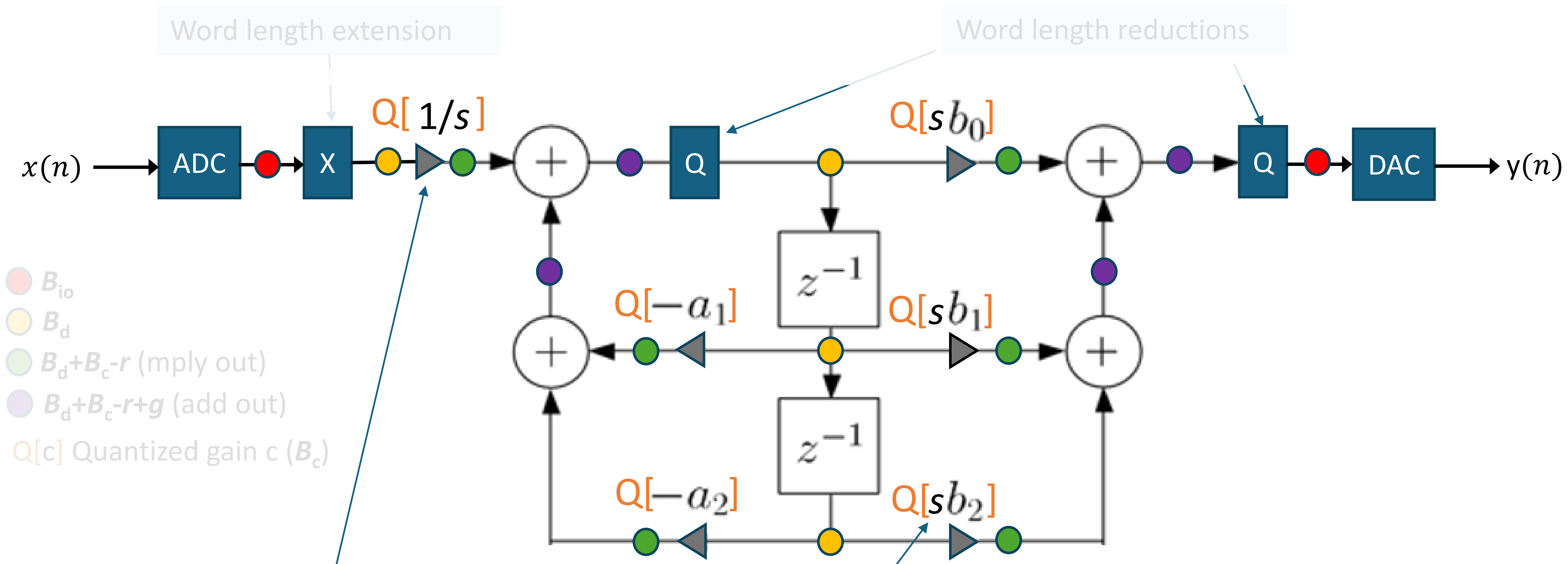


- Assumed DSP platform:
  - Integer arithmetic
  - ***B*$_{io}$**-bit I/O
  - ***B*$_d$**-bit registers for delays
  - ***B*$_c$**-bit memories for coefficients (incl. pre-scaling factor)
  - Multiplier output is **(B_d+B_c-r)**-bit (***r*** is the number of LSB bits discarded from the result by rounding)
  - Sums stored into **(B_d+B_c-r+g)**-bit accumulators (***g*** is the number of guard bits); saturating adder used

Signal flow with DSP platform details



Pre-scaling

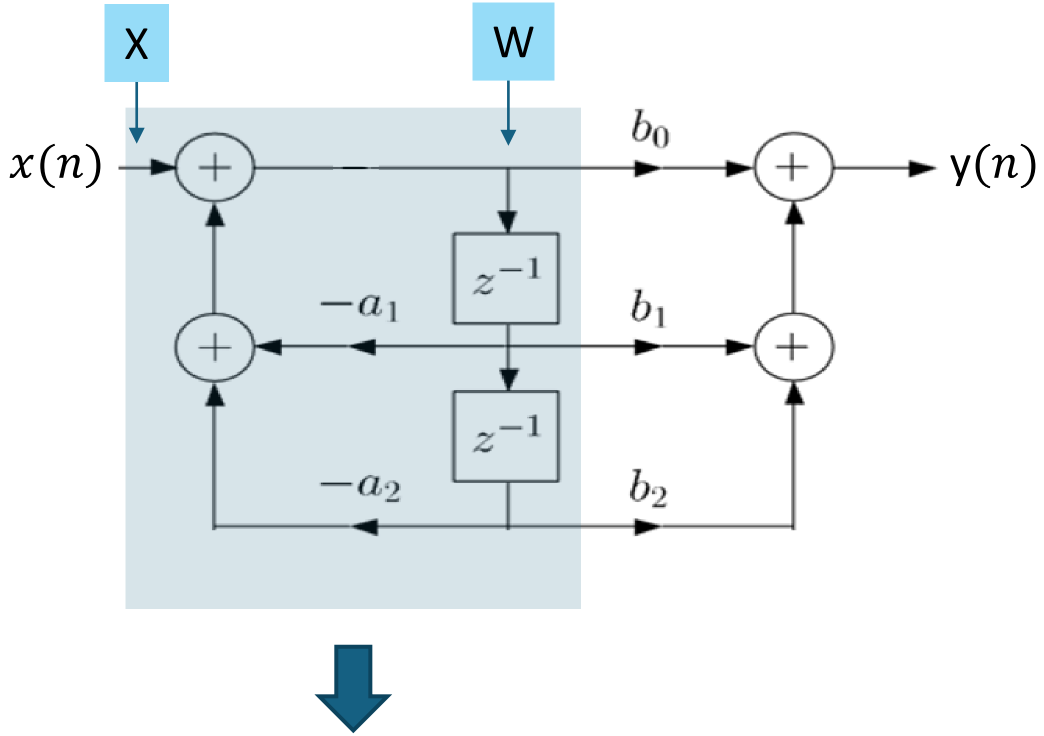


One additional multiplication:

$1/s$ A **pre-scaling** factor (<1) that is used to reduce/avoid overflows in the feedback portion
Compensation for it is done in scaling of feedforward coefficients

3.2. Scaling for overflow control

- $x(n) \in [-1, +1)$
- We reduce the input range so that overflows in the feedback portion are reduced or even avoided completely
- To find the pre-scaling $1/s$, we consider the response at point W to the impulse at point X
  - Transfer function $H'(z)$
  - A norm of its impulse response $f(n)$ provides the gain s that we compensate for
  - There are various norms that can be applied, computed in time or frequency domain



$$H'(z) = \frac{W(z)}{X(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Three common norms for scaling

- L_1 norm applied to impulse response $f(n)$
  - $s = \sum_{n=0}^{\infty} |f(n)|$
  - Overflow avoided, but may be too conservative
- L_2 norm applied to impulse response
  - $s = \sqrt{\sum_{n=0}^{\infty} f^2(n)}$
  - Provides better SNR than previous one (if overflows do not occur)
- L_∞ norm applied to frequency response $F(w)$
  - $s = \max\ |F(w)|$
  - Overflow with full scale sinusoidal input avoided
- Relationship: $L_2 < L_\infty < L_1$

