Embedded DSP: Introduction to Digital Filters

- Digital filters are a important part of DSP. In fact their extraordinary performance is one of the keys that DSP has become so popular.
  - Audio processing
  - Speech processing (detection, compression, reconstruction)
  - Modems
  - Motor control algorithms
  - Video and image processing

- Historically, electronic designers implemented filters with analog components, as resistors, capacitors and inductors.
- With the develpment of special DSP processor (>1980), designers have a alternative: filter implementation by software on DSPs.
- Since the last 10 years, the designers can choose between the implementation on several technologies as
  - General purpose DSP
  - Gate-Arrays

Embedded DSP: Introduction to Digital Filters

Analog versus digital filters

- Analog filters
  - Electronic components are cheap.
  - Large dynamic range in amplitude and frequency.
  - Real-time.
  - Low stability of resistors, capacitors and inductors due to temperature.
  - Difficult to get the components accuracy as calculated by the formula.
- Digital filters:
  - Better performance than analog filters concerning.
    - Sharp Cut-off in the transition band.
  - DSP filters are programmable. The transfer function of the filter can be changed by exchanging coefficiants in the memory. One hardware design can implement many different, loadable filters by executing a software development process.
  - The charachteristics of DSP filters are predictible.
  - Filter design software packages can accurately evaluate the performance of a filter by simulation before it is implemented in hardware.
  - Alternative digital designs are available by tools to adapt the filter to the application.

Embedded DSP:
Example of analog and digital filter !

Embedded DSP: Introduction to Digital Filters

- Unlike analog filters, the performance of digital filters is not dependant from the environment, such as temperature or voltage.
- In general, complex digital filters can be implemented at lower cost than complex analog filters.

Digital filters are used for two general tasks:

- <u>Separation</u> of different frequency components in signals if contaminated by
  - noisy
  - interference
  - other signal
- <u>Restoration</u> of signals which have been distorted in some ways
  - Improvement and correction of an audio signal recording which is distorted by poor equipment
  - Deblurring of an image from improperly focused lens

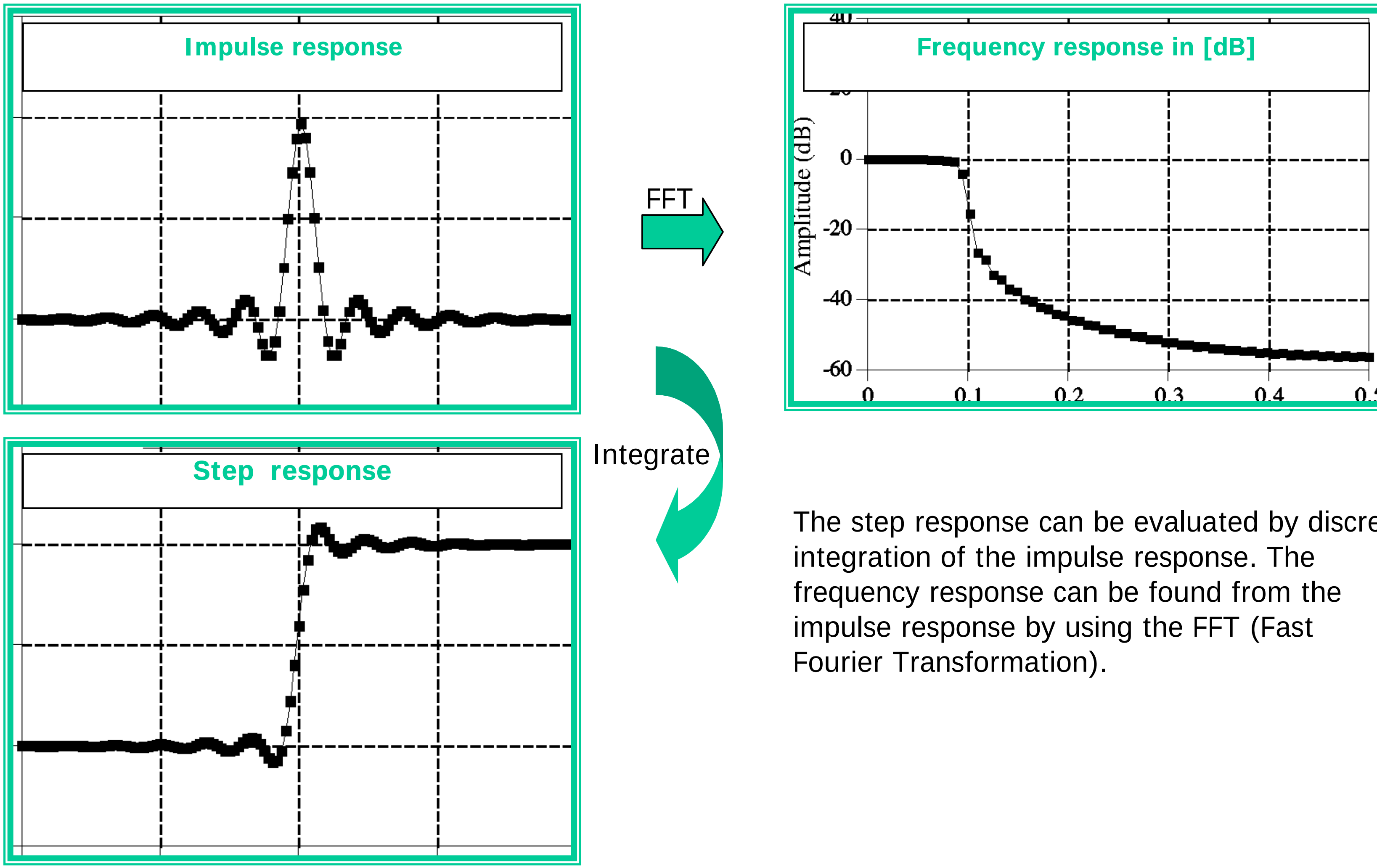
Embedded DSP: Introduction to Digital Filters

- Every linear filter has an
  - Impulse response
  - Step response
  - Frequency response

- Each of these responses contain the same information about the filter, but in different form.

- All representations are important because they describe how the filter will react under various circumstances.

Embedded DSP: Introduction to Digital Filters



The step response can be evaluated by discrete integration of the impulse response. The frequency response can be found from the impulse response by using the FFT (Fast Fourier Transformation).

Embedded DSP: Introduction to Digital Filters

Implementation of a digital filter

By convolution:

- Convolving the input signal with the digital filter impulse response.
- Each sample in the output is calculated by weighting the samples in the input and adding them together.
- All linear filters can be realized by convolution (by a filter kernel)
- FIR-Filter (Finite Impuls Response)

By recursion:

- Extension of the convolution by using previously calculated values from the output, besides the points from the input.
- Made of recursion coefficients.
- IIR-Filter (Infinite Impuls Response)

Embedded DSP: Introduction to Digital Filters

How Information is Represented in Signals

- Information is represented in the time domain
- Information is represented in frequency domain

- The step response describes how information in the time domain is being modified by the system.
- The frequency response shows how information repesented in the frequency domain is being changed.

- It is not possible to optimize a filter for both applications !
- High performance in time domain results in poor performance in the frequency domain, and vica versa.

Embedded DSP: Introduction to Digital Filters

Time Domain Parameters

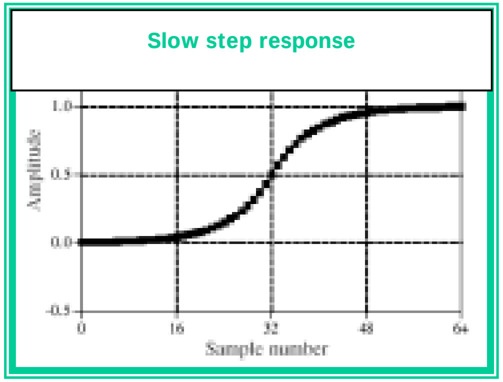
- **Rise-time**: To select events in a signal, the duration of the step response must be shorter than the spacing of the events, looked for.
  - ----> The step response should be as fast as possible for the given system.
  - In general the rise-time time is the time between 10 and 90 %.

