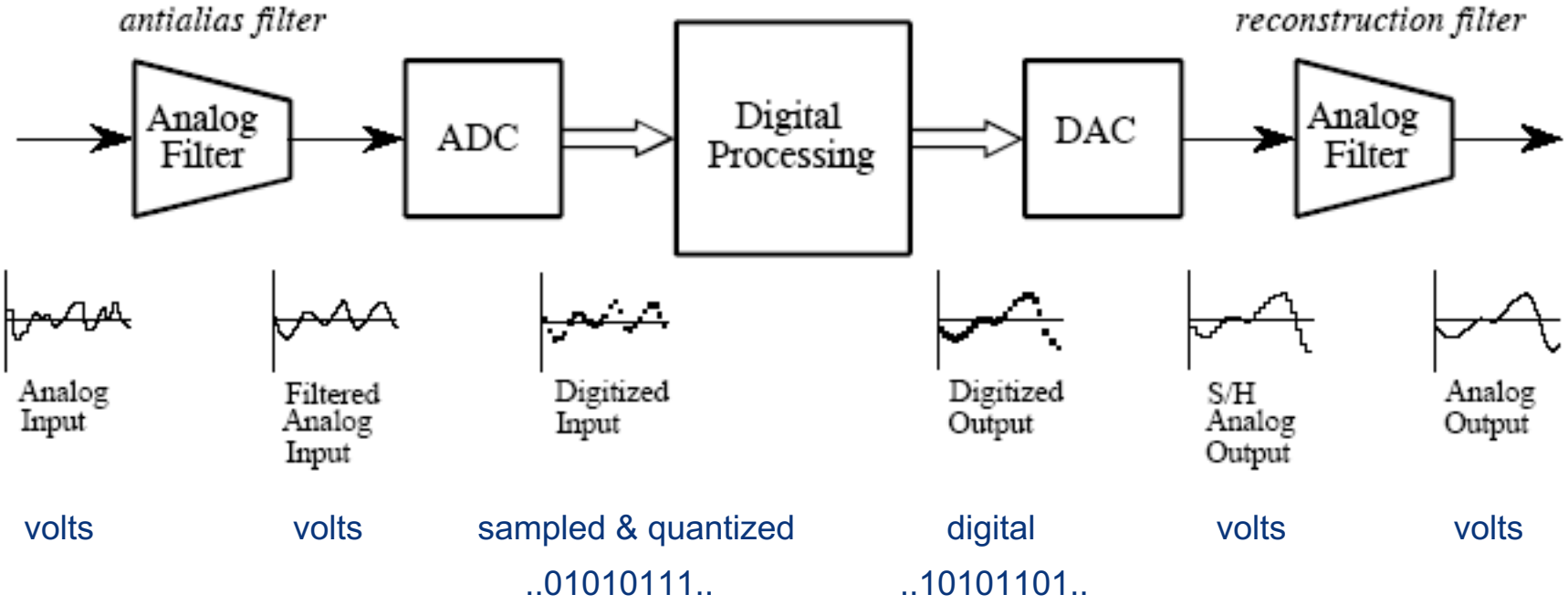
Digital Filtering
online lecture

Prof. Brad Nelson
Dr. Naveen Shamsudhin

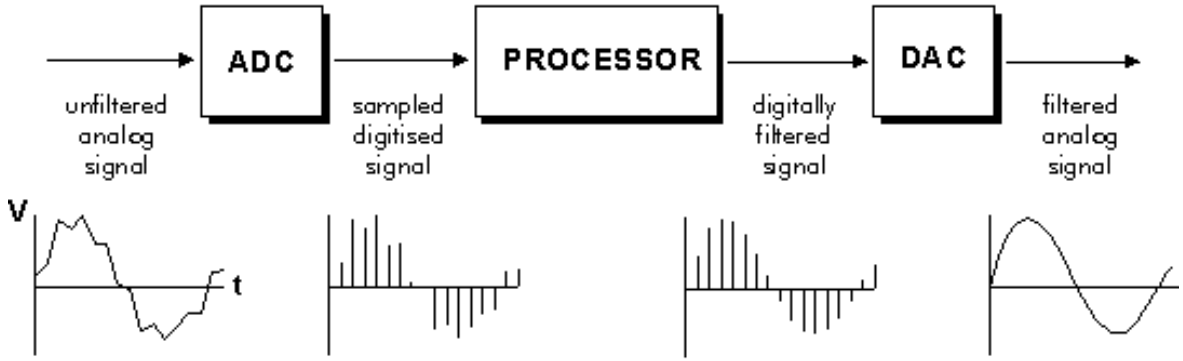
Questions?

