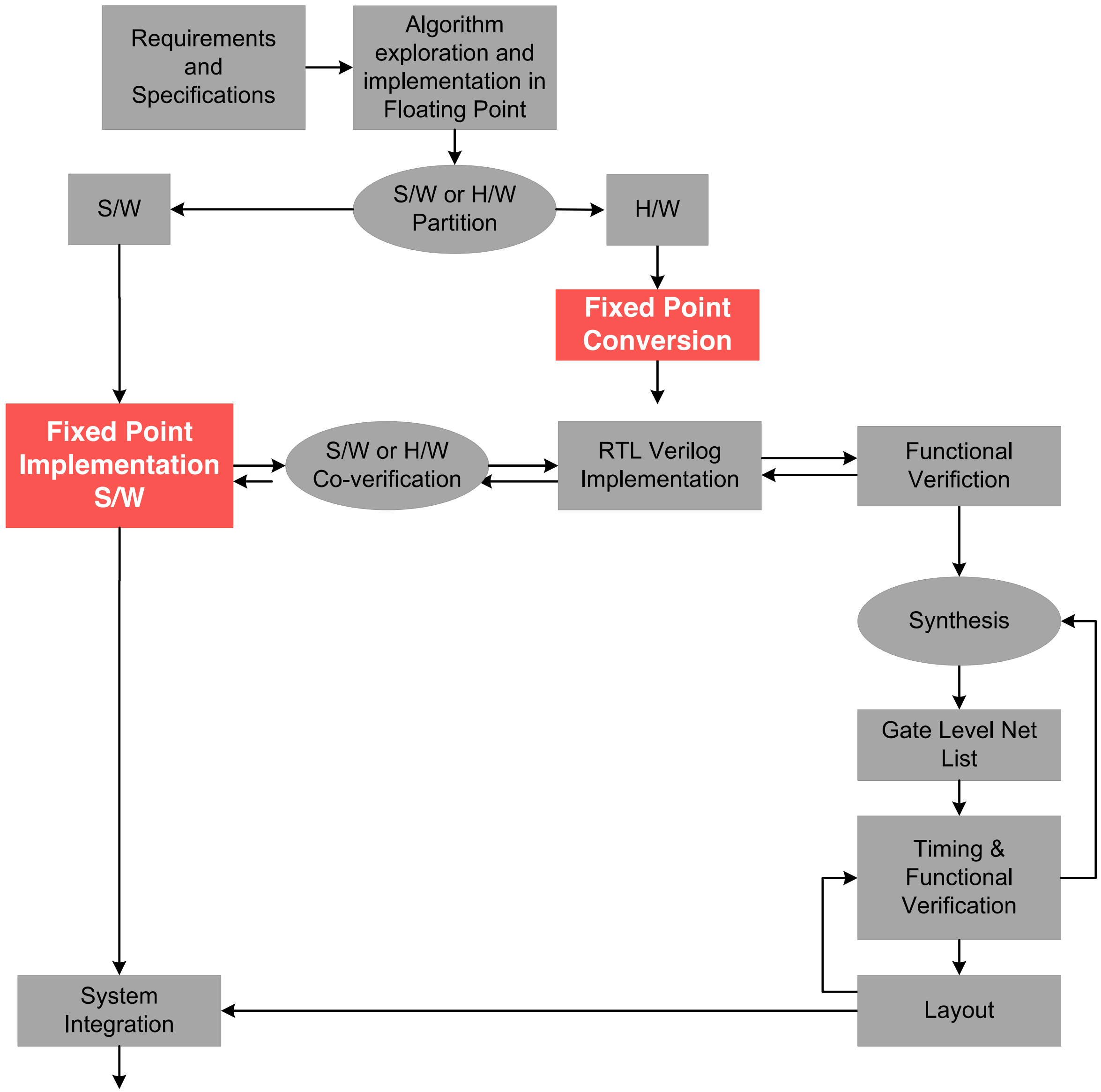
ECSE 682
Topics in Computers and Circuits
“VLSI Signal Processing”

Fixed-point Arithmetic

Prof. Warren Gross

Materials from “*Digital Design of Signal Processing Systems*” By S. A. Khan

System-level Design Flow and Fixed-point Arithmetic



Floating-point

- Three integers define the value:

$$(-1)^s \times c \times b^q$$

b is the base or radix: 2 or 10
c is the significand
S is the sign

e.g. -4.5677 x 2^{17}

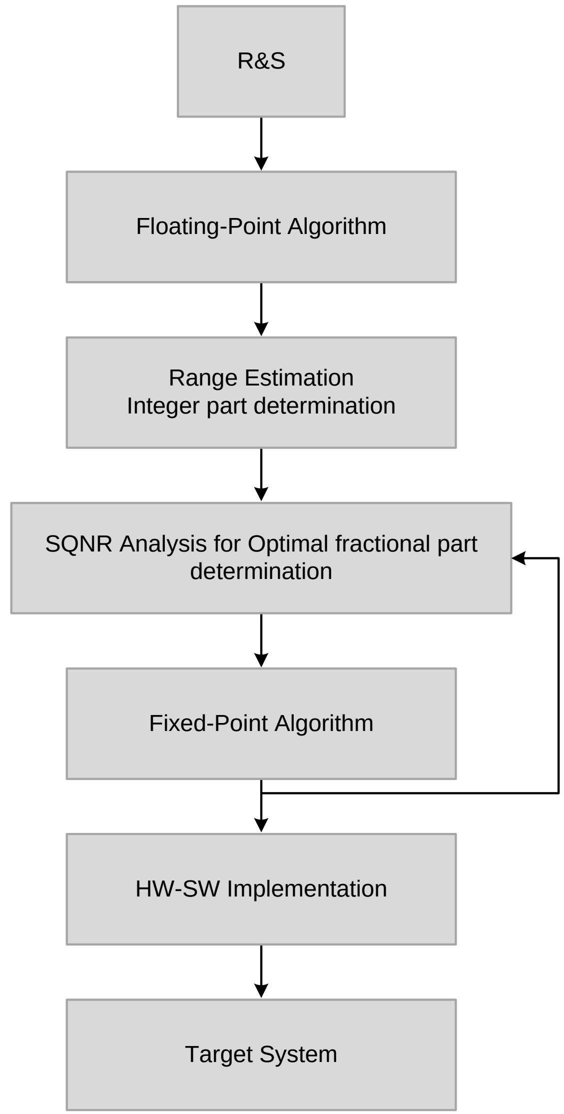
- Way to approximate real numbers
- Trade-off between range and precision
- IEEE floating-point provides error handling and rounding rules
- Floating-point h/w automatically scales the significand and updates the exponent to make the result fit in the required number of bits in a defined way
- Expensive in hardware

