



# Embedded System Architectures

## Digital Signal Processing Concepts and Theory

Maurizio Palesi

Dipartimento di Ingegneria Informatica e delle Telecomunicazioni

University of Catania, Italy

mpalesi@diit.unict.it

<http://www.diit.unict.it/users/mpalesi>

## Contents

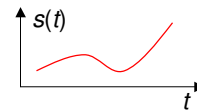
- Introduction to Digital Signal Processing
- Converting analogue signals
  - Aliasing and Quantisation problems
- Time domain processing
  - Correlation and Convolution
- Frequency domain processing
  - Fourier transform and windowing
- Digital filters
  - FIR and IIR filters
- Digital Signal Processing using DSP
  - Special features for arithmetics
  - Addressing modes

# What is a DSP?

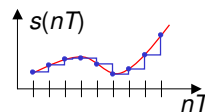
- Digital
  - Operating by the use of discrete signals to represent data in the form of numbers
- Signal
  - A variable parameter by which information is conveyed through an electronic circuit
- Processing
  - To perform operations on data according to programmed instructions
- Digital Signal Processing
  - Changing or analysing information which is measured as discrete sequences of numbers

# Signals

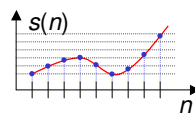
- Continuous-time (analog) signals



- Discrete-time signals



- Digital signals

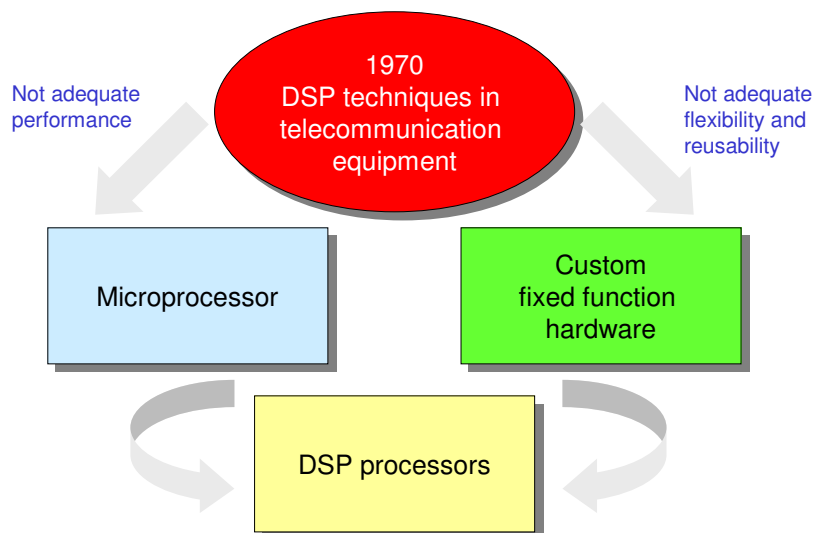


Signal	Time	Amplitude
Analog	Continue	Continue
Discrete	Discrete	Continue
Digital	Discrete	Discrete

## Main Characteristics

- Compared to other embedded computing applications, DSP applications are differentiated by the following
  - Computationally demanding
    - ✓ Iterative numeric algorithms
  - Sensitivity to small numeric errors (audible noise)
  - Stringent real-time requirements
  - Streaming data
  - High data bandwidth
  - Predictable (though often eccentric) memory access pattern
  - Predictable program flow (nested loops)

## DSP Processors

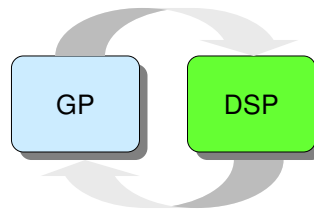


# DSP vs. General Purpose

- DSPs adopt a range of specialized features

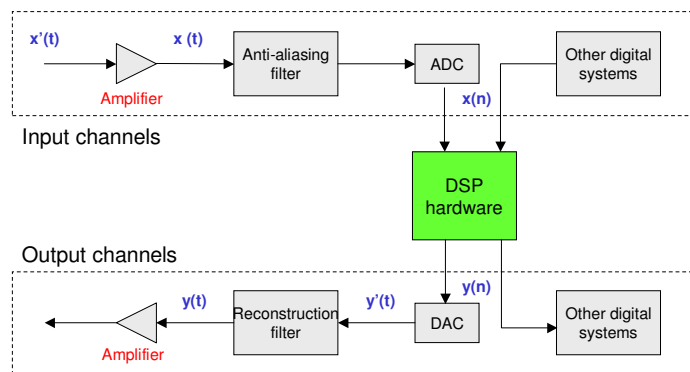
- Single-cycle multiplier
- Multiply-accumulate operations
- Saturation arithmetic
- Separate program and data memories
- Dedicated, specialized addressing hw
- Complex, specialized instruction sets

VLIW, Superscalar, SIMD, multiprocessing, ...

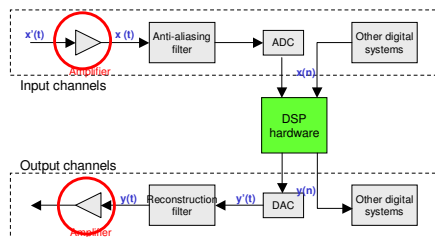


- Today, virtually every commercial 32-bit microprocessor architecture (from ARM to 80x86) has been subject to some kind of DSP-oriented enhancement

# Basic Elements of DSP Systems

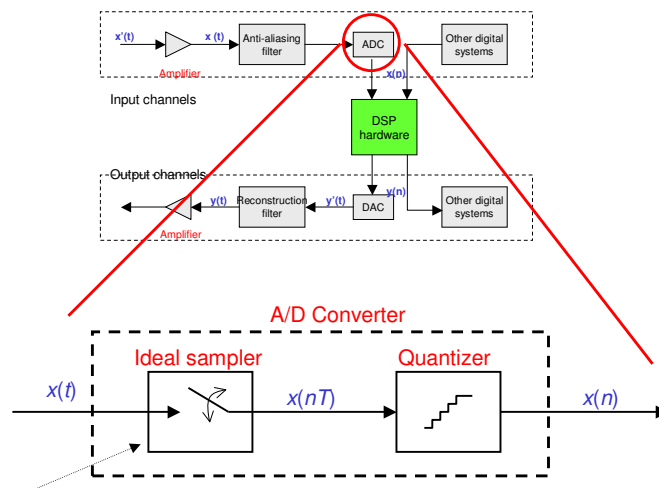


# Amplifier



- $x(t) = g x'(t)$
- The gain  $g$  is determined such that  $x(t)$  has a dynamic range that matches the ADC
- Very difficult in practice
  - $x'(t)$  may be unknown and changing with time
    - ✓ Especially for signals with larger dynamic range (e.g., speech)

# ADC

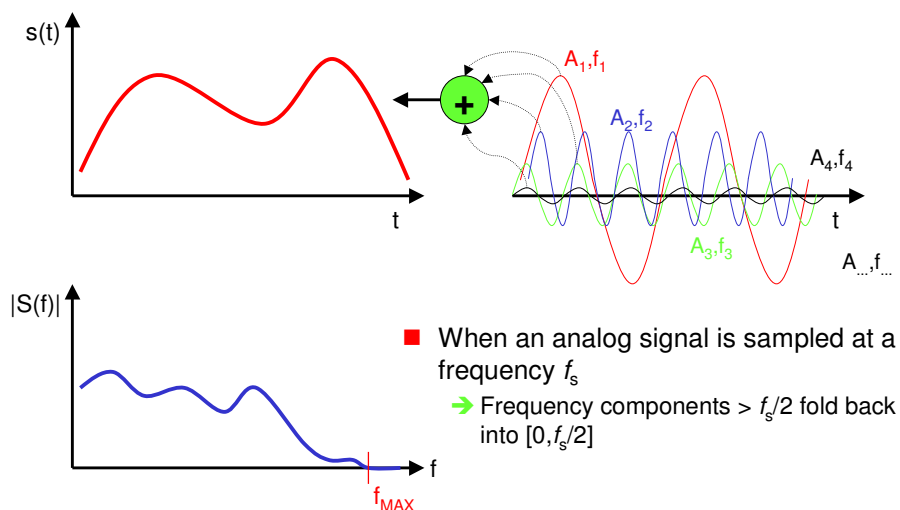


Sample-and-hold circuit

# Sampling

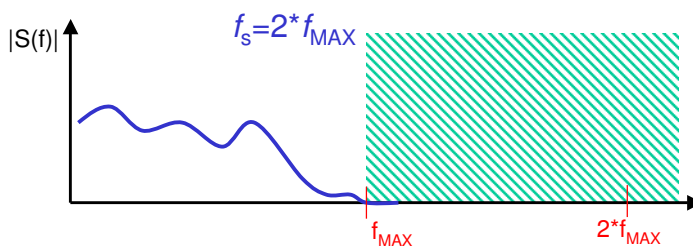
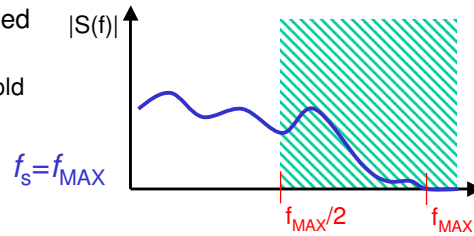
- In order to represent an analog signal  $x(t)$  by  $x(nT)$  accurately, two conditions must be met
  - $x(t)$  must be bandlimited by  $f_M$
  - The sampling frequency,  $f_s$ , must be  $>2f_M$
- Shannon's theorem
  - When the sampling frequency is greater than twice the highest frequency component contained in  $x(t)$ , it can be **perfectly reconstructed** from  $x(nT)$