Digital Signal Processing



Topics for Today

- Digital Filters
  - Background
  - Finite Impulse Response (FIR) Filter
    - Moving average
    - Windowed-sinc
  - Infinite Impulse Response (IIR) Filter
    - Single pole



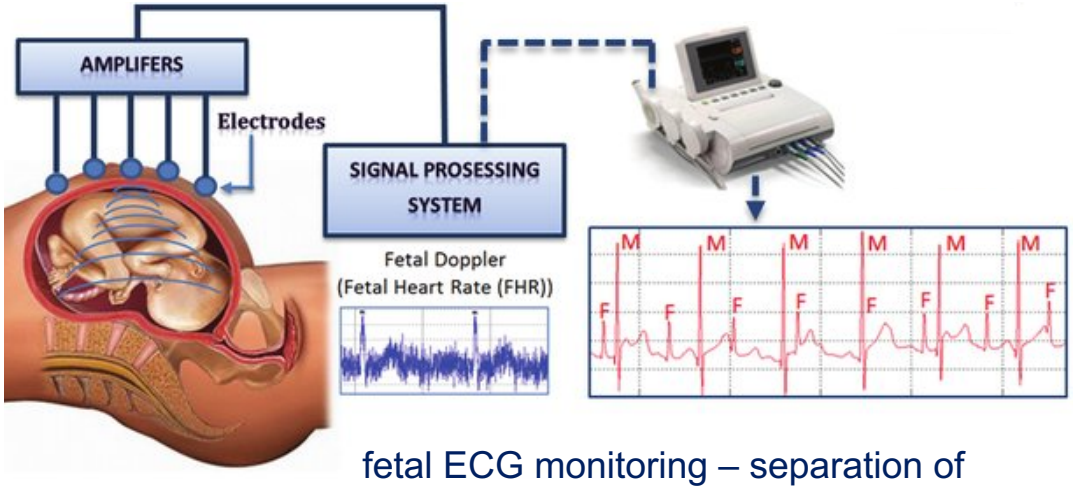
Filter Overview

- Filters are used for
  - Signal Restoration
    - Remove unwanted parts of the signal such as random noise, or correct for signal distortion.

Image deblurring

Filter Overview

- Filters are used for
  - Signal Restoration
    - Remove unwanted parts of the signal such as random noise
  - Signal Separation
    - Extract useful parts of the signal such as components within a certain frequency range



fetal ECG monitoring – separation of fetal heartbeat from breathing & heartbeat of mother

Analog vs. Digital

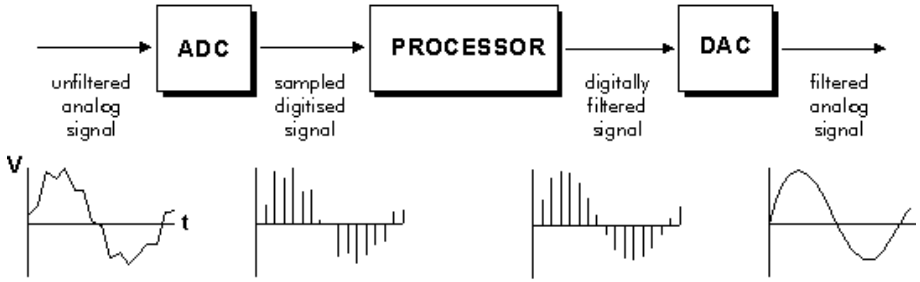
- A filter can be either analog or digital
- Analog filters are made from components such as resistors, capacitors, op amps.
  - Fast and Cheap
  - Have a large dynamic range in amplitude & frequency
- For examples (see lecture notes on Signal Processing)

Digital Filters Are Not Perfect

- Digital filters use digital processors. The signal must first be digitized by an ADC
- They are programmable, so they are flexible and can be easily modified
- They are versatile
- They can handle very slow signals
- They are not subject to drift from temperature variations over time

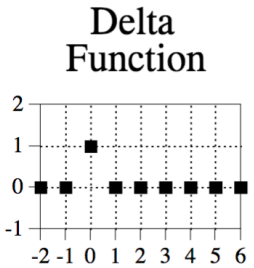
But,

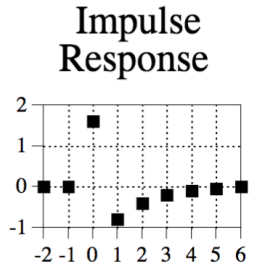
  - They introduce a delay into the signal (more components in signal path)
  - They might attenuate the part of the signal you want
  - They might not (probably won't) ignore all the noise you want them to ignore
  - Relatively expensive

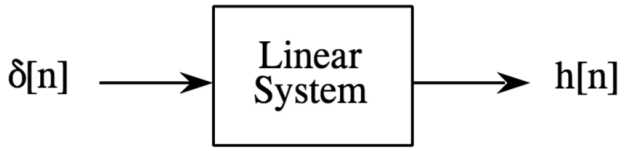


Digital Filtering – Background

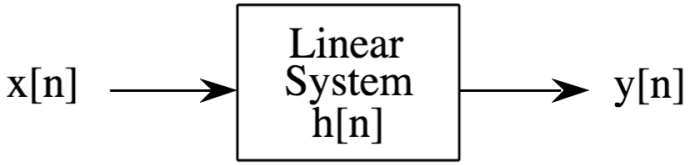
- Delta function δ [n]
  - Normalized impulse function
- Impulse response h [n]
  - Also known as filter kernel
- Convolution
  - y [n] = x [n] ∗ h [n]







