Design flows.
Matlab’s fixed-point toolbox.
Fixed-point FIR filter design.

Signal Processing Systems Fall 2025

Lecture 4 (Thursday 6.11.)

Outline

- Design flows
  - Analysis-based, Simulation-based
- Matlab’s Fixed Point Toolbox
  - Basic features
- Fixed-point FIR filter design
  - Coefficient quantization
  - Word length analysis
  - Roundoff error study (tool used in DT2 problem 3)

I. Design flows

Design flows

- The first step in the development of a signal processing system is the development and selection of the algorithms:
  - design of a **floating-point reference model**
  - exploration of the numerical properties of the algorithm using the floating-point model
  - the model serves in evaluation of the final implementation quality

Algorithm reference model

- Then, a model of the system implementation should be developed
- Details of the model reflect the primitives of the hardware
  - Primitive = split ALUs, CORDIC processors, saturation/wrapping arithmetic, etc.
  - System operation is mimicked in the **fixed-point reference model**
  - An accurate model may act as a bit-exact verification model for final implementation.
  - Sufficient word lengths (range, precision) for each number object must be determined

Implementation reference model

- Two basic approaches for conversion work
  - **Analytic approach**. Favored by algorithm designers who do not have complete understanding of the hardware, or when one is looking for completely novel implementation options.
  - **Simulation approach**. Favored by HW designers who do not like mathematics of the models and already have an idea on the characteristics of the potential implementation platform and tool chain.

Analytic approach

- Design outline:
  - Algorithm design via mathematical modelling
  - Analysis of the coefficient quantization effects
  - Analysis of the rounding and scaling effects
  - Selection of the word lengths
  - Simulation to verify the design results

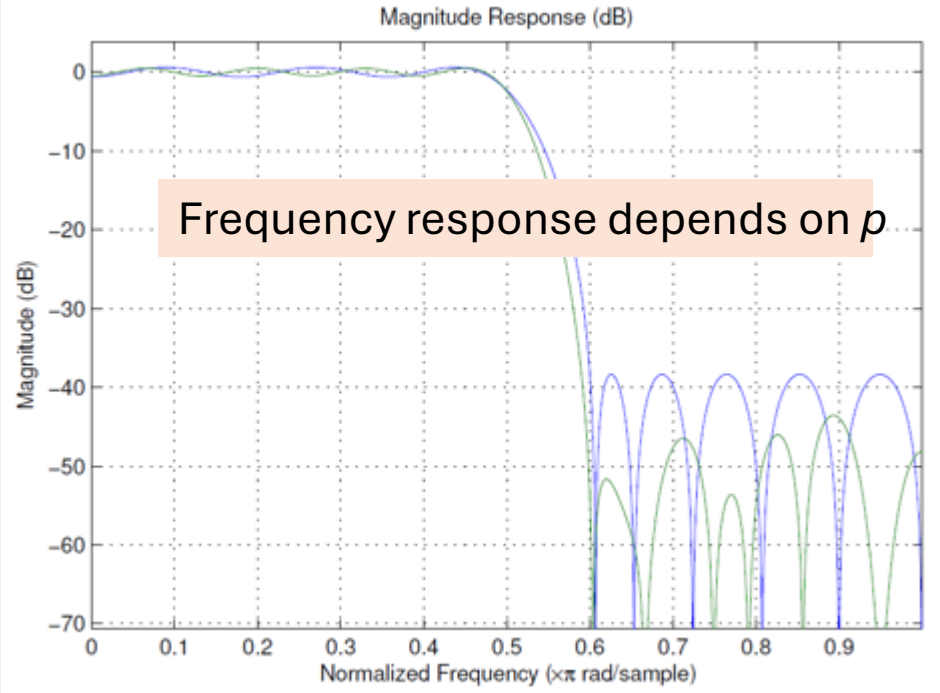
Analytic approach

- Design outline:
  - Algorithm design via mathematical modelling
  - Analysis of the coefficient quantization effects
  - Analysis of the rounding and scaling effects
  - Selection of the word lengths
  - Simulation to verify the design results
- Example of fixed-point FIR filter analysis on the right
  - (1) coefficient quantization analysis gives an upper bound for word length *p*

(1) Coefficient quantization: one must check that the specifications are met, and in the case of IIR filtering that the filter remains stable

Analysis example:

- assume FIR filter with N coefficients
- $sp.(p-1)$ format for the coefficients is to be selected
- quantization introduces a parallel filter: $H_q(z) = H(z) + E(z)$
- max. quantization error for the format is (unit-of-last-place) ulp/2. ulp=$2^{-(p-1)}$. Thus, $|e(n)| \leq 2^{-p}$.
- assuming the worst-case error for all coefficients, we get $E(\omega) = 2^{-p}N$
- maximum stop-band attenuation is therefore bounded by $20\log_{10}(2^{-p}N)$
- having a specification available, we get an upper bound for p



Analytic approach

- Design outline:
  - Algorithm design via mathematical modelling
  - Analysis of the coefficient quantization effects
  - Analysis of the rounding and scaling effects
  - Selection of the word lengths
  - Simulation to verify the design results
- Example of fixed-point FIR filter analysis on the right
  - (1) coefficient quantization analysis gives an upper bound for word length *p*
  - (2) roundoff noise analysis – a problem is that error sources can be correlated

(1) Coefficient quantization: one must check that the specifications are met, and in the case of IIR filtering that the filter remains stable

Analysis example:

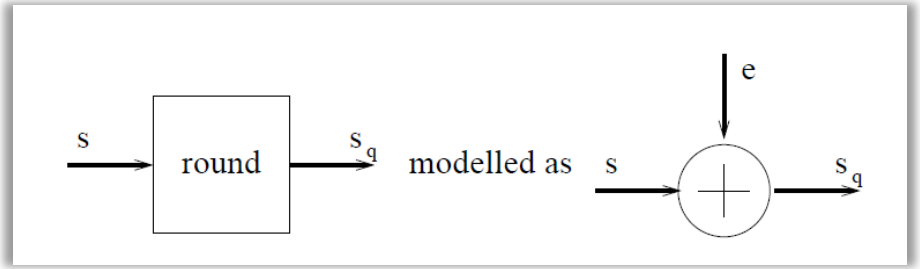
- assume FIR filter with N coefficients
- $sp.(p-1)$ format for the coefficients is to be selected
- quantization introduces a parallel filter: $H_q(z) = H(z) + E(z)$
- max. quantization error for the format is (unit-of-last-place) ulp/2. ulp=$2^{-(p-1)}$. Thus, $|e(n)| \leq 2^{-p}$.
- assuming the worst-case error for all coefficients, we get $E(\omega) = 2^{-p}N$
- maximum stop-band attenuation is therefore bounded by $20\log_{10}(2^{-p}N)$
- having a specification available, we get an upper bound for p

(2) Effect of the roundoff noise - one can think in terms of the impulse response $h(n)$ from the round-off location to the output

- if the quantization interval is q then the signal quantization noise power is $\sigma_{ir}^2 = q^2/12$
- the effect at output is

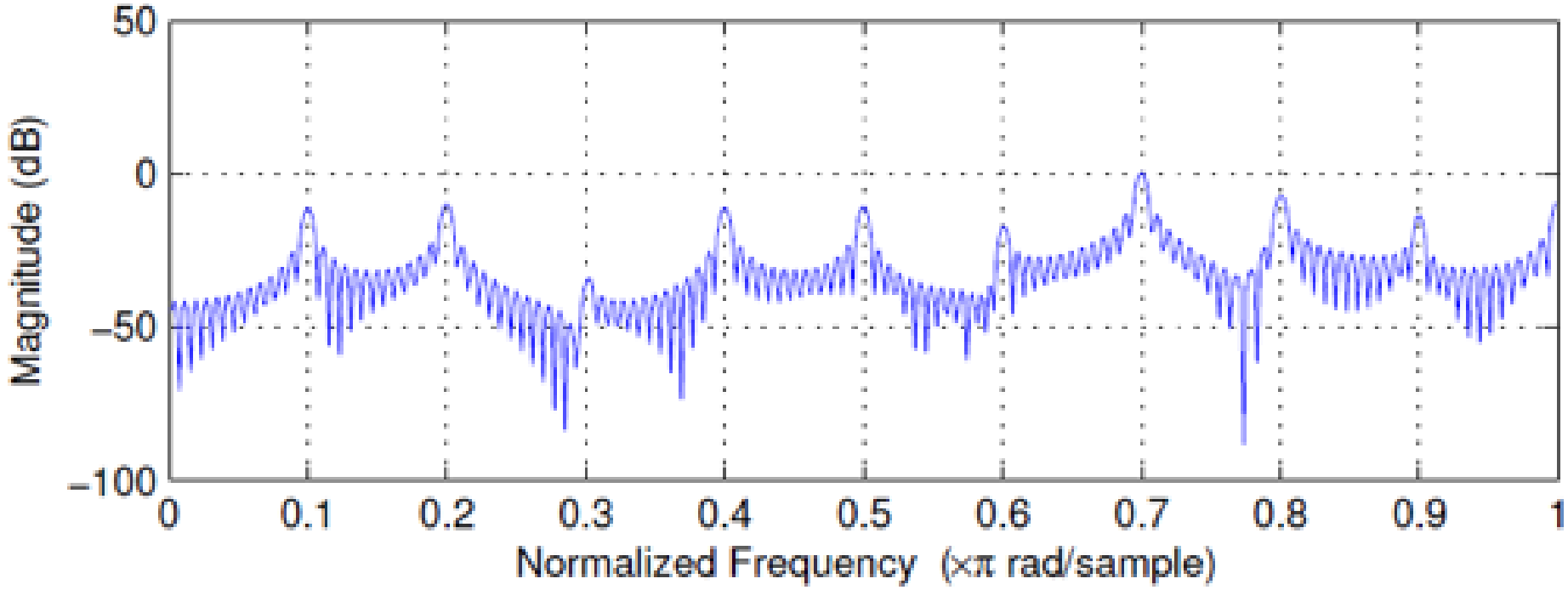
$$\sigma_{or}^2 = \sigma_{ir}^2 \sum_{n=0}^{\infty} h^2(n)$$

- if noises are assumed uncorrelated the effects of round-off locations sum up
- however, the noises can be correlated. E.g. in telecommunications the signals are often periodic
- such correlation leads to peaking in the noise spectrum



What can happen when roundoff errors are correlated?