# Folding Effect (1/2)



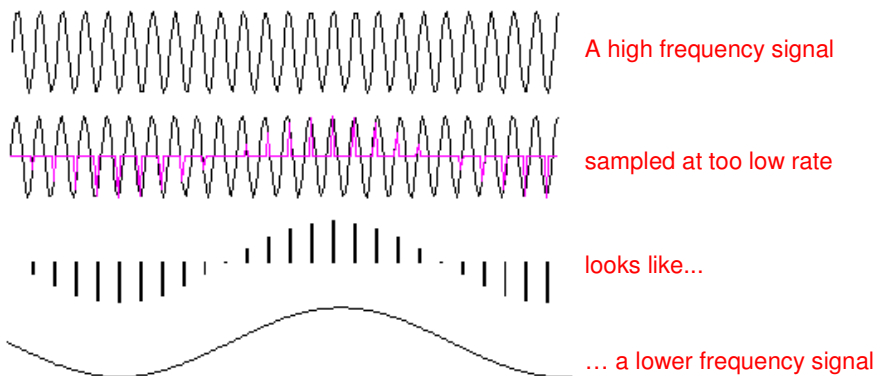
## Folding Effect (2/2)

- When an analog signal is sampled at a frequency  $f_s$ 
  - Frequency components  $> f_s/2$  fold back into  $[0, f_s/2]$



## Aliasing

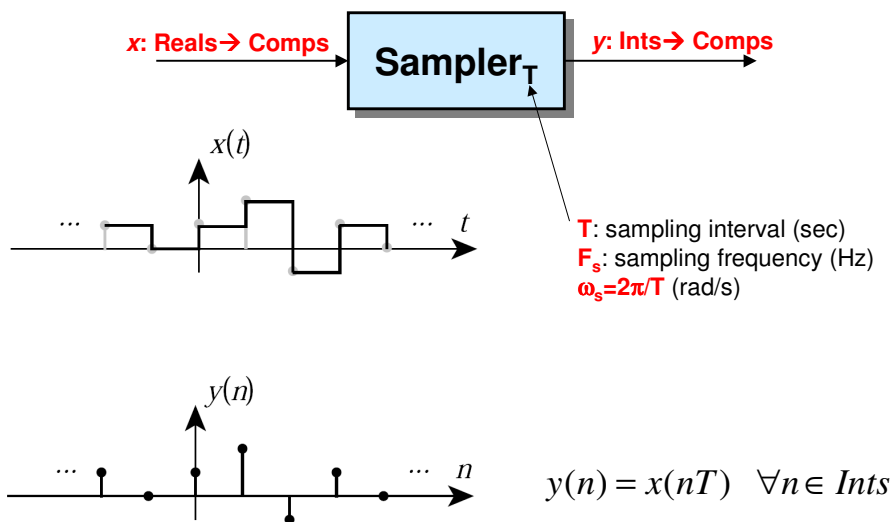
- Some higher frequencies can be incorrectly interpreted
  - **Aliasing problem**: One frequency looks like another



# Sampling and Reconstruction

- Formal explanation of
  - Aliasing
  - Reconstruction

# Sampling



# Aliasing

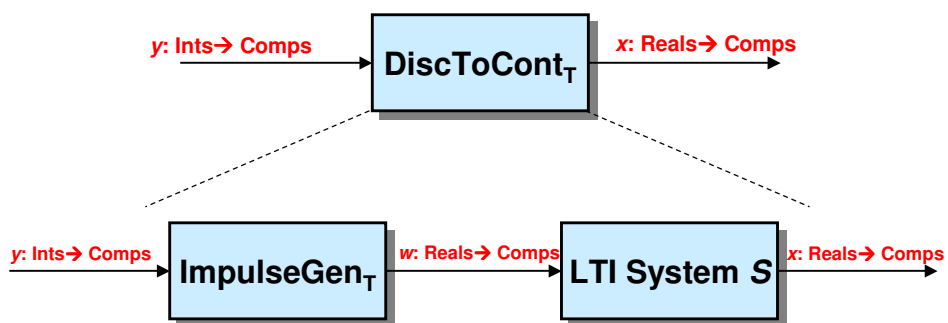
$$x(t) = \cos(2\pi ft) \quad u(t) = \cos(2\pi(f + Nf_s)t)$$

$$y = \text{Sampler}_T(x) \Rightarrow y(n) = \cos(2\pi fnT)$$

$$\begin{aligned} w = \text{Sampler}_T(u) \Rightarrow w(n) &= \cos(2\pi(f + Nf_s)nT) = \\ &= \cos(2\pi fnT + 2\pi Nn) = \cos(2\pi fnT) = \\ &= y(n) \end{aligned}$$

- Even though  $u \neq x$ ,  $\text{Sampler}_T(u) = \text{Sampler}_T(x)$   
 → After sampling,  $x$  and  $u$  are indistinguishable

# Reconstruction



$$x = S(\text{ImpulseGen}_T(y))$$

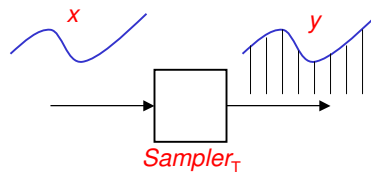
## Three-step Reconstruction Proof

- Let  $x(t)$  be a continuous-time signal with Fourier transform  $X$
- Let  $X(\omega)$  be zero outside the range  $-\pi/T < \omega < \pi/T$

## 1st Step

- Sampling  $x$  with sampling interval  $T$

$$y = \text{Sampler}_T(x)$$



- It can be shown that

$$Y(\omega) = \frac{1}{T} \sum_{k=-\infty}^{+\infty} X\left(\frac{\omega - 2\pi k}{T}\right)$$

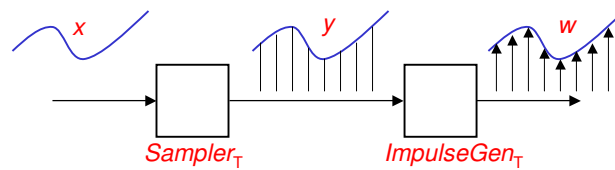
- Since  $X(\omega)=0$  outside the range  $-\pi/T < \omega < \pi/T$

$$Y(\omega) = \frac{1}{T} X\left(\frac{\omega}{T}\right), \quad |\omega| < \pi$$

## 2nd Step

- Let  $w$  be the signal produced by the

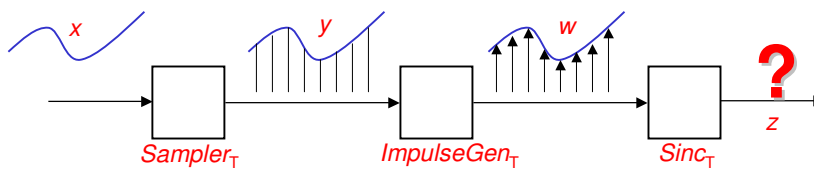
$$w(t) = \sum_{n=-\infty}^{+\infty} y(n)\delta(t - nT)$$