Fixed-point

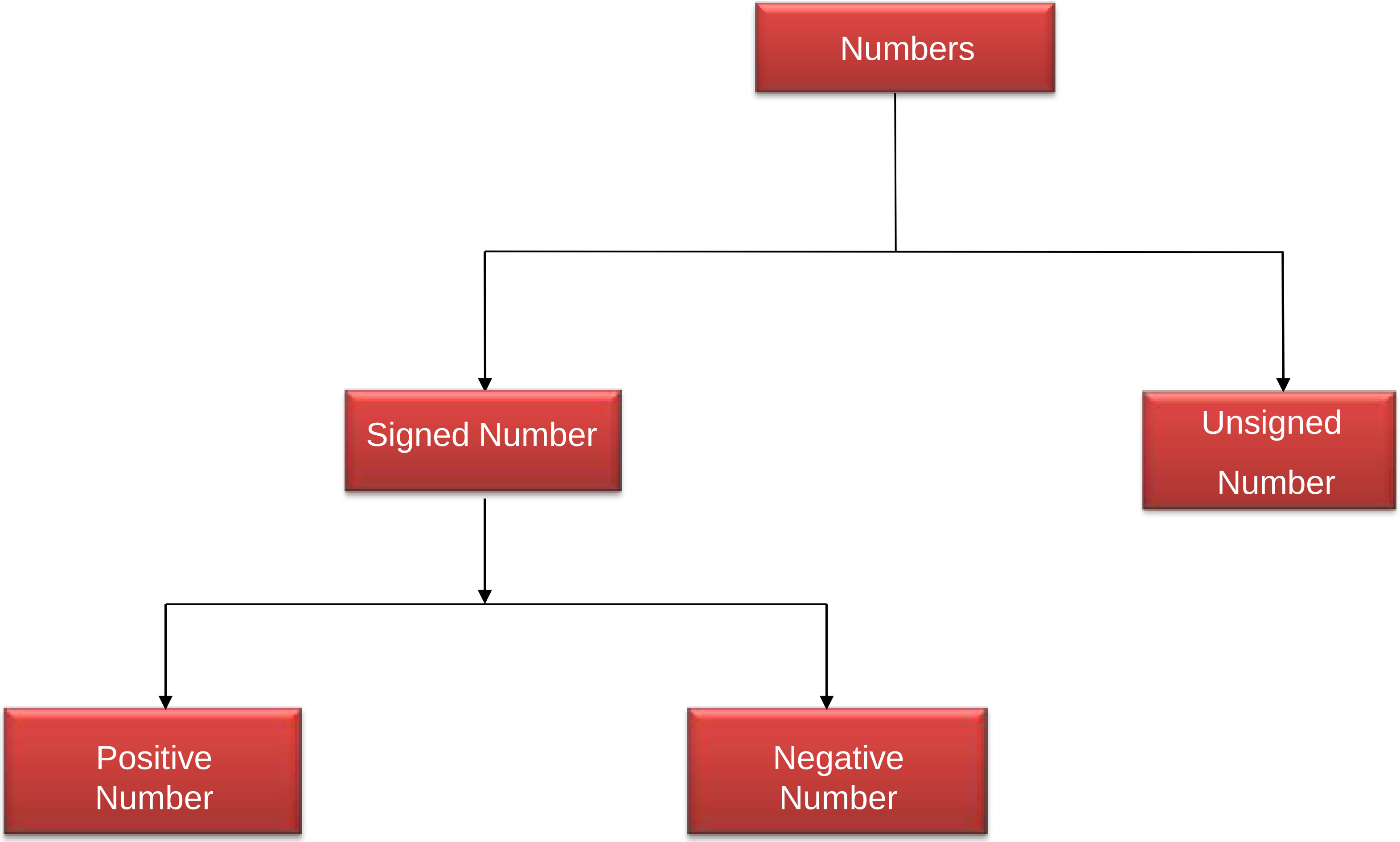
- Another way to approximate real-numbers
- If you imagine a "binary point" at a fixed place in a binary number, then regular integer arithmetic works

```
    1 0 1 0    (10)          10 . 10      (2.5)
+   0 0 0 1    (1)         + 00 . 01      (0.25)
-------------------        -----------------
     1011    (11)            10. 11       (2.75)
```

Floating-Point to Fixed-Point Conversion



2's Complement Arithmetic



n-bit binary representation

$$a_{n-1}a_{n-2}\ldots a_2a_1a_0$$

msb

lsb

2’s complement representation

$$a_{n-1}a_{n-2}\dots a_2a_1a_0$$

For positive numbers:

$$a = \sum_{i=0}^{n-2} a_i 2^i$$

For negative numbers:

$$a = -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

2's Complement Arithmetic

- MSB has negative weight,
  - Positive number $a_{N-1} = 0$
  - Negative number $a_{N-1} = 1$

e.g. 1 0 1 (-3) = $1 * -2^2 + 0 * 2^1 + 1 * 2^0$

Example

1 0 1 1 (Negative number as MSB = 1)

2^3 2^2 2^1 2^0

-8 + 2 + 1 = -5

Equivalent Representation

Many design tools do not display numbers as 2’s complement signed numbers
A signed number is represented as an equivalent unsigned number
Equivalent unsigned value of an N-bit negative number is

$2^N - |a|$

Example
for -5 = 1011

N=4
a=-5
$2^4 - |-5| = 16 – 5 = +11$

In binary it is equivalent rep is 1011

Four-bit representation of two's complement and equivalent unsigned numbers

Decimal number	Two's complement representation				Equivalent unsigned number
	-2^3	2^2	2^1	2^0	
0	0	0	0	0	0
+1	0	0	0	1	1
+2	0	0	1	0	2
+3	0	0	1	1	3
+4	0	1	0	0	4
+5	0	1	0	1	5
+6	0	1	1	0	6
+7	0	1	1	1	7
−8	1	0	0	0	8
−7	1	0	0	1	9
−6	1	0	1	0	10
−5	1	0	1	1	11
−4	1	1	0	0	12
−3	1	1	0	1	13
−2	1	1	1	0	14
−1	1	1	1	1	15

Computing Two's Complement of a Signed Number

- Refers to the negative of a number
- Invert all bits and add 1 to the LSB
- Adding 1 can be expensive in HW

Sign Extension

- An N bit number is extended to an M bit number M > N, by replicating M-N sign bits to the most significant bit positions
  - Positive number: M-N extended bits are filled with 0s
    - The number unsigned value remains the same
  - Negative number: M-N extended bits are filled with 1s,
    - Signed value remains the same
    - Equivalent unsigned value is changed

4'b1000 2's complement sign number is sign extend to 8'b1111_1000

Dropping Redundant Sign bits

- When a number has redundant sign bits, these redundant bits can be dropped
- This dropping of bits does not affect the value of the number

Example

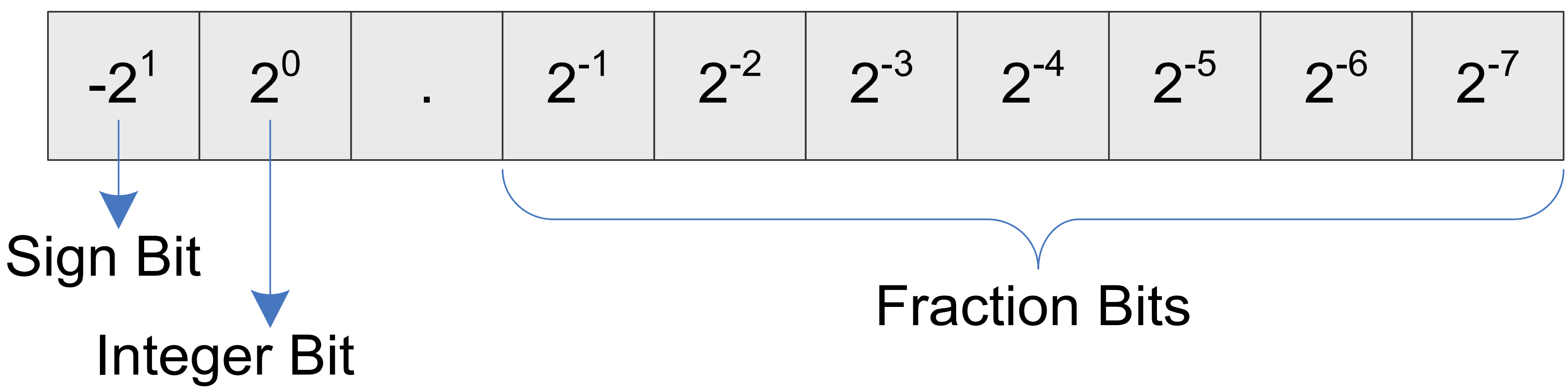
8’b1111_1000 = -8

Is same as

4’b1000 = -8

Qn.m Format for Fixed-point Arithmetic

- Qn.m format is a fixed positional number system for representing floating-point numbers
- A Qn.m format N-bit binary number assumes n bits to the left and m bits to the right of the binary point