L2 : Closed form solution for SOS

$$s = \sqrt{\frac{1}{1-a_2^2-a_1^2(1-a_2)/(1+a_2)}}$$

Matlab:

```
f = impz(1,[1,a1,a2])
L1 = norm(f,1)
L2 = norm(f,2)
F = freqz(1,[1,a1,a2])
Linf = max(abs(F))
```

Note:

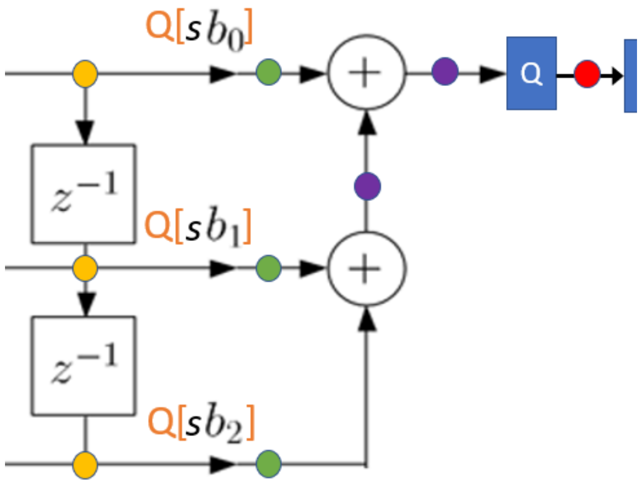
```
L2 = norm(F,2)/sqrt(numel(F))
```

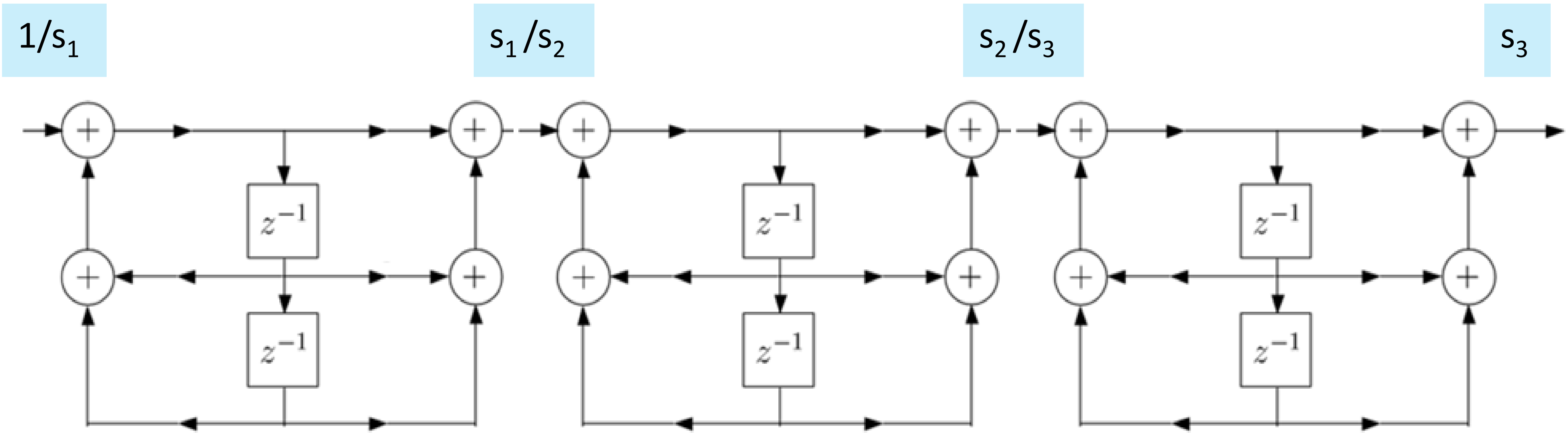
Notes on scaling

COMPENSATION for 1/s ?

Alternatives

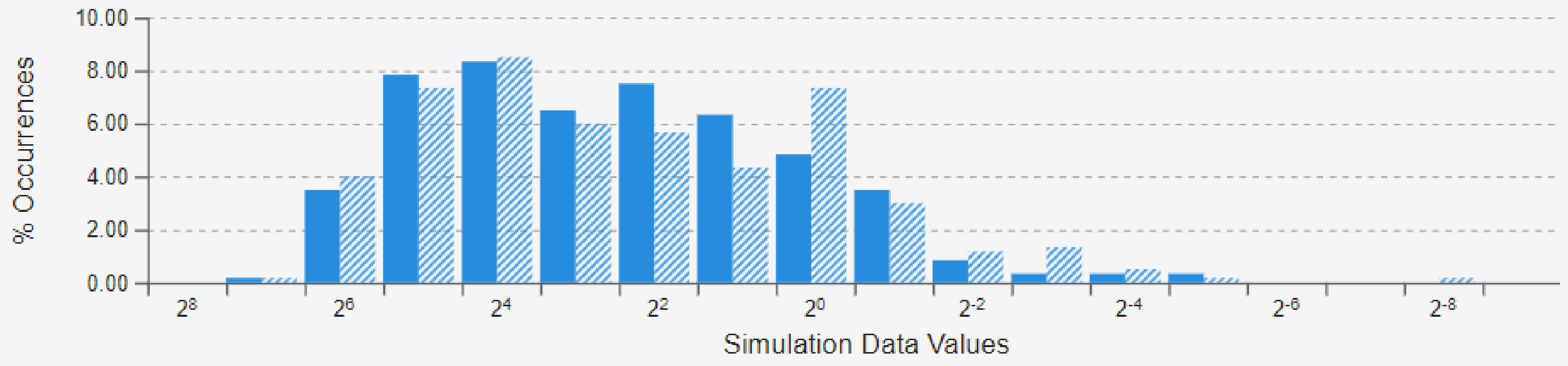
- **Scaling of feedforward coefficients b_i by s** => filter gain unchanged
- A scaling factor ≠ s might be used for them
  - What is the range of the multiplier/accumulator outputs?
- Scaling of b_i might also be neglected
  - Do scaling before rounding to output word length (extra multiplication)
  - In SOS cascades, integrate it to pre-scaling of the following SOS





Tool: NumericTypeScope

- A tool in Fixed Point Toolbox
- Can visualize what kind of values observed for a variable
    - If it has fixed-point type, provides information about overflows and underflows => information on adjusting word and fraction lengths



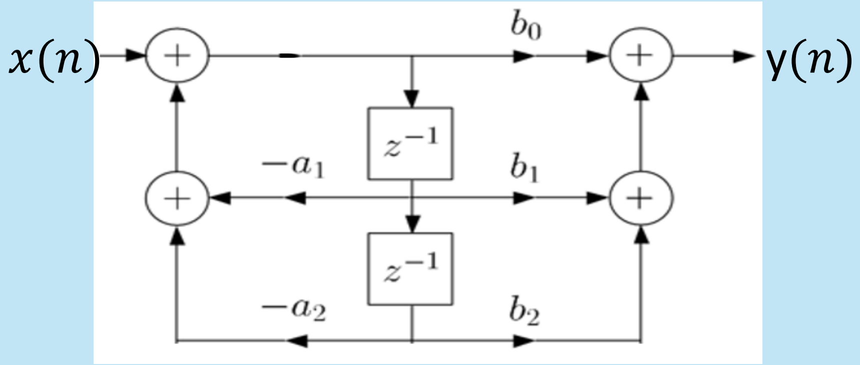
Values	Potential Overflows	In-Range	Potential Underflows
Positive	0	302	0
Negative	0	298	0
Zero	0	0	0