- It can be shown that

$$W(\omega) = Y(\omega T)$$

## 2nd Step (cont.)



- Let  $z$  be the output of the  $Sinc_T$

$$Z(\omega) = W(\omega)H(\omega) = Y(\omega T)H(\omega)$$

Frequency response of the  
reconstruction filter  $Sinc_T$

## 2nd Step (cont.)

- $H(\omega)$  is the frequency response of the reconstruction filter  $Sinc_T$

$$h(t) = \frac{\sin(\pi/T)}{\pi/T}$$

- It can be shown that

$$H(\omega) = \begin{cases} T & -\pi/T < \omega < \pi/T \\ 0 & \text{otherwise} \end{cases}$$

- Substituting for  $H$  and  $Y$ , we get

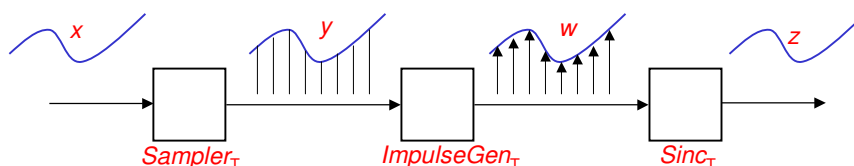
$$Z(\omega) = \begin{cases} \sum_{k=-\infty}^{\infty} X(\omega - 2\pi k/T) & -\pi/T < \omega < \pi/T \\ 0 & \text{otherwise} \end{cases}$$

## 3rd Step

- As  $X(\omega)$  is zero for  $|\omega|$  larger than the Nyquist frequency  $\pi/T$ , then

$$Z(\omega) = X(\omega) \quad \forall \omega \in \mathfrak{R}$$

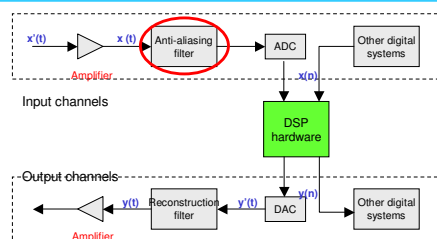
- That is,  $z$  is identical to  $x$



## Practical Applications (1/2)

- In general  $x(t)$  is not bandlimited
- Significant energies outside the highest frequency of interest
- Voice communication systems use 8 kHz sampling rate
  - Maximum frequency component in a speech signal is much higher than 4kHz
  - Out-of-band components at the input of ADC can become in-band because the folding effect

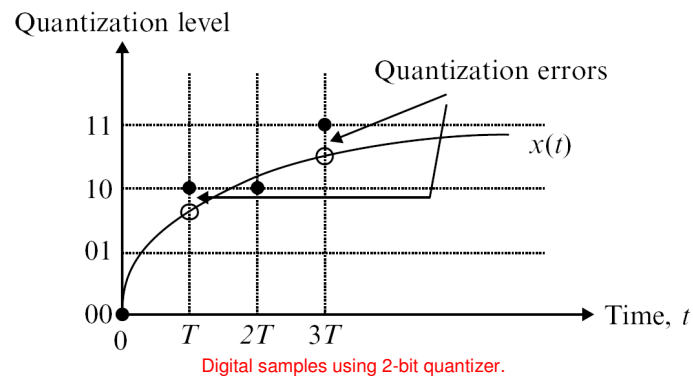
## Practical Applications (2/2)



- Solution
  - Filter  $x(t)$  with a lowpass filter (*anti-aliasing filter*)
  - In many practical system a bandpass filter is used
    - ✓ In order to prevent undesidered DC offset

## Quantizing and Encoding

- $x(nT)$  assume infinite wordlength
- $x(n)$  assume finite wordlength



## Signal to Noise Ratio (SNR)

- The *Signal to noise ratio* (SNR) is the ratio of a signal power to the noise power corrupting the signal
  - It compares the level of a desired signal to the level of background noise
  - The higher the ratio, the less obtrusive the background noise is

## SNR

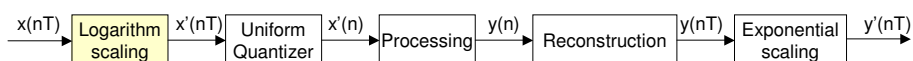
- Signal-to-quantization Noise Ratio is approximated by

$$\text{SNR} \approx 6B \text{ dB}$$

- Where  $B$  is the wordlength
- This is a theoretical maximum
- Simple guideline for determining the wordlength of the quantizer for a given application

## Non-uniform Quantizer

- If the uniform quantization can adequately represent loud sounds
  - Most of the softer sounds may be pushed into the same small value
  - *i.e.*, soft sounds may not be distinguishable
- Non-uniform quantizer



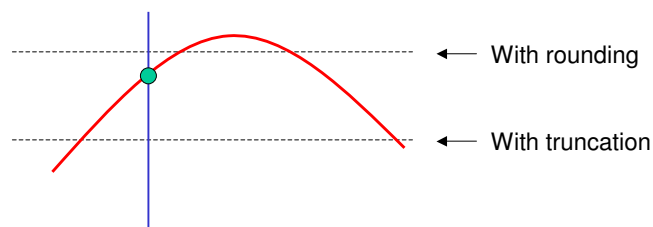
## Input Quantization Noise

- Let us assume that  $x(n)$  is scaled such that  $-1 \leq x(n) < 1$
- If the quantizer employs  $B$  bits
  - The number of quantization levels for representing  $x(nT)$  is  $2^B$
  - The spacing between two quantization levels is

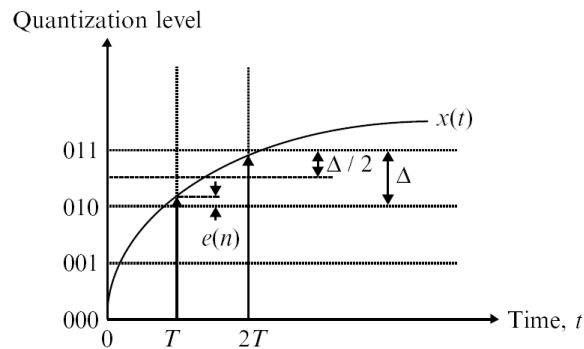
$$\Delta = \frac{\text{full scale range}}{\text{number of quantization levels}} = \frac{2}{2^B} = 2^{-B+1} = 2^{-M}$$

## Common Methods of Quantization

- Rounding
  - The signal value is approximated using the **nearest quantization level**
- Truncation
  - The signal value is approximated using the **highest quantization level that is not greater than the signal itself**
  - ✓ Produces bias effects

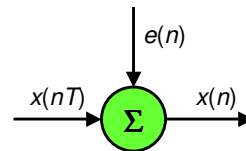


## Quantization Error



$$e(n) = x(n) - x(nT)$$

$$|e(n)| \leq \frac{\Delta}{2}$$



## Quantization Error (mean)

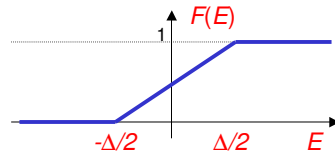
- For an arbitrary signal with fine quantization ( $B$  is large),  $e(n)$  may be assumed to be **uncorrelated** with  $x(n)$ 
  - Can be assumed to be random noise that is **uniformly distributed** in  $[-\Delta/2, \Delta/2]$

$$E[e(n)] = \frac{-\Delta/2 + \Delta/2}{2} = 0$$

**The quantization noise  $e(n)$  has zero mean.**

## Quantization Error (variance)

$$\begin{aligned}\sigma_e^2 &= E[(e - m_e)^2] = E[e^2] = \int_{-\infty}^{+\infty} E^2 f(E) dE = \\ &= \frac{1}{\Delta} \int_{-\Delta/2}^{\Delta/2} E^2 dE = \frac{\Delta^2}{12} = \frac{2^{-2B}}{3}\end{aligned}$$



**The larger the wordlength, the smaller the input quantization error.**

## SQNR

- If the quantization error is regarded as noise

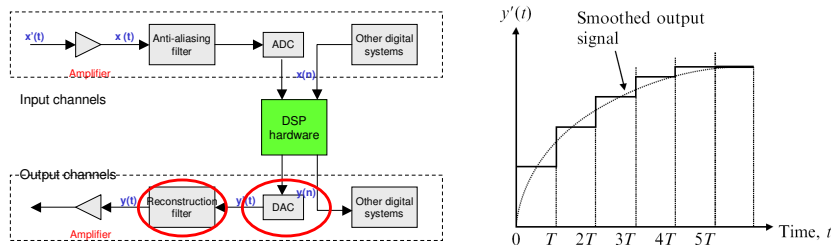
$$SQNR = \frac{\sigma_x^2}{\sigma_e^2} = 3 \cdot 2^{2B} \sigma_x^2$$