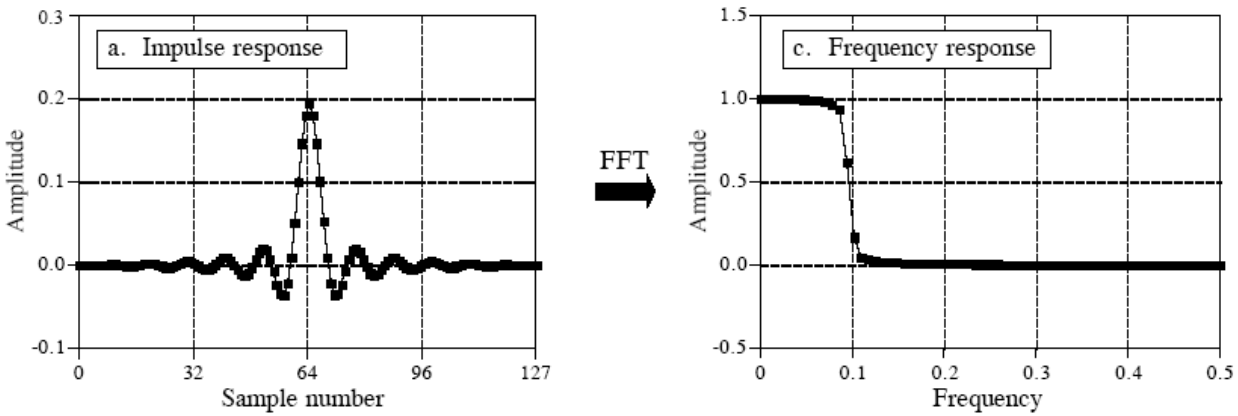
$$y[i] = \sum_{j=0}^{M-1} h[j]\, x[i-j]$$



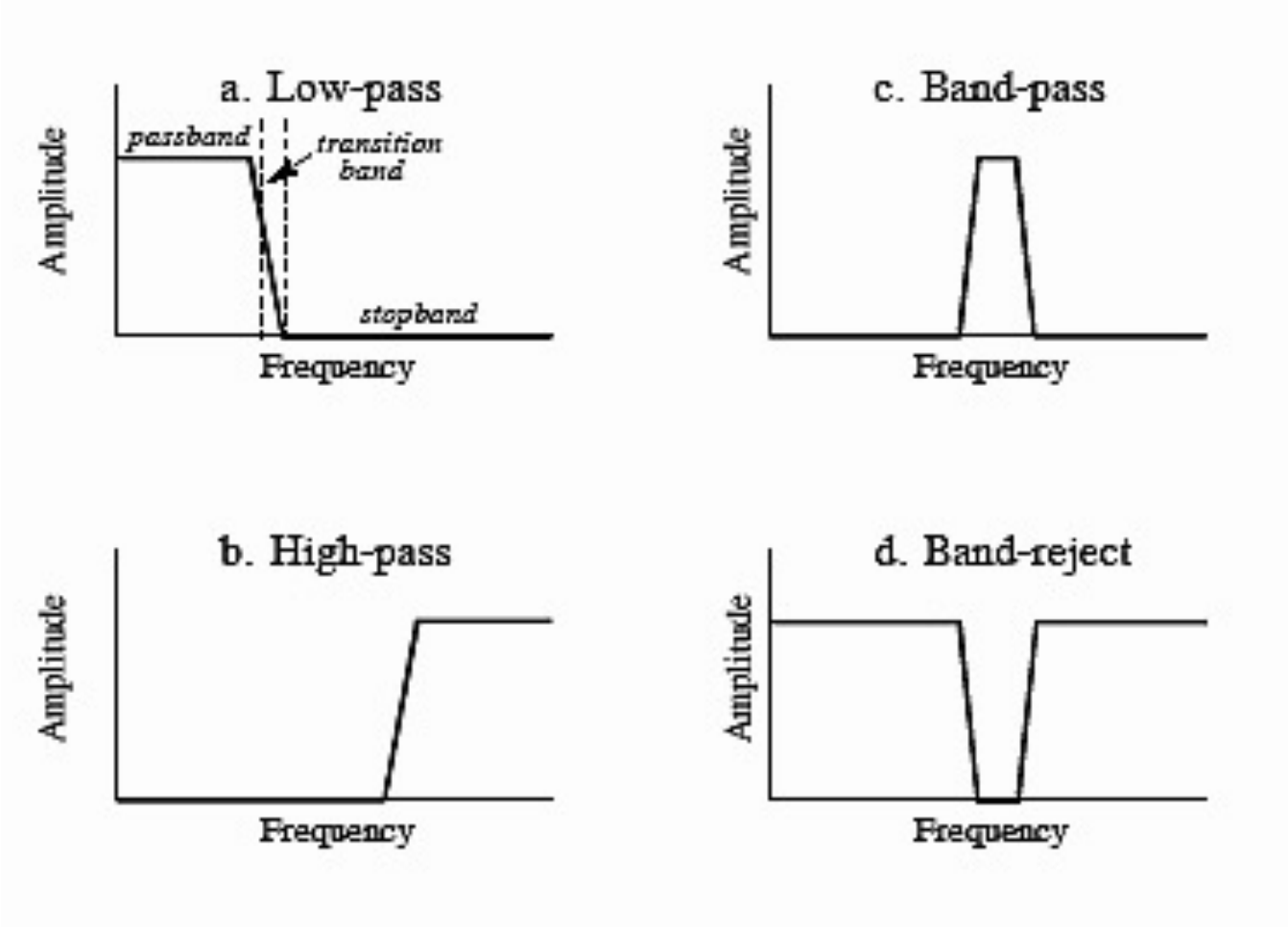
x[n] ∗ h[n] = y[n]