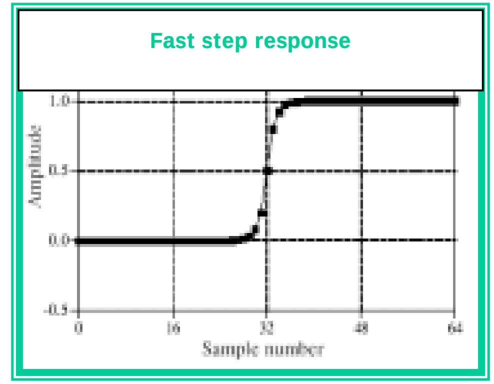
- **Overshoot**: Overshoot is generally a real distortion of the signal, and must therefore be eliminated or at least decreased. Overshoot changes the amplitude of the signal !.

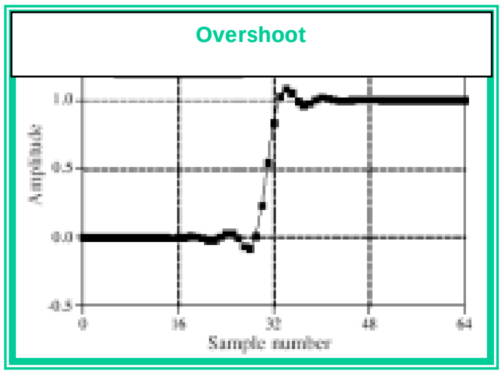
- **Linear phase**: Often symmetry is necessary: rising edges looking similar to the falling edges.

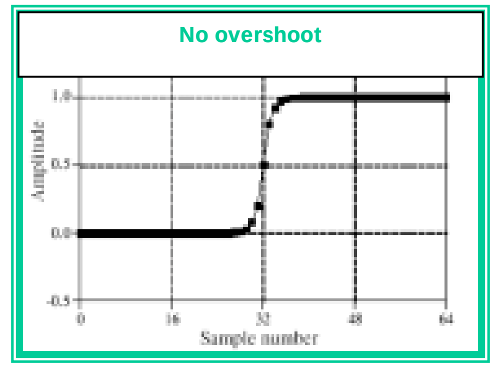
Poor results

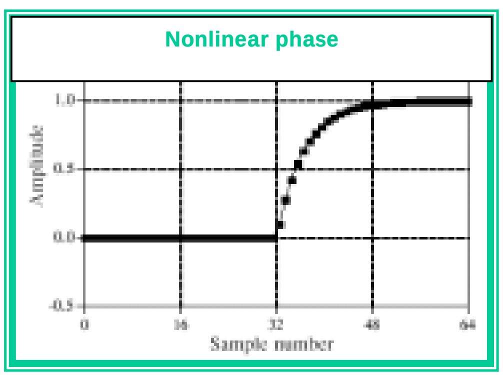
Good results

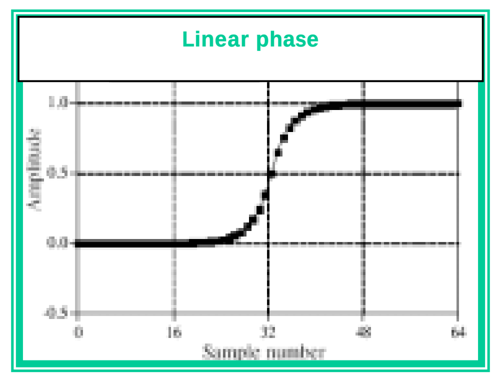






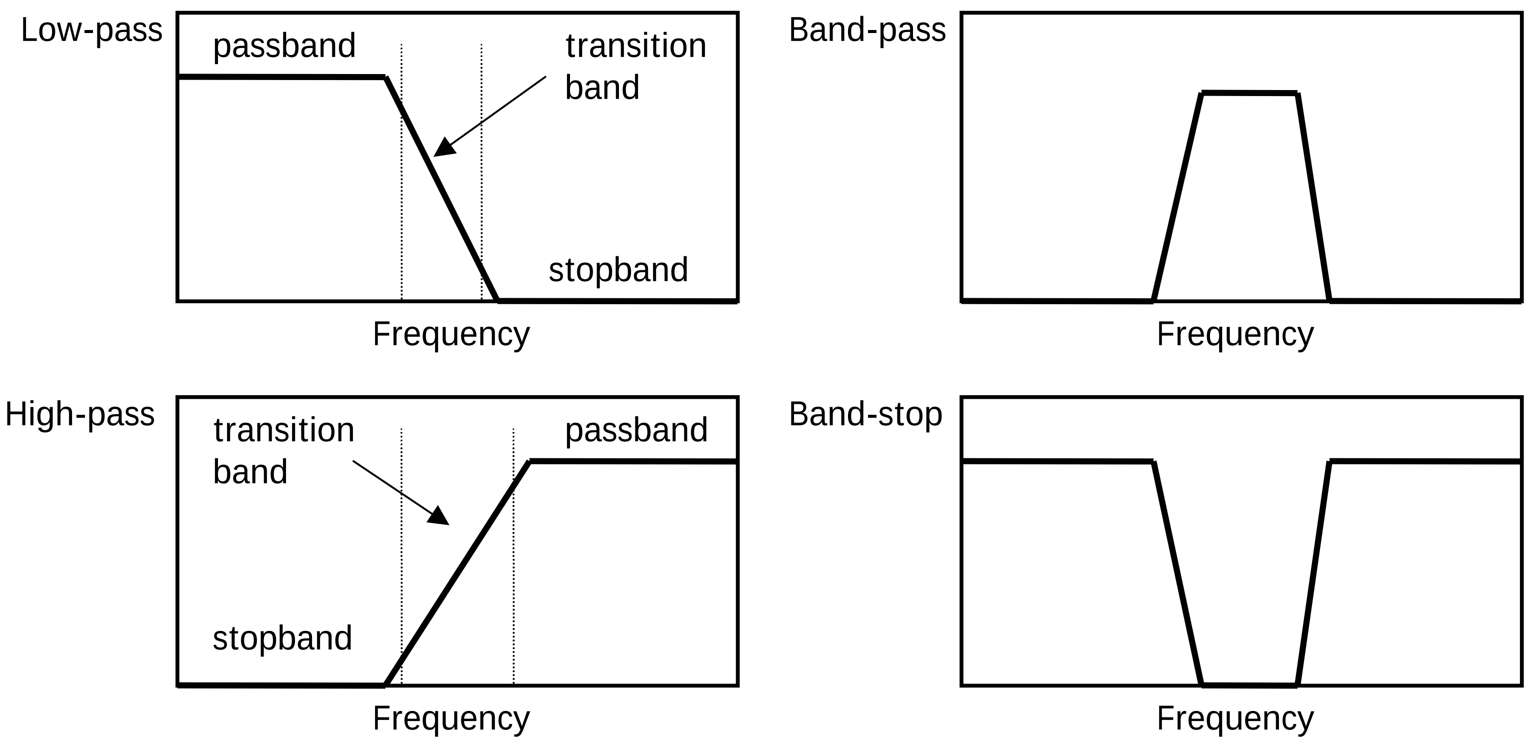






Embedded DSP: Introduction to Digital Filters

- The four basic frequency responses:
  - low-pass, high-pass, band-pass and band-stop.
  - A fast roll-off means that the transition band is very narrow.
  - The division between the passband and the transition band is called the cutoff frequency.



Embedded DSP: Introduction to Digital Filters

Frequency Domain Parameters

- The purpose of this filters is to allow some frequencies to pass unchanged, while other frequencies are fully blocked.
- In analog filter design , the cut-off frequency is normaly defined as the -3dB point (the signal is reduced to 0.707 of the max value).
- In digital filters it is common to use 90%, 70.7% (-3 dB) or 50 % as the cut-off frequency.