- Usually, the SNR is expressed in dB

$$\begin{aligned}SQNR &= 10 \log_{10} (3 \cdot 2^{2B} \sigma_x^2) = \\ &= 4.77 + 6.02B + 10 \log_{10} \sigma_x^2\end{aligned}$$

**For each additional bit used in the ADC, the converter provides about 6-dB SNR gain.**

## DAC & Reconstruction



- Most commercial DACs are zero-order-hold
  - Convert binary to analog level and hold that value for  $T$  second
  - DAC produces a staircase shape analog waveform
- Reconstruction smoothes the staircase-like output signal generated by the DAC
  - Lowpass filter (the same of the anti-aliasing filter)

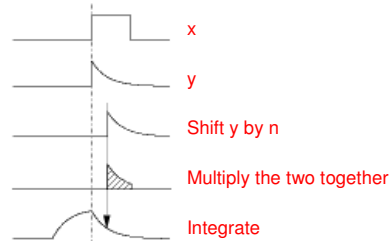
## Time Domain Processing

- Correlation
  - Extract a signal from noise (*autocorrelation*)
  - Locate a know signal (*cross correlation*)
  - Identify a signal (*cross correlation*)
- Convolution

# Correlation

- Correlation is a weighted moving average

$$r(n) = \sum_k x(k) \times y(k+n)$$



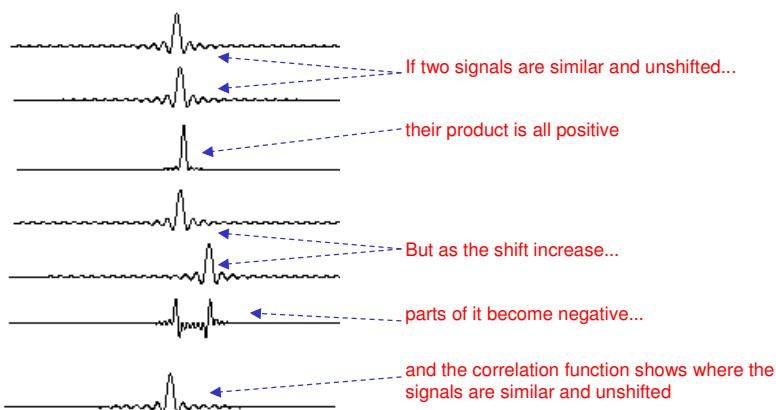
- Requires a lot of calculation

→ If one signal is of length  $M$  and the other is of length  $N$ , then we need  $(N * M)$  multiplications, to calculate the whole correlation function

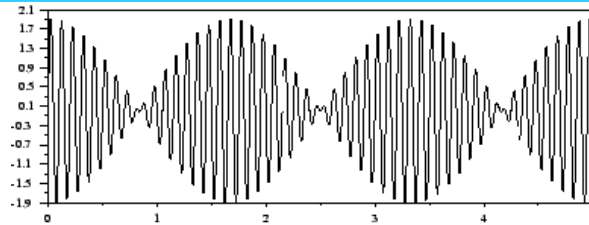
- ✓ Note that really, we want to multiply and then accumulate the result - this is typical of DSP operations and is called a *multiply & accumulate operation*

# Correlation

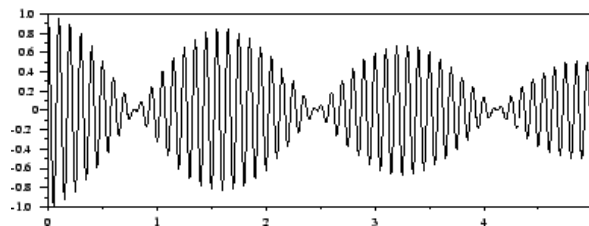
- Correlation has a maximum when two signals are similar in shape
- Correlation is a measure of the similarity between two signals as a function of time shift between them



## Detecting Periodicity



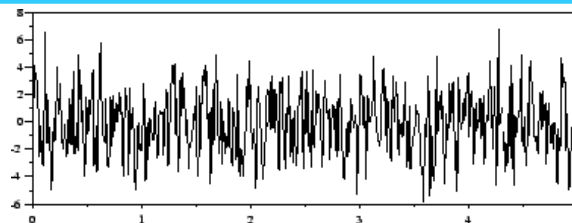
EEG signal



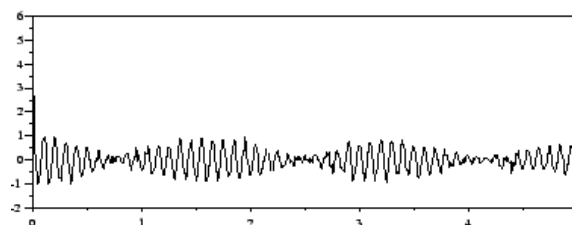
EEG autocorrelation

- Autocorrelation as a way to detect periodicity in signals

## Detecting Periodicity



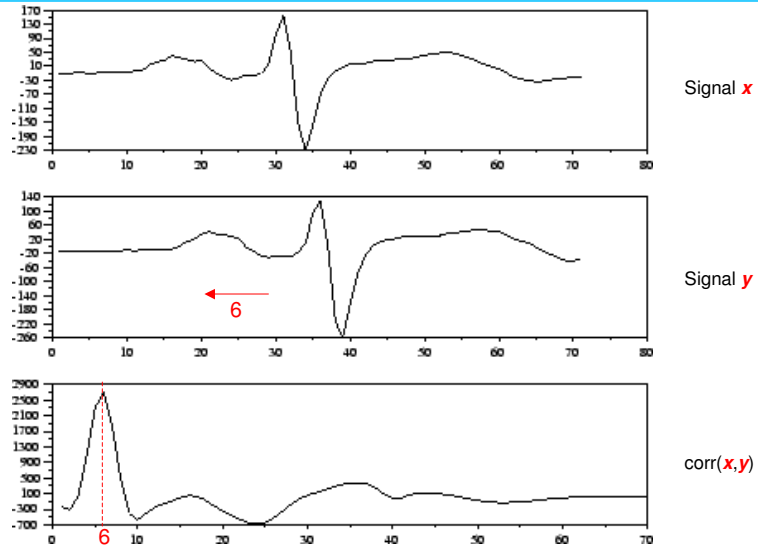
EEG signal with noise



EEG with noise autocorrelation

- Although a rhythm is not even visible (upper trace) it is detected by autocorrelation (lower trace)

# Align Signals

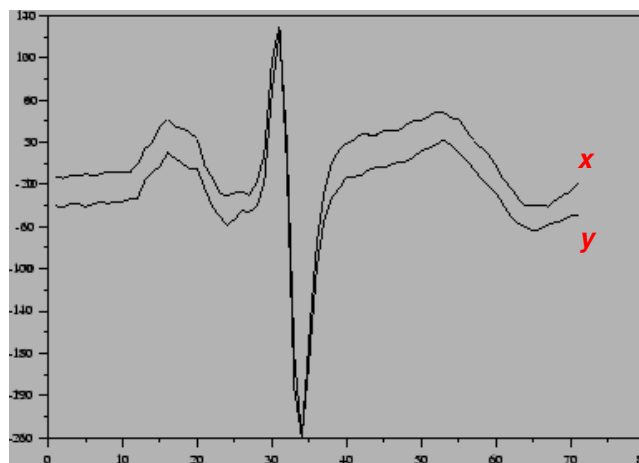


Maurizio Palesi

Embedded System Architectures – Autumn 2008

43

# Align Signals



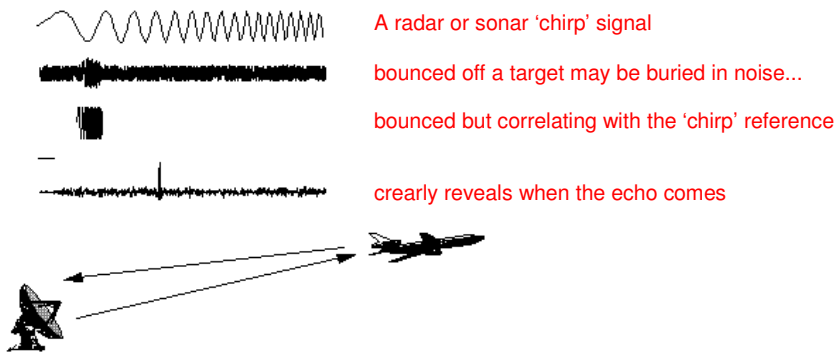
Maurizio Palesi

Embedded System Architectures – Autumn 2008

44

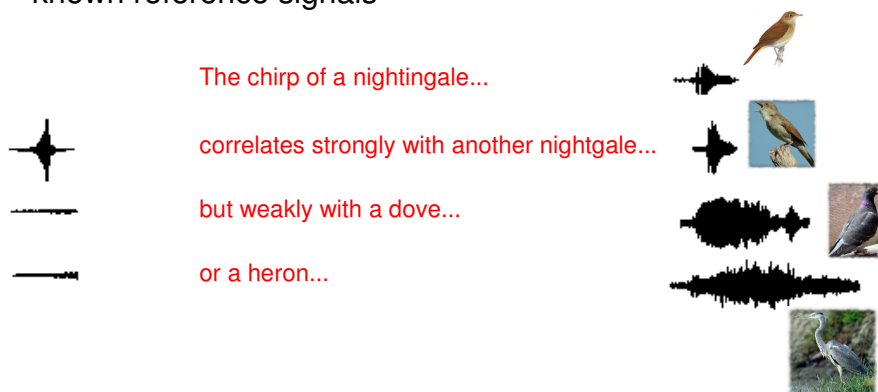
## Cross correlation

- Cross correlation (correlating a signal with another) can be used to detect and locate known reference signal in noise