Spectrum of error signal contains peaks



Mathematical analysis cannot see this. Simulation may reveal

How to solve the problem: increase word lengths, implement stochastic rounding ..

Analytic approach

- Design outline:
  - Algorithm design via mathematical modelling
  - Analysis of the coefficient quantization effects
  - Analysis of the rounding and scaling effects
  - Selection of the word lengths
  - Simulation to verify the design results
- Example of fixed-point filter analysis on the right
  - (1) coefficient quantization analysis gives an upper bound for word length *p*
  - (2) roundoff noise analysis – problem is that error sources can be correlated [illustrated later]
  - (3) overflow study is necessary with feedback structures (IIR filter)
- Analysis might be used in the beginning of design – simulations cannot be avoided

(1) Coefficient quantization: one must check that the specifications are met, and in the case of IIR filtering that the filter remains stable

Analysis example:

- assume FIR filter with N coefficients

- $sp.(p-1)$ format for the coefficients is to be selected

- quantization introduces a parallel filter: $H_q(z) = H(z) + E(z)$

- max. quantization error for the format is (unit-of-last-place) ulp/2. ulp=$2^{-(p-1)}$. Thus, $|e(n)| \leq 2^{-p}$.

- assuming the worst-case error for all coefficients, we get $E(\omega) = 2^{-p}N$

- maximum stop-band attenuation is therefore bounded by $20\log_{10}(2^{-p}N)$

- having a specification available, we get an upper bound for p

(2) Effect of the roundoff noise - one can think in terms of the impulse response $h(n)$ from the round-off location to the output

- if the quantization interval is q then the signal quantization noise power is $\sigma_{ir}^2 = q^2/12$

- the effect at output is

$$\sigma_{or}^2 = \sigma_{ir}^2 \sum_{n=0}^{\infty} h^2(n)$$

- if noises are assumed uncorrelated the effects of round-off locations sum up

- however, the noises can be correlated. E.g. in telecommunications the signals are often periodic

- such correlation leads to peaking in the noise spectrum

(3) Overflows in IIR structures

- the adders on the feedback path may overflow

- use of saturation arithmetic may lead to instability, and spoil the response

- another solution is to perform input scaling to reduce its range

Note on analysis:

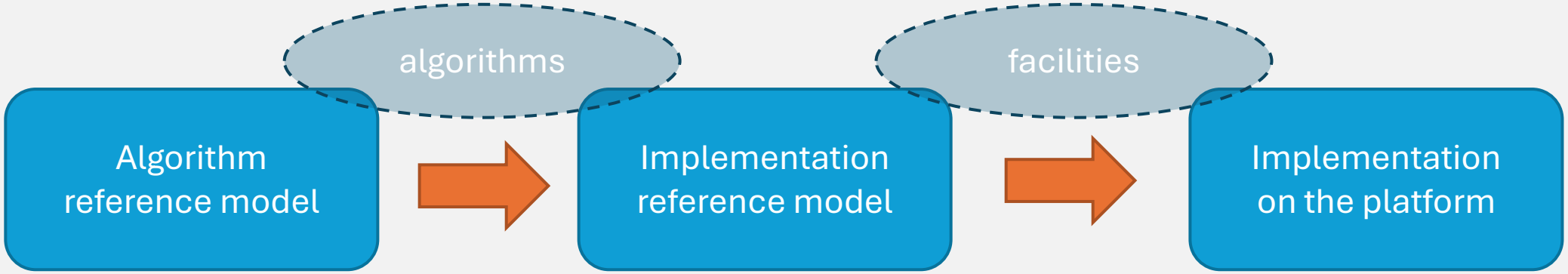
- approximation formulas may only provide quick checks and starting points for the design

- more detailed derivations needed, or simulation-based approach taken

2. Simulation approach

- Outline of fixed-point design process:
  - Algorithm design using full floating-point precision
  - Conversion into a fixed-point model (or custom floating-point)
  - Simulation used to determine
    - coefficient quantization effects
    - word length effects on precision
    - overflow problem detection: determination of the max values via data logging
  - Reduction of the word lengths, adaptation of the algorithm for the implementation platform
- Simulation can be based on writing C/C++ code
- Alternative: using tools like Matlab's Fixed Point Toolbox
  - Matlab's Simulink environment also contains features that can be used to implement fixed-point simulations
- One would like to model the algorithm with a generic package, which can be instantiated with any fixed-point/floating-point types
  - Comparison of type alternatives by simulation
  - Some support for this in Matlab and Simulink

Steps in simulation/tool –based design



Basic algorithm description, accurate simulation using ”full resources” (e.g. double precision floating-point in Matlab).

Description of the algorithm implementation on a specific platform, reflecting available **primitives** on it. Simulation on computer, compared to algorithm reference model.

Answering question: Will the performance of the implementation be sufficiently close to the algorithm reference model?

Tools: C/C++ programming, Matlab Fixed-Point Toolbox, Simulink, …

Implementation work using the tools associated with the platform. Testing verifies that the result corresponds to the implementation reference model (bit-exactly)

”Primitives” includes things like

- real number formats: fixed-point, custom floating-point, word lengths …
- functional units available for arithmetics
- possibilities for tuning word lengths
- etc.

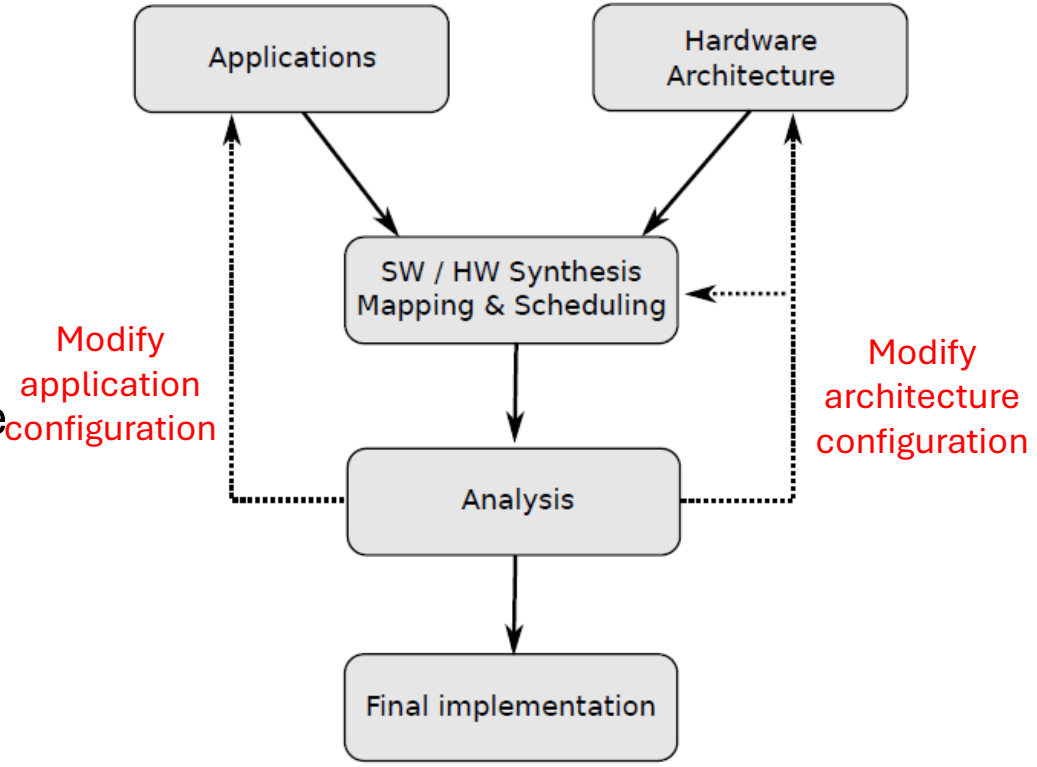
Two kinds of matching

- Matching algorithm to architecture
  - **Available facilities to be used are given** (e.g. going to use some specific signal processor)
  - Architecture is given: thing to do is to select the best algorithm for it. For example, algorithm A1 might be good for sequential computation, A2 good for parallel SIMD implementation, A3 good for SIMT etc.
  - Recall matrix inversion algorithms from last lecture example: authors used divide-and-conquer for floating-point, matrix decomposition techniques for fixed-point (issue: numerical stability)
- Matching architecture to algorithm
  - Case where an application-specific processor (ASP) is designed (**HW/SW codesign**)
  - Options for choosing and tuning the FUs, ALUs, interconnections, word lengths …
  - One can also think that algorithm options are open here and the goal is to find the **best architecture-algorithm combination**
  - Example from the previous lecture: support for complex-valued arithmetics included in the cores of MIMO-OFDM detection (PAPER-2,3)

Advanced tool-based approach:

Decoupling application and architecture modelling

- **Y-chart system design** provides another viewpoint for mapping application to implementation
- The application is described using a model that captures **the functionality of the application without reference to a possible hardware architecture** which is run by the software.
- The idea is to make software and hardware architecture design independent from each other
  - allows the use of high-level abstractions to describe the application
  - no need to consider timing, HW/SW partitioning or exclusion of any interesting implementation options in early design phases
- Synthesis of solutions: formal models and transformations needed
  - Various formal dataflow semantics like synchronous data flow (SDF) semantics may act as a basis



From: Ilkka Hautala (PhD thesis 2019) Acta Universitatis Ouluensis C 714

II. Matlab Fixed-Point Toolbox

Matlab's Fixed-Point Toolbox (FPT)

- Fixed-Point Designer
- A tool for building implementation reference models
- As an example, modelling fixed-point implementation of a FIR filter
- Introduction of features first …

```
>> help fixedpoint
  Fixed-Point Designer
  Version 24.2 (R2024b) 21-Jun-2024

  MATLAB Functionality
    Data and Data Types      - Fixed-Point Data and Data Types
    Math                     - Fixed-Point Math
    Float-to-Fixed Workflow  - Float-to-Fixed Workflow
    Code Acceleration        - Fixed-Point on Simulink Platform

  Simulink Functionality
    Data Types and Math      - Fixed-Point Data Types and Math
    Float-to-Fixed Workflow  - Float-to-Fixed Workflow

    Fixed-Point Designer Documentation
```