Interpretation: the bars labeled 2^k corresponds to observed magnitudes $2^k \leq |x| < 2^{k+1}$

Input scaling

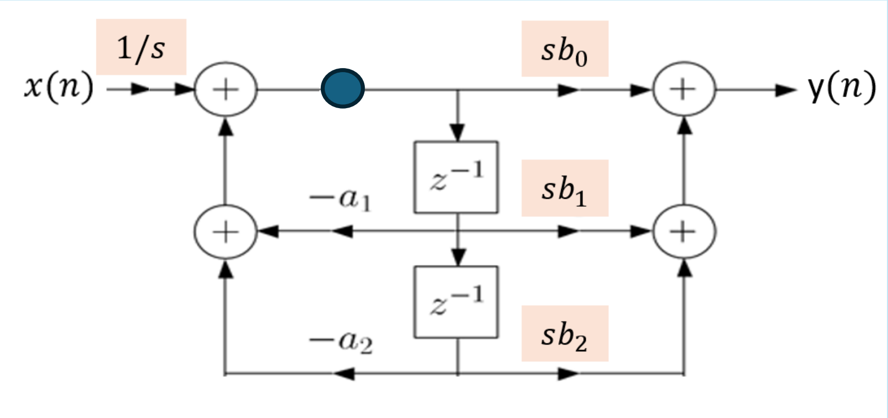
The idea is to prevent overflows in feedback section (e.g., keep results below 1)

```
for k=1:length(x)
    z0 = x(k) - a1 * z1 - a2 * z2;
    y(k) = b0 * z0 + b1 * z1 + b2 * z2;
    z2 = z1; z1 = z0;
end
```



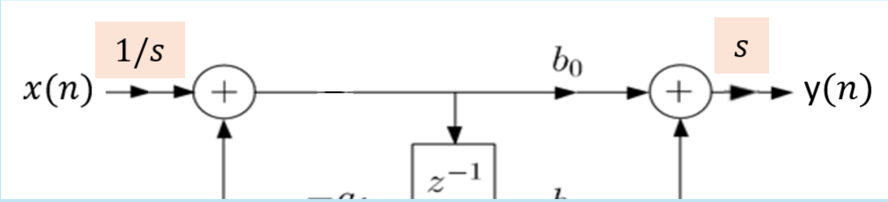
pre-scaling, compensation by scaled feedforward coefficient

```
for k=1:length(x)
    z0 = si * x(k) - a1 * z1 - a2 * z2;
    y(k) = sb0 * z0 + sb1 * z1 + sb2 * z2;
    z2 = z1; z1 = z0;
end
```



pre-scaling, compensation by post-multiplication

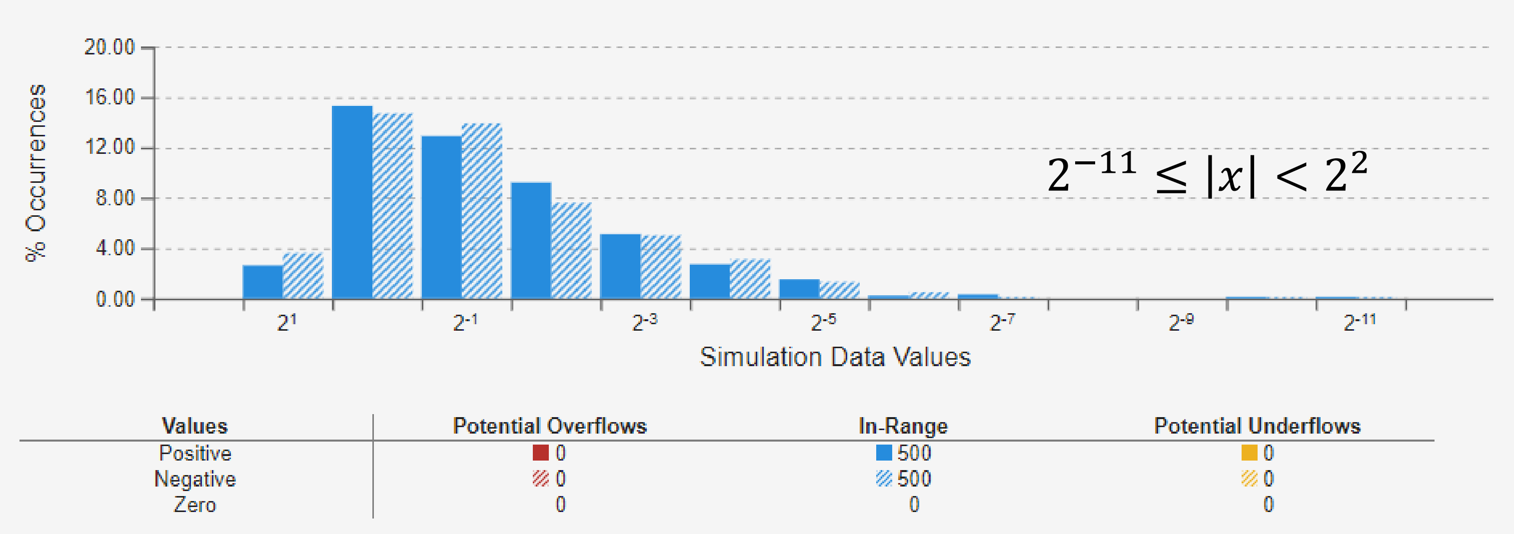
```
for k=1:length(x)
    z0 = si * x(k) - a1 * z1 - a2 * z2;
    siy = b0 * z0 + b1 * z1 + b2 * z2;
    y(k) = s * siy;
    z2 = z1; z1 = z0;
end
```