## Cross Corelation to Identify a Signal

- Cross correlation (correlating a signal with another) can be used to identify a signal by comparison with a library of known reference signals



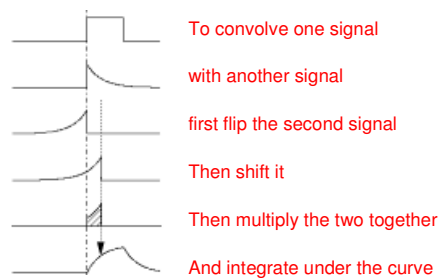
## Cross Correlation to Identify a Signal

- Cross correlation is one way in which sonar can identify different types of vessel
  - Each vessel has a unique sonar *signature*
  - The sonar system has a library of *pre-recorded echoes* from different vessels
  - An unknown sonar echo is *correlated* with a library of *reference echoes*
  - The *largest correlation* is the most likely match

## Convolution

- Correlation is a weighted moving average with one signal flipped back to front

$$r(n) = \sum_k x(k) \times y(k - n)$$



- Requires a lot of calculation
  - If one signal is of length  $M$  and the other is of length  $N$ , then we need  $(N * M)$  multiplications, to calculate the whole convolution function
    - ✓ We need to multiply and then accumulate the result - this is typical of DSP operations and is called a *multiply & accumulate operation*

## Convolution vs. Correlation

- Convolution is used for digital filtering
  - Convolving two signals is equivalent to *multiplying the frequency spectra* of the two signals together
    - ✓ It is easily understood, and is what we mean by filtering
  - Correlation is equivalent to *multiplying the complex conjugate of the frequency spectrum* of one signal by the frequency spectrum of the other
    - ✓ It is not so easily understood and so convolution is used for digital filtering
- Convolving by multiplying frequency spectra is called *fast convolution*

## Fourier Transform

- The *Fourier Transform* is a mathematical procedure that allows to convert a signal from the time domain to the frequency domain
- Any signal or waveform could be made up just by adding together a series of sine waves with appropriate *amplitude* and *phase*



A square wave can be made by adding...



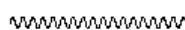
the fundamental



minus 1/3 of the third harmonic



plus 1/5 of the fifth harmonic...



minus 1/7 of the 7th harmonic...



# Fourier Transform

- The Fourier transform is an equation to calculate the *frequency*, *amplitude* and *phase* of each sine needed to make up any given signal
  - The *Fourier Transform* (FT) is a mathematical formula using integrals
  - The *Discrete Fourier Transform* (DFT) is a discrete numerical equivalent using sums instead of integrals
  - The *Fast Fourier Transform* (FFT) is just a computationally fast way to calculate the DFT
- The Discrete Fourier Transform involves a summation

$$H(f) = \sum_k c(k) \times e^{-2\pi j k (f\Delta)}$$

- DFT and the FFT involve a lot of multiply and accumulate the result
  - This is typical of DSP operations and is called a *multiply & accumulate* operation

# Frequency Spectra

- With some signals it is easy to see that they are composed of different frequencies
  - A chord played on the piano is obviously made up of the different pure tones generated by the keys pressed
  - You can use a piano **as an acoustic spectrum analyser** to show that a hand clap has a frequency spectrum
    - ✓ Open the lid of the piano and hold down the 'loud' pedal
    - ✓ Clap your hands loudly over the piano
    - ✓ You will hear (and see) the strings vibrate to echo the clap sound
      - The strings that vibrate show the *frequencies*
      - The amount of vibration shows the *amplitude*

## Short Term Fourier Transform

$$H(f) = \sum_k c(k) \times e^{-2\pi jk(f\Delta)}$$

### ■ Fourier transform

- The signal is analysed over all time
- ...an infinite duration

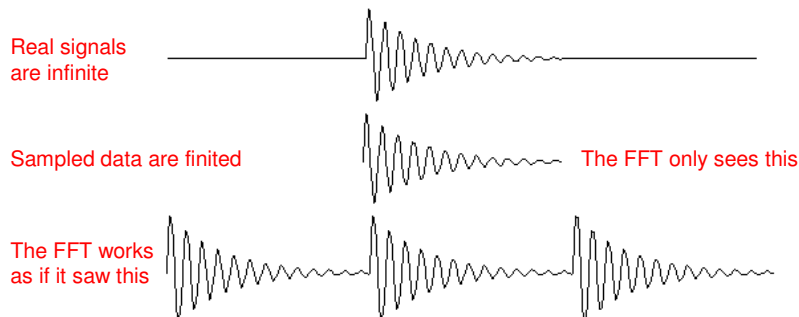
### ■ Short Time Fourier Transform (*STFT*)

- Evaluates the way frequency content changes with time

## Short Signals

### ■ If we measure the signal for a short time

- What happened to the signal before and after we measured it?
- FFT makes an assumption about what happened before and after
- The FFT assumes the data are periodic for all time

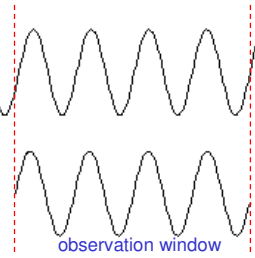


# Short Signals

This is the signal



Period fits the sample time



The FFT works as if it saw this

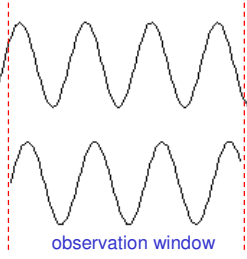


# Short Signals

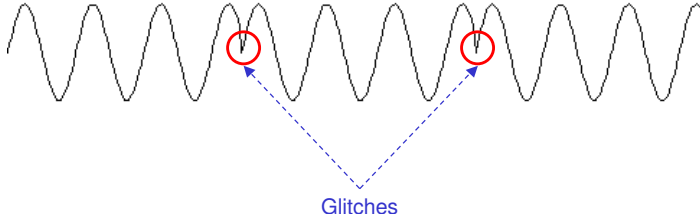
This is the signal



Not quite an integral number of cycles fit into the total duration of the measurement



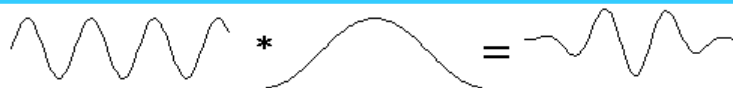
The FFT works as if it saw this



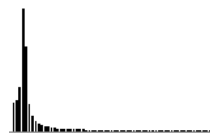
## Short Signals

- If the period **exactly fits the measurement time**
  - The frequency spectrum is correct
- If the period **does not match the measurement time**
  - The frequency spectrum is incorrect - it is broadened
- The size of the glitch depends on when the first measurement occurred in the cycle
  - The broadening will change if the measurement is repeated