- MSB is the sign bit
- Positive numbers: MSB is 0

$$b = 0b_{n-2} \dots b_1 b_0 b_{-1} b_{-2} \dots b_{-m}$$

- Equivalent floating-point value is:

$$b = b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 + b_{-1}2^{-1} + b_{-2}2^{-2} + \cdots + b_{-m}2^{-m}$$

- For negative numbers, the MSB has negative weight and its equivalent value is

$$b = -b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 + b_{-1}2^{-1} + b_{-2}2^{-2} + \cdots + b_{-m}2^{-m}$$

Floating-point to fixed-point

- Convert floating point to Qn.m
  - Bring m fractional bits to the integer part
  - Drop the rest of the bits with or without rounding
  - This gives an integer with an *implied* binary point
    - The designer needs to remember where the point is

- In Matlab,

$$numfixed = round(numfloat \times 2^m)$$

$$numfixed = fix(numfloat \times 2^m)$$

Saturation

$$\mathtt{num\_fixed} = \mathtt{round}(\mathtt{num\_float} \times 2^{m})$$

$$\mathtt{if}\ (\mathtt{num\_fixed} > 2^{\mathtt{N-1}} - 1)$$

$$\mathtt{num\_fixed} = 2^{\mathtt{N-1}} - 1$$

$$\mathtt{elseif}\ (\mathtt{num\_fixed} < -2^{\mathtt{N-1}})$$

$$\mathtt{num\_fixed} = -2^{\mathtt{N-1}}$$

Example: Conversion to Q1.15 on a 16-bit DSP

```
num_fixed_long = (long)(num_float × 2^15)
if (num_fixed_long > 0x7fff)
    num_fixed_long = 0x7fff
elseif (num_fixed_long < 0xffff8000)
    num_fixed_long =0xffff8000
num_fixed_Q15 = (short)(num_fixed_long & 0xffff))
```

Example (Q2.3)

- 1.75

Examples (Q1.15)

- 0.5
- -0.5
- 0.9997
- 0.213
- -1.0

Equivalent Q formats

- In many cases Q format of a number is to be changed

Convert $Q_{n1.m1}$ to $Q_{n2.m2}$

- If n2>n1, we simply append sign bits=n2-n1 to the MSB location of n1
- If m2>m1 we simply append zeros to the LSB locations of the Fractional part of m1

Example

- Let a=11.101 (Q2.3)

is supposed to be added to a number b of $Q_{7.7}$ format.
So extending a will result in:

1111111.1010000 ($Q_{7.7}$)

Arithmetic: Addition in Q Format

Addition of two fixed-point numbers a and b of Qn1.m1 and Qn2.m2 formats, respectively, results in a Qn.m format number, where n is the larger of n1 and n2 and m is the larger of m1 and m2.

Example

implied decimal

$Qn_1.m_1$	1	1	1	1 •	1	0			= Q4.2 = -2+1+0.5 = -0.5
$Qn_2.m_2$	0	1	1	1	0	1	1	0	= Q4.4 = 1+2+4+025+0.125 = 7.375
Qn.m	0	1	1	0	1	1	1	0	= Q4.4 = 2+4+0.5+0.25+0.125 =6.875

Multiplication in Q-Format

$$\mathbf{Q}_{\mathbf{n1.m1}}\ \mathbf{X}\ \mathbf{Q}_{\mathbf{n2.m2}} = Q\ (n1+n2)\ .\ (m1+m2)$$

Four types of Fractional Multiplication:

Unsigned Unsigned

Unsigned Signed

Signed Unsigned

Signed Signed

Signed x Signed multiplication, results in a redundant sign bit

Unsigned by Unsigned

- The partial products are added without any sign extension logic

```
        1 1 0 1 = 11.01 in Q2.2 = 3.25
        1 0 1 1 = 10.11 in Q2.2 = 2.75
      ---------
        1 1 0 1
      1 1 0 1 X
   0  0 0 0 X X
  1 1 0 1 X X X
---------------
1 0 0 0 1 1 1 1= 1000.1111 in Q4.4 i.e.8.9375
```

Signed by Unsigned

- Sign extension of each partial product is necessary in signed-unsigned multiplication.
- The partial products are first sign-extended and then added

				1	1	0	1	= 11.01 in Q2.2 = -0.75
				0	1	0	1	= 01.01 in Q2.2 = 1.25
1	**1**	**1**	**1**	1	1	0	1	extended sign bits shown in bold
0	**0**	**0**	0	0	0	0	X	
1	**1**	1	1	0	1	X	X	
0	0	0	0	0	X	X	X	
1	1	1	1	0	0	0	1	= 1111.0001 in Q4.4 i.e.-0.9375

Unsigned by Signed

- All partial products except for the last one are unsigned.
- Must sign extend the last partial product
- For the last partial product, compute the 2's complement of the unsigned multiplicand

```
              1 0 0 1 = 10.01 in Q2.2 = 2.25 (unsigned)
              1 1 0 1 = 11.01 in Q2.2 = -0.75 (signed)
              -------
              1 0 0 1
            0 0 0 0 X
          1 0 0 1 X X
  1 0 1 1 1 X X X      2's compliment of the positive multiplicand 01001
  -------------------
  1 1 1 0 0 1 0 1     = 1110.0101 in Q4.4 i.e.-1.6875
```

Signed by Signed

- Sign extend all partial products
- Takes 2's complement of the last partial product if multiplier is a negative number.
- The MSB of the product is a redundant sign bit
  - Removed the bit by shifting the product to left, the product is in

$$Q_{(n1 + n2 - 1) \, . \, (m1 + m2 + 1)}$$

```
          1 1 0 = Q1.2 = -0.5 (signed)
          0 1 0 = Q1.2 = 0.5 (signed)
        -------
    0 0 0 0 0 0
    1 1 1 1 0 X
    0 0 0 0 X X
  ---------------
    1 1 1 1 0 0 = Q1.5 format 1_11000=-0.25
```

Example: Signed x Signed

```
          1 1. 0 1 = -0.75 in Q2.2 format
          1. 1 0 1 = -0.375 in Q1.3 format
      ---------------
  1 1 1 1 1 1 0 1
  0 0 0 0 0 0 0 X
  1 1 1 1 0 1 X X
  0 0 0 1 1 X X X
-----------------
  0 0 0 0 1 0 0 1 = shifting left by one 00.010010 in Q2.6 format is 0.2815
```

Corner Case:

Signed-Signed Fractional Multiplication

- -1x-1 = -1 in Q fractional format is a corner case, it should be checked and result should be saturated as max positive number

```
          1 0 0  = Q1.2 = -1
          1 0 0  = Q1.2 = -1
    -----------
    0 0 0 0 0 0
    0 0 0 0 0 X
    0 1 0 0 X X     2's Compliment
    -----------
    0 1 0 0 0 0  Dropping redundant sign bit
                 results 1000000 = -1 in Q1.5
```