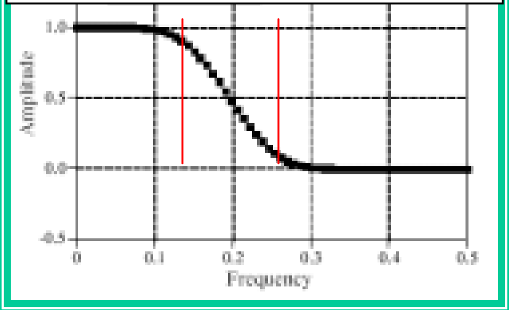
General Guidelines for Frequency Domain Parameters

- To separate closely spaced frequencies, the filter must have a fast roll-off.
- For the passband frequencies it is important, that no passband ripple exists.
- To prevent the stop-band frequencies, it is necessary to have a good stop-band attenuation.
- The phase is not really important for the most applications in the frequency domain.
  - Example: The phase of an audio signal is completely random.
  - If the phase is important in the application, we need a filter with a perfect phase response.

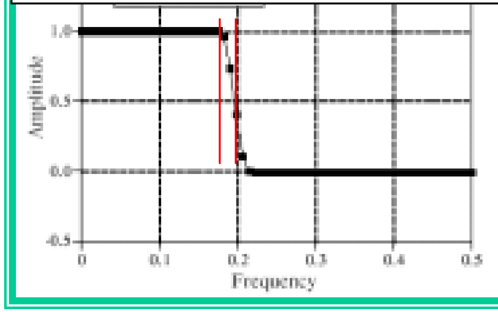
Poor results

Good results

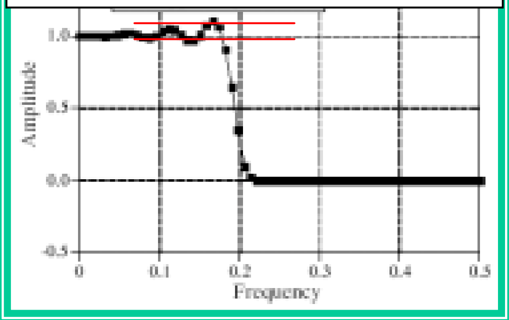
Slow roll-off



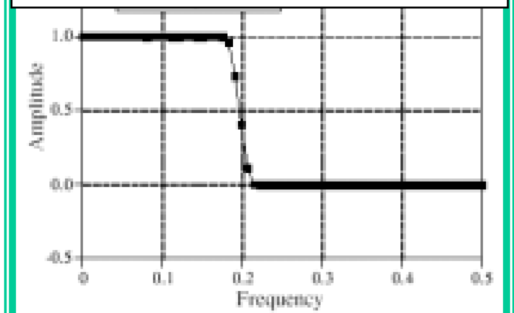
Fast roll-off



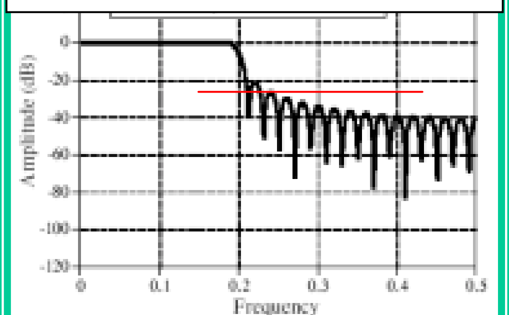
Ripple in pass-band



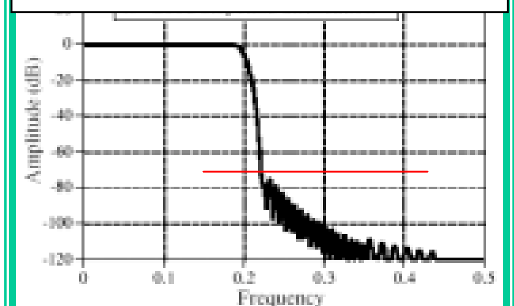
Flat in pass-band



Poor stop-band attenuation



Good stop-band attenuation



Embedded DSP: Introduction to Digital Filters

Filter Classification in time domain, frequency domain and custom

- Time domain filtering : when the information is encoded in the shape of the signals waveform
  - Smoothing
  - DC-removal
  - waveform shaping
  - ...
- Frequency domain filtering : when the information is contained in amplitude, frequency and phase of the component sinusoids.
  - Separation of different frequency bands.
- Custom filtering: when a special action is required and not realizable by the basic filters (low-pass, high-pass, band-pass and band-stop).

Embedded DSP: Introduction to Digital Filters

Filter implemented by

Time Domain	Moving Average	Single pole
Frequency Domain	Windowed-sinc	Chebyshev
Custom Domain	FIR-custom	Iterative design

Embedded DSP: Moving Average Filters

- Moving average is the most common filter in DSP
  - Easy to understand
  - Easy to implement for DSP and FPGA
  - Less computation time
  - FIR filter

- the moving average filter operates by averaging a number of points from the input signal to produce each point in the output signal. In equation form, this is written

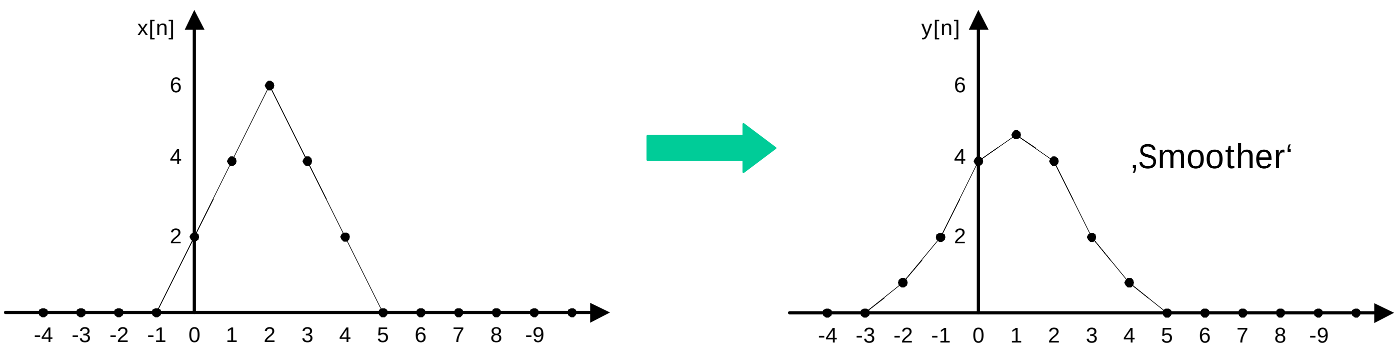
$$y[i] = 1/M \sum_{j=0}^{M-1} x[i+j] \quad \text{or symmetrical form: when } j=-(M-1)/2 \text{ to } (M-1)/2$$

- Optimal filter for the following tasks:
  - Reducing random noise while retaining the sharp step response
  - Therefore usefull for time domain encoded signals, but
- Worst filter concerning frequency encoded signals (no frequency separation capabilities !)
- Relatives of the moving average filter include Gaussian and Blackman.

Embedded DSP: Moving Average Filters

- Example: A three point averager:
- y[n] = 1/3 [x(n) + x(n+1) + x(n+2)]

n	n < -2	-2	-1	0	1	2	3	4	5	n > 5
x[n]	0	0	0	2	4	6	4	2	0	n > 5
y[n]	0	**2 / 3**	2	4	14/3	2	3	5	0	n > 5



Embedded DSP: Moving Average Filters

- Examples:

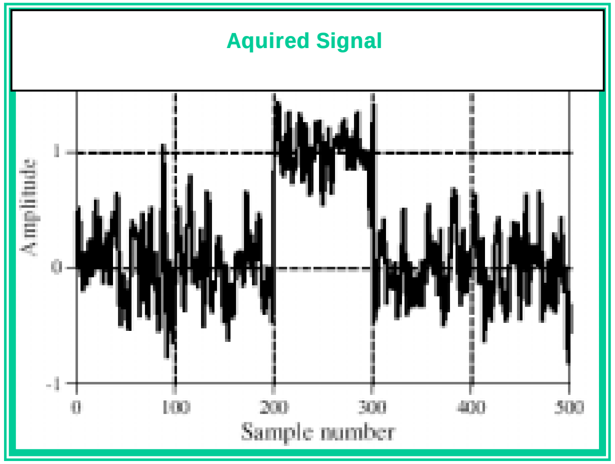
- Smothing filter 5 points:
  - $y[n] = 1/5\ [x(n-2) + x(n-1) + x(n) + x(n+1) + x(n+2)]$
    - Kernel is a rectangular pulse

- In contrast: Least square cubic:
  - $y[n] = 1/35\ [-3\ x(n-2) + 12\ x(n-1) + 17\ x(n) + 12\ x(n+1) - 3\ x(n+2)]$
    - Kernel is a more complex function !