Example

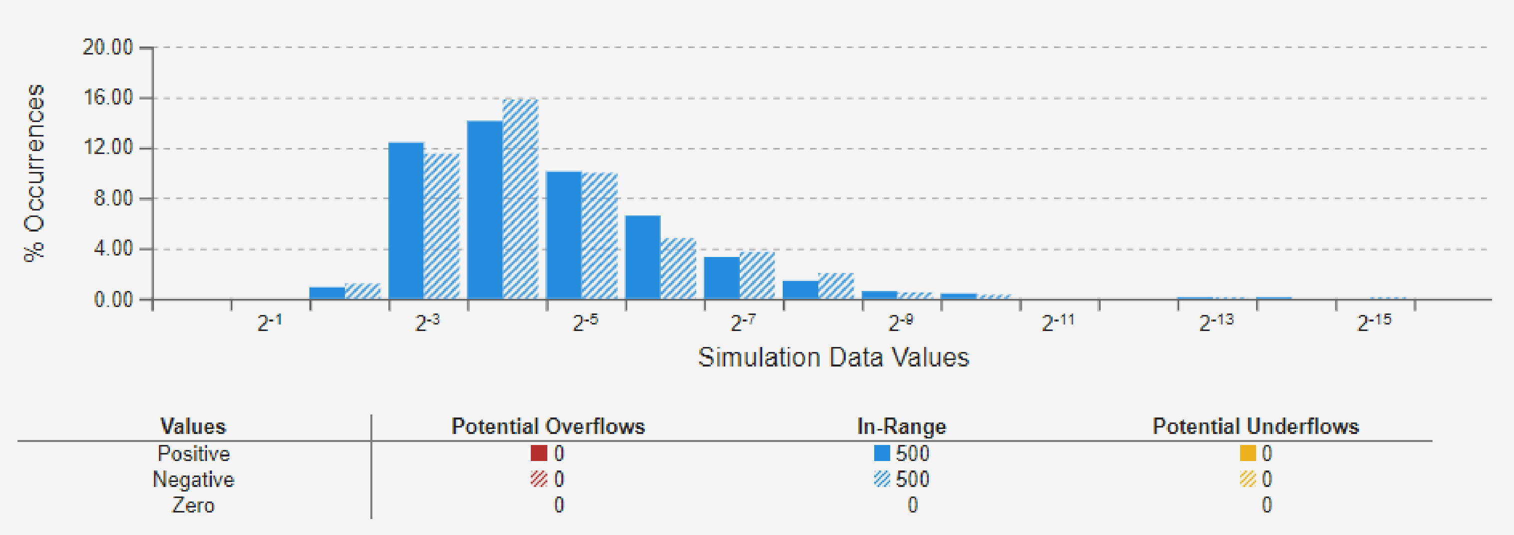
- scopeDemo.mlx
- Delay register variable ranges
- If one needs to keep delay register values below 1, probably latter solution would work

No scaling



$2^{-11} \leq |x| < 2^2$

Prescaling with s = 10



Range $2^{-15} \leq |x| < 2^{-1}$

3.3. Data path: fixed-point formats

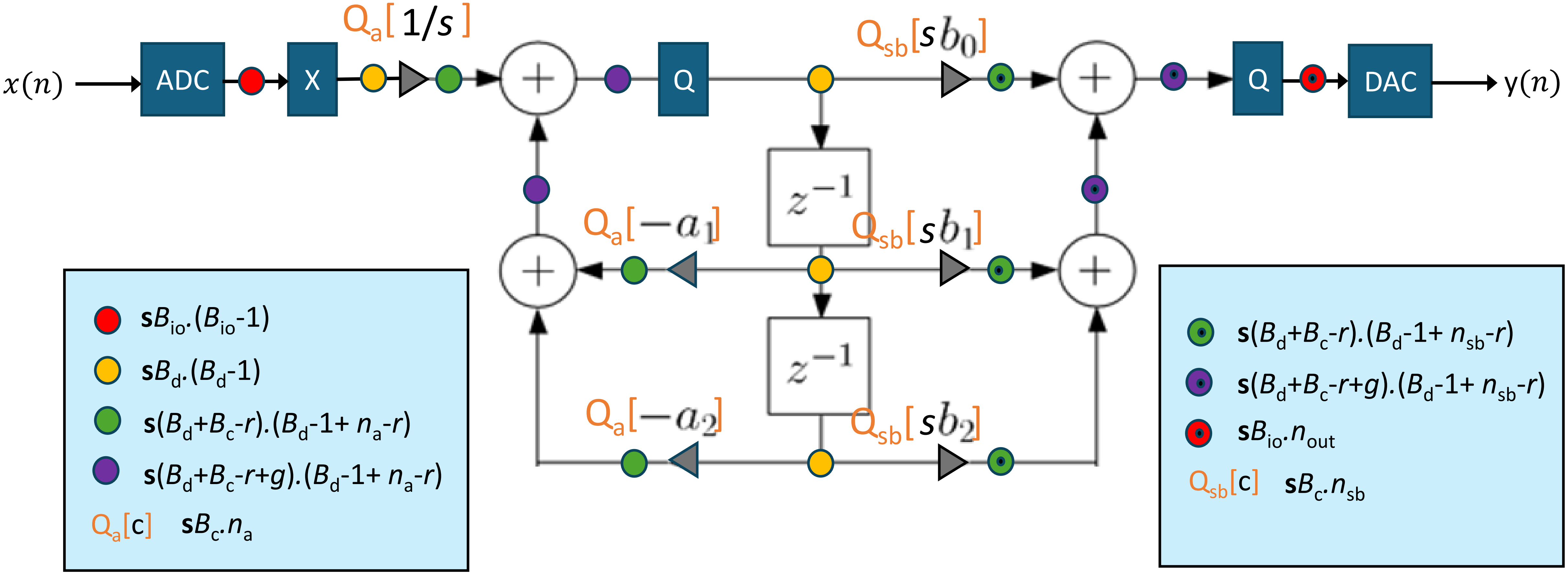
- B_{io}
- B_d
- B_d+B_c-r (mply out)
- B_d+B_c-r+g (add out)

Q[c] Quantized gain c (B_c)

- In previous, only **word lengths** of DSP platform were given
- We must make decisions related to the **fraction lengths**
  - In some cases, we can infer them
  - Some decisions require simulation studies

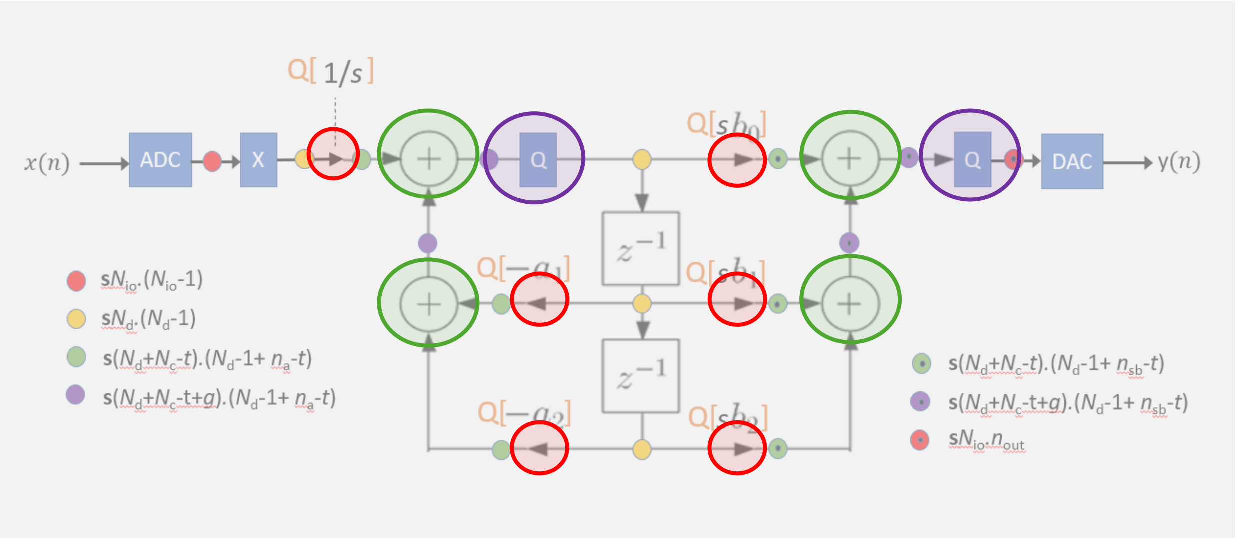
- Fraction lengths are defined for
  1) <u>Input</u>: assuming range (-1,+1) so the fraction length is B_{io}**-1**
  2) <u>Output</u>: set by user, n_{out} depends on the observed gain of the filter
  3) <u>Delay-line</u>: pre-scaling => range (-1,+1) assumed so the fraction length is B_d**-1**
  4) <u>Coefficients</u>: separate choice of fraction length (n_a, n_{sb}) for feedback/-forward portions
     - Guided by the range of coefficients {$1/s$, a_1, a_2} and {sb_0, sb_1, sb_2}
  5) <u>Multiplier output</u>: choice of fraction length is guided by inputs and amount of LSB reduction r
  6) <u>Adder output</u>: follows from the multiplier output format
     - one or two guard bits g may be added, but they do not affect the fraction length

