# ETISB
## Embedded Signal Processing

# Indhold

# Introduction, number formats and Blackfin

## 1.1 Lektion 30-01-2018

1. Course introduction
2. Typical embedded system
3. Number formats (fixed- and floating-point)
4. Conversion between different number formats
5. (Blackfin) DSP Architecture
6. Software development flow

- ESP Chapter 1.1 + 1.2
- ESP 5.1 + 5.2.1
- ESP Chapter 6.1.1 (only p.217-p.222) and 6.1.3 - 6.1.5

### 1.1.1 Typical embedded system

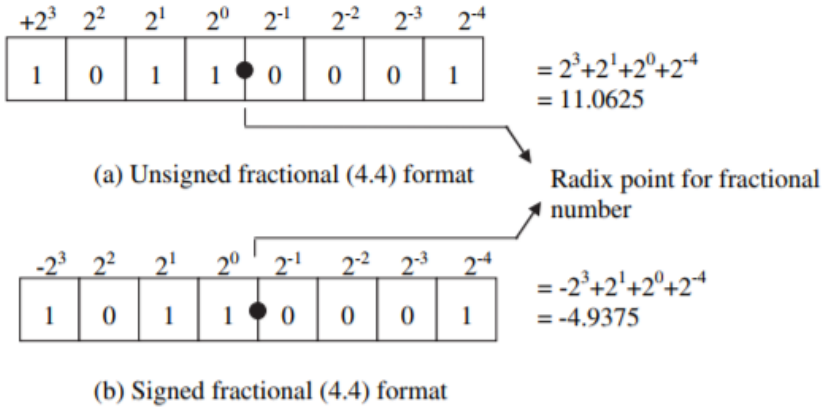
- **Dedicated functions:** Embedded systems usaully execute a specific task repeatedly.
- **Tight constraints:** There are many constraints in designing an embedded system, such as; cost, processing speed, size, power consuption.
- **Reactive and real-time performance:** Many embedded systems must continously react to changes of the system's input.

### 1.1.2 Number formats (fixed- and floating-point)

- Fixedpoint
- Floatingpoint
- Blockfloatingpoint

**Fixed-point**

- Binary data format - signed and unsigned
  - The 2's complement format is the most popular signed number in DSP processors.
  - Most DSP processors support both integer and fractional data formats.
    - In an integer format, the radix point is located to the right of the least significant bit (LSB).
    - In a fractional number format, the radix point is located within the binary number.
      - The number to the right of the radix point assumes a fractional binary bit, with a weighting of $2^{-p}$ where the lowest fractional increment is $2^4$ (or 0.0625) in 1.1.
      - For the number to the left of the radix point, the weighting increases from $2^q$. The weighting of the MSB (or sign bit) depends on whether the number is signed or unsigned.
    - (N.M) notation
      - N is the number of bits to the left of the radix point (integer part).
      - M is the number of bits to the right of the radix point (fractional part).
      - The symbol "." represents the radix point.
      - Total number of bits in the data word is B = N + M.




Figur 1.1: Example of 8-bit binary data formats for a fractional number.

- Dynamic Ranges and Precisions
  - The maximum positive number in (1.15) format is $12^{15}$(= 0.999969482421875) (0x7FFF).
  - The minimum negative number in (1.15) format is 1 (0x8000).
  - The 1.15 format has a **dynamic range** of [+0.999969482421875 to 1]
    - Numbers exceeding this range cannot be represented in 1.15 format.
  - The smallest increment (or precision) within the (1.15) format is $2^{15}$.
- Scaling Factors
  - A number in (N.M) format cannot be represented in the programs because most compilers and assemblers only recognize numbers in integer or (16.0) format.
    - Convert the fractional number in (N.M) format into its integer equivalent.
    - Its radix point must be accounted for by the programmers.
      - To convert a number 0.6 in (1.15) format to its integer representation, multiply it by $2^{15}$ (or 32 768) and round the product to its nearest integer to become 19 661 (0x4CCD).

In table 1.2 all 16 possible (N.M) formats for 16-bit numbers. Different formats give different dynamic ranges and precisions. There is a trade-off between the dynamic range and precision.