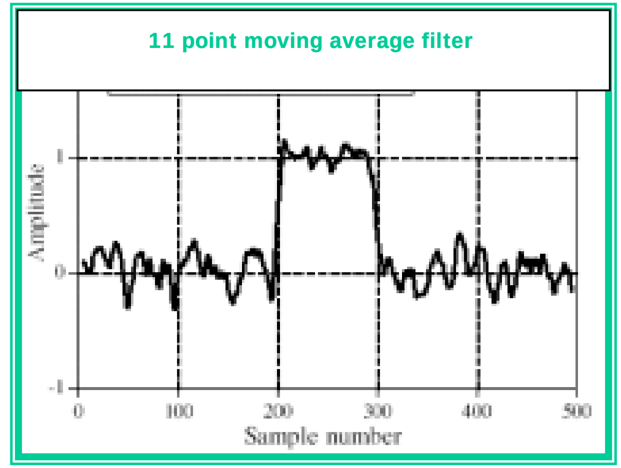
- Symmetrical averaging requires that M be an odd number !
- An unsymmetrical filter produces an time offset of the output !

- Moving average is a convolution using a simple filter kernel: For 5-points the kernel is
  - ....0, 0, 1/5, 1/5, 1/5, 1/5, 1/5, 0, 0, ...   (a rectangular pulse)

Embedded DSP: Moving Average Filters

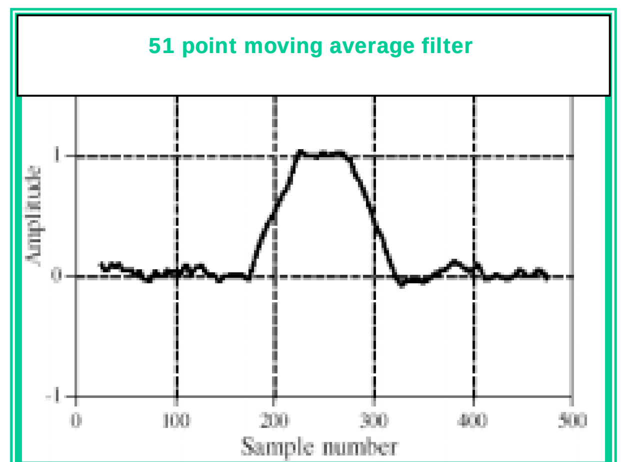


A rectangular pulse with noise



Filtering by a 11 point moving average filter

Processing time ++

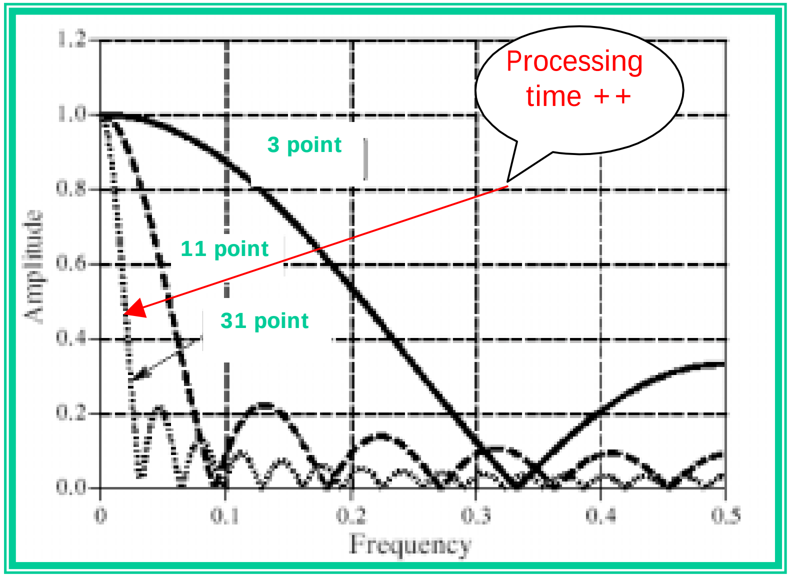


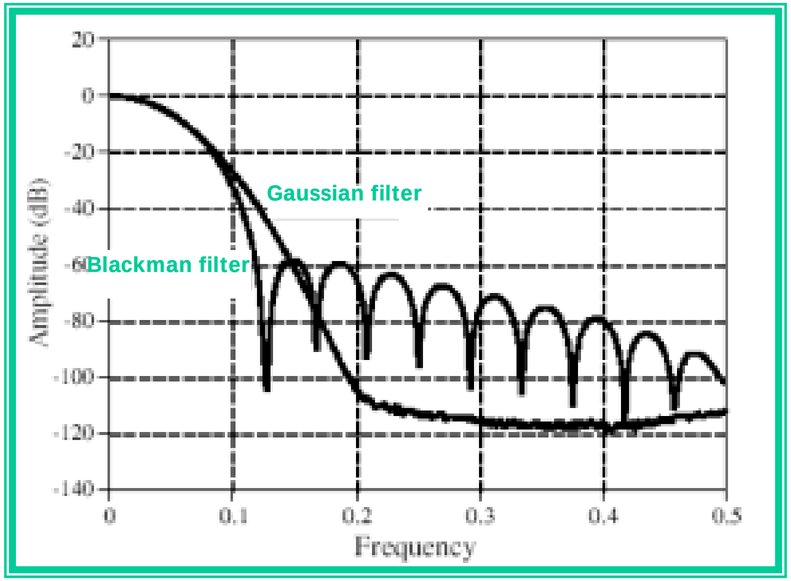
Filtering by a 51 point moving average filter

Increasing the number of points in the filter leads to a better noise performance. But the edges are then less sharp. This filter is the best solution providing the lowest possible noise level for a given sharpness of the edges. The possible amount of noise reduction is equal to the square-root of the number of points in the average (a 16 point filter reduces the noise by a factor of 4)

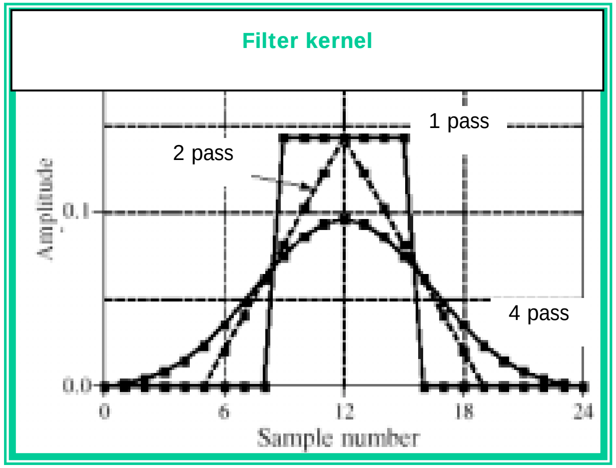
Embedded DSP: Moving Average Filters

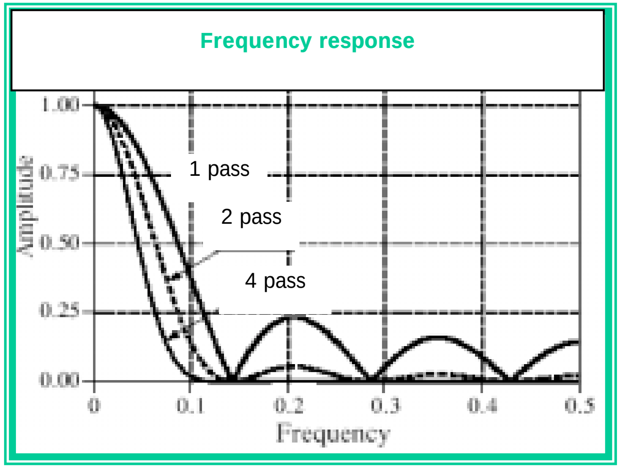
- The frequency response is mathematically described by the Fourier Transform of the rectangular pulse.
- H[f]=sin(Pi f M) / M sin(Pi f)
- The roll-off is very slow, the stopband attenuation is very weak !
- The moving average filter is a good smoothing filter but a bad low-pass-filter !