Signal flow with fixed-point **sp.n** formats



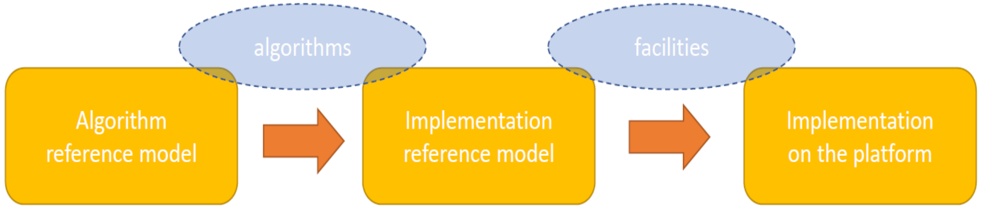
Summary of noise sources

- Multiplications
  - Source of round-off errors
  - Only if $r > 0$, that is, if we drop LSBs by truncation (or some other rounding)
- Additions
  - Potential source of overflow errors
- Conversions
  - Q blocks before the delay line and DAC
  - round-off errors, overflow errors possible
- In addition, coefficient quantizations have effect on DSP noise (change response from optimal)



3.4. Simulator for experiments

- To experiment with the implementation outlined in the preceeding, some Matlab code were written, utilizing Fixed Point Toolbox
  - *The code, **SOSsimulator.m**, is provided in zip package*
  - Example of an **implementation reference model**




```
% SOSsimulator   Simulate a second-order section
%
% << Syntax >>
%    h = SOSsimulator(b,a,sp,[Bi,Bc,Bd],r,g,no,mround,qround)
%
% << Arguments >>
%    b          Numerator coefficients : [b(0),b(1),b(2)]
%    a          Denominator coefficients : [a(0),a(1),a(2)]
%    sp         Scaling factor : computation method ('L1t', 'L2t', 'Lif') or a numeric value
%               - note: coefficients "b" are multiplied by this factor, input by 1/sp
%    Bi,Bc,Bd   Word length used for I/O, coefficients and delay memories
%               - multiplier outputs have word length Bc+Bd
%               - adder outputs have word length Bc+Bd+g
%    r          Number of bits discarded at multiplier output
%               - if r > 0, there will be round-off errors in multiplications
%    g          Number of accumulator guard bits (0-2)
%               - if necessary, can be used to prevent overflows in additions
%    no         Number of fraction bits used for output
%    mround     Multiplier rounding method : 'floor', 'nearest', 'round', 'zero' or 'convergent'
%               - see fimath RoundingMethod for meaning
%               - note: 'floor' is same as truncation, which would be easiest to implement in HW
%    qround     Rounding method for word length reduction (Q) blocks
%    h          Output: structure of access function handles
```

SOSsimulator.m 's procedure for setting up a fixed-point implementation for SOS filter

1. Quantize the feedback coefficients

* Design parameter: word length used for coefficients (p_c)
* Determine the fraction length under the assumption that all feedback coefficients must use the same format
* Quantize coefficients a_1 and a_2

2. Set the scaling factor

* The transfer function analyzed is $H'(z) = \frac{1}{1+a_1 z^{-1}+a_2 z^{-2}}$.
* Design parameter: Select one of the norms of $H'(z)$ or provide some other value
* Quantize pre-scaling factor 1/s using the format used for feedback coefficients

3. Scale and quantize the feedforward coefficients

* Design parameter: word length used for coefficients (p_c)
* To compensate for pre-scaling, multiply feedforward coefficients by inverse of the quantized pre-scaling factor
* Determine the fraction length under the assumption that all scaled feedforward coefficients must use the same format
* Quantize coefficients sb_0 , sb_1 and sb_2

4. Determine fixed-point formats on the data paths

* Design parameters: p_{io}, p_d, p_c, t, g, and n_o
* Input format is $sp_{io}.(p_{io}-1)$, where p_{io} is the word length for input/output
* Delay memory format is $sp_d.(p_d-1)$, where p_d is the word length for internal memory
* Multiplier output formats: $s(p_c+ p_d-t).(n+ p_d-1-t)$, where n depends on the portion (feedback/-forward) of the filter
* Accumulator formats: $s(p_c+ p_d-t+g).(n+ p_d-1-t)$, where g is the number of guard bits used
* Output format is $sp_{io}.n_o$, where n_o is the fraction length chosen by the user

Simulate using test signals and repeat with other design parameters, if not satisfactory

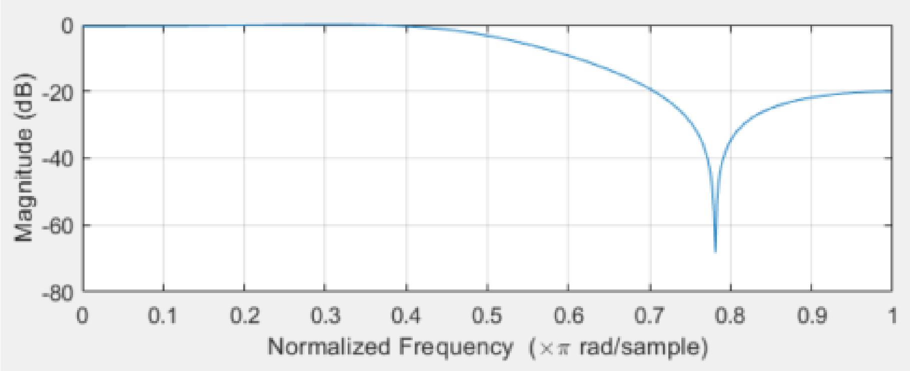
* Simulation interests: Overflows and underflows at acceptable level? SNR acceptable? Resource use optimal?
* Compare to floating-point!

3.5. Experiments*

In the experiments, a low-pass SOS filter is constructed with the command

>> [b,a] = ellip(2,0.5,20,0.4);

Frequency response of the filter on the right. Passband cutoff is at 0.4 x Fs / 2, stopband cutoff at 0.7 x Fs/2, and stopband attenuation 20 dB.