Digital Filtering – Background

- Impulse response h [n]
- Frequency response H [f]



Digital Filtering – Background

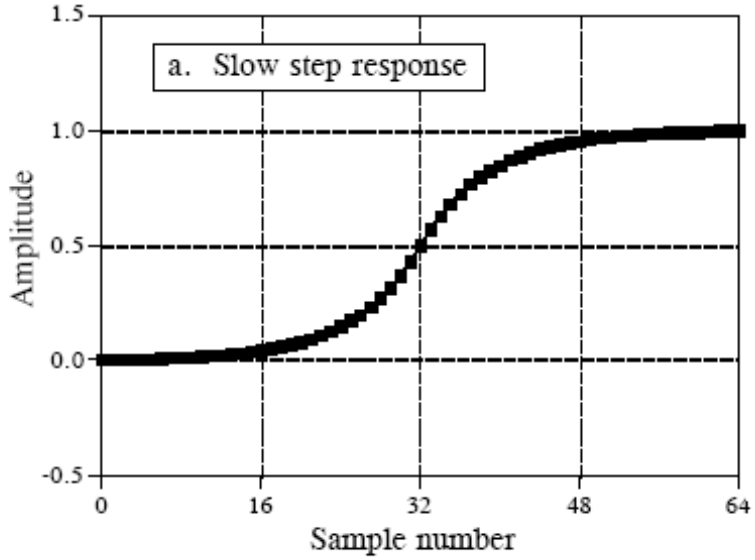


Digital Filters – How?

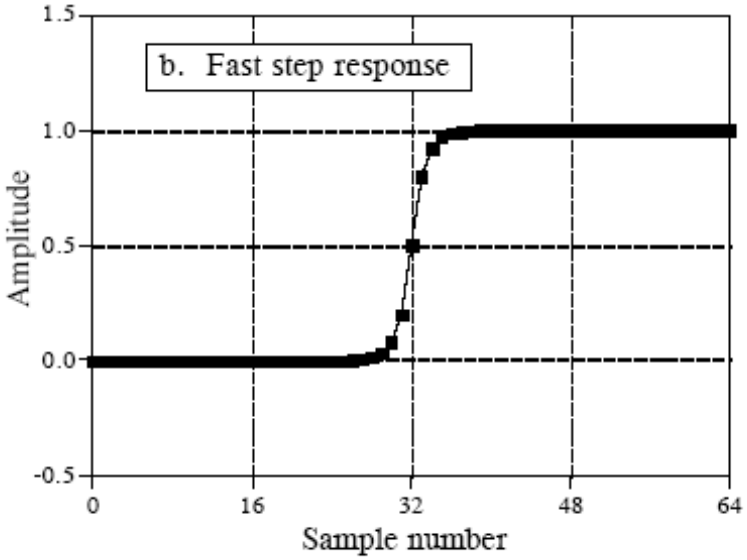
- Two different approaches for implementing a digital filter
  - **Convolve** the input signal with the digital filter's impulse response
    - Each sample in the output is calculated by weighting the samples in the input
    - The impulse response is called a **filter kernel**
    - **Finite Impulse Response** (FIR)
  - Recursive
    - Each sample in the output is calculated by weighting the samples in the input **AND** the previous samples in the output
    - Instead of filter kernel, defined by a set of **recursion coefficients**
    - **Infinite Impulse Response** (IIR)