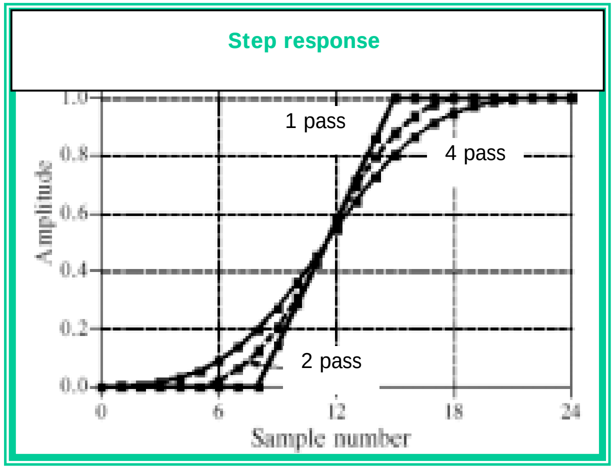


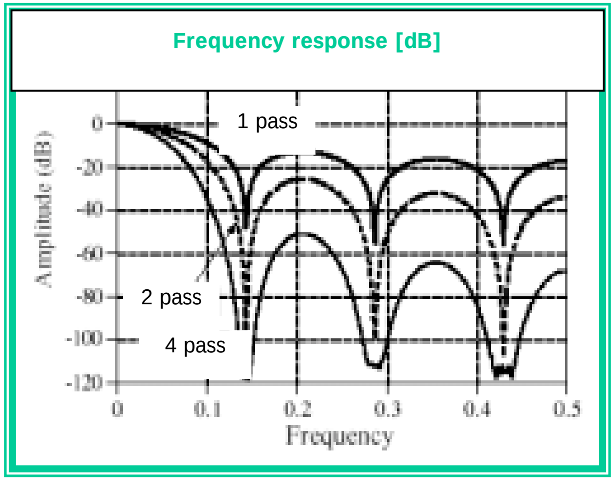


Embedded DSP: Moving Average Filters









- Multiple-pass averaging filter:
- passing the input data several times through a moving average filter.

Embedded DSP: Moving Average Filters

- A great advantage of the moving average filter is that the filter can be implemented with an algorithm which is very fast.
- Example: 9-point moving average filter

- Y[30] = x[26] + x[27] + x[28] + x[29] + x[30] + x[31] + x[32] + x[33] + x[34]

- Y[31]= x[27] + x[28] + x[29] + x[30] + x[31] + x[32] + x[33] + x[34] + x[35]

- x[27] to x[34] must be calculated for y[30] and y[31] !
- If y[27] has already been calculated the most efficient way for y[31] is:

- y[31] = y[30] + x[35] - x[26]

- **y[i] = y[i-1] + x[i+p] - x[i-q];   with: p = (M - 1) / 2, q = p + 1**

Embedded DSP: Windowed-Sinc Filters

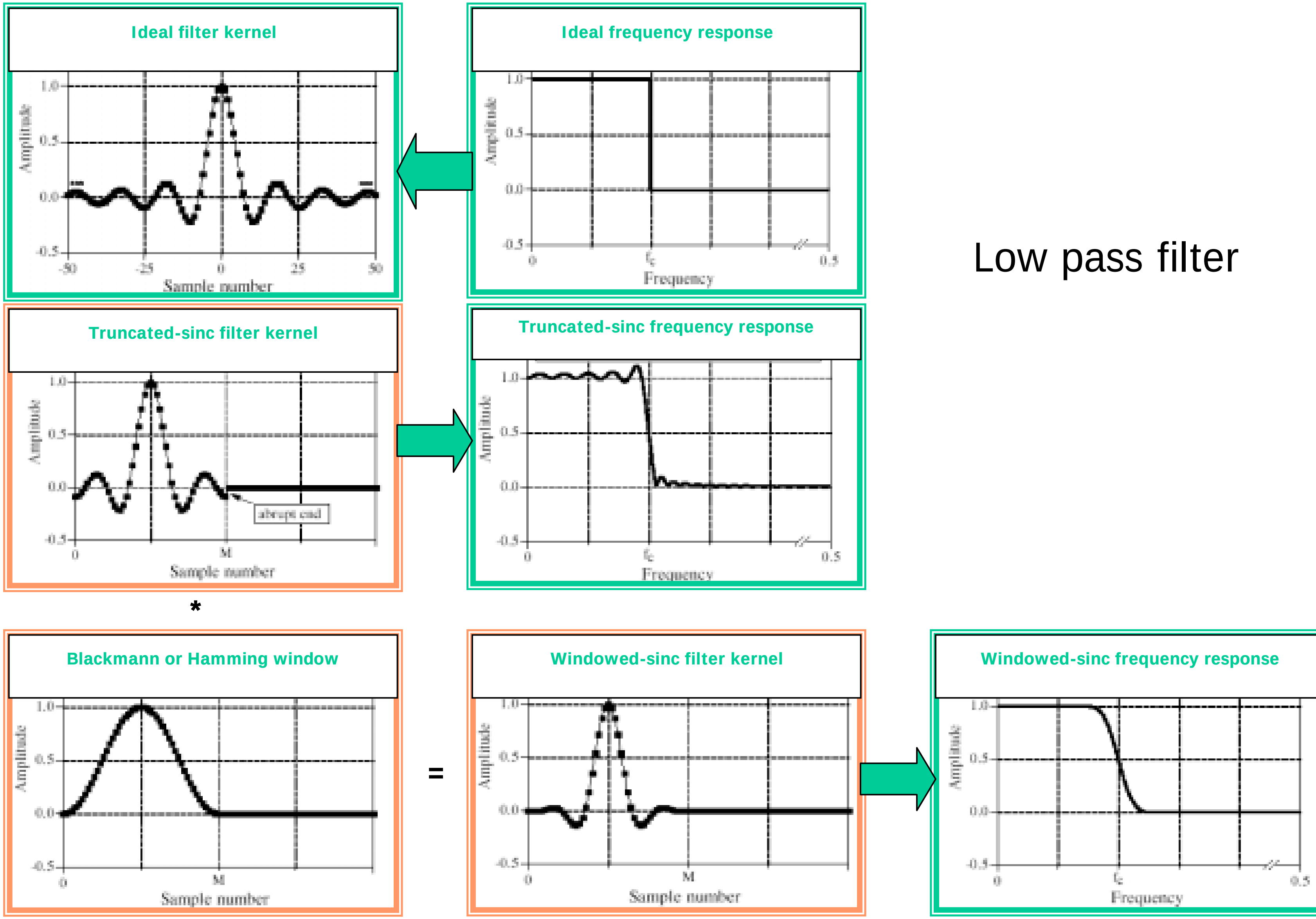
- Windowed-sinc filters are used to separate different bands of frequencies.
- Features are:
  - stable
  - very good overall performance in the frequency domain.
  - but therefore --> poor performance in time domain.
  - if done by standard convolution: needs much computation power.
  - less execution time necessary when realized by FFT-programm.

- Idea: use of a windowed-sinc filter kernel !

$h[i]=\sin(2\ \mathrm{Pi}\ f\ \ i)\ /(\ i\ \mathrm{Pi})$ (has infinite length)

- The basic is the sinc function which is first cutted in time and then smothed with a special time window (Hamming or Blackman).

Low pass filter



Embedded DSP: Windowed-Sinc Filters

- Equation of the Hamming window:
  - h[i] = 0.54 - 0.46 cos(2 Pi i/M)
- Equation of the Hanning window:
  - h[i] = 0.5 - 0.5 cos(2 Pi i/M)
- Equation of the Blackmann window:
  - h[i] = 0.42 – 0.5 cos(2 Pi i/M) + 0.8 cos(4 Pi i/M)
- Bartlet-window == Triangel window
- Rectangular window == No window !