Fixed Point Multiplication

```
Word32 L_mult(Word16 var1,Word16 var2)
  {
   Word32 L_var_out;

   L_var_out = (Word32)var1 * (Word32)var2;
   if (L_var_out != (Word32)0x40000000L)    // 0x8000 x 0x8000 =
   0x40000000
     {
       L_var_out *= 2;          //remove the redundant bit
     }
   else
     {
       Overflow = 1;
       L_var_out = 0x7fffffff; //if overflow then clamp to max +ve value

                  }

   return(L_var_out);
  }
```

Bit Growth in Fixed-Point Arithmetic

- Multiplication of $Q(n_1, m_1)$ by $Q(n_2, m_2)$ results in $Q(n_1 + n_2, m_1 + m_2)$ or $Q(n_1 + n_2 - 1, m_1 + m_2 + 1)$ if signed $\times$ signed.
- Addition of $Q(n_1, m_1)$ and $Q(n_2, m_2)$ gives $Q(\max(n_1, n_2), \max(m_1, m_2))$

Several rounds of computation, or iterative computation:

reduce bit growth by truncation

Truncation

- In multiplication of two Q format numbers as the number of bits in the product increases
- We sacrifice precision by throwing some low precision bits of the product
- Qn1.m1 is truncated to Qn1.m2 where m2 < m1

Let the product is

8'b01110110 ($Q_{4.4}$)

$4 + 2 + 1 + 0.25 + 0.125 = 7.375$

Truncate it to $Q_{4.2}$ results

6'b011101 = 7.25

Rounding with Truncation

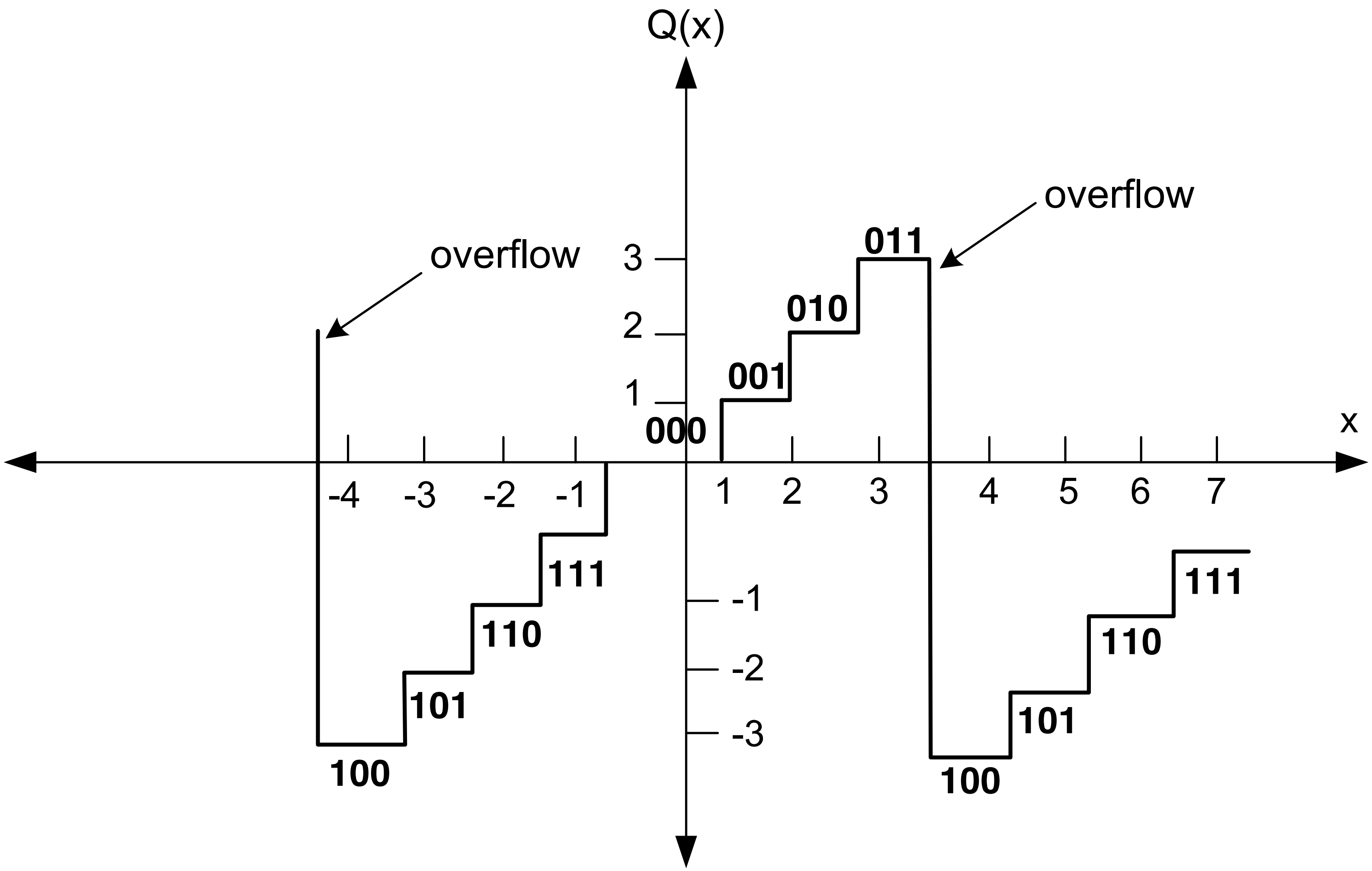
- Sometimes we truncate with rounding

  Example

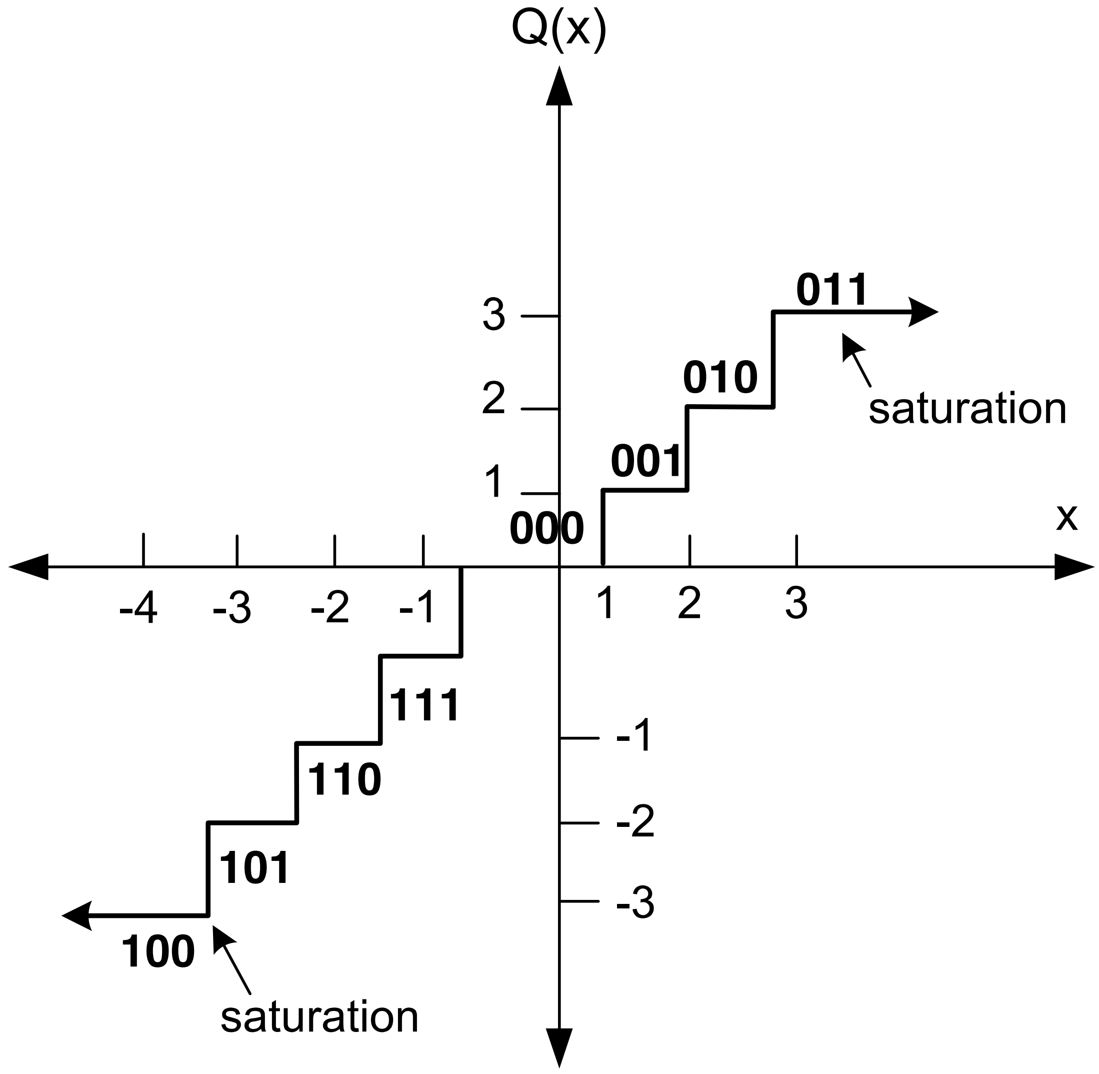
  3.726 can be rounded off to 3.73
- We do similar things in Digital Design that is “First Round off then Truncate”
- Before rounding add 1 to the right side of the truncation point and then truncate

```
                         truncation point
0 1 1 1_0 1 1 1  in Q4.4 is 7.4375
            1
________________ add 1 for rounding
0 1 1 1_1 0 0 1
0 1 1 1_1 0     = 7.5 Q4.2 format
```