Filter Characteristics / Design Requirements – in Time domain

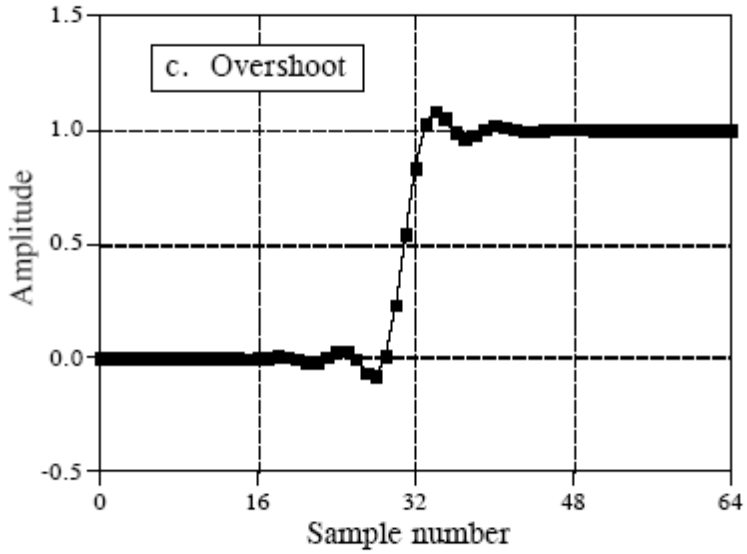
POOR

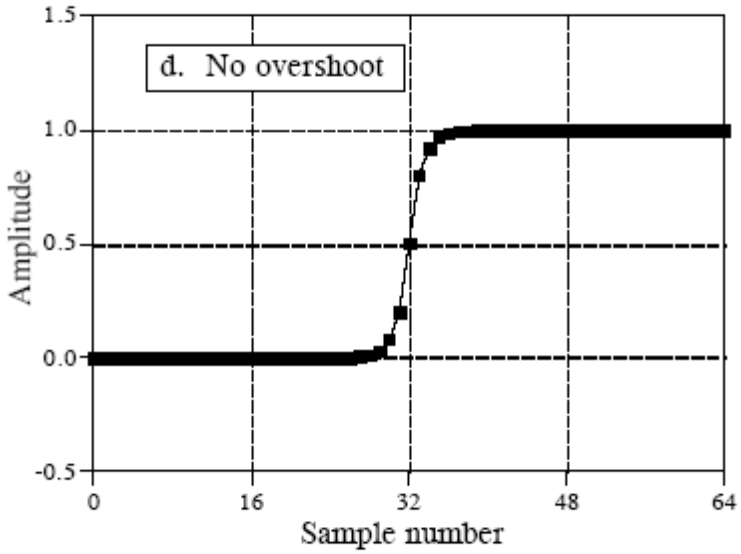


GOOD

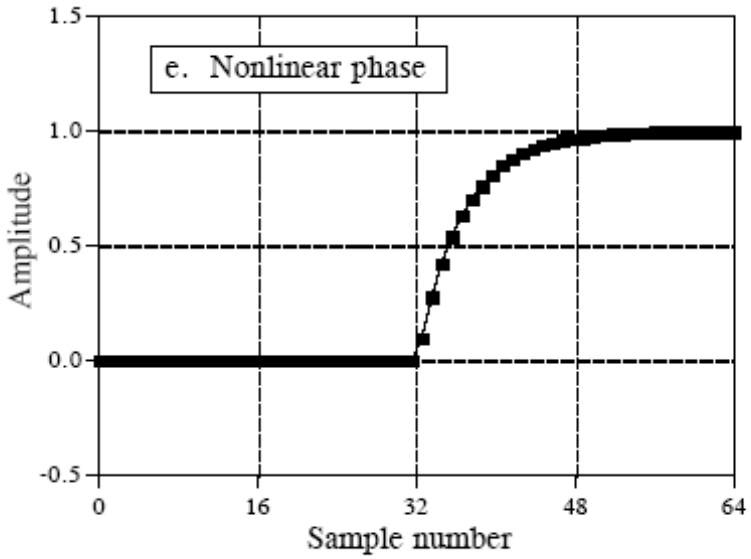


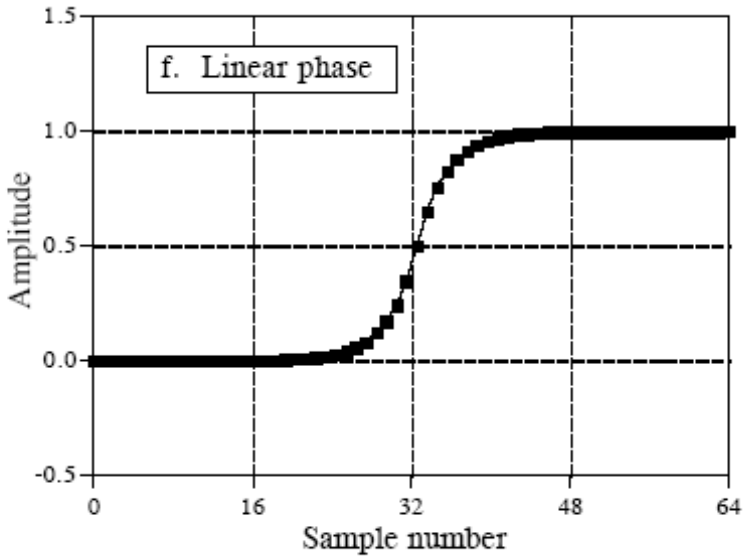
Filter Characteristics / Design Requirements – in Time domain