- Tools for Python?
  - **deModel**, see http://www.dilloneng.com/demodel.html
  - **fxpmath**: https://github.com/francof2a/fxpmath

Strengths of FPT

- Allows defining the word lengths and the positions of the binary point in an arbitrary manner, for inputs and outputs
- Includes possibility to override fixed point definitions, returning back to floating point
  - easy to check if the base algorithm is still the same and the deviations from the specification are just due to the fixed-point conversion process
- Enables data logging to record minimum and maximum values
  - simplifies the tuning of the word lengths
- Compatible with other Matlab tools like Signal Processing Toolbox and Simulink

Tools in FPT

- The key tools (ie. FPT functions) are
  - **fi**: instantiation of a fixed-point numeric object
  - **fimath**: construction of a fixed-point math object
    - used to describe the behavior of additions and multiplications associated with the numeric objects, how overflows are handled, how rounding is done
  - **fipref**: sets fixed-point preferences for the current Matlab session
    - display properties, data logging properties, data type override properties
- Other useful tools are
  - **numerictype**: instantiation of an object which encapsulates numeric type information (e.g. can be used as a parameter for **fi**)
  - **quantizer**: construction of a quantizer object, can be used with **quantize** command
- Using these tools, one can build models of fixed-point computation

Note: FPT uses data abstractions

- FPT does not define objects like 16-bit multipliers
  - This would be a functional abstraction
- Instead, **operator overloading** is used
  - Provides specific meaning for operations
- For example, **fimath** assigned to **fi** objects A and B defines what kind of output is produced by * and + in expressions A * B and A + B

FPT: fi object

- Can be used to instantiate both unsigned and signed (two's complement) fixed point numbers
- The command for defining numbers with binary point scaling is

```
fixed_v = fi(v, s, p, n, F)
```

where

v is a scalar, vector or matrix containing the numbers to be mapped,

s defines the signedness (1=signed, 0=unsigned),

p is the word length,

n is the fraction length

F is a fimath object, which assigns computational features to the object

- Objects with slope-bias scaling can be instantiated with

```
fixed_v = fi(v, s, p, slope, bias, F)
```

- Instantiation can also use separate definition of the fixed-point type:

```
T = numerictype(s, p, n)
fixed_v = fi(v, T, F)
```

Fi object – examples (1)

- `h = exp(1.0)`

```
h = 2.7183
```

- `h_fixed = fi(h,1,8,3)`

```
h_fixed = 2.75
```

- `h_fixed = 2.88`

```
h_fixed = 2.88
```

- `h_fixed = fi(h,1,8,3)`

```
h = 2.75
```

- **`h_fixed(1)`** `= 2.88`

```
h_fixed = 2.875
```

If you refer to the items of fi object using (), then it stays as a fi object. Assigned values are rounded

Fi object – examples (2)

```
➢ h_fixed = fi(h,1,8)
h_fixed = 2.7188
```

Result is fi –object (s8.5) – best precision!

```
➢ h_fixed + h_fixed
ans = 5.4375
```

Full precision result: fi –object (s9.5)

```
➢ bin(h_fixed + h_fixed)
ans = '010101110'
```

Fixed-point number 0101.01110

By default, best/full precision results are obtained:
* Minimum word and fraction length without overflows and loss of precision

In fixed-point modelling, we change this behavior using **fimath** objects

```
➢ a = fi(1.1,1,8,3)
a = 1.125
```

```
➢ b = fi(1.2,1,6,4)
b = 1.1875
```

```
➢ a * b
ans = 1.3359
```

Full precision result: fi –object (s14.7) ; recall law of conservation of bits

```
➢ fi(2,1,8,6)
ans = 1.9844
```

Closest representable value 01.111111 in s8.6; we have ”saturation”

```
➢ fi(1.1,1,8,3,'RoundingMethod','Floor')
ans = 1
```

Changing the rounding method. Default method is 'Nearest'

FPT: fimath object

The command `F = fimath('PropertyName1',PropertyValue1,...)` constructs a fixed-point math object

Properties that can be set are (important ones in **bold face**):

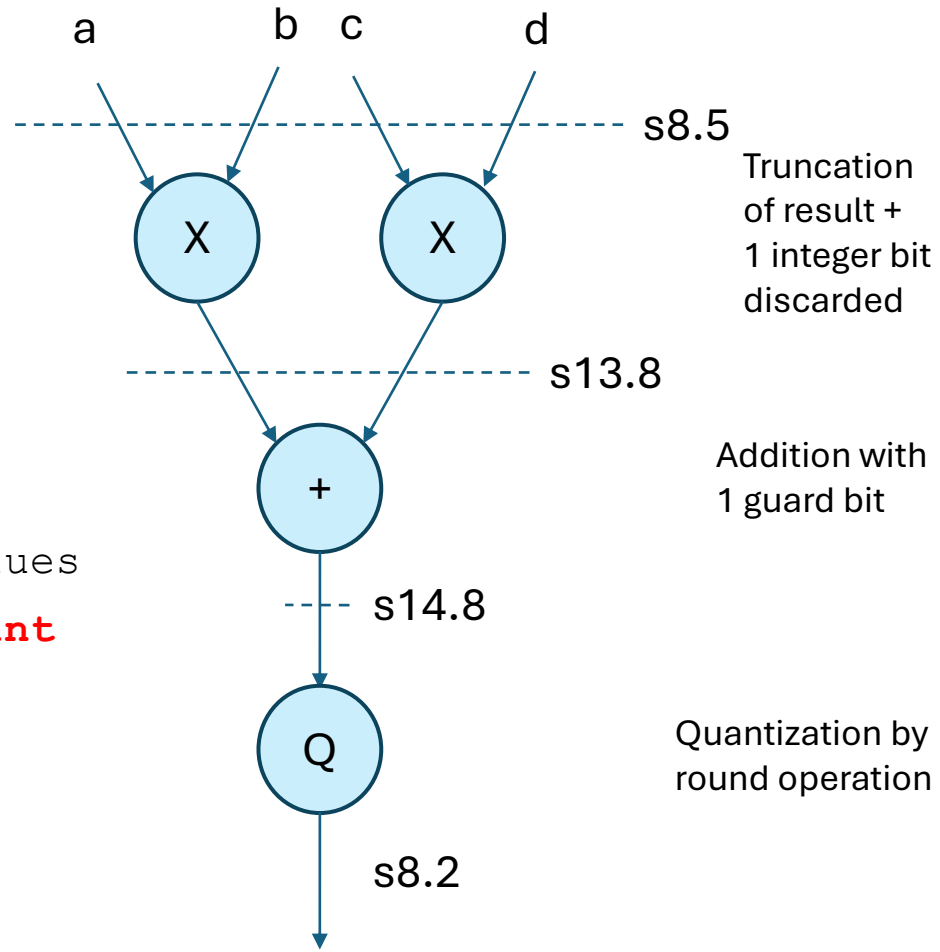
- **`CastBeforeSum`** - Whether both operands are cast to the sum data type before addition
- `MaxProductWordLength` - Maximum allowable word length for the product data type (range: 2 to 65535)
- `MaxSumWordLength` - Maximum allowable word length for the sum data type (range: 2 to 65535)
- **`OverflowAction`** - Overflow action: {Saturate, Wrap}
- **`RoundingMethod`** - Rounding method: {Ceiling, Convergent, Floor, Nearest, Round, Zero}
- `ProductBias` - Bias of the product data type
- `ProductFixedExponent` - Fixed exponent of the product data type
- **`ProductMode`** - Defines how the product data type is determined: {FullPrecision, KeepLSB, KeepMSB, **SpecifyPrecision**}
- **`ProductWordLength`** - Word length, in bits, of the product data type
- **`ProductFractionLength`** - Fraction length, in bits, of the product data type
- `ProductSlope` - Slope of the product data type
- `ProductSlopeAdjustmentFactor` - Slope adjustment factor of the product data type
- `SumBias` - Bias of the sum data type
- `SumFixedExponent` - Fixed exponent of the sum data type
- **`SumMode`** - Defines how the sum data type is determined: {FullPrecision, KeepLSB, KeepMSB, **SpecifyPrecision**}
- **`SumWordLength`** - Word length, in bits, of the sum data type when ProductMode is KeepLSB, KeepMSB, or SpecifyPrecision
- **`SumFractionLength`** - Fraction length, in bits, of the sum data type
- `SumSlope` - Slope of the sum data type
- `SumSlopeAdjustmentFactor` - Slope adjustment factor of the sum data type

Note: same OverflowAction and RoundingMethod associated with multiplication and addition.
If different methods are used in the modelled system, we must construct separate fimath objects and be careful in assignments.

Fimath - example

Modelling the computation on the right

```
F = fimath('ProductMode','SpecifyPrecision',...
           'ProductWordLength',13,...
           'ProductFractionLength',8,...
           'SumMode','SpecifyPrecision',...
           'SumWordLength',14,...
           'SumFractionLength',8,...
           'RoundMode', 'floor',...
           'OverflowMode', 'wrap');
Tin = numerictype(1,8,5);
abcd = [ 0.3125, 1.125, 3.28125, -3.125 ]; % Test values
abcd_in = fi(abcd, Tin, F); % Map values to fixed-point
sum_out = abcd_in(1)*abcd_in(2)+abcd_in(3)*abcd_in(4)
sum_out = -9.9023 (exact value: -9.90234375)
bin(sum_out)
ans = '11011000011001' (fixed point: 110110.00011001)
Tout = numerictype(1,8,2);
out = fi(sum_out, Tout, 'RoundMode', 'round')
out = -10  (fixed point: 110110.00)
```



FPT: fipref object

Command **`P = fipref`** provides access to the preferences of fixed-point simulation and display

Properties that can be set are (default values underlined):

`NumberDisplay`: {RealWorldValue, bin, dec, hex, int},

`NumericTypeDisplay`: {full, none, short} and

`FimathDisplay`: {full, none}

set how some properties of fixed-point objects are shown

`LoggingMode`: {Off, On}

When the logging mode is on, statistics about fixed-point object values and events are collected during simulation.