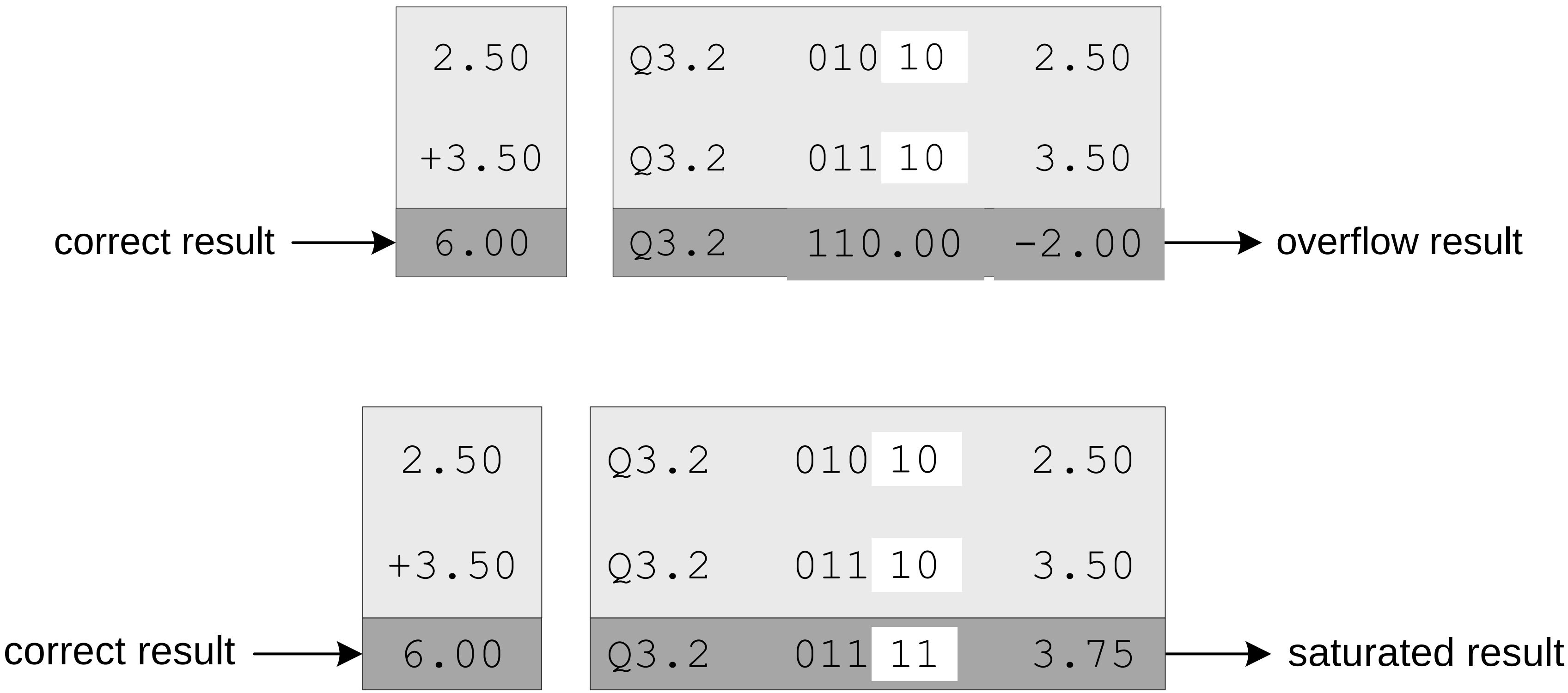
Overflow introduces an error equal to the dynamic range of the number



Saturation clamps the value to a maximum positive or minimum negative level



Overflow and Saturation



2’s Complement Intermediate Overflow Property

In an iterative calculation using 2’s complement arithmetic if it is guaranteed that the final result will be within precision bound of assigned fixed-point format then any number of intermediate overflows will not affect the final answer.

1.75	Q2.2	0111	1.75
+1.25	Q2.2	0101	1.25
3.00	Q2.2	1100	-1.00

intermediate overflow

3.00	Q2.2	1100	-1.00
-1.25	Q2.2	1011	-1.25
1.75	Q2.2	0111	1.75

correct final answer

Digital Signals

- Digital signals appear to the hardware as a sequence of numbers with index n
- Analog signals are sampled and encoded in binary form with a sampling period of T seconds
- The digital signal sampled at index n corresponds to time nT
- The sampling period is

  $$T = \frac{1}{f_s}$$

  where f_s is the sampling rate in cycles per second (Hz)

Digital Signals

- Usually we normalize the sampling period T to 1

$$x(n) = x(nT), \quad \infty \leq n \leq \infty$$

- *Causal* signal: Assume that any element of a sequence whose time index is less than zero has a value of zero:

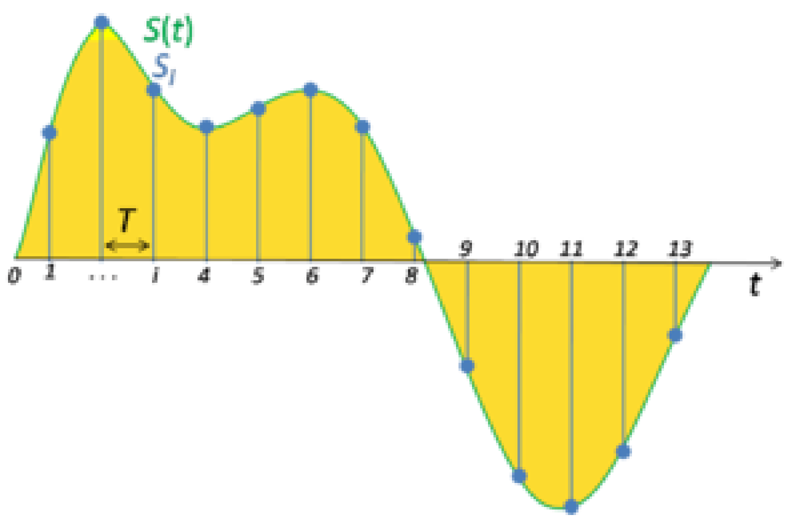
$$x(n) = 0, \quad n < 0$$

Basic Signals

- Digital unit-impulse

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

- Equally-spaced (T seconds) train-of-unit impulses are used as sampling functions to get discrete-time signals.