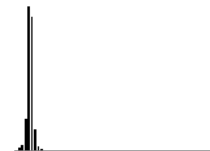
## Windowing



If the period does not fit the time, spurious spectral lines result



But windowing the data can narrow the spectrum

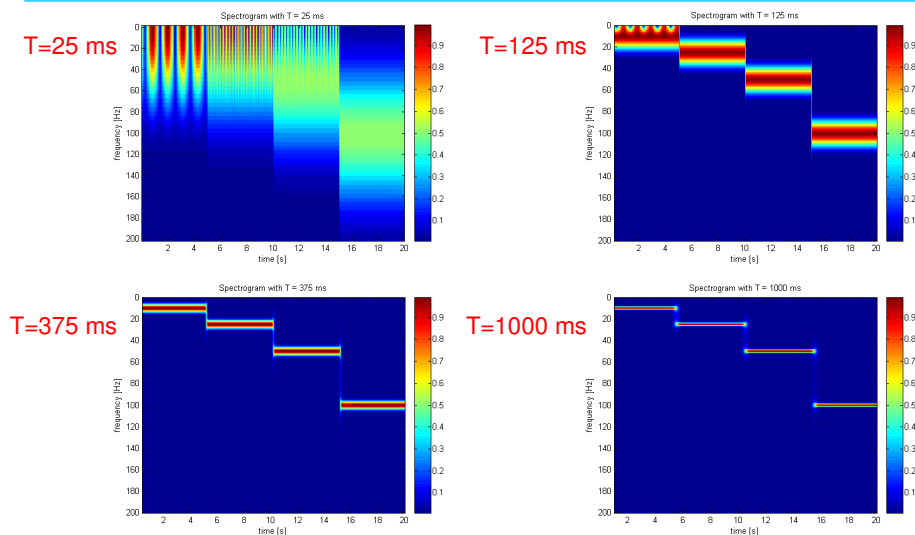


## Windowing Tradeoffs (1/2)

- A **wide window** gives better frequency resolution but poor time resolution
- A **narrower window** gives good time resolution but poor frequency resolution

$$x(t) = \begin{cases} \cos(2\pi 10t) & 0 \leq t \leq 5s \\ \cos(2\pi 25t) & 5 \leq t < 10s \\ \cos(2\pi 50t) & 10 \leq t < 15s \\ \cos(2\pi 100t) & 15 \leq t < 20s \end{cases}$$

## Windowing Tradeoffs (1/2)



# Filtering

---



- The function of a filter is to remove unwanted parts of the signal
  - Random noise
  - Extract useful parts of the signal
    - ✓ Components lying within a certain frequency range
- Filters
  - Analog
  - Digital

# Analog Filters

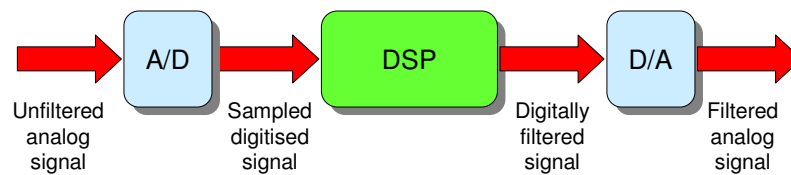
---

- An *analog filter* uses analog electronic circuits
  - Use components such as resistors, capacitors and op amps
- Widely used in such applications
  - Noise reduction
  - Video signal enhancement
  - Graphic equalisers in hi-fi systems
  - ..., and many other areas

## Digital Filters

- A *digital filter* uses a digital processor to perform numerical calculations on sampled values of the signal

→ Specialised DSP chip



## Advantage of Digital Filters

- Programmability
  - The digital filter can easily be changed without affecting the circuitry
- Analog filter circuits are subject to drift and are dependent on temperature
- Digital filters can handle low frequency signals accurately
- As the speed of DSP technology continues to increase, digital filters are being applied to high frequency signals in the RF domain
- Versatility
  - Adapt to changes in the characteristics of the signal

## Digital Filter Types & Specifications

### ■ Types

- Lowpass
- Highpass
- Bandpass
- Bandstop

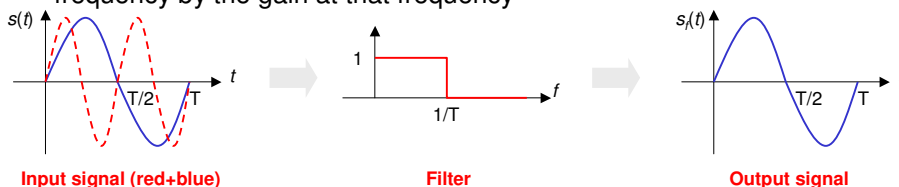
### ■ Graphical specifications

- Starting point to determine the coefficients of the digital filter
- Ideal plot versus the frequency of where the gain curve of the digital filter is allowed to go and not allowed to go

## Plot of Gain

$$\text{gain} = 20 \lg_{10}(\text{magnitude transfer function}) = 20 \lg_{10} \frac{\text{amplitude of output sinusoid at frequency } \omega}{\text{amplitude of input sinusoid at frequency } \omega}$$

- With a plot of gain vs. frequency of a filter it is easy to see what the filter does to the output by multiplying the input amplitude at any frequency by the gain at that frequency



Input signal (red+blue)

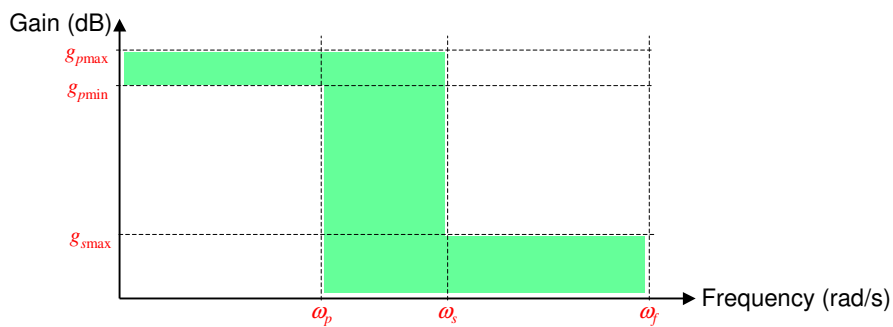
Filter

Output signal

- As all DSP systems start to repeat their output spectrums in magnitude every 0.5 times the sampling frequency, the frequency axis is usually limited to this value

## Lowpass Digital Filter Specification

- $g_{pmax}/g_{pmin}$  max/min allowed gain in the passband
- $g_{smax}$  max allowed gain in the stopband
- $\omega_p$  highest frequency in the passband
- $\omega_s$  lowest frequency in the stopband
- $\omega_f$  folding frequency (half of the sampling frequency  $\pi/T$  rad/sec)



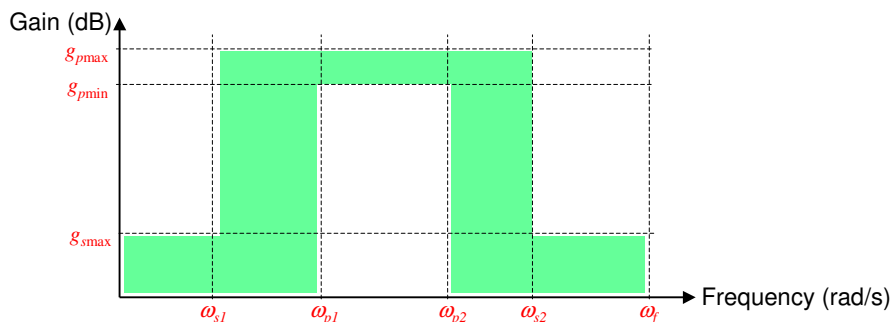
Maurizio Palesi

Embedded System Architectures – Autumn 2008

67

## Bandpass Digital Filter Specification

- $g_{pmax}/g_{pmin}$  max/min allowed gain in the passband
- $g_{smax}$  max allowed gain in the stopband
- $\omega_{s1}$  upper frequency limit of the lower stopband
- $\omega_{p1}$  lower frequency limit of the passband
- $\omega_{p2}$  upper frequency limit of the passband
- $\omega_{s2}$  lower frequency limit of the upper stopband
- $\omega_f$  folding frequency (half of the sampling frequency  $\pi/T$  rad/sec)



Maurizio Palesi

Embedded System Architectures – Autumn 2008

68

## FIR and IIR Filters

### ■ Finite Impulse Response (*FIR*)

→ Non-recursive filter

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \dots + a_mx_{n-m}$$

→ Order is  $m$

### ■ Infinite Impulse Response (*IIR*)

→ Recursive filter

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \dots + a_mx_{n-m} + b_1y_{n-1} + b_2y_{n-2} + b_3y_{n-3} + \dots + b_py_{n-p}$$

→ Order is  $\max\{m,p\}$

## Recursive vs. Non-Recursive

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \dots + a_mx_{n-m}$$

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \dots + a_mx_{n-m} + b_1y_{n-1} + b_2y_{n-2} + b_3y_{n-3} + \dots + b_py_{n-p}$$