After simulation, they can be studied using commands minlog, maxlog, noverflows, nunderflows. Logging data associated with an object is reset using resetlog

DataTypeOverride: {ForceOff, ScaledDoubles, TrueDoubles, TrueSingles}

Allows changing of behaviour of fi objects during simulation.

Important case is when we want to collect information about the range of possible values, which helps in deciding the word and fraction lengths. Turn TrueDoubles mode on and after simulation, use minlog and maxlog commands

FPT: quantize command

- Useful when you just want to quantize some values
  - signed fixed point mode ’fixed’, unsigned mode ’ufixed’
  - also custom-precision floating-point mode ’float’
  - also quantizer object can be defined and used with this command
  - the output values have type double

- Example: quantizing values to floating-point binary16
  - recall: has 5 exponent bits

```
➢ q = quantizer('Mode','float','Format',[16,5])
q = DataMode = float
    RoundMode = floor
    Format = [16  5]
➢ quantize(q, 40002)
ans = 40000
```

More:

Take a look at the links below in Matlab

```
>> help fixedpoint
  Fixed-Point Designer
  Version 24.2 (R2024b) 21-Jun-2024

  MATLAB Functionality
    Data and Data Types      - Fixed-Point Data and Data Types
    Math                     - Fixed-Point Math
    Float-to-Fixed Workflow  - Float-to-Fixed Workflow
    Code Acceleration        - Fixed-Point on Simulink Platform

  Simulink Functionality
    Data Types and Math      - Fixed-Point Data Types and Math
    Float-to-Fixed Workflow  - Float-to-Fixed Workflow

    Fixed-Point Designer Documentation
```

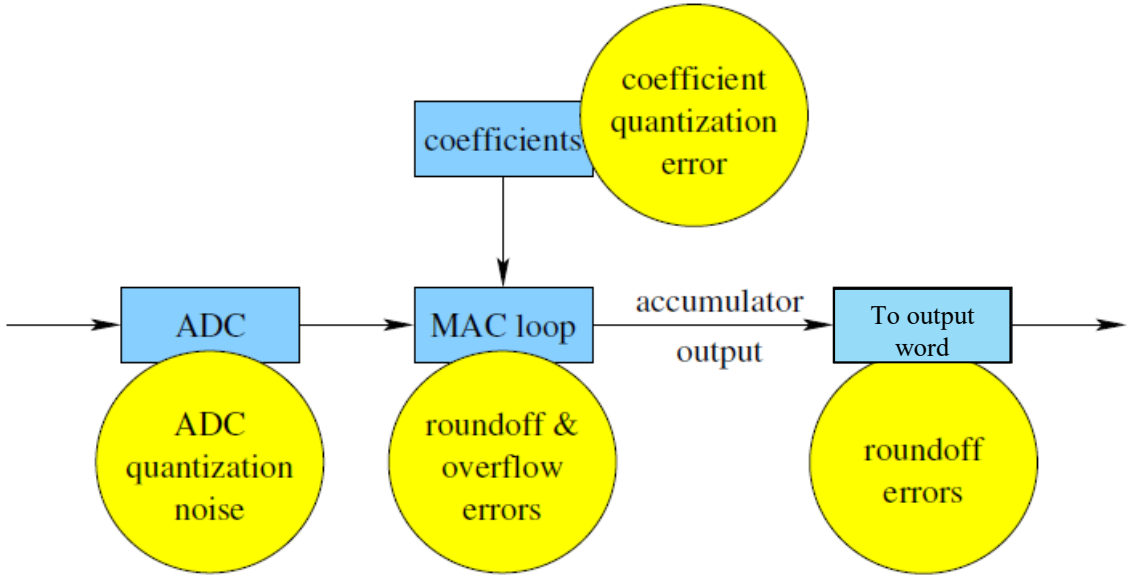
III. DT2 – Fixed-point FIR filter design

Fixed-point FIR filter design

- Design Task 2, problems 2 and 3

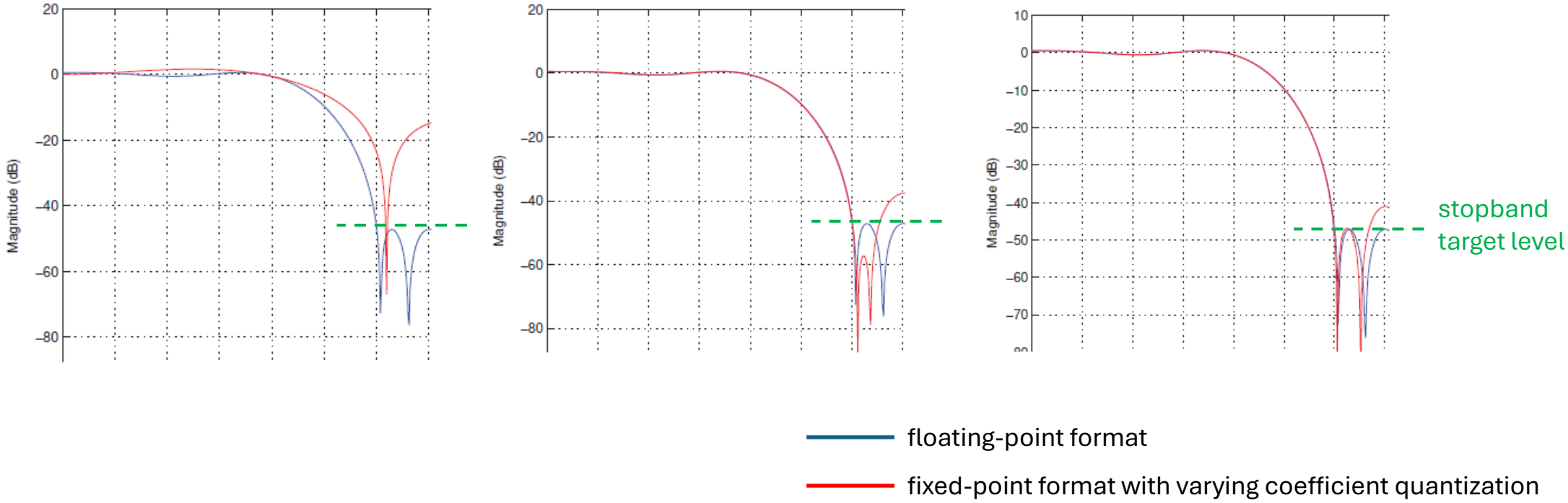
- Modelling and analyzing fixed-point FIR filter implementations
  - Finding a numerically working solution with Fixed-Point Toolbox
  - Available primitive on the assumed platform is a MAC unit

- Note: Four sources of error
  - A/D conversion
  - Coefficient quantization
  - Computation errors
  - Mapping to output (D/A input)



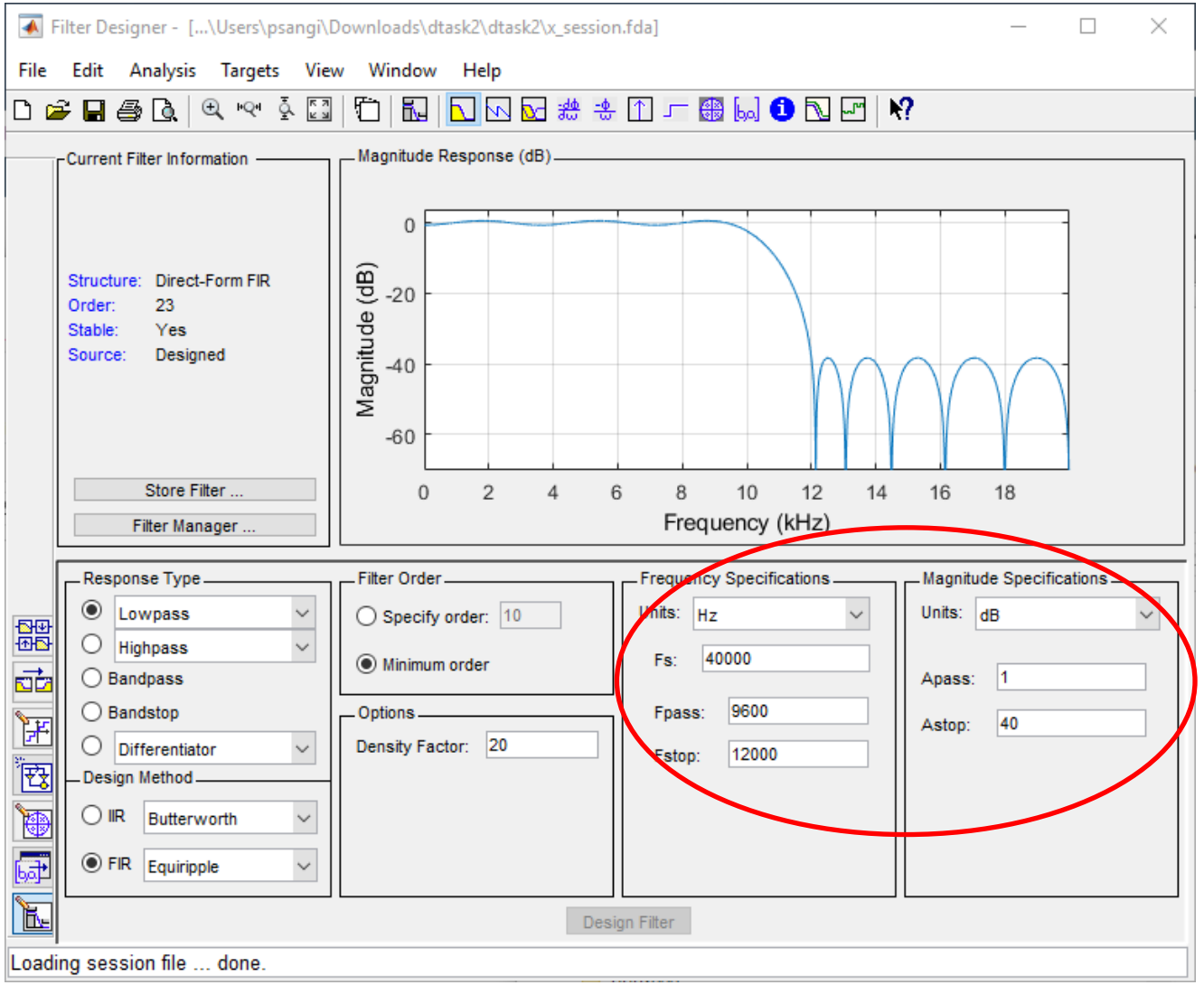
FIR filter

- No poles => no stability problems due to quantization
- Achieving sharp transition bands and sufficient depth for the stopband can be problematic