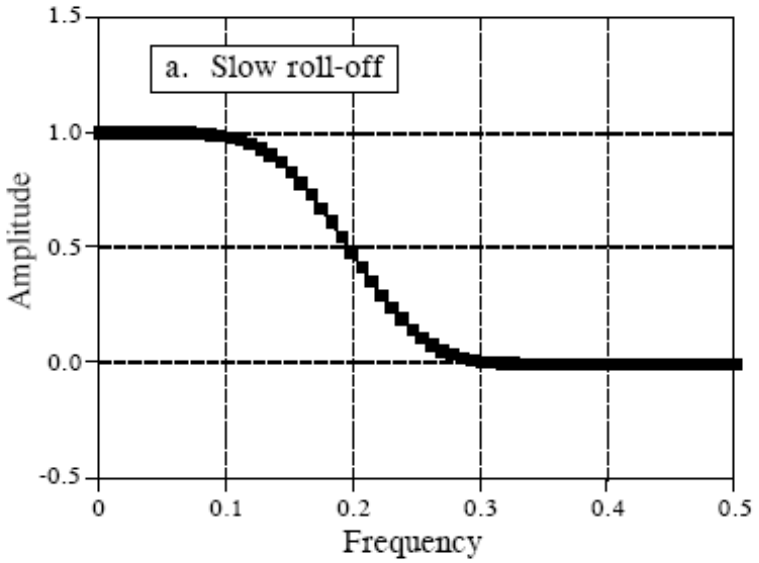
Filter Characteristics / Design Requirements – in Time domain