■ It might seem that recursive filters require more calculations to be performed

→ To achieve a given frequency response characteristic using a recursive filter generally requires a much lower order filter

$$y_n = x_n + y_{n-1}$$

**Recursive**

$$y_0 = x_0 + y_{-1}$$

$$y_1 = x_1 + y_0$$

$$y_2 = x_2 + y_1$$

$$y_3 = x_3 + y_2$$

...

$$y_{10} = x_{10} + y_9$$

**Non-recursive**

$$y_0 = x_0 + y_{-1}$$

$$y_1 = x_1 + x_0 + y_{-1}$$

$$y_2 = x_2 + x_1 + x_0 + y_{-1}$$

$$y_3 = x_3 + x_2 + x_1 + x_0 + y_{-1}$$

...

$$y_{10} = x_{10} + x_9 + x_8 + \dots + x_3 + x_2 + x_1 + x_0 + y_{-1}$$

## Transfer Function (IIR)

### ■ $z^{-1}$ unit delay operator

$$\rightarrow z^{-1}x_n = x_{n-1}$$

$$\rightarrow z^{-2}x_n = x_{n-2}$$

$$b_0y_n + b_1y_{n-1} + b_2y_{n-2} = a_0x_n + a_1x_{n-1} + a_2x_{n-2}$$

$$(b_0 + b_1z^{-1} + b_2z^{-2})y_n = (a_0 + a_1z^{-1} + a_2z^{-2})x_n$$

$$\frac{y_n}{x_n} = \frac{a_0 + a_1z^{-1} + a_2z^{-2}}{b_0 + b_1z^{-1} + b_2z^{-2}}$$

Transfer function  
for a second-order  
recursive (IIR)  
filter

## Transfer Function (FIR)

### ■ $z^{-1}$ unit delay operator

$$\rightarrow z^{-1}x_n = x_{n-1}$$

$$\rightarrow z^{-2}x_n = x_{n-2}$$

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2}$$

$$y_n = (a_0 + a_1z^{-1} + a_2z^{-2})x_n$$

$$\frac{y_n}{x_n} = a_0 + a_1z^{-1} + a_2z^{-2}$$

Transfer function  
for a second-order  
non-recursive  
(FIR) filter

## Filter Frequency Response

$$y(n) = \sum_k c(k) \times x(n-k) + \sum_j d(j) \times y(n-j)$$

$$H(f) = \frac{\sum_k c(k) \times e^{-2\pi j k (f\Delta)}}{1 - \sum_k d(k) \times e^{-2\pi j k (f\Delta)}}$$

- When designing a digital filter we want to do the *inverse* operation
  - Calculate the filter coefficients having first defined the desired frequency response
  - Additional constraint
    - ✓ We usually want to design a filter that meets the requirement but which requires the least possible amount of computation
      - Using the **smallest number of coefficients**

## FIR Filters

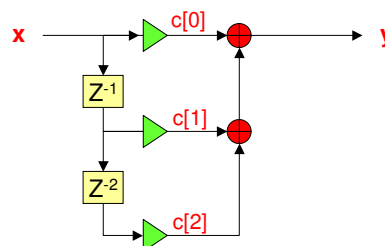
- The filter equation is simplified by excluding the possibility of feedback

$$y(n) = \sum_k c(k) \times x(n-k)$$

- The filter response is

$$H(f) = \sum_k c(k) \times e^{-2\pi j k (f\Delta)}$$

- This frequency response is just the Fourier transform of the filter coefficients



## FIR Filters

---

- So, the coefficients for an FIR filter can be calculated “*simply*” by taking the *Inverse Fourier Transform* of the desired frequency response
- Here is a recipe for **calculating FIR filter coefficients**
  - Decide upon the desired frequency response
  - Calculate the inverse Fourier transform
  - Use the result as the filter coefficients
- **...BUT...**

## FIR Filters

---

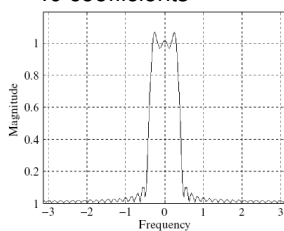
- **...BUT...**
  - The iFT has to take samples of the continuous desired frequency response
  - To define a sharp filter needs closely spaced frequency samples - **so a lot of them**
  - So the iFT will give us a lot of filter coefficients
  - But **we don't want a lot of filter coefficients**
- A better recipe for calculating FIR filter is:
  - Specify the desired frequency response using lots of samples
  - Calculate the inverse iFT
  - This gives us a lot of filter coefficients
  - So truncate the filter coefficients to give us less
  - Then calculate the FT of the truncated set of coefficients to see if it still matches our requirement
- **...BUT...**

# FIR Filters

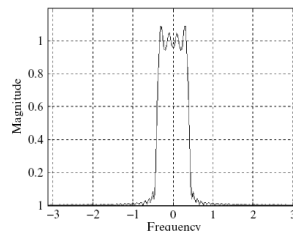
- Truncating the filter coefficients means we have a truncated signal...
- ...And a truncated signal has a broad frequency spectrum
- Applying a window function is a simple way to sharpen up the frequency spectrum of a truncated signal
  - Specify the desired frequency response using lots of samples
  - Calculate the inverse Fourier transform
  - This gives us a lot of filter coefficients
  - So truncate the filter coefficients to give us less
  - Apply a window function to sharpen up the filter's frequency response
  - Then calculate the Fourier transform of the truncated set of coefficients to see if it still matches our requirement
- This is called the **window method of FIR filter design**

# Example

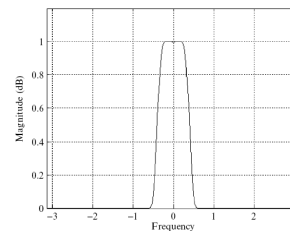
Fourier series method  
40 coefficients



Fourier series method  
60 coefficients



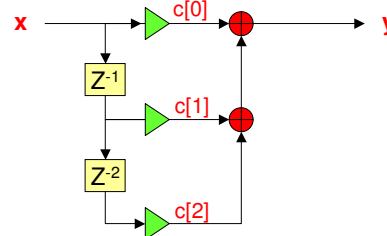
Window method  
60 coefficients



## Characteristics of DSP Processors

### ■ The basic DSP operations

- Additions and multiplications
  - ✓ Fetch two operands
  - ✓ Perform the addition or multiplication (usually both)
  - ✓ Store the result or hold it for a repetition
- Delays
  - ✓ Hold a value for later use
- Array handling
  - ✓ Fetch values from consecutive memory locations
  - ✓ Copy data from memory to memory



## Address Generation

- The ability to **generate new addresses** efficiently is a characteristic feature of DSP processors
- Usually, the next needed address can be generated during the data fetch or store operation, and with **no overhead**
- DSP processors have **rich sets** of address generation operations

*rP	register indirect	read the data pointed to by the address in register rP
*rP++	postincrement	having read the data, postincrement the address pointer to point to the next value in the array
*rP--	postdecrement	having read the data, postdecrement the address pointer to point to the previous value in the array
*rP++rl	register postincrement	having read the data, postincrement the address pointer <i>by the amount held in register rl</i> to point to <i>rl</i> values further down the array
*rP++rlr	bit reversed	having read the data, postincrement the address pointer to point to the next value in the array, <i>as if the address bits were in bit reversed order</i>

## Bit Reversed Addressing

- DSPs are tightly targeted to a small number of algorithms
  - It is surprising that an addressing mode has been specifically defined for just one application (the FFT)

### Addresses generated by a radix-2 FFT

0 (000 <sub>2</sub> )	→	0 (000 <sub>2</sub> )
1 (001 <sub>2</sub> )	→	4 (100 <sub>2</sub> )
2 (010 <sub>2</sub> )	→	2 (010 <sub>2</sub> )
3 (011 <sub>2</sub> )	→	6 (110 <sub>2</sub> )
4 (100 <sub>2</sub> )	→	1 (001 <sub>2</sub> )
5 (101 <sub>2</sub> )	→	5 (101 <sub>2</sub> )
6 (110 <sub>2</sub> )	→	3 (011 <sub>2</sub> )
7 (111 <sub>2</sub> )	→	7 (111 <sub>2</sub> )

- Without special support such address transformations would
  - Take an extra memory access to get the new address
  - Involve a fair amount of logical instructions

## Memory Addressing

- As DSP programmers migrate toward larger programs, they are more attracted to compilers
  - Such compilers are not able to fully exploit such specific addressing modes
  - DSP community routinely uses library routines
    - ✓ Programmers may benefit even if they write at a high level