- As the dynamic range increases, precision becomes coarser.

| Format (*N.M*) | Largest Positive Value (0x7FFF) | Least Negative Value (0x8000) | Precision (0x0001) |
|---|---|---|---|
| (1.15) | 0.999969482421875 | −1 | 0.00003051757813 |
| (2.14) | 1.99993896484375 | −2 | 0.00006103515625 |
| (3.13) | 3.9998779296875 | −4 | 0.00012207031250 |
| (4.12) | 7.999755859375 | −8 | 0.00024414062500 |
| (5.11) | 15.99951171875 | −16 | 0.00048828125000 |
| (6.10) | 31.9990234375 | −32 | 0.00097656250000 |
| (7.9) | 63.998046875 | −64 | 0.00195312500000 |
| (8.8) | 127.99609375 | −128 | 0.00390625000000 |
| (9.7) | 255.9921875 | −256 | 0.00781250000000 |
| (10.6) | 511.984375 | −512 | 0.01562500000000 |
| (11.5) | 1,023.96875 | −1,024 | 0.03125000000000 |
| (12.4) | 2,047.9375 | −2,048 | 0.06250000000000 |
| (13.3) | 4,095.875 | −4,096 | 0.12500000000000 |
| (14.2) | 8,191.75 | −8,192 | 0.25000000000000 |
| (15.1) | 16,383.5 | −16,384 | 0.50000000000000 |
| (16.0) | 32,767 | −32,768 | 1.00000000000000 |

Figur 1.2: Dynamic ranges and precisions of 16-bit numbers using different formats.

| Format | Scaling Factor ($2^M$) | Range in Hex (fractional value) |
|---|---|---|
| (1.15) | $2^{15} = 32{,}768$ | 0x7FFF (0.99) → 0x8000 (−1) |
| (2.14) | $2^{14} = 16{,}384$ | 0x7FFF (1.99) → 0x8000 (−2) |
| (3.13) | $2^{13} = 8{,}192$ | 0x7FFF (3.99) → 0x8000 (−4) |
| (4.12) | $2^{12} = 4{,}096$ | 0x7FFF (7.99) → 0x8000 (−8) |
| (5.11) | $2^{11} = 2{,}048$ | 0x7FFF (15.99) → 0x8000 (−16) |
| (6.10) | $2^{10} = 1{,}024$ | 0x7FFF (31.99) → 0x8000 (−32) |
| (7.9) | $2^9 = 512$ | 0x7FFF (63.99) → 0x8000 (−64) |
| (8.8) | $2^8 = 256$ | 0x7FFF (127.99) → 0x8000 (−128) |
| (9.7) | $2^7 = 128$ | 0x7FFF (511.99) → 0x8000 (−512) |
| (10.6) | $2^6 = 64$ | 0x7FFF (1,023.99) → 0x8000 (−1,024) |
| (11.5) | $2^5 = 32$ | 0x7FFF (2,047.99) → 0x8000 (−2,048) |
| (12.4) | $2^4 = 16$ | 0x7FFF (4,095.99) → 0x8000 (−4,096) |
| (13.3) | $2^3 = 8$ | 0x7FFF (4,095.99) → 0x8000 (−4,096) |
| (14.2) | $2^2 = 4$ | 0x7FFF (8,191.99) → 0x8000 (−8,192) |
| (15.1) | $2^1 = 2$ | 0x7FFF (16,383.99) → 0x8000 (−16,384) |
| (16.0) | $2^0 = 1$(integer) | 0x7FFF (32,767) → 0x8000h (−32,768) |

Figur 1.3: Scaling factors and dynamic ranges for 16-bit numbers using different formats.

### 1.1.3 Conversion between different number formats

#### Fixed-Point Data Types

The Blackfin C compiler supports eight scalar data types and two fractional data types.