Transfer Function

$$y(n) = b_0 x(n) + b_1 x(n-1)$$

Impulse response:

$$y(n) = h(n) = \begin{cases} b_0, & n = 0 \\ b_1, & n = 1 \\ 0, & \text{otherwise} \end{cases}$$

Transfer function:

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

Linear Time-Invariant Systems

- linearity:

$$x(n) = a_1 x_1(n) + a_2 x_2(n)$$

$$y(n) = a_1 y_1(n) + a_2 y_2(n)$$

- time-invariant:

$$y(n) = O[x(n)]$$

$$y(n-k) = O[x(n-k)]$$

Upper Bound on Output Value

- For an LTI system, the upper bound on the output values with input $x(n)$ and impulse response $h(n)$ is found by the Cauchy-Schwarz inequality:

$$|y(n)| \leq \sqrt{\sum_{n=\infty}^{\infty} h^2(n) \sum_{n=\infty}^{\infty} x^2(n)}$$

- This can be used to find the maximum number of integer bits. The number of fractional bits depends on the tolerance of the system to quantization noise.

Finite-Impulse Response Filters

$$y(n) = b_0 x(n) + b_1 x(n-1)$$

has a finite impulse response of length 2. This can be generalized to a system with a FIR of length L, i.e.

$$h(i) = \{b_i, i = 0, 1, \ldots, L-1\}$$

Such a filter is called an FIR filter since its response to an impulse input becomes zero after a finite number L of output samples.

$$\begin{aligned} y(n) &= \sum_{i=0}^{L-1} h(i)x(n-i) \\ &= \sum_{i=0}^{L-1} b_i x(n-i) \\ &= b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + \cdots + b_{L-1} x(n-L+1) \end{aligned}$$

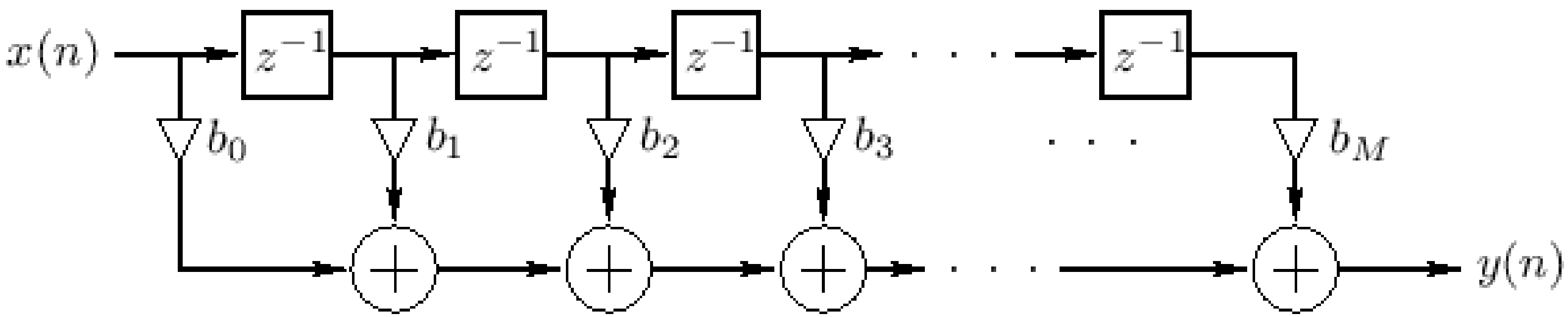
FIR Filter Transfer Function

Taking z-transform of both sides,

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{L-1} z^{-(L-1)} = \sum_{i=0}^{L-1} b_i z^{-i}$$

Setting $H(z) = 0$, we obtain $(L-1)$ zeros. Therefore the FIR filter of length L has order $L-1$.

Direct-Form I



- The signal buffer
  is also called a *delay buffer* or a *tapped delay line*.
- The MATLAB function $y = \text{filter}(b, 1, x)$ implements the FIR filtering where vector b contains the filter coefficients $\{b_i\}$ and vectors x and y contain input and output signals.
- The finite length of the impulse response guarantees that FIR filters are stable:

$$h(n) \to 0 \quad \text{as} \quad n \to \infty$$

- No phase distortion: linear-phase response
- Disadvantage: may require a high-order to achieve a given frequency response

IIR Filters

- If the impulse response of a filter is not a finite-length sequence, the filter is called an *infinite-impulse response* (IIR) filter.
- IIR filters can achieve sharp cuttoffs in frequency response with fewer coefficients.
- The transfer function of an IIR filter is:

$$
\begin{aligned}
H(z) &= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{L-1} z^{-(L-1)}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} \\
&= \frac{\sum_{i=0}^{L-1} b_i z^{-i}}{1 + \sum_{m=1}^{M} a_m z^{-m}}
\end{aligned}
$$

Poles and Zeros

$$H(z) = \frac{b_0(z - z_1)\dots(z - z_i)\dots(z - z_{L-1})}{(z - p_1)\dots(z - p_m)\dots(z - p_M)},$$

where z_i and p_m denote the zeros and poles of $H(z)$

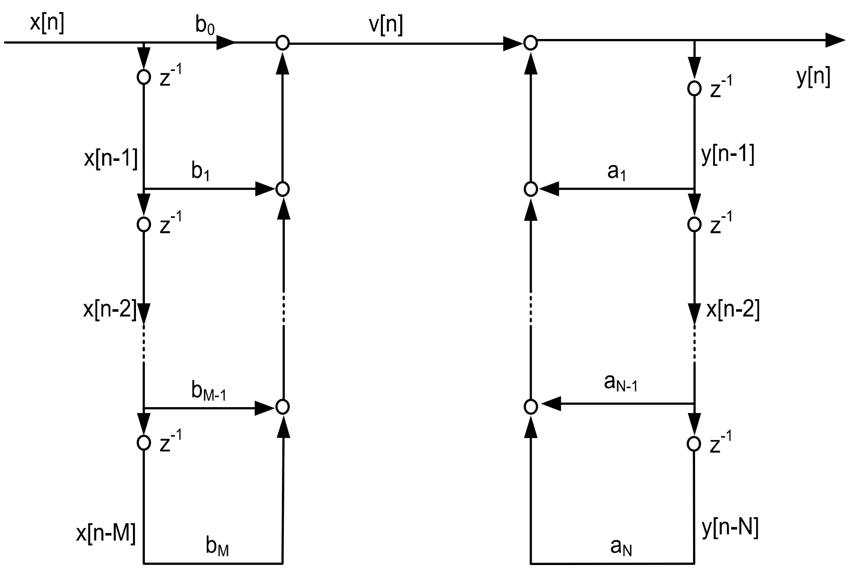
- A causal system is stable if and only if the transfer function has all of its poles inside the unit circle, i.e.