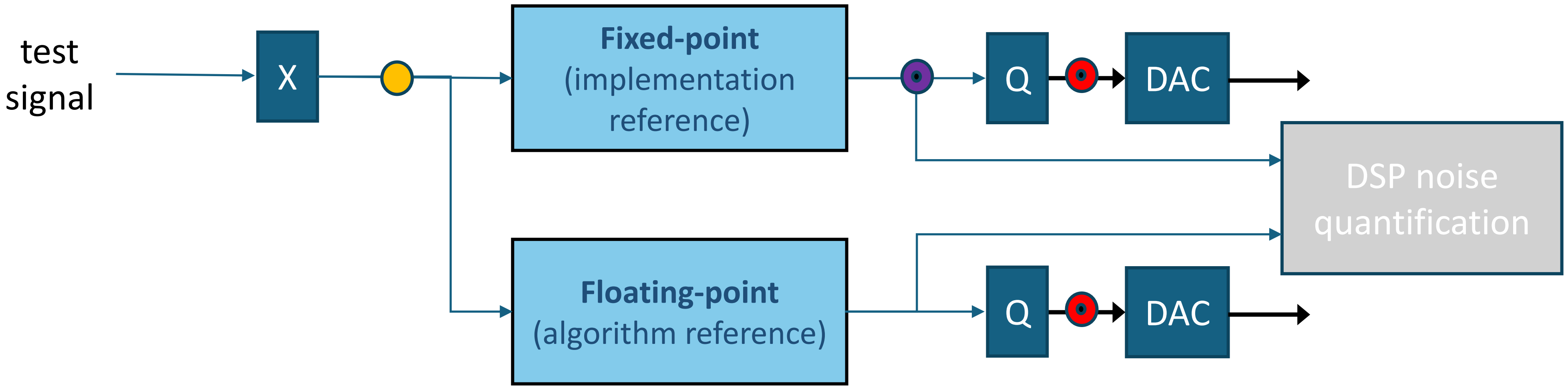


Three experiments:
Experiment 1 – truncation versus rounding
Experiment 2 – required precision of multiplications
Experiment 3 – error correlations with periodic signals

Fixed-point constructs are compared to the full-precision floating-point filter, which can be simulated with Matlab's **filter** command.

Evaluation setup (1)

Following setup is used in order to make output comparison fair



use same
quantized
input

take outputs that work as the
input to final quantization
(before DAC)

Evaluation setup (2)

- **Sine wave test signal**
  - Varying frequency within the range (0,Fs/2)

$$x(n) = \sin(2\pi f n + r)$$

where $f \in (0, 1/2)$ and r is a small random phase shift

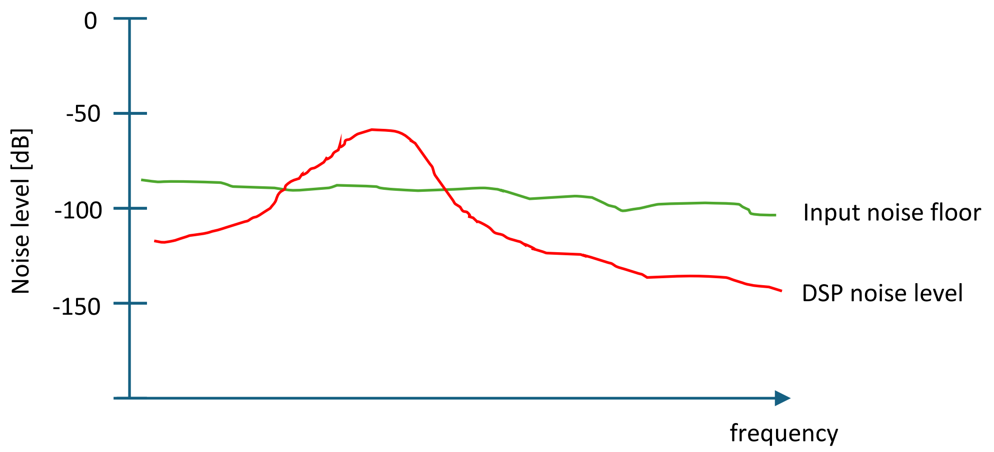
- **ADC noise at output**
  - Evaluated in order to compare the DSP noise to ADC noise
  - Model-based formula used

$$\sigma_{OA}^2 = \sigma_A^2 \sum_{k=0}^{\infty} h^2(k) \qquad \sigma_A^2 = \frac{2^{-2B}}{3}$$

where $h(k)$ is impulse response of the filter and B is the ADC word length.

- Plotting ratio $\dfrac{\text{noise power}}{\text{signal power}}$ [dB]

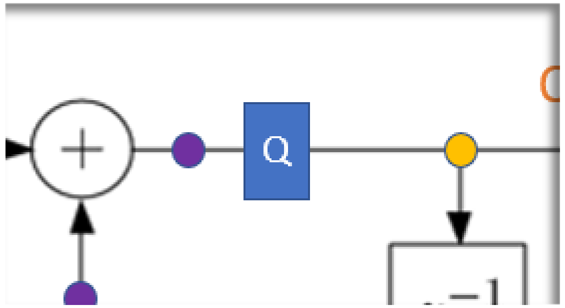
- Recall earlier noise floor figure:



Code for following experiments, **SOSexperiments.m, is in zip package*

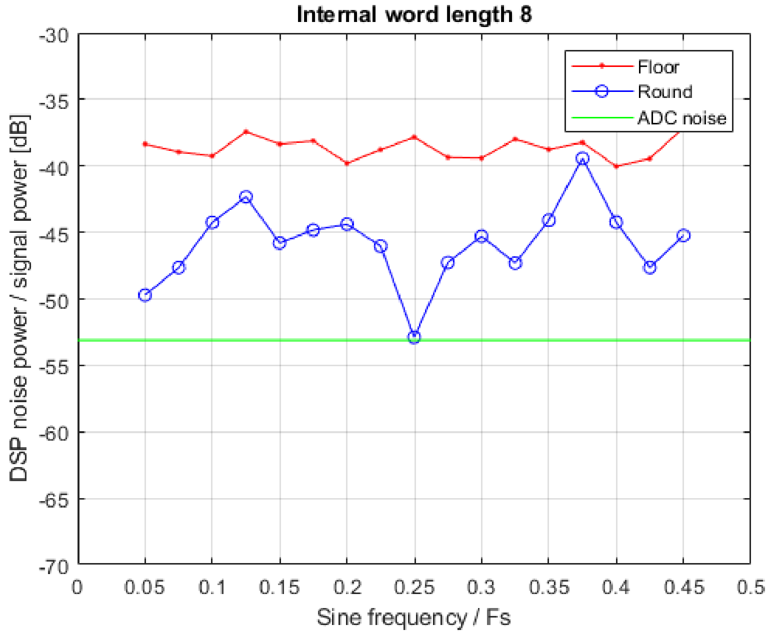
Experiment 1 – truncation study

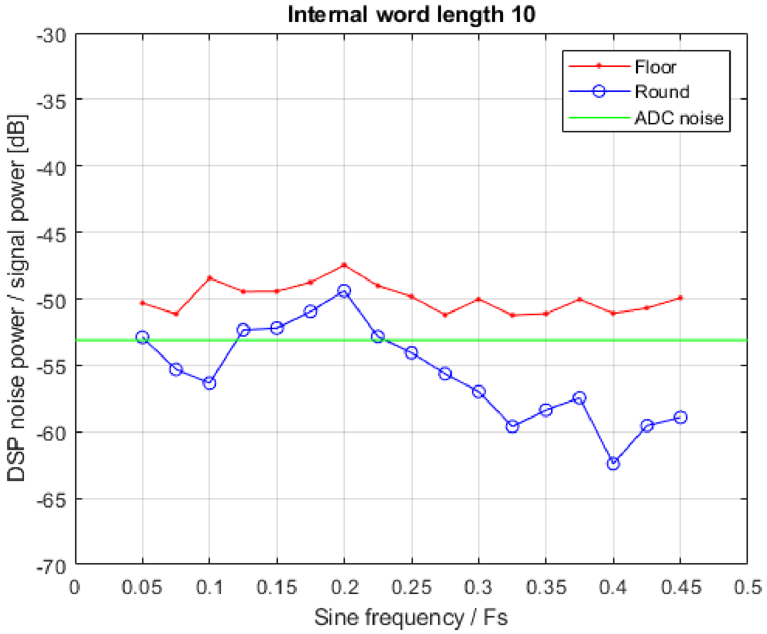
- Quantization by truncation (= floor) is biased. Does it show up?
- Small experiment, where we compare truncation to rounding
- 8-bit ADC, 8-bit coefficients
- **delay register word length** (N_d) varied from 8 to 12 bits
- Result:

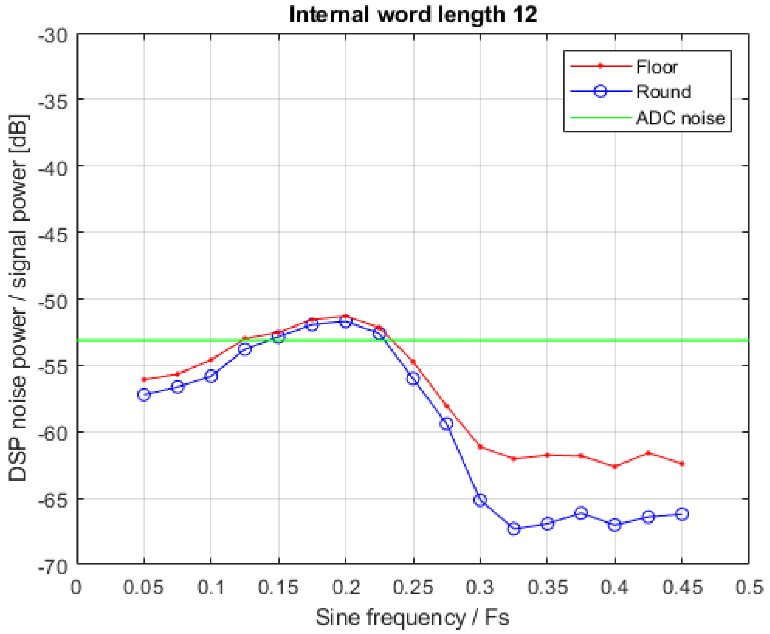


Truncation/rounding point

$$N_d = 8$$



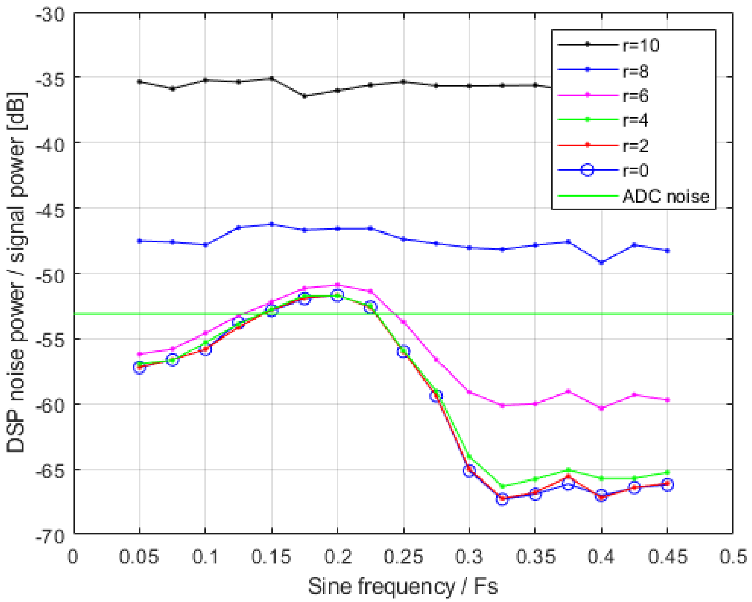




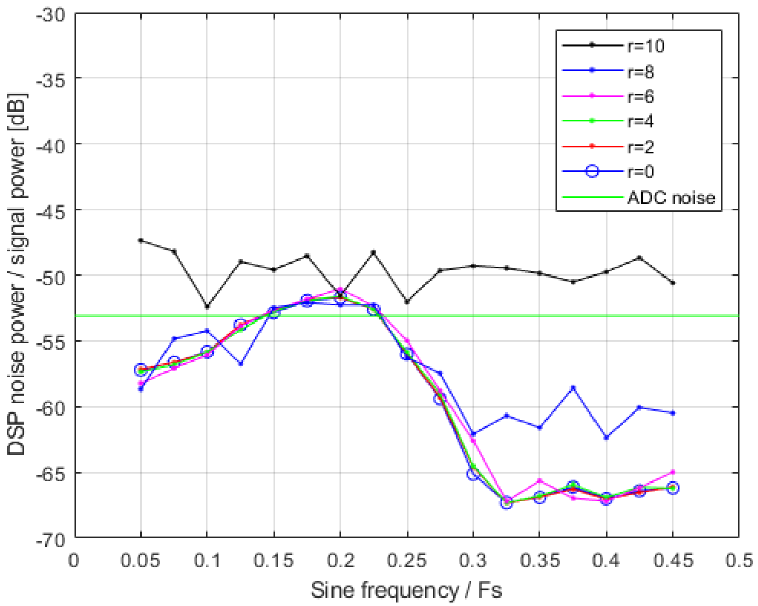
Experiment 2 – precision of multiplications

- How much can we discard LSB bits in multiplication?
- Experiment where we increase the parameter ***r***, which controls discarding
- Delay register word length set to 12 bits, Q operation uses rounding
- Multiplier output word length (20 – ***r***), testing for ***r*** = 0, 2, 4, 6, 8, 10 (floor / round)
- Result:

Floor:



Round:

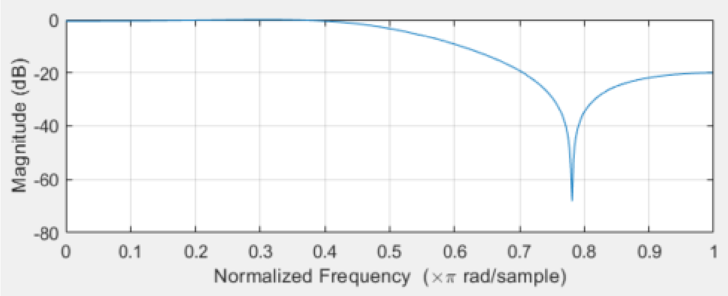


More bits can be discarded in the case of rounding. Compare results with r = 8

Experiment 3 – error correlations

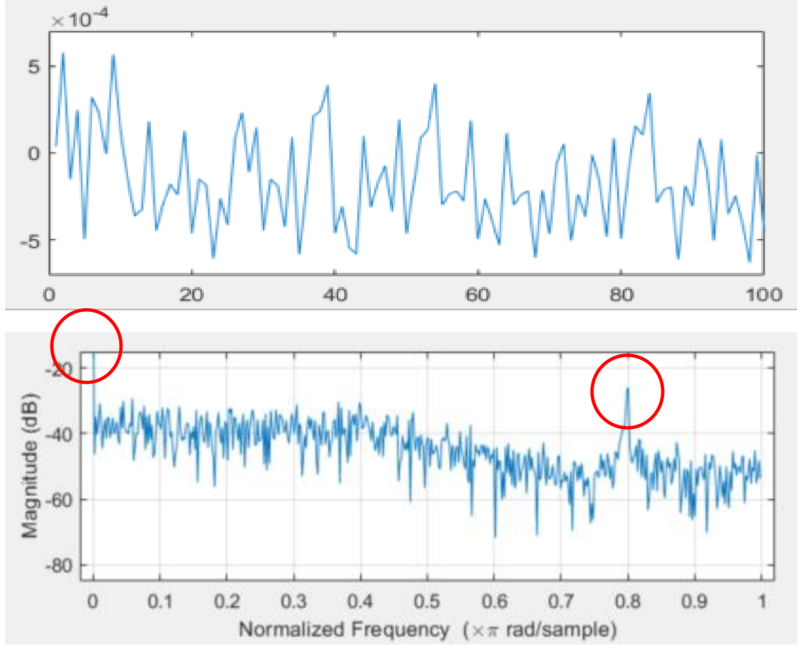
- DSP error spectra of periodic and aperiodic signals compared
- The test signal $x(n) = \sin(2\pi f n + r)$

  where $f \in (0, 1/2)$ and r is a phase shift
- Comparison for sine frequency $f = 0.4$:

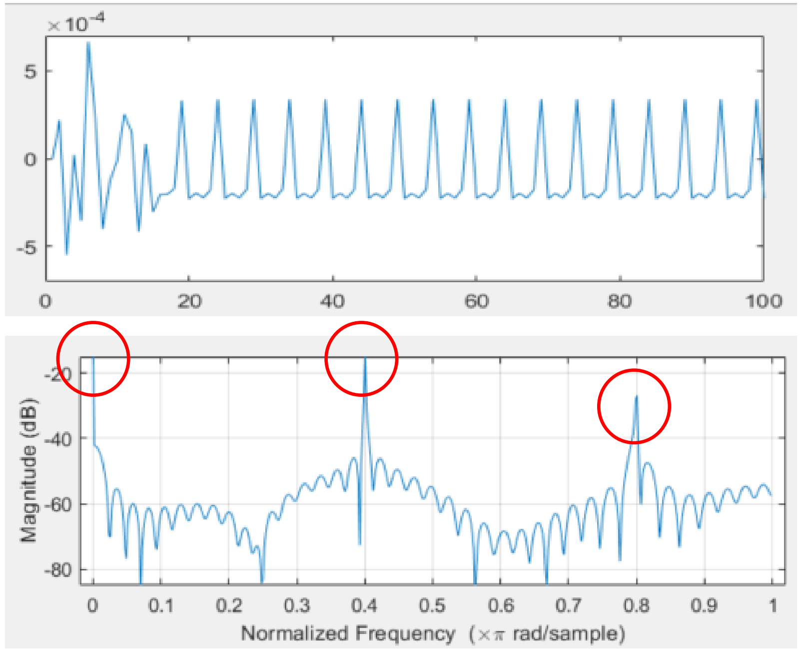


Response of the filter

With **random** phase shift *r*



With **zero** phase shift *r (r=0)*

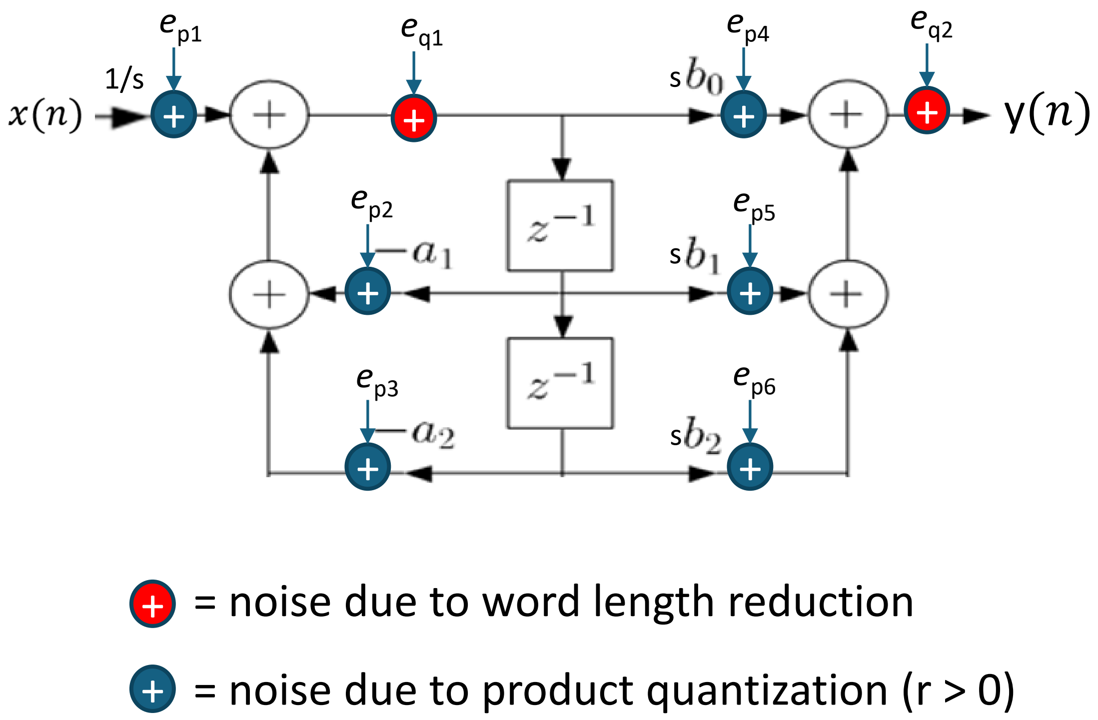


Error correlations cause peaks to the error spectra

Error at zero frequency : dc bias in output

3.6. Model-based analysis*

- The alternative for simulation-based approach was the analytic approach
- In analysis, noise sources have certain assumed statistics (spectrally white, independent)
- Effect at the filter output computed
  - the impulse response from the noise source to the output must be determined
  - Uncorrelated => effects of noise sources summed up
- Note 1: moving an added input across a sum has no effect
  - Direct effect of e_{p4}, e_{p5}, e_{p6}, e_{q2} at output
- Note 2: many noise sources have the same transfer function
  - e_{p1}, e_{p2}, e_{p3}, e_{q1} have the same path through the filter
- See Ifeachor & Jervis (2002) for in-depth discussion



Summary

- A/D conversion
  - Tuning – Finding balance between quantization and clipping noises
  - Noise floor – Setting target for quality of signal processing
- IIR filter coefficient quantization
  - Filter stability must be addressed
  - Need to consider structure of computation
- IIR filter processing
  - Multiple fixed-point formats involved
  - Scaling of input data helps to prevent overflows in feedback sections
  - NumericTypeScope in Matlab FPT provides a tool for analyzing data collected during floating-point or fixed-point simulation