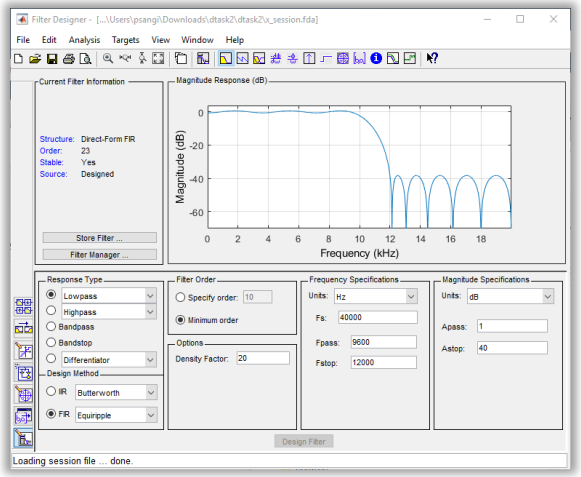
Coefficient quantization

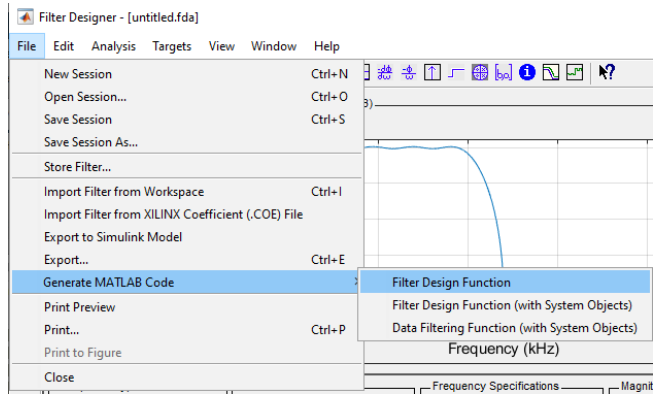
- Example: Designing a lowpass FIR filter for the requirements
  - Sample rate 40 kHz
  - Passband < 9.6 kHz, stopband > 12 kHz
  - 1 dB passband ripple
  - 40 dB stopband attenuation
- Designed using **filterDesigner** (**fdatool)**
  - data in dtask2.zip
- **Note**: the tool has features for designing realization. However, in this course we use other tools for fixed-point realization.
- Let's consider fixed-point quantization of this filter using binary-point scaling



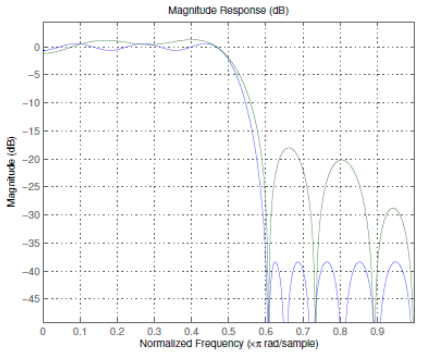
Procedure

- Design a filter using filterDesigner
- Generate Matlab code which returns the filter data
  - In File menu
- Generated function is called from Matlab command line, provides coefficients (in Hd.Numerator)
- Quantization actions are applied to the coefficients
  - many alternatives: **quantize** command, **fi** objects, **round** command ...)
  - Note: **max(abs(Hd.Numerator))** gives the largest absolute coefficient value which can be used to select scaling
- Frequency responses of the quantized and unquantized filter are compared (**fvtool**)
- See intro2.pdf Section 3






```
>> fvtool(Hd,Hdq);
```

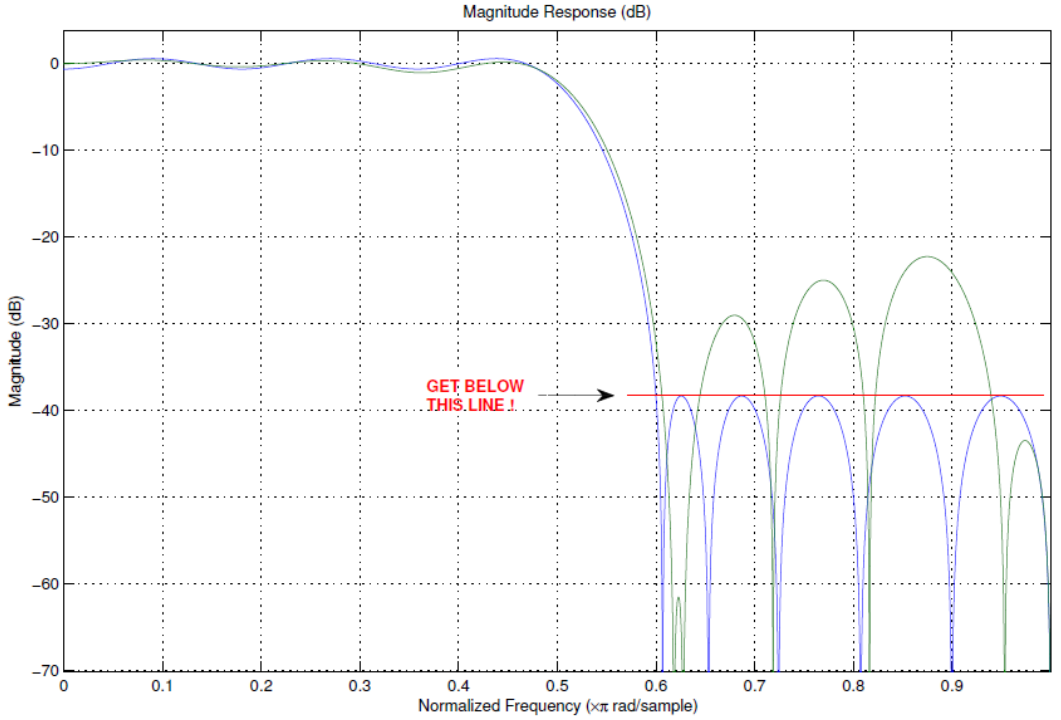


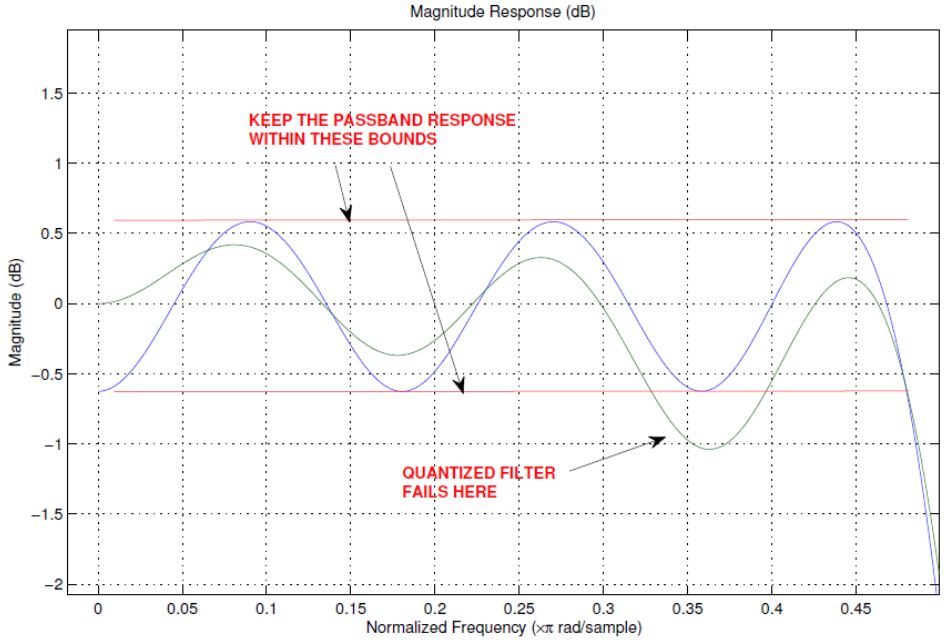
Trying 5-bit wordlength

- Filter was designed using filterDesigner
- Matlab code was generated: x_filter.m in dtask2.zip
- `max(abs(Hd.Numerator))=0.4701`
  - Less than 0.5 => with binary point scaling, the fixed-point format **s**5.5 can be used
- Quantizing coefficients to this format

```
>> Hdq = dfilt.dffir;
>> Hdq.Numerator = fi(Hd.Numerator,1,5,5);
```

- Comparison of frequency responses:
  - Clearly seen that the 5-bit word length is not sufficient