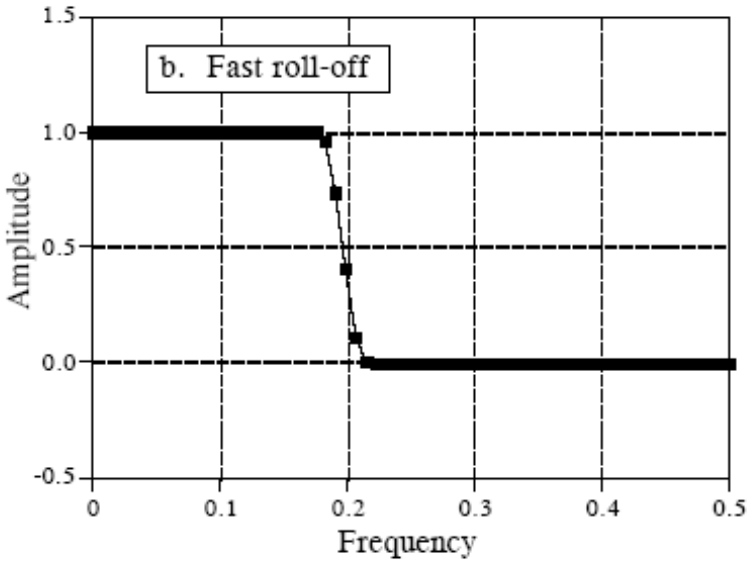


Filter Characteristics / Design Requirements – in Frequency domain

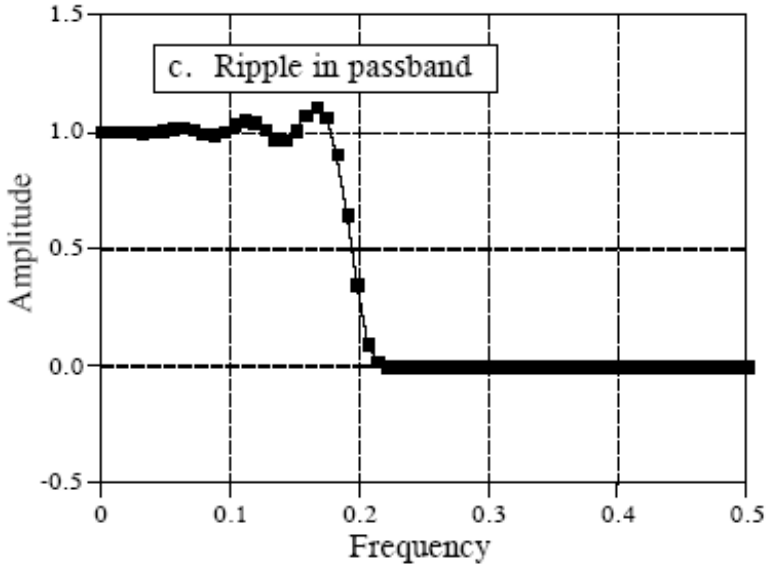
POOR

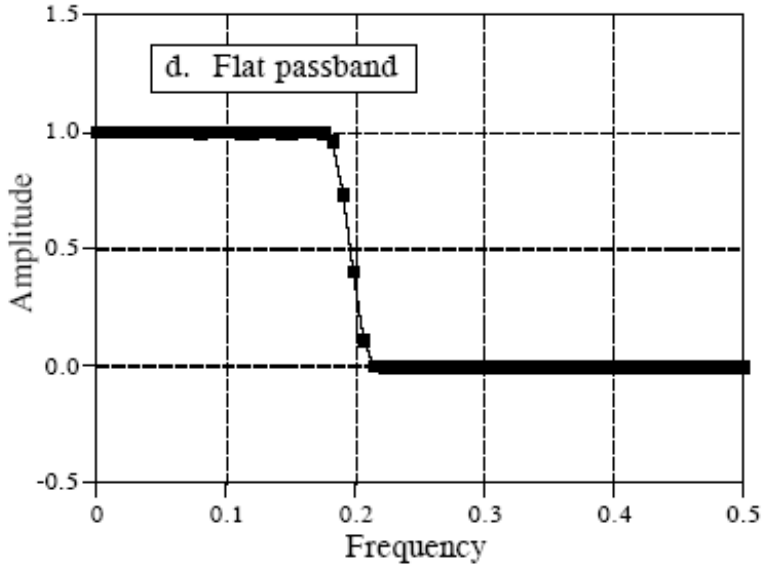


GOOD

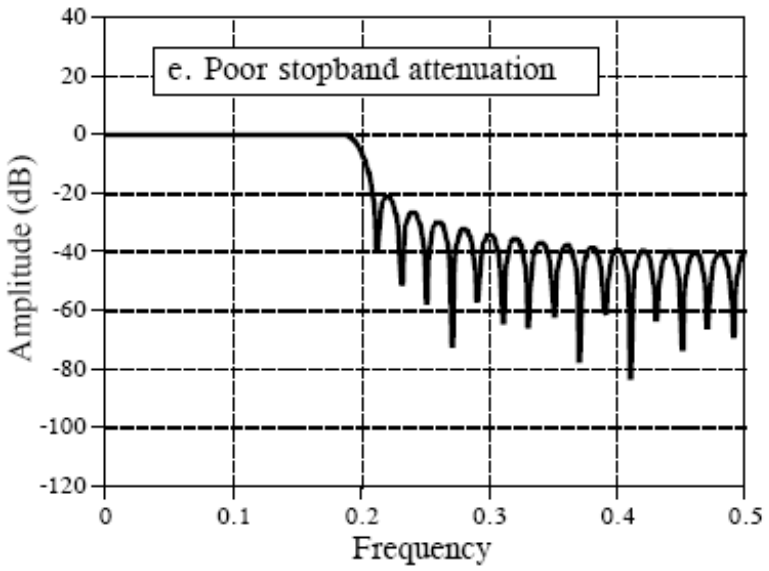


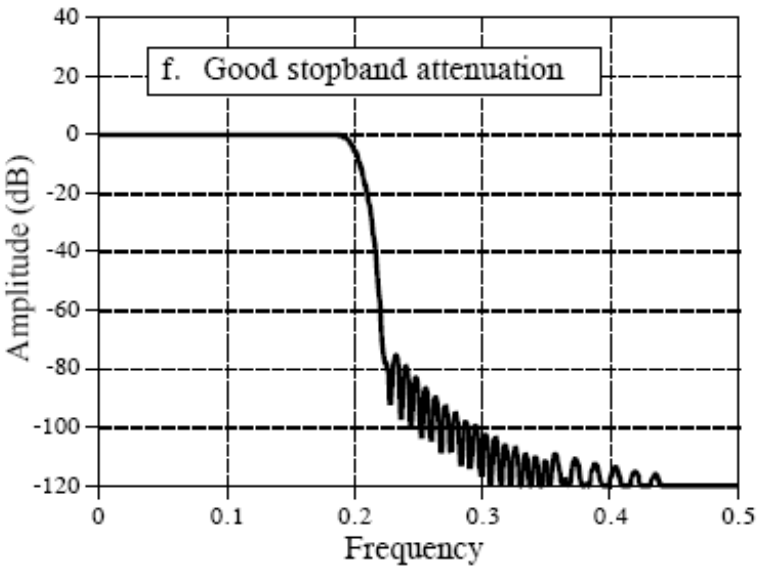
Filter Characteristics / Design Requirements – in Frequency domain





Filter Characteristics / Design Requirements – in Frequency domain





Finite Impulse Response (FIR) – Moving Average Filter

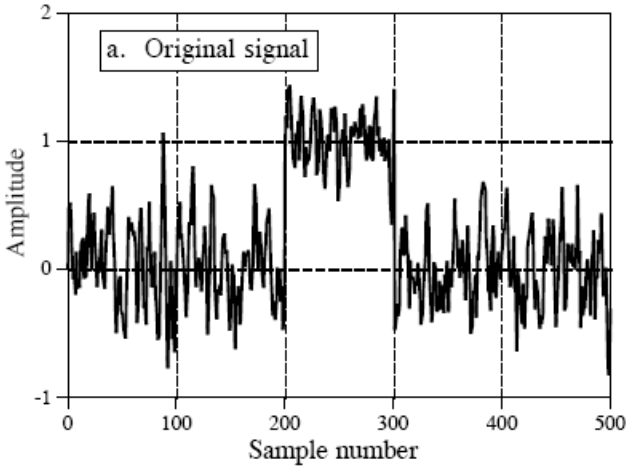
- Most common filter in DSP
  - (mainly because it's the easiest to use)