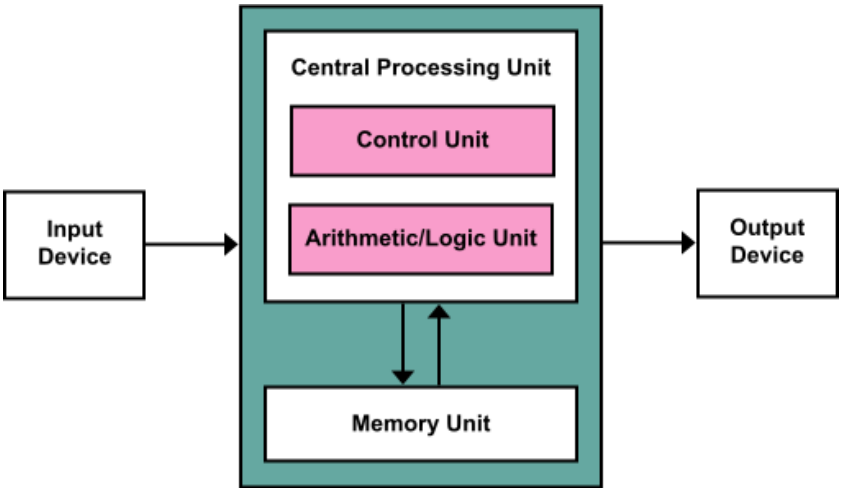
| Type | Number Representation |
|---|---|
| `char` | 8-bit signed integer |
| `unsigned char` | 8-bit unsigned integer |
| `short` | 16-bit signed integer |
| `unsigned short` | 16-bit unsigned integer |
| `int` | 32-bit signed integer |
| `unsigned int` | 32-bit unsigned integer |
| `long` | 32-bit signed integer |
| `unsigned long` | 32-bit unsigned integer |
| `fract16` | 16-bit signed (1.15) fractional number |
| `fract32` | 32-bit signed (1.31) fractional number |

Figur 1.4: Fixed-Point Data Types.

### 1.1.4 (Blackfin) DSP Architecture

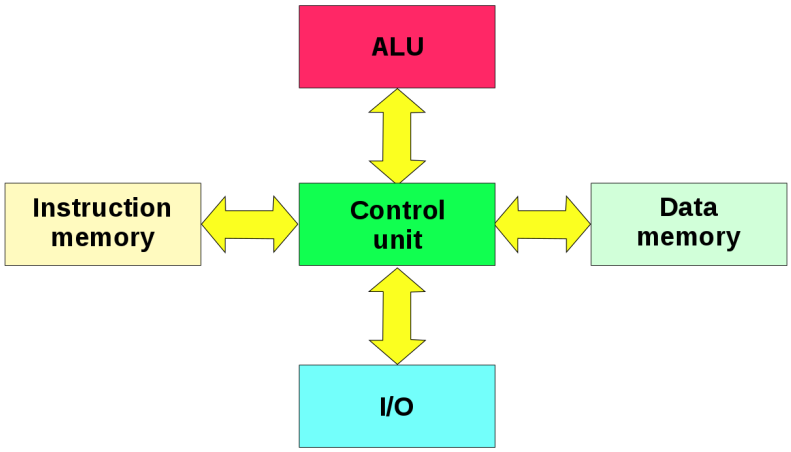
The Von Neumann architecture uses a single memory to hold both data and instructions. In comparison, the Harvard architecture uses separate memories for data and instructions, providing higher speed. The Super Harvard Architecture improves upon Harvard design by adding an instruction cache and a dedicated I/O controller.

### Von Neumann Architecture




Figur 1.5: Von Neumann Architecture (single memory).

### Super Harvard Architecture




Figur 1.6: Super Harvard Achitecture (dual memory, instruction cache, I/O controller)

## 1.1.5 Choice of processor

- DSP
  - Harvard memory architecture (optimized for the operational needs of digital signal processing).