Improving fixed-point design

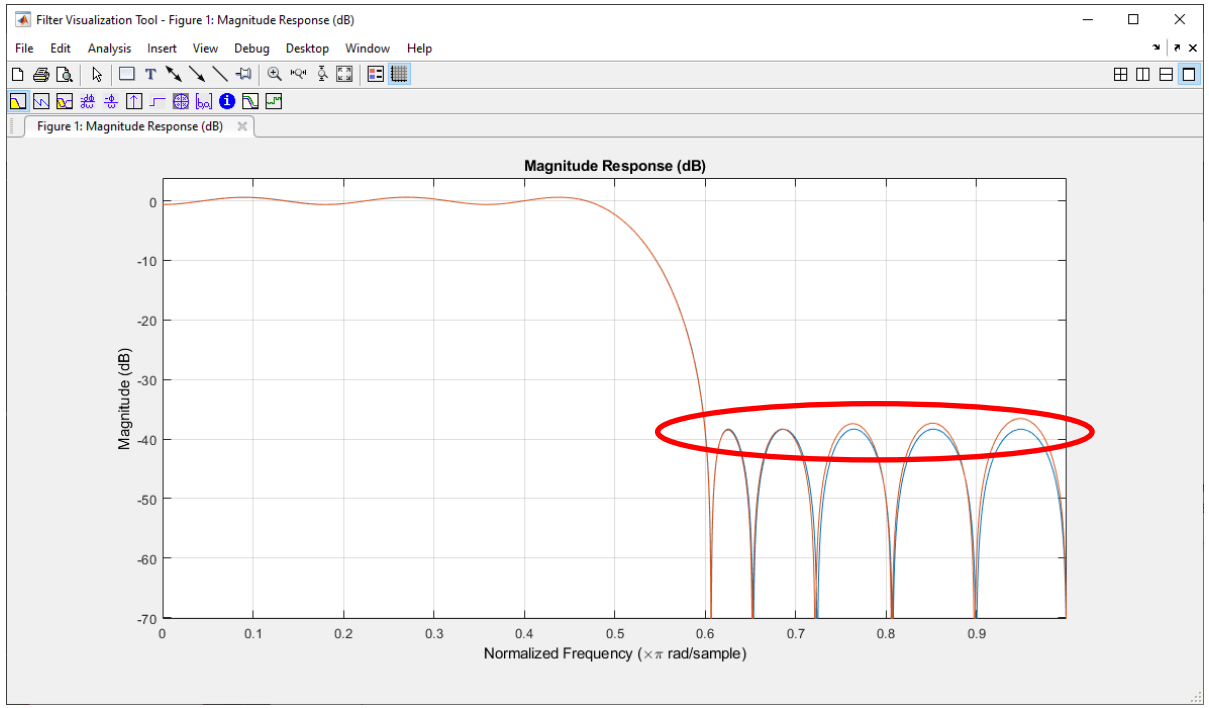
- To find a solution, we may

  1. Increase the coefficient word length until the performance comes sufficiently close to specification

  2. Another approach is to design a floating-point filter with **tighter specification** and consider its quantization

- **An example of using the first option: gradually increase the word length from 5 bits (6, 7, ..)**
- In the case of 10 bits
  - We have s10.10 format
  - About 2 dB error in the stopband can still be seen on the right
- If we increase the word length more, we can reduce the error more

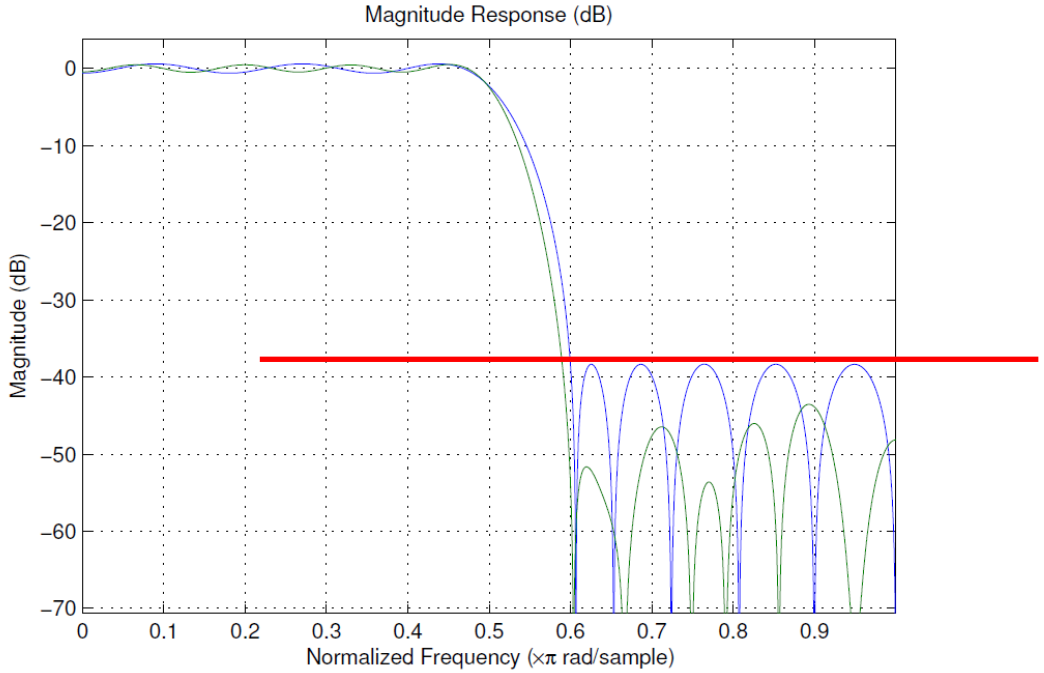
```
>> Hd = x_filter;
>> Hdq = dfilt.dffir;
>> max(abs(Hd.Numerator))

ans =

    0.4701

>> fraction_bits = 10;
>> mplier = 2^fraction_bits;
>> Hdq.Numerator = round(Hd.Numerator * mplier) / mplier;
>> fvtool(Hd,Hdq)
```

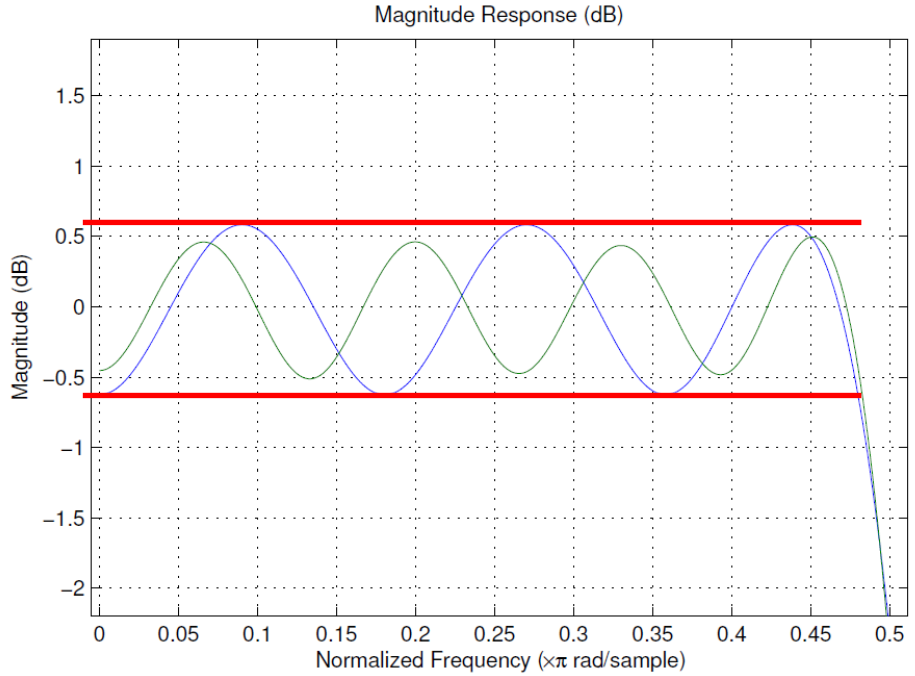


Improving design …

- Trying the second option: knowing that quantization reduces the performance, we put higher demands for the original filter
- Example: in filterDesigner, setting
  - 0.75 dB passband ripple (< 1 dB)
  - 60 dB stopband attenuation (> 40 dB)
- Resulting filter:
  - 35 coefficients (previous design has 24)
  - Largest coefficient now 0.5232
- Quantizing this to s10.9 format:
  - Now, the requirement is clearly fulfilled
  - Longer filter and perhaps too good => one could iterate with the design parameters and quantization to find more optimal solution

Topic of Design Task 2, Problem 2





DT 2 Problem 2

Task 2. [Filter design (Sec. 3), 1.5p] A list of filter specifications is given on the last page of this handout. Take the specification assigned to your group.

(a) Design a fixed-point FIR filter which fulfills the specification requirements. As explained in Chapter 3 of the background material, you can do it in the following steps:

1. Design a floating-point FIR filter using `filterDesigner` or `fdatool`.
2. Quantize the coefficients provided by the tool.
3. Check out the response of the quantized filter using `fvtool`.

Steps 2-3 are repeated as many times as needed to find a sufficient quantization. What is the number of coefficients (L) in that filter and what is the sp_c.n_c format of the quantized coefficients?

(b) Chapter 3.3. in the material explains how the filter specification can be tightened to find a proper fixed-point filter meeting the original specification. Try out this approach:

1. Using the values from your specification, decrease the passband ripple by 10-20% and increase the stopband attenuation by 5-10 dB. Design a floating-point FIR filter with these specifications.
2. Quantize the coefficients provided by the tool.
3. Check out the response of the quantized filter using `fvtool`.

Repeat steps 2-3 until the filter performance is close to the original specification. What is the number of the coefficients (L) in the filter? What is the sp_c.n_c format of suitable quantized coefficients?

(c) Compare the filters you found in subtasks (a) and (b). Compute for both the number of bits required for storing the coefficients, N_{tot}:

$$N_{\text{tot}} = Lp_c,$$

where L is the length of the filter, and p_c is the coefficient word length. Select the filter for which N_{tot} is lower, and use it in Task 3.

(d) For the filter you selected in (c), include in the report `fvtool` images to show that it works in passband/stopband.

Analysis related to coefficient word length*

- how to find some estimate before experiment

- Consider FIR filter having N coefficients (N-tap filter)
- **s***p*.(*p* − 1) format for the coefficients is to be selected, what is *p*?
- Quantization error introduces a parallel filter: $H_q(z) = H(z) + E(z)$

- Max. quantization error = the unit in least position (ulp) divided by two. As ulp = $2^{-(p-1)}$, $|e(n)| \leq 2^{-p}$.
- Assuming the worst-case error for all coefficients, we get $E(\omega) \leq 2^{-p}N$ (equality holds for DC)

- Assuming we have equality over all frequencies, stop band attenuation is bounded by $20 \log_{10}(2^{-p}N)$
- For a filter specification, we get an **upper bound** for the word length *p*
  - ”Upper bound” = the required word length is at most p
  - Equation is $p \leq -\log_2\left(\frac{10^{A/20}}{N}\right)$, where A is specified stopband attenuation
  - Due to assumptions, the bound is a conservative estimate (i.e. we can use lower *p*)
- See Ifeachor & Jervis (2002, Ch. 7.11) for more discussion and other bounds.