Addressing mode	Percent
Immediate	30,02%
Displacement	10,82%
Register indirect	17,42%
Direct	11,99%
Autoincrement, postincrement	18,84%
Autoincrement, preincrement with 16 bit immediate	0,77%
Autoincrement, preincrement with circular addressing	0,08%
Autoincrement, postincrement by contents of AR0	1,54%
Autoincrement, postincrement by contents of AR0, with circular addressing	2,15%
Autodecrement, postdecrement	6,08%

~90%

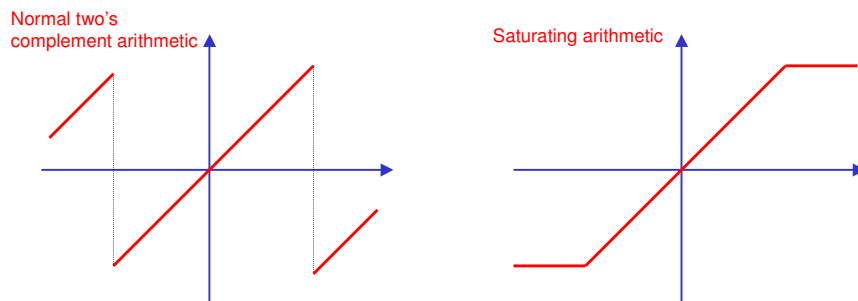
# Compiler for DSPs

- Despite the well documented advantages in programmer productivity and software maintenance...

TMS320C54 D (C54) for DSPstone kernels	Ratio to assembly in execution time (>1 means slower)	Ratio to assembly in code space (>1 means bigger)	TMS320C6203 (C62) EEMBC Telecom kernels	Ratio to assembly in execution time (>1 means slower)	Ratio to assembly in code space (>1 means bigger)
Convolution	11,8	16,5	Convolution encoder	44,0	0,5
FIR	11,5	8,7	Fixed-point complex FFT	13,5	1,0
Matrix 1x3	7,7	8,1	Viterbi GSM decoder	13,0	0,7
FIR2dim	5,3	6,5	Fixed-point bit allocation	7,0	1,4
Dot product	5,2	14,1	Autocorrelation	1,8	0,7
LMS	5,1	0,7			
N real update	4,7	14,1			
IIR n biquad	2,4	8,6			
N complex update	2,4	9,8			
Matrix	1,2	5,1			
Complex update	1,2	8,7			
IIR one biquad	1,0	6,4			
Real update	0,8	15,6			
<b>C54 geometric mean</b>	<b>3,2</b>	<b>7,8</b>	<b>C62 geometric mean</b>	<b>10,0</b>	<b>0,8</b>

# Saturating Arithmetics

- DSPs are often used in real-time applications
  - No exception on arithmetic overflow
    - ✓ It could miss an event
  - To support such an environment, DSP architectures use saturating arithmetic
    - ✓ If the result is too large to be represented, it is set to the largest representable number



## The TMS320C3x

---

- The TMS320C3x generation of DSPs are high performance 32-bit floating-point devices in the TMS320 family
- Extensive internal busing
  - 2 operands from memory & 2 operands from the registers file
- Powerful DSP instruction set
- 60 MFLOPS
- High degree of on-chip parallelism
  - Up to 11 operations in a single instruction

## General Features

---

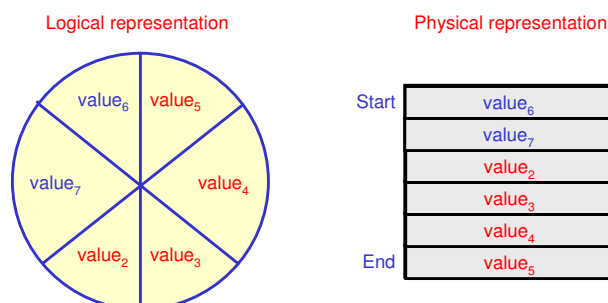
- General-purpose register file
- Program cache
- Dedicated auxiliary register arithmetic units (ARAU)
- Internal dual-access memories
- Direct memory access (DMA)
- Short machine-cycle time

## Addressing Modes

- Five types of addressing
  - Register addressing
  - Direct addressing
  - Indirect addressing
  - Immediate addressing
  - PC-relative addressing
- Plus two specialized addressing modes
  - Circular addressing
  - Bit-reverse addressing

## Circular Addressing

- Many DSP algorithms, such as convolution and correlation, require a circular buffer in memory
- In convolution and correlation, the circular buffer acts as a sliding window that contains the most recent data to process
- As new data is brought in, the new data overwrites the oldest data



# Parallel Operations

- The 13 parallel-operations instructions make a high degree of parallelism possible
- Some of the 'C3x instructions can occur in pairs that are executed in parallel
  - Parallel loading of registers
  - Parallel arithmetic operations
  - Arithmetic/logical instructions used in parallel with a store instruction

Mnemonic	Description
MPYF3    ADDF3	Multiply and add floating-point value
MPYF3    SUBF3	Multiply and subtract floating-point value
MPYI3    ADDI3	Multiply and add integer
MPYI3    SUBI3	Multiply and subtract integer

...and many many other...

# Matrix-Vector Multiplication

$$\mathbf{[P]}_{K \times 1} = \mathbf{[M]}_{K \times N} \times \mathbf{[V]}_{N \times 1}$$

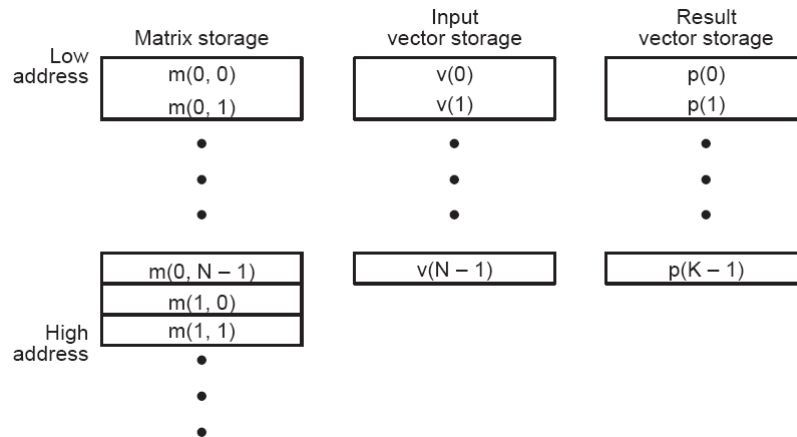
```
for (i=0; i<K; i++)
{
    p[i] = 0.0;
    for (j=0; j<N; j++)
        p[i] = p[i] + m[i][j] * v[j];
}
```



```
for (i=0; i<K; i++)
{
    p[i] = m[i][0] * v[0];
    for (j=1; j<N; j++)
        p[i] = p[i] + m[i][j] * v[j];
}
```

# Matrix-Vector Multiplication

## ■ Data memory organization



# Matrix-Vector Multiplication

```

* AR0 : ADDRESS OF M(0,0)
* AR1 : ADDRESS OF V(0)
* AR2 : ADDRESS OF P(0)
* AR3 : NUMBER OF ROWS - 1 (K-1)
* R1  : NUMBER OF COLUMNS - 2 (N-2)

MAT   LDI    R1, IR0           ; Number of columns-2 -> IR0
        ADDI   2, IR0           ; Number of columns -> IR0
ROWS  LDF    0.0, R2          ; Initialize R2
        MPYF3  *AR0++(1), *AR1++(1), R0 ; m(i,0) * v(0) -> R0
        RPTS   R1               ; Multiply a row by a column
        MPYF3  *AR0++(1), *AR1++(1), R0 ; m(i,j) * v(j) -> R0
        || ADDF3 R0, R2, R2      ; m(i, j-1) * v(j-1) + R2 -> R2
        SUBI   1, AR3
        BNZD   ROWS             ; Counts the no. of rows left
        Delay slot { ADDF    R0, R2          ; Last accumulate
                    STF     R2, *AR2++(1) ; Result -> p(i)
                    NOP     *--AR1 (IR0)  ; Set AR1 to point to v(0)
    
```

## Summary and Conclusions

---

- Introduction to Digital Signal Processing
  - Time domain processing
  - Frequency domain processing
  - Digital filters
- Signal Processing using DSP
  - Special features for arithmetic
  - Addressing modes
- More reading
  - Introduction to DSP
    - ✓ <http://www.bores.com>
  - TMS320C3x General-Purpose Applications User's Guide
    - ✓ <http://focus.ti.com/general/docs/tecdocs.tsp>