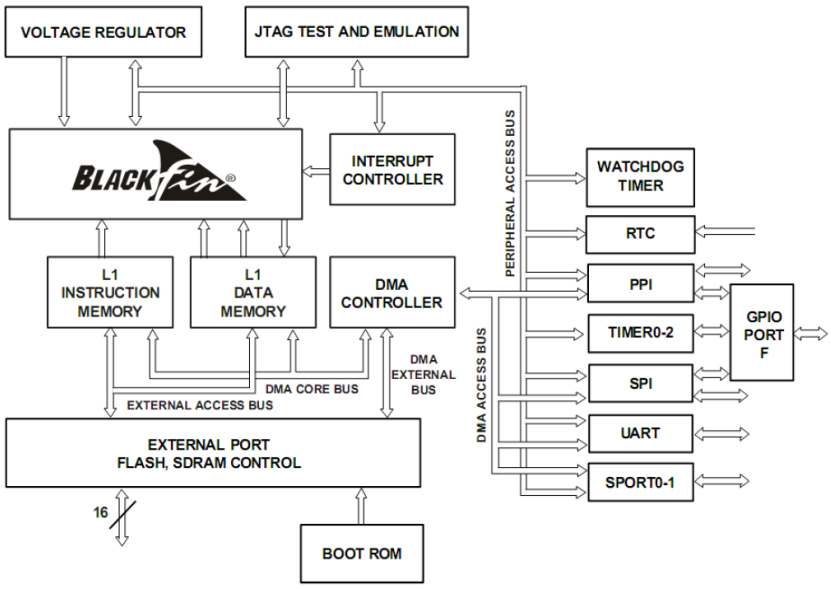
  - Handle real time processing, architecture is specially designed to fetch multiple data at the same time.
  - Can handle floating numbers directly in the data flow path.
  - Calculations are usually carried out by fixed point arithmetic process in order to speed them up.

- Microcontroller
- Generalpurpose CPU
  - Most general-purpose CPU can also execute DSP algorithms successfully, but dedicated DSPs usually have better power efficiency.
- FPGA
- GPU

### 1.1.6 Blackfin

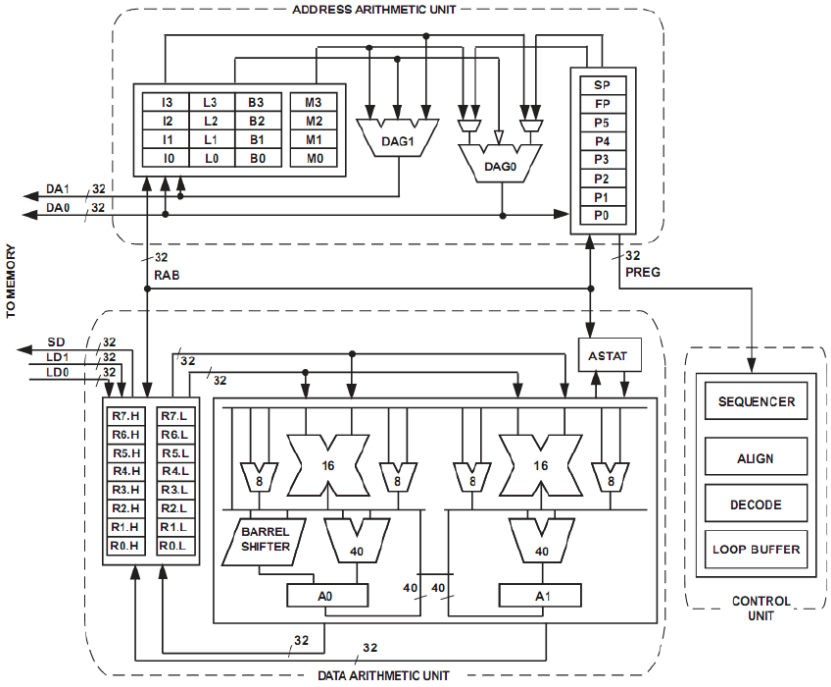
- Blackfin is a *MSA* Processor.
  - MCU and DSP in same processor.
    * Cheap and low power consumption.
- Architecture optimized for multimedia processing (Audio and Video).
- Dataflow oriented tasks (DSP).
- Control flow oriented tasks (MCU).
- Good tools and compiler support.

### BF533 Overview




Figur 1.7: BF533 Overview.

### Blackfin Core




Figur 1.8: BF533 Core, Fixed-point processor.