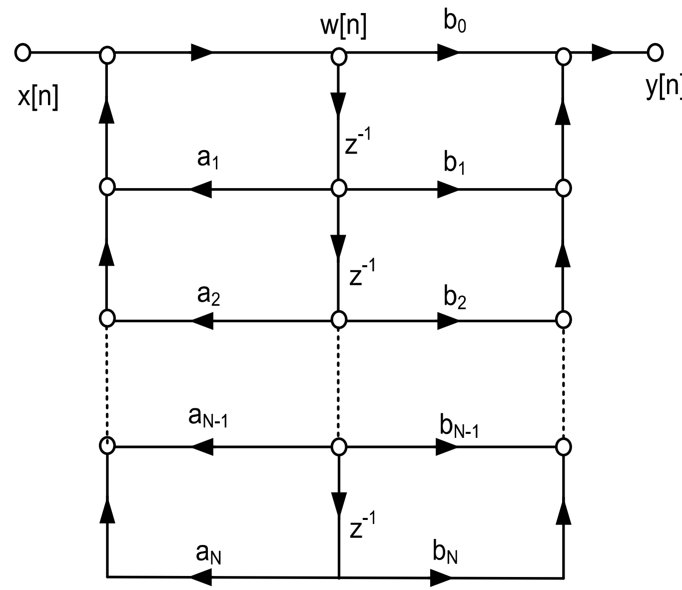
$$|p_m| < 1, \quad m = 1, 2, \dots, M$$

- In general, IIR filters require fewer coefficients to approximate a desired frequency response than FIR filters. However, they are more difficult to design and stability, finite-precision effects and nonlinear phases must be considered.

- Filter design in MATLAB: fdatool

Filter Structures: DF-I and DF-II



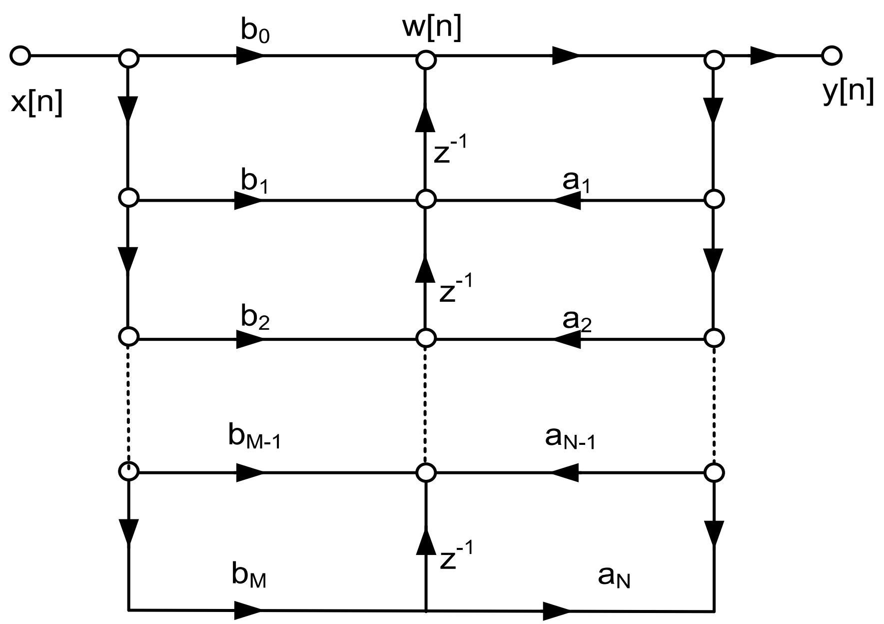


$$H(z) = H_1(z)H_2(z)$$

$$= \left(\sum_{i=0}^{L-1} b_i z^{-i}\right)\left(\frac{1}{1+\sum_{m=1}^{M} a_m z^{-m}}\right)$$

TDF-II

- Transposed Direct Form-II is another implementation that reduces the number of delays
- DF-I, DF-II, TDF-I and TDF-II all suffers from coefficient quantization
  - A filter designed using double precision may get unstable after quantization

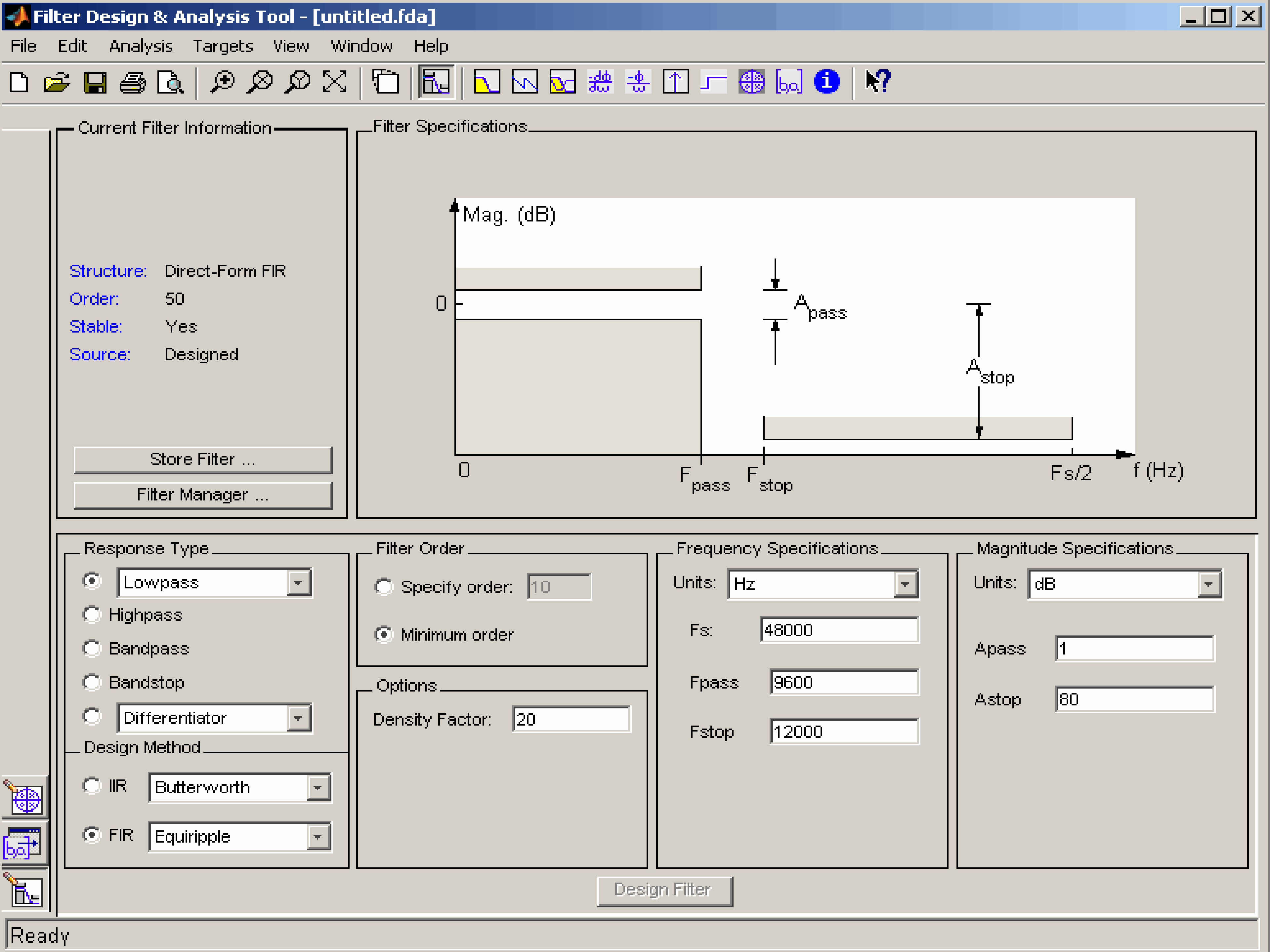


Quantization of IIR Filter Coefficients

- E.g. eighth-order, passband ripple of 0.5 dB, stop-band attentuation of 50 dB and normalized cutoff frequency ω_c = 0.15.