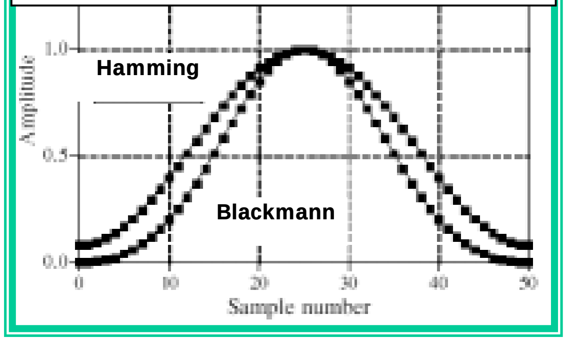
- Important parameters are:
  - Cut-off frequency according to the sampling rate
  - Filter length M which defines the transition bandwith

- M = 4 / BW

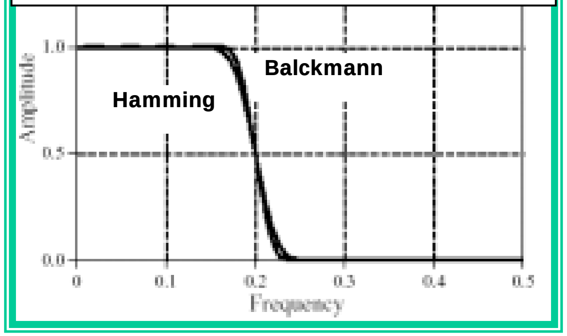
- BW is the width of the transition band (e.g. 99% -> 1 % of the curve)

Embedded DSP: Windowed-Sinc Filters

Blackman and Hamming window

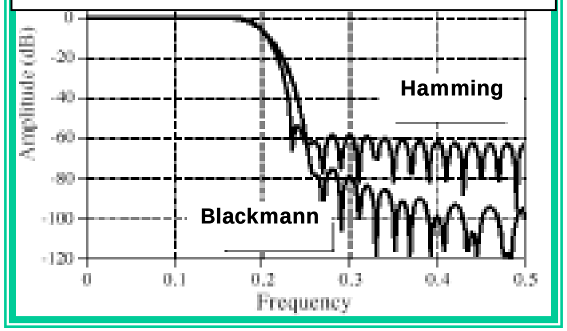


Frequency response



Characteristics of the Blackman and Hamming window:

Frequency response [dB]



Hamming

Embedded DSP: FIR Filters

- An FIR filter is a weighted sum of a limit set of inputs. The equation for a FIR filter is

$$y[n] = \sum_{k=0}^{m-1} b_k x[n-k]$$

x(n-k) is a previous history of inputs

y(n) is the filter output at time n

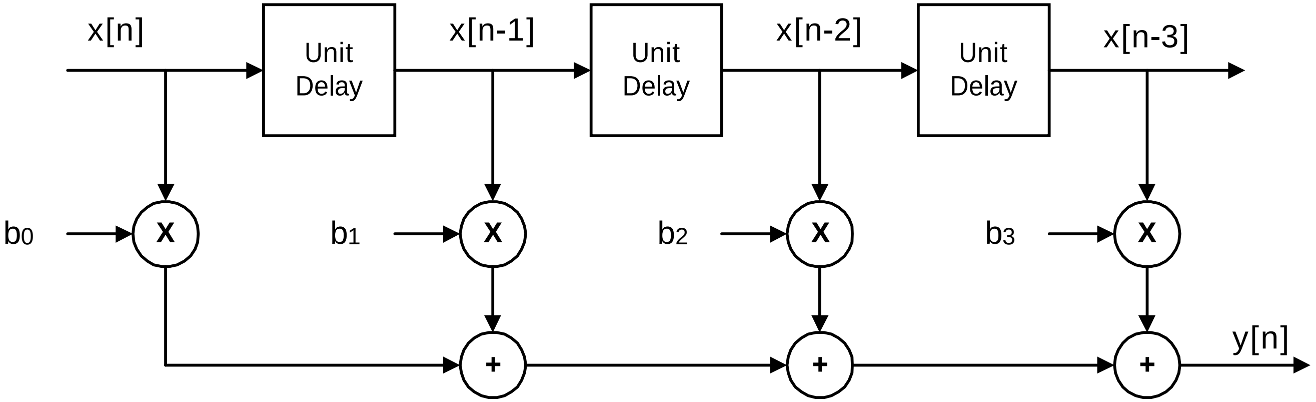
b_k is the vector of filter coefficients

$$y(n) = b_0\, x(n) + b_1 x(n-1) + b_2 x(n-2) + \ldots + b_M x(n-m-1)$$

- For linear phase FIR filters, all coefficients are real and symmetrical
- FIR filters are easy to realize in either hardware (FPGA) or DSP software
- FIR filters are inherently stable
- FIR filtering is a convolution in time: $y[n] = \sum_{k=0}^{\infty} h(k)x[n-k]$

Embedded DSP: FIR Filters

- The basic building-block system are the multiplier, the adder and the unit-delay-operator.



$$y(n) = b_0\, x(n) + b_1 x(n\text{-}1) + b_2 x(n\text{-}2) + b_3 x(n\text{-}3)\ ..\ + b_M x(n\text{-}m\text{-}1)$$

Embedded DSP: FIR Filters

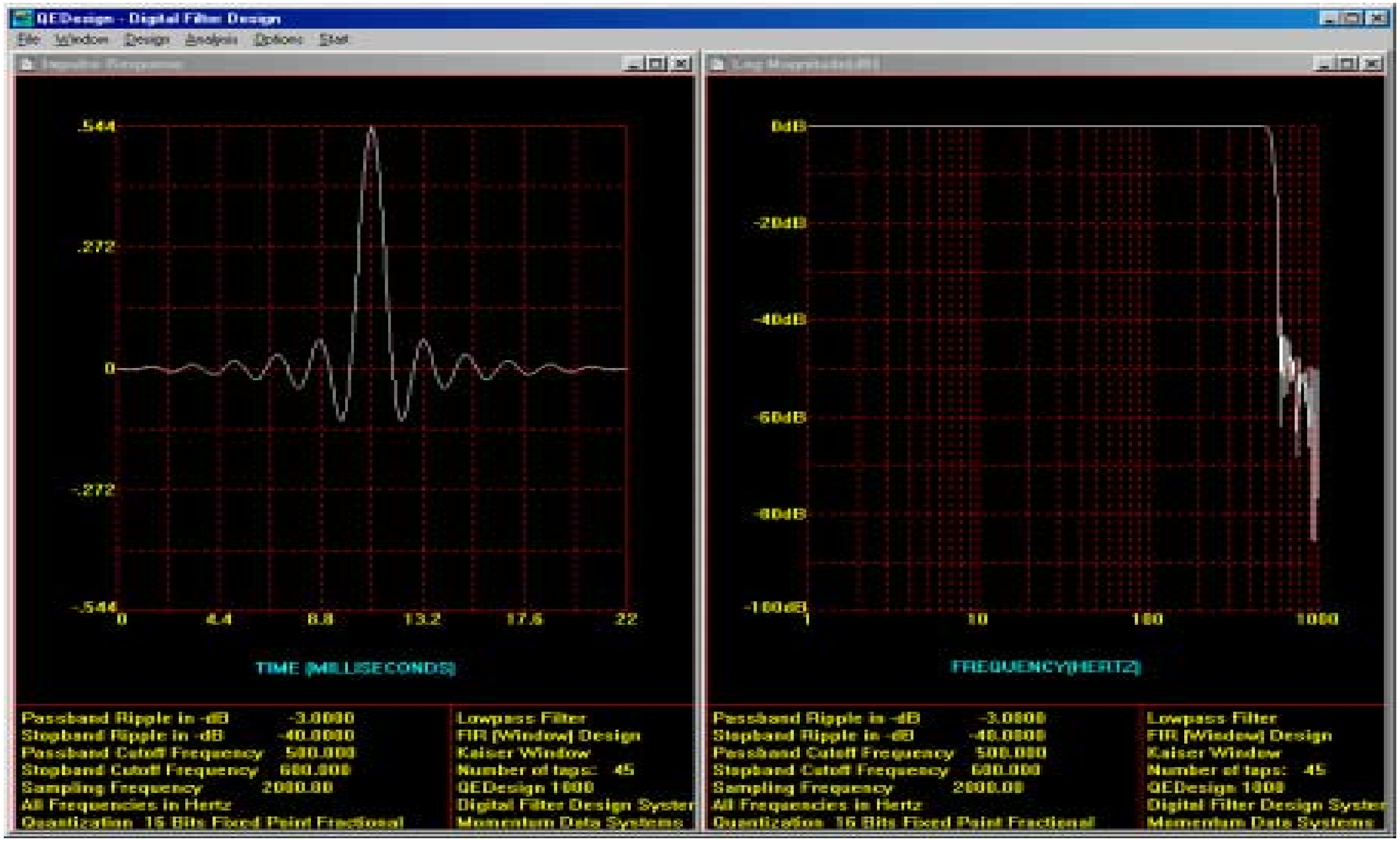
- FIR filter have several advantages that make them more desirable than IIR filters for certain design applications:
  - FIR can be designed to have linear phase. In some applications phase is critical to the output. For example, in video processing, if the phase information is corupted the image becomes fully distorted.
- FIR filters are always stable, because they are made only of zeros in the complex plane.
- Overflow errors are not problematic because the sum of products operation is realized ona finite set of data.
- FIR filters are easy to understand and implement.
- FIR filter costs computation time (dependant from filter length !)