# Quantization and fixed-point effects

## 2.1 Lektion 06-02-2018

1. ADC Quantization
2. Coefficient- and product-quantization
3. Overflow / underflow and coefficient scaling
4. Notch and peak filters as example

- ESP 6.1.1 (only p. 222 - 229)
- ESP 6.2.2, 6.2.3 (p. 240 - 243)
- ESP 3.4.2 + 3.4,3

### 2.1.1 ADC Quantization

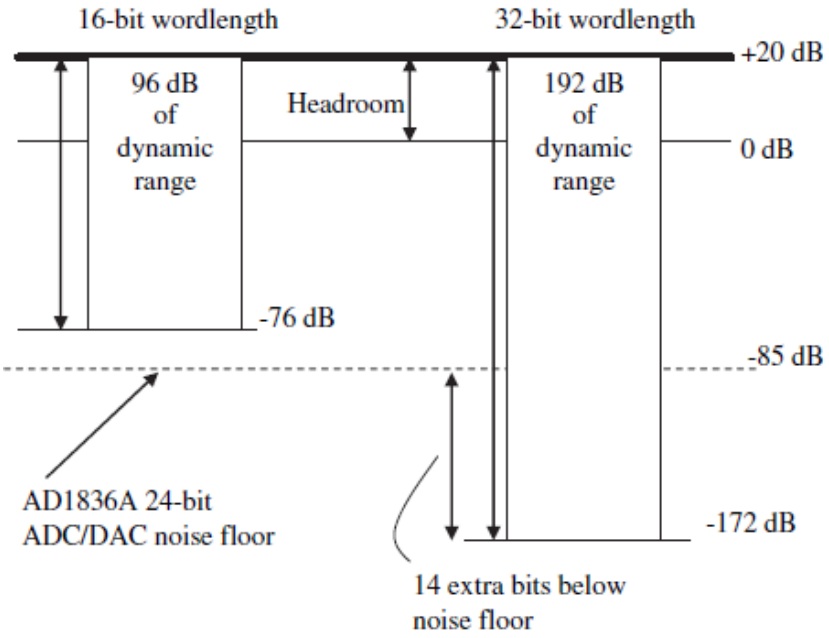
#### Dynamic Range and Signal-to-Quantization Noise Ratio

The dynamic range of a digital signal is defined as the difference between the largest and smallest signal values. If noise is present in the system, the dynamic range is the difference between the largest signal level and the noise floor. When performing signal processing, a higher precision (>16 bits) is normally preferred to maintain this level of dynamic range.

- Dynamic range
  - Typical in relation to "0 dB full scale"

$$dynamicrange = 20\log(\frac{max\ signal\ level}{noise\ floor})[\text{dB}] \tag{2.1}$$

- SQNR [dB] $\approx 6 \cdot n$ dB (n = bits precision)
  - Signal-to-quantization noise ratio = $20\log(2^n)$
  - Different wordlengths of ADC's
    * 16-bit fixed-point 96 dB
    * 24-bit fixed-point 144 dB
    * 32-bit fixed-point 192 dB
- Dynamic [dB] = Peak Level [dB] - Noise Floor [dB]




Figur 2.1: Comparison of different wordlenghts.

### Other sources of quantization errors

Besides analog-to-digital and digital-to-analog quantization noise, there are other sources of quantization errors in digital systems.

- Coefficient quantization
- Computational overflow
- Quantization error due to truncation and rounding

### 2.1.2 Coefficient- and product-quantization

#### Coefficient quantization

When a digital system is designed for computer simulation, the coefficients are generally represented with floating-point format. However, these parameters are usually represented with a finite number of bits in fixed-point processors with a typical wordlength of 16 bits.

Coefficient quantization of an IIR filter can affect pole/zero locations, thus altering the frequency response and even the stability of the digital filter.

- Only specific values possible for eg. filter coefficients
- Only discrete points in pole-zero plane
- Deterministic effect  known in advance

#### Product quantization

Truncation or rounding is used after multiplication to store the result back into the memory.

In general, the distribution of errors caused by rounding is uniform, resulting in quantization errors with zero mean and variance of $\Delta 2/12$. Truncation has a bias mean of $\Delta/2$ and a variance of $\Delta 2/12$.

- Happens after multiplication (=product) of both fixed- and floating-point numbers (unless using double number bits)
- Can be considered as an added noise term (stochastic)
- Limit-cycles may occur

### 2.1.3 Quantization and noise models

Noise is introduced by multiplications between variables, a noise signal is added e(n).