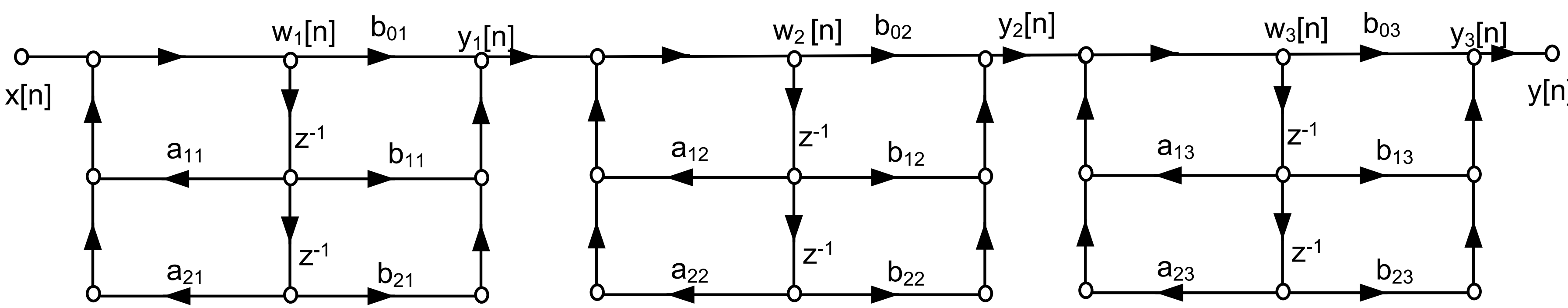
```
[b,a] = ellip(8,0.5,50,0.15)
```

- Can also use `fdatool`
- `E.g. lowpass IIR, 8’th order, Fs=2000 Hz, Fpass=100 Hz, Apass = 0.5 dB, Astop = 50 dB`


Filter Design & Analysis Tool - [untitled.fda]
File Edit Analysis Targets View Window Help
Current Filter Information
Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed
Store Filter ...
Filter Manager ...
Filter Specifications
Mag. (dB)
0
Apass
Astop
0
Fpass
Fstop
Fs/2
f (Hz)
Response Type
Lowpass
Highpass
Bandpass
Bandstop
Differentiator
Design Method
IIR
Butterworth
FIR
Equiripple
Filter Order
Specify order: 10
Minimum order
Options
Density Factor: 20
Frequency Specifications
Units: Hz
Fs: 48000
Fpass 9600
Fstop 12000
Magnitude Specifications
Units: dB
Apass 1
Astop 80
Design Filter
Ready

Second Order Cascaded Sections

- Conversion to Second Order Sections before quantization is highly desired
- Quantization of coefficient effects only the conjugate pole pair



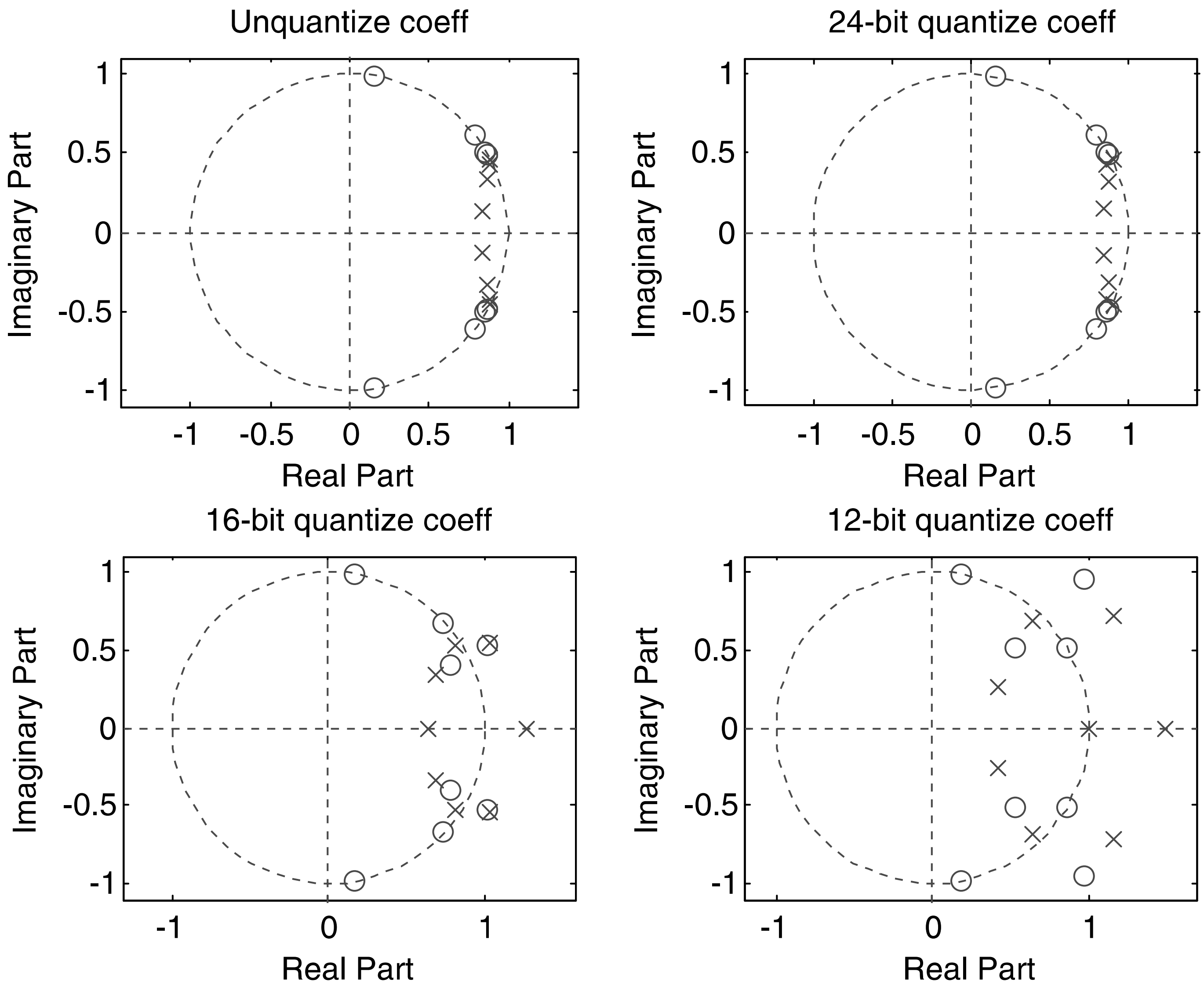


Figure 3.25 Effect of coefficient quantization on stability of the system. The system is unstable for 16-bit and 12-bit quantization as some of its poles are outside the unit circle

FIR Filter Quantization

- Stability is not a concern
- Quantization affects frequency response

$$h_Q(n) = h(n) + \Delta h(n)$$

$$H_Q(e^{j\omega}) = \sum_{n=0}^{M} \left(h(n) + \Delta h(n)\right) e^{j\omega n}$$

$$H_Q\left(e^{j\omega}\right) = H\left(e^{j\omega}\right) + \sum_{n=0}^{M} \Delta h(n) e^{j\omega n}$$