”Quantization introduces a parallel filter”

Unquantized filter: $H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$

Quantized filter: $H_q(z) = \sum_{n=0}^{N-1} h_q(n)z^{-n} = \sum_{n=0}^{N-1} [h(n) + e(n)]z^{-n}$

$e(n)$ = quantization error

$$= \underbrace{\sum_{n=0}^{N-1} h(n)z^{-n}}_{H(z)} + \underbrace{\sum_{n=0}^{N-1} e(n)z^{-n}}_{E(z)}$$

”$E(\omega) \leq 2^{-p}N$ (equality holds for DC)”

Error filter: $$E(z) = \sum_{n=0}^{N-1} e(n)z^{-n}$$

Worst-case error: $e(n) = 2^{-p}$ Half of distance between quantization levels

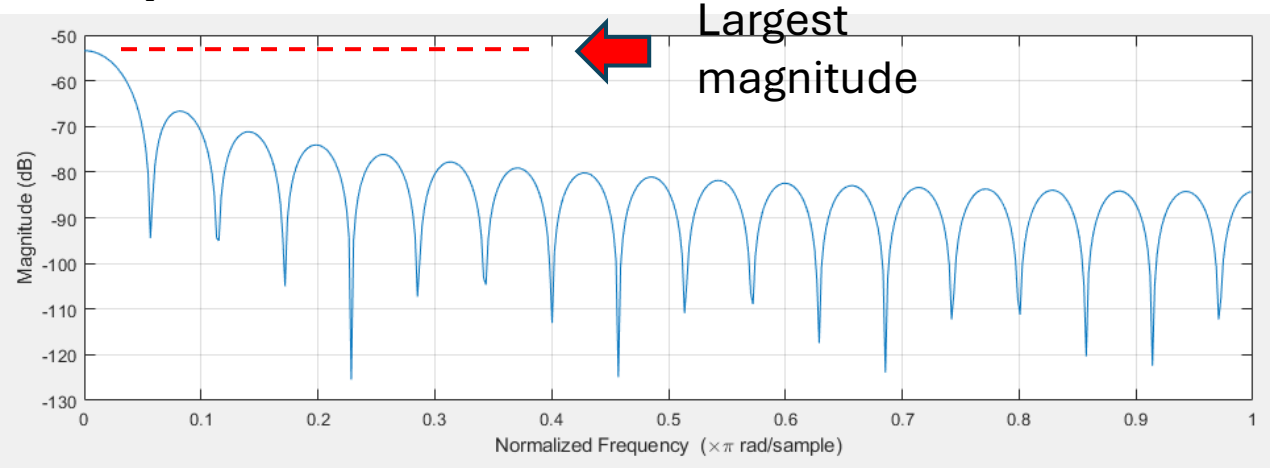
$$E(z) = \sum_{n=0}^{N-1} 2^{-p}z^{-n}$$

E.g. case: p = 14, N = 35
$E(z)$’s frequency response

Magnitude seen at frequency zero:

$20 \log_{10}(2^{-p}N) = -53.4$

```
>> freqz(2^-14*ones(1,35))
```



Example.

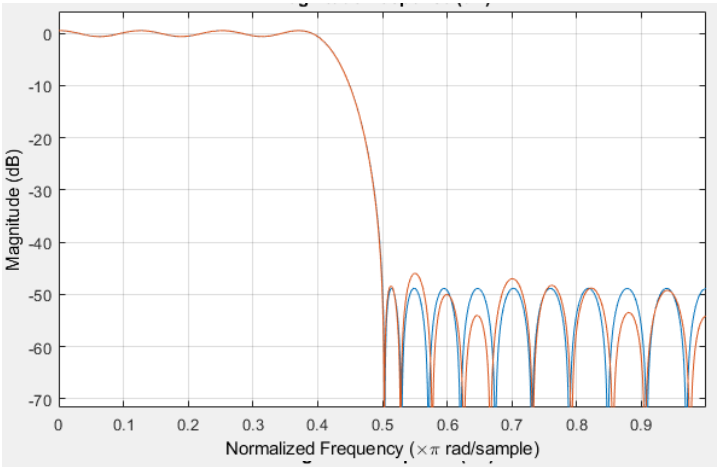
```
% Body of the code generated using filterDesigner (fdatool)
Fs = 48000; % Sampling Frequency
Fpass = 9600; % Passband Frequency
Fstop = 12000; % Stopband Frequency
Dpass = 0.057501127785; % Passband Ripple
Dstop = 0.0031622776602; % Stopband Attenuation A = -50 dB
dens = 20; % Density Factor
[N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0], [Dpass, Dstop]);
b = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);
% Hd.Numerator contains unquantized coefficients, N = 35
```

$$p \leq -\log_2\left(\frac{10^{\frac{A}{20}}}{N}\right) = -\log_2\left(\frac{10^{-2.5}}{35}\right) = 13.4341$$

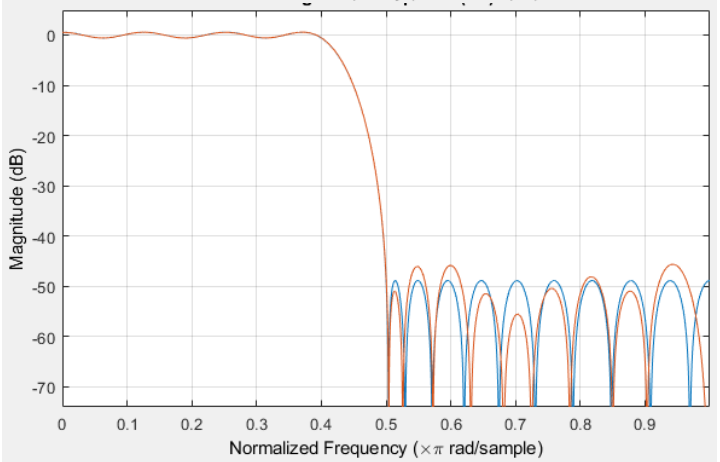
```
% Quantizing and plotting
Hdq = dfilt.dffir;
Hdq.Numerator = fi(Hd.Numerator,1,p,p-1)
fvtool(Hd,Hdq)
```

With p = 11, the result is close to wanted for stopband

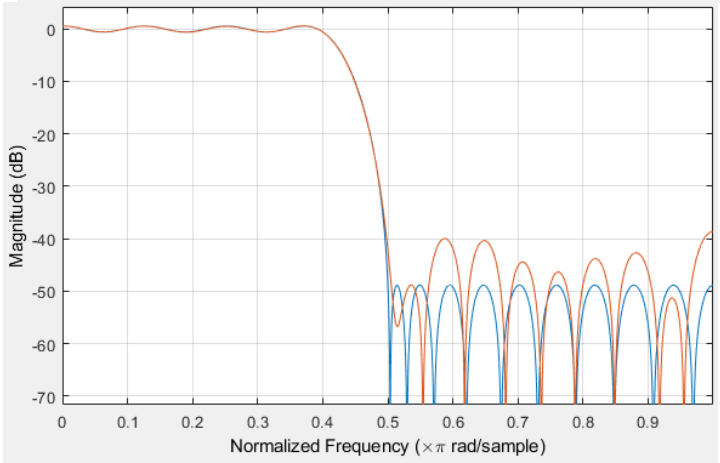
p = 12



p = 11



p = 10

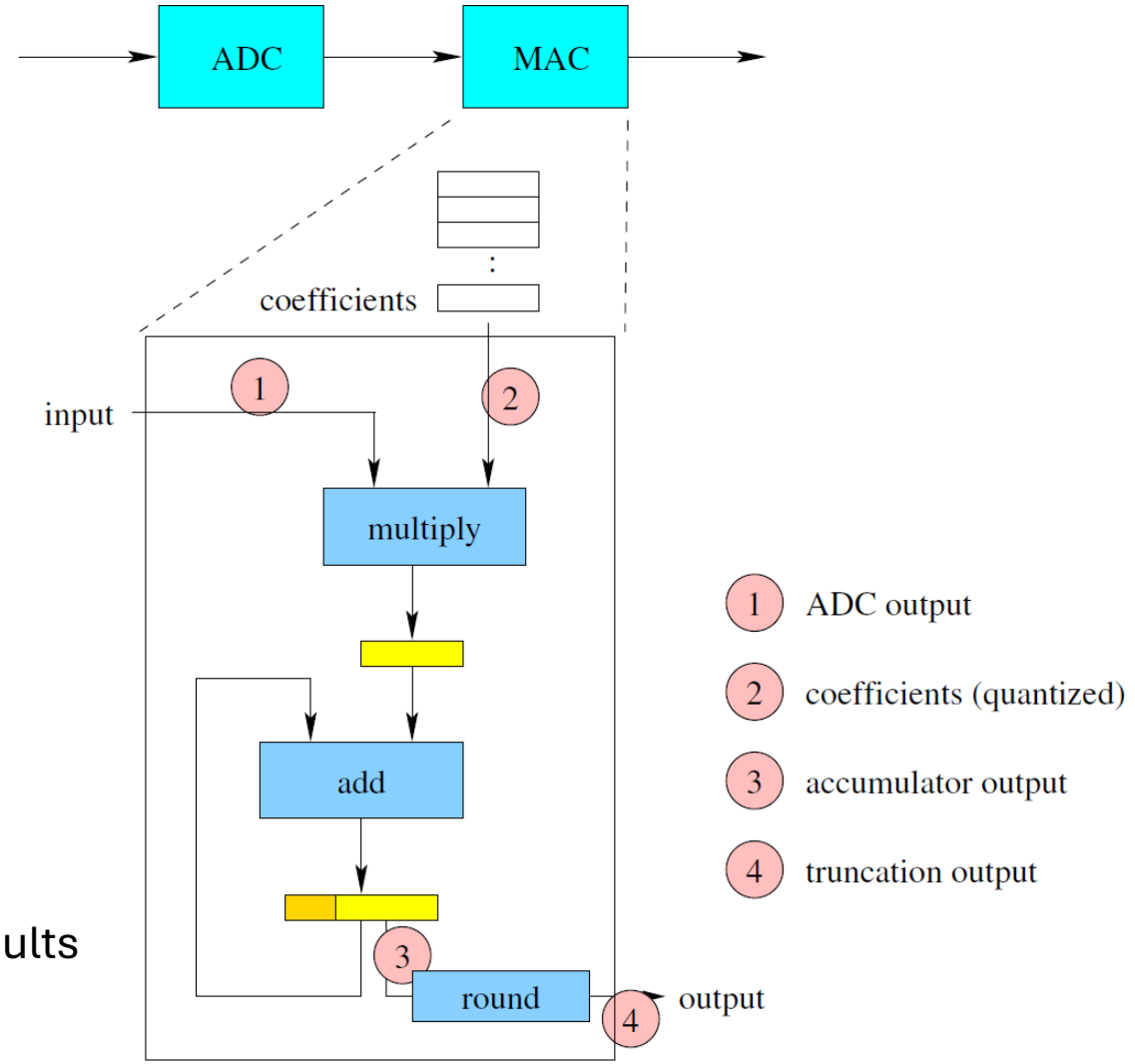


As an exercise, you may try to use similar analysis to estimate necessary word length for the filter in Design Task 2 Problem 2