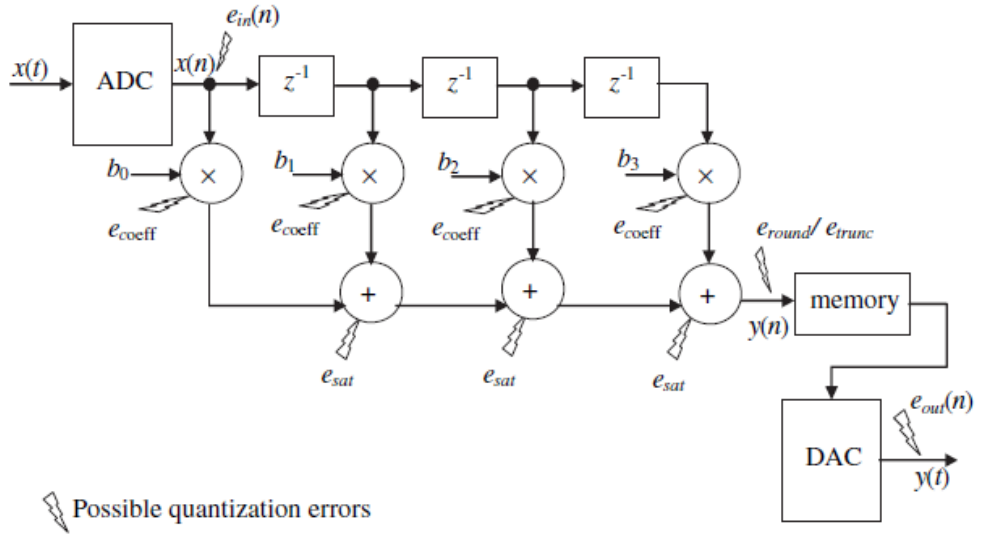
#### Signal flow graphs

- Illustration of implementation
- Useful for illustration of quantization errors

- Different filter-forms available  most try to reduce effects of quantization for fixed-point implementations

Assumed that noise $e$ is equal distributed

- Product quantization $0 \leq |e| \leq 2^{-(m+1)}$




Figur 2.2: Quantization error sources in a 4-tap FIR filter.

### 2.1.4 Overflow / underflow and coefficient scaling

Because of the finite memory/register length, the results of arithmetic operations may have too many bits to be fitted into the wordlength of the memory or register.

- Causing "random"results or no output

A sum of $M$ $B$-bit numbers requires $B + \log_2 M$ bits to represent the final result without overflow.

- eg. If 256 32-bit numbers are added, a total of $32 + \log_2 256 = 40$ bits are required to save the final result.

Overflow/underflow can be avoided with "clever"scaling or, at least, avoided "with a high probability"

- Block floating point
- Exponent checking
- Saturation
- Coefficient scaling

### 2.1.5 Notch and peak filters

- Can be designed by pole-zero placement
- Pole radii and angle determine frequency response
- Common buildingblocks for higher-order IIR

#### Notch filter

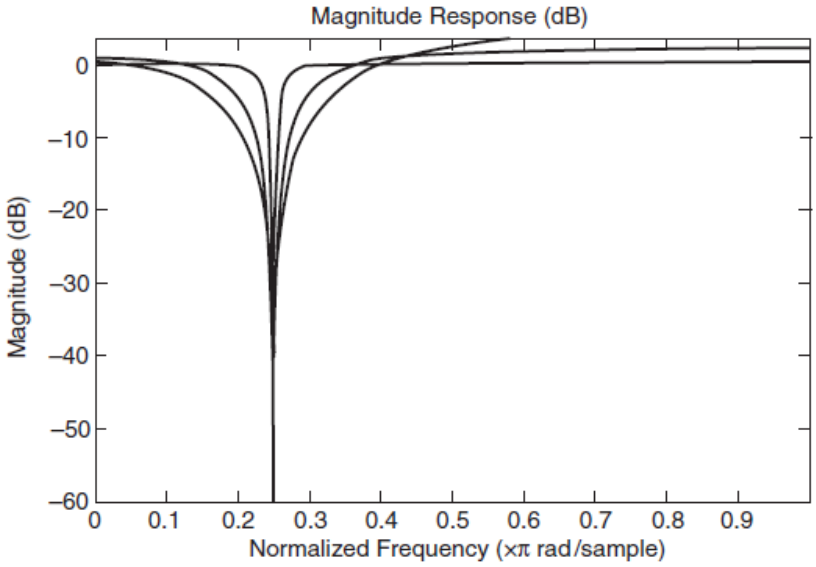
A notch filter contains one or more deep notches in its magnitude response. To create a notch at frequency $\omega_0$, a pair of complex-conjugate zeros on the unit circle at angle $\omega_0$ as $z = e^{\pm j\omega_0}$.