Embedded DSP: FIR Filters

- Design Example by a Software-tool (QEDesign from Momentum Data Systems):
- Lowpass filter spec:
  - 500 Hz passband cutoff frequency, less than 3 dB
  - 600 Hz stopband cutoff frequency, with at least 40 dB attenuation
  - sampling frequency of 200 Hz

rp=3	Passband ripple
rs=40	Stopband ripple
fs = 2000	Sampling frequency
f=[500,600]	Cutoff frequencys
a=[10]	Desired amplitude

FIR: Filter Filter Design with ‚QEDesign‘



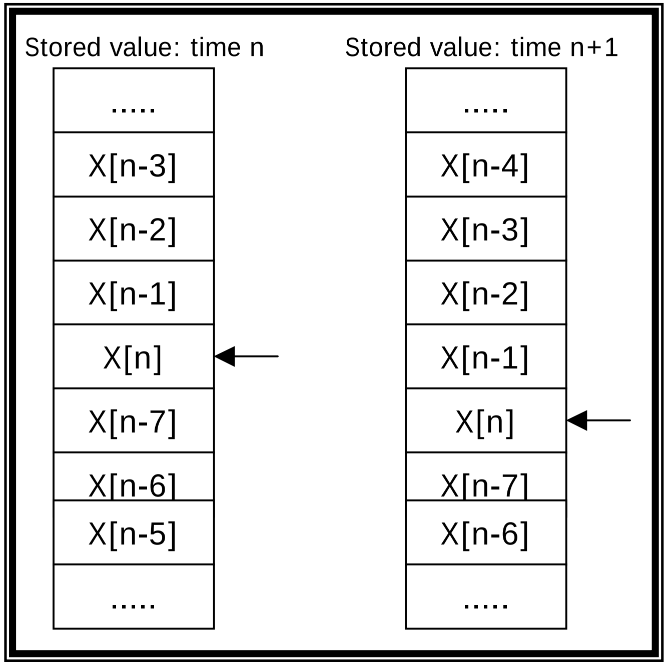
Embedded DSP: Realization of a FIR Filters by DSP

- Steps:
  - Design the filter by a DSP-Design-Tool
  - Create coefficients
  - Put the coefficients in reverse order to the circular buffer
  - call a function e.g. fir.asm

FIR Implementation on DSP SHARC 21062

Embedded DSP: Realization of FIR Filters by DSP

- The solution for fast filter alghorithms is the use of circular buffer technique.
- Circular buffer are used to store the most recent values of a continually sampled signal.
- Four parameters are necessary to manage a circular buffer
- First, there must be a pointer that indicates the start of the circular buffer in memeory.
- Second, there must be a pointer indicating the end of the buffer.
- The step size of the ememory must be specified.
- A pointer to the most recent sample, must be modified after a new sample is fetched.
- There must be a method how this value has to be updated based on the first three values.
- An easy technique for that problem is to use
- a DSP, which is optimized for this task.



Embedded DSP: Realization of a FIR Filters by DSP

- Which basic steps are necessary to implement an FIR filter with circular buffers for both the coefficients and the input signal ?

- Get a sample by the ADC (by interrupt !)
- Detect and manage the interrupt
- Move the sample into the circular input buffer
- Update the pointer for the circular input buffer
- Zero the accumulator
- Control the loop for the main algorithm
  - Get the coefficient from the circular coefficient buffer
  - Update the pointer for the circular coefficient buffer
  - Get the sample from the circular input buffer
  - Update the pointer for the circular input buffer
  - Add the product to the accumulator
- Store the output sample from the accumulator to a output buffer
- Store the output sample from the output buffer to the DAC

```
// Main programm for FIR.ASM
#define SAMPLES         512
#define TAPS            16
.EXTERN                 fir;
.VAR                    coefs[TAPS];            /* circular buffer coefficients */
.VAR                    dline[TAPS];            /* circular  buffer that holds dline */
.VAR                    inbuf[SAMPLES];         /* buffer coefficients */
.VAR                    outbuf[SAMPLES];        /* buffer that holds dline */

    L0=TAPS; B0=dline; M0=1;                    /* circular buffer dline */
    I1=0; B1=inbuf;
    I2=0; B2=outbuf;
    L8=TAPS; B8=coefs;
    call init_fir (db);                         /* all elements of delay line = 0, not shown ! */
    M8=1;
    R0=TAPS;

    lcntr=SAMPLES, do filter until ce;
            call fir(db);                       /* input sample passed in F0, output returned in F0 */
              R1=TAPS-1;
              F0=dm(i1,1);                      /* actual data sample */
    filter: dm(i2,1)=F0;                        /* result to outbuf */
....
....
```

FIR Implementation on DSP SHARC 21062

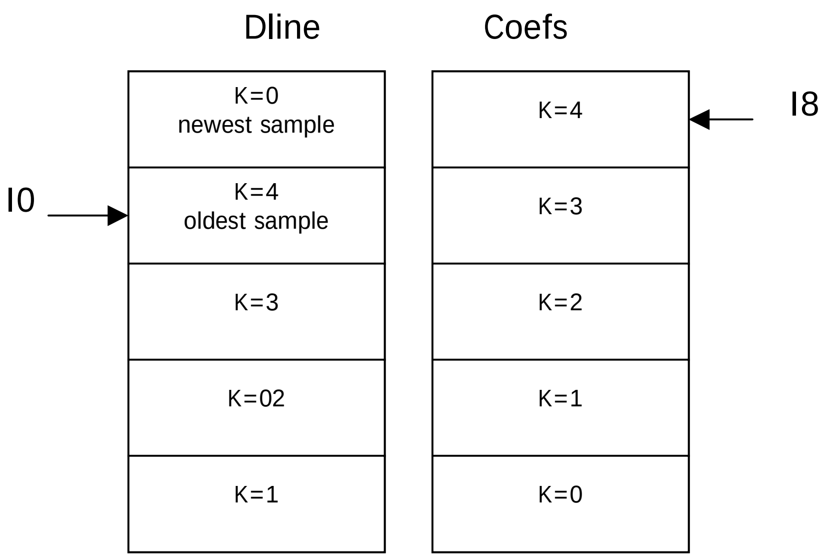
- The actual filtering algorithm is implemented as a loop.
- The loop is executed SAMPLE number of times (once for each input sample) and ends at the label filter.
- The loop calls FIR with a delayed branch. In the two cycles two transfers are performed:
  - A value from the input buffer is placed in F0.
  - R1 is set to a value that is the amount of times that the loop in the FIR mosule has to repeat (M-1).
- After the code in FIR.ASM completes, control returns to main.
  - The instruction at the end of label filter writes the result to the output buffer.