Roundoff error study

- A simulator was developed for experimenting with FIR filtering on a MAC unit
- Implemented using Fixed Point Toolbox
- Construction:

h = FIR_MAC_Simulator(qxp,qap,qpp,gbits,qop)
where
- qxp specifies the ADC parameters
- qap specifies coefficient quantization
- qpp specifies the pipeline register for multiplication results
- gbits is the number of guard bits in the accumulator
- gop specifies the output format.



The output h is a structure that contains nested function handles.

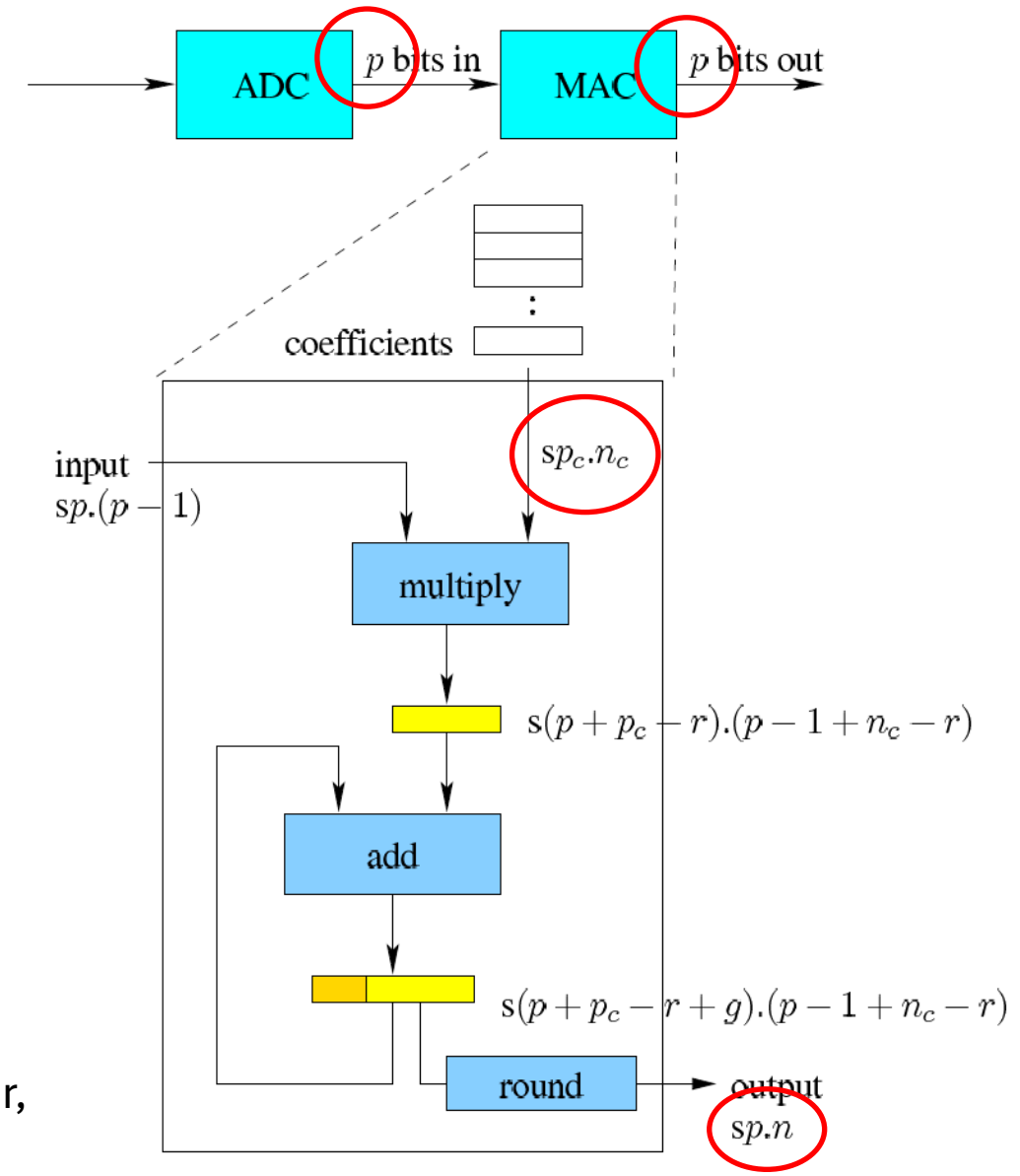
Experimentation tool

- In **Design Task 2 Problem 3**, this simulator is configured and executed in another function that is used to experiment with word length effects
- Execution:

  [y_fixpt,y_float,y_error] = **FIR_Experiment**(signal, a, bitspec) where
  - **signal** selects the test signal
  - **a** contains the unquantized filter coefficients
  - **bitspec** contains parameters, that control the word and fraction lengths used in the simulation
  - **y_fixpt** contains the fixed-point filtering result
  - **y_float** contains the floating-point result
  - **y_error** is their difference (y_fixpt – y_float)

  bitspec = [p, p_c, n_c, r, g, n], which define fixed-point types for the data paths as shown on the right

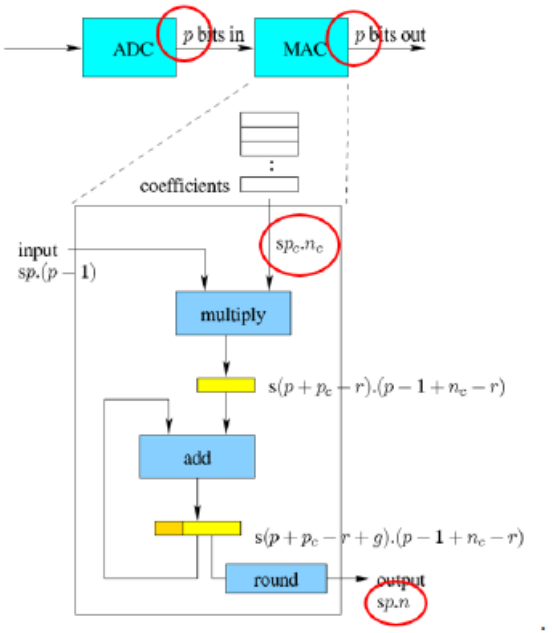
  Note how **r** reduces the word and fraction length of the multiplier, and thereby also modifies the size of the accumulator.



DT 2 Problem 3

Task 3. [Roundoff errors (Sec. 4), 1.5p] The filter you designed in Task 2 is implemented on a MAC unit. The Matlab function **FIR_Experiment.m** (in **dtask2.zip**) is used for simulating the filtering. Its operation is described in Chapter 4 of the background material. Note that you do not have to make any changes to this function.

(a) Experiments are done using `FIR_Experiment`, which takes as an input argument a bit specification `bitspec=[` p, p_c, n_c, r, g, n `]`, whose values control the data path as shown below.



The word length for the signal input and output, p, is provided in the last column of the parameter table, take the value from there. p_c and n_c are the coefficient word and fraction lengths you determined in Task 2. r is used to reduce the fraction length of the multiplier output and will be varied in subtask (d).

In this subtask, consider the choice of g and n, which control the number of guard bits in the accumulator and the fraction length of the MAC output. The incoming signal from ADC is considered to be in the fractional fixed-point format $sp.(p-1)$. Then, to quarantee that there is no overflow in the accumulator, one must be able to represent the numbers in the range $[-A, +A]$, where $A = \sum_i |h_i|$ (h_i are the filter coefficients). A is the maximum possible value in the accumulator and therefore also the maximum possible value in the MAC output.

Based on A, we can select the values of g and n. For example, if $A < 1$ we can use $g = 0$ and $n = p - 1$. Explain why. Analyze your filter and select appropriate g and n.

(b) Perform next the following experiment using the values found in (a) for `bitspec` and set $r = 0$.

Firstly, use `'sine'` as the first argument in the function call. As a result, the test signal is a sum of three sine waves (3.5 kHz, 7.5 kHz, 8.5 kHz; sampling frequency is 30 kHz). Using `freqz` function, plot the **spectrum** of the error signal and include the figure in your report. Note that the error signal is the third output of **FIR_Experiment**().

Secondly, repeat the simulation using a random signal as input. To do that, use `'rand'` as the first argument. Include a figure of the error spectrum in the report also in this case. How are the error spectra different? Why?

(c) Change the fraction length of the MAC unit output by increasing/decreasing the 6th parameter of `bitspec`. Go through the values $\{n-4,\ n-3, ..., n+4\}$ where n is the value determined in subtask (a). Simulate each case for the random signal input and compute the power of the error signal in decibels. Tabulate or plot the result in the report (i.e. error power as a function of the fraction length). Explain the dependence observed between the fraction length and the error power.

(d) The length of the pipeline register used for the multiplication output can be reduced by increasing the value of r, the 6th parameter of `bitspec`. Set other parameters as in subtask (b), simulate, and compute the power of the error signal for $r = 0, 1, 2, ...$ How quantization of the multiplication results changes the power of the error signal? What length of the pipeline register seems to be sufficient?

To be continued next Monday ...