Transfer function of **FIR** notch filter is $H(z) = (1-e^{j\omega_0}z^{-1})(1-e^{-j\omega_0}z^{-1})$. This is a filter of order 2 because there are two zeros in the system. Note that a pair of complex-conjugate zeros guarantees that the filter will have real-valued coefficients.

The magnitude response has a relatively wide bandwidth, which means that other frequency components around the null are severely attenuated. To reduce the bandwidth of the null, we may introduce poles into the system $z_p = re^{\pm j\phi_0}$. $r$ is the radius, $\phi_0$ is the angle of poles.

Transfer function of **IIR** notch filter is $H(z) = \frac{(1-e^{j\omega_0}z^{-1})(1-e^{-j\omega_0}z^{-1})}{(1-r\cdot e^{j\phi_0}z^{-1})(1-r\cdot e^{-j\phi_0}z^{-1})}$.

This is a IIR filter of order 2 because there are two poles in the system.




Figur 2.3: Magnitude responses of notch filter for different values of $r$.

### Peak filter

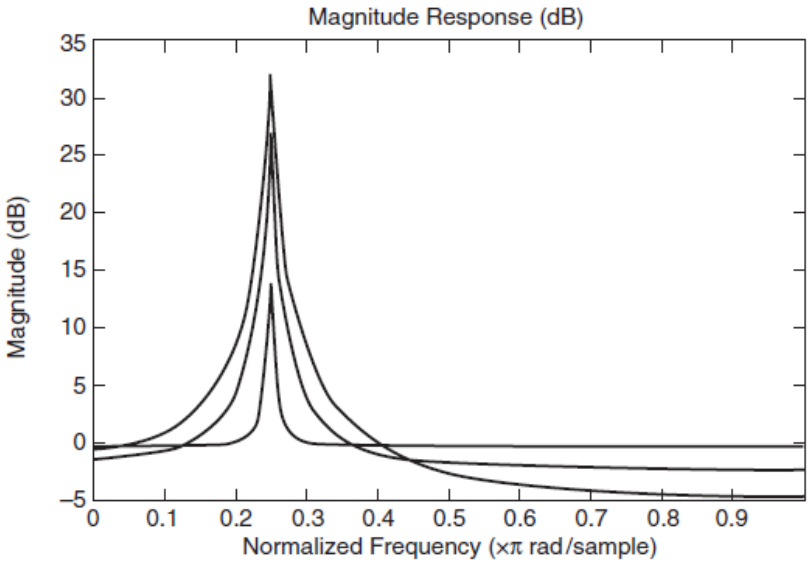
To create a peak (or narrow passband) at frequency $\omega_0$, we may think we can simply follow the example of designing a notch filter by introducing a pair of complex-conjugate poles on the unit circle at angle $\omega_0$. However the resulting second-order IIR filter will be unstable. To solve this problem, we have to move the poles slightly inside the unit circle ($r_p < 1$) as $z_p = r_p e^{\pm j\omega_0}$.

Transfer function of IIR peak filter is $H(z) = \frac{1}{1-r_p e^{j\omega_0} z^{-1})(1-r_p e^{-j\omega_0} z^{-1})}$.

Similar to the second-order IIR notch filter, the magnitude response has a relatively wide bandwidth, which means that other frequency components around the peak will also be amplified. To reduce the bandwidth of the peak, we may introduce zeros into the system.

Transfer function of **IIR** peak filter is $H(z) = \frac{1-r_z e^{j\omega_0} z^{-1})(1-r_z e^{-j\omega_0} z^{-1})}{1-r_p e^{j\omega_0} z^{-1})(1-r_p e^{-j\omega_0} z^{-1})}$.

The poles must be closer to the unit circle as $r_p > r_z$.




Figur 2.4: Magnitude responses of peak filter with poles ($r_p = 0.99$) and different radius of zeros.