- FIR.ASM
  - The pointer is post modified when the input sample moves to the delay buffer.
  - The coefficients array must rearranged so thet the coefficients assocoiated mit K=max is the first element in the array.
  - Use a tricky XOR to reset R12.
  - The modify instructions moves the delay line pointer to the oldest value in the delay line.
  - The multiplication works on the operands fetched in the previous cycle.
  - The operands for the addition are the results of the multiplication; valid operands for the addition are not generated until the loop executes twice. For the first two iterations of the loop, the code uses dummy operands of zero. The third time through the loop and after, the multiplication produces two valid operands for the addition.

```
.GLOBAL       ___fir;
___fir: R12=R12 xor R12, dm(i0,m0)=F0;              /* set r12 =0 and store input sample in dline */
        R8=R8 xor R8, F0=dm(i0,m0), F4=pm(i8,n8);  /* set r8 = 0 and take data from dline and coef */

              LCNTR=R1, DO macs UNTIL LCE;
macs:            F12=F0*F4, F8=F8+F12, F0=DM(i0,m0), F2=PM(i8,m8);         /* Multifunctions !!! */

              rts;
              F12=FF0*F4, F8=F8+F12;                /* perform mult on last pieces of data and 2nd last */
              F8=F8+F12;                            /* perform last add and store result in F0*/
.ENDSEG;
```



Cycles: 3 + (TAPS-1) + 3 + 2 cache misses=
7 + number TPAS

IIR Filters

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

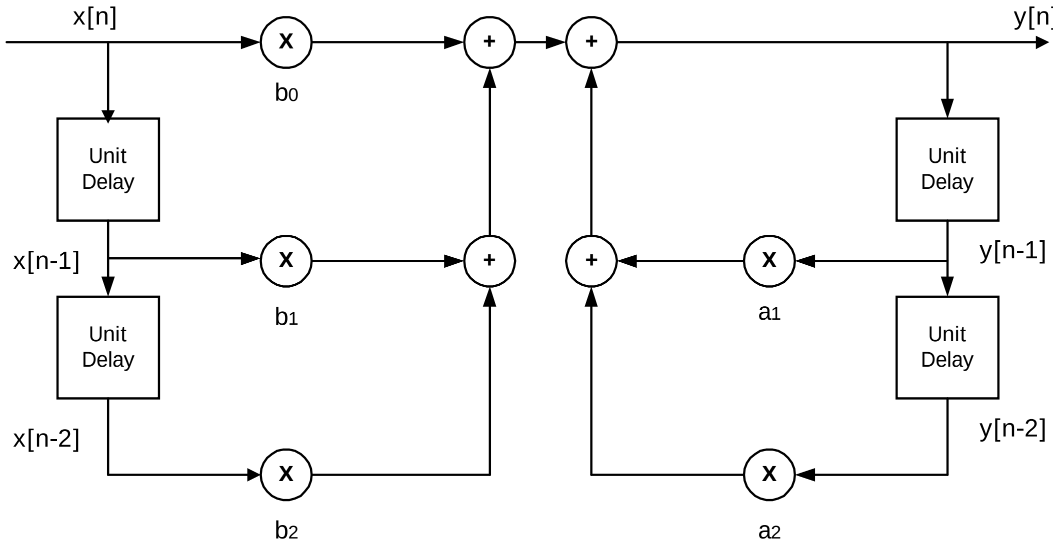
- Advantages
  - Fewer coefficients for sharp cutoff filters.
  - Able to calculate coefficients for standard filter (Bessel, Butterworth, Dolph-Tschebyscheff, Elliptic ....).

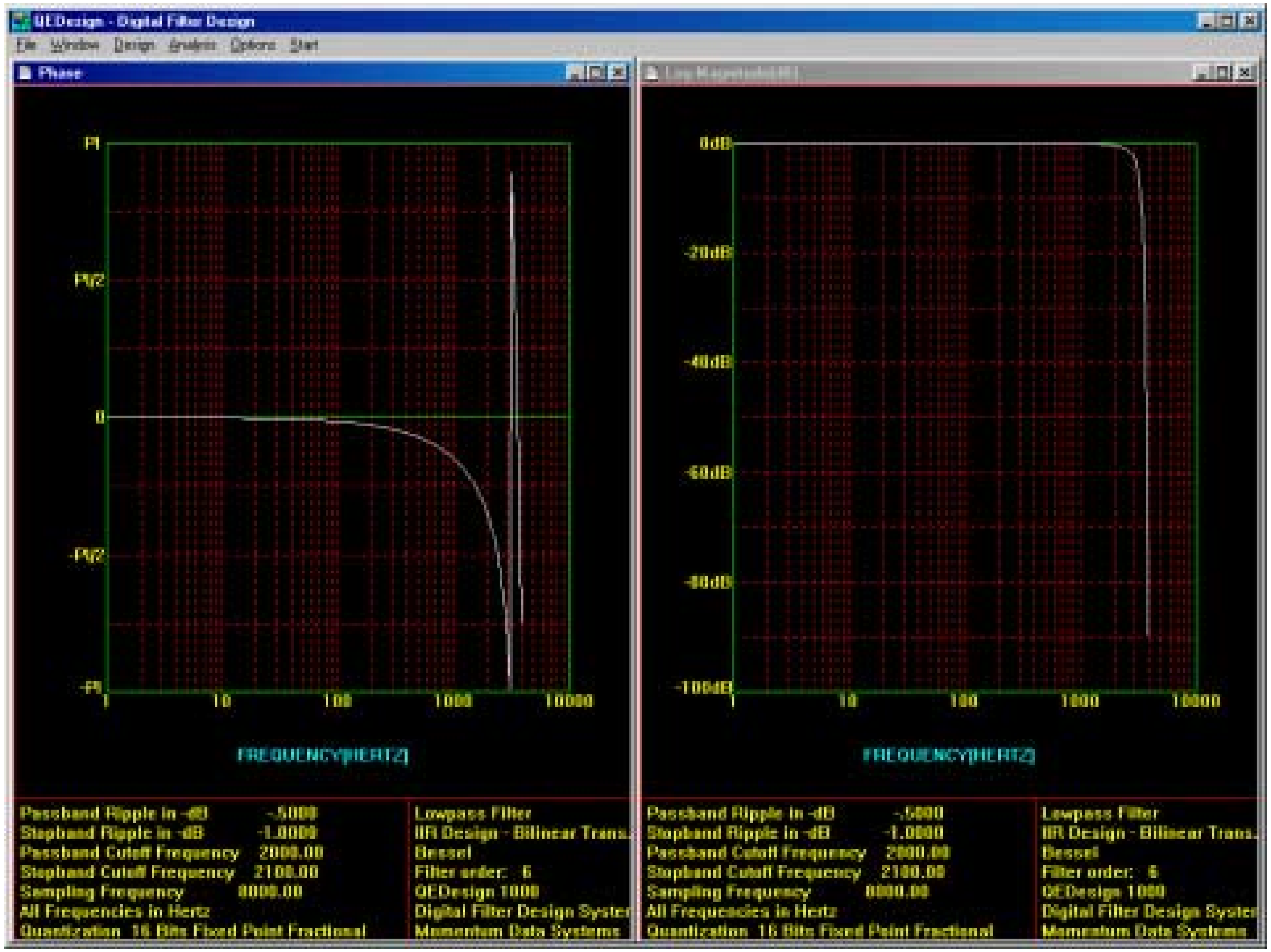
- Disadvantages
  - Existing non-linear phase response.
  - Filter can be unstable: precision of coefficients is important, adaptive filters are difficult to realize.

Embedded DSP: IIR Filters

- Because IIR filters corresponds directly to analog filters, one way to design IIR filters is to create a desired transfer function in the analog domain and then transform it to the z-domain. Then the coefficients of a direct form IIR filter can be calculated from the z-domain equation. The following equation is the direct form of the biquad difference equation of an IIR filter:

$$y(n) = b_0\, x(n) + b_1 x(n\text{-}1) + b_2 x(n\text{-}2) + a_1 y(n\text{-}1) + a_2 y(n\text{-}2)$$





- 6th order IIR (Bessel)
- IIR filter has strong cutoff but extreme non-linear phase
- the 180° phase jumps comes when the response changes sign

Reference:

,The Scientiest and Engineer‘s Guide to
Digital Signal Processing
Second Edition
by Steve Smith

www.DSPguide.com

Next Lecture

- Digital Signal Processors
- Getting Started with DSP