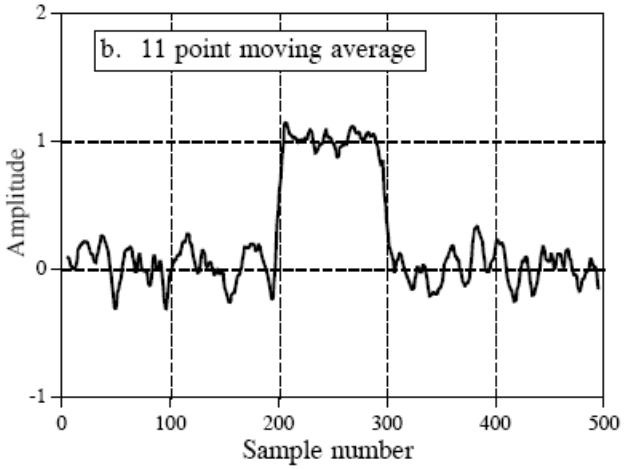
$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i-j]$$

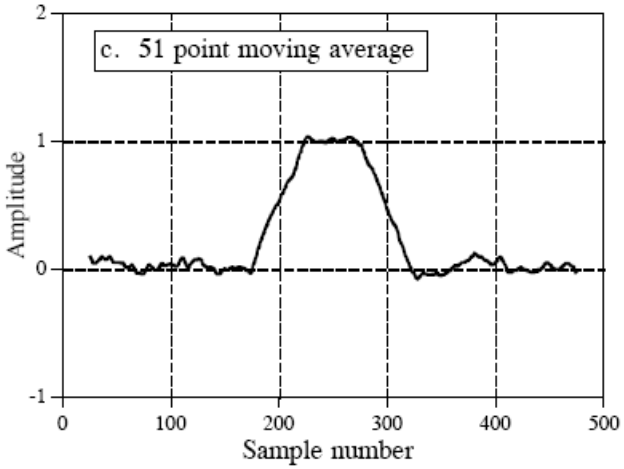
y[53] = (x[47] + x[48] + x[49] + x[50] + x[51] + x[52] + x[53])/7

- The moving average filter is **optimal** for reducing random noise while retaining a sharp step response
  - Other filters can be equally as good, but not **better**
  - Used as smoothing filter
- Good for time domain signals, which means bad for frequency domain
  - Little ability to separate one band of frequencies from another

Moving Average Filter







Example ot a moving average filter. In (a), a rectangular pulse is buried in random noise. In (b) and (c), this signal is filtered with 11 and 51 point moving average filters, respectively. As the number of points in the filter increases, the noise becomes lower; however, the edges becoming less sharp. The moving average filter is the *optimal* solution for this problem, providing the lowest noise possible for a given edge sharpness.

Moving Average Filter – Pseudo code

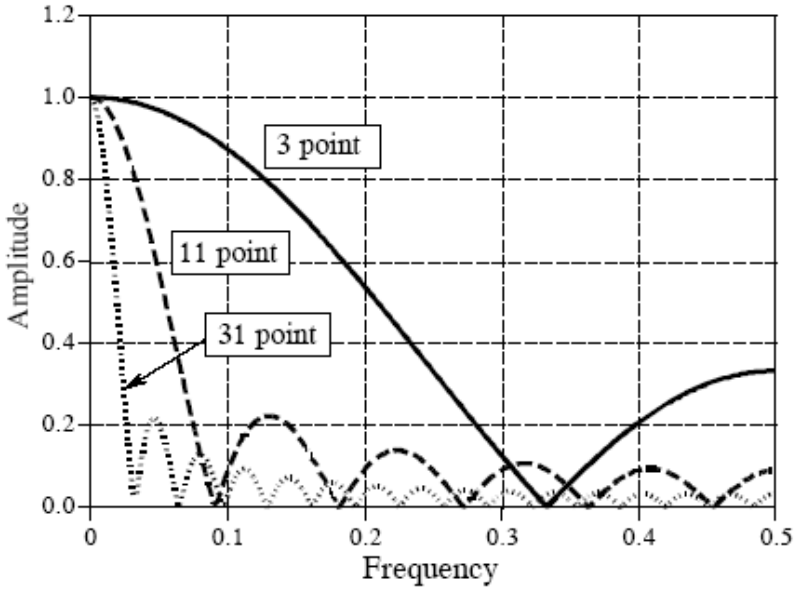
```
100 'MOVING AVERAGE FILTER
110 'This program filters 5000 samples with a 101 point moving
120 'average filter, resulting in 4900 samples of filtered data.
130 '
140 DIM X[4999] 'X[ ] holds the input signal
150 DIM Y[4999] 'Y[ ] holds the output signal
160 '
170 GOSUB XXXX 'Mythical subroutine to load X[ ]
180 '
190 FOR I% = 50 TO 4949 'Loop for each point in the output signal
200   Y[I%] = 0 'Zero, so it can be used as an accumulator
210   FOR J% = -50 TO 50 'Calculate the summation
220     Y[I%] = Y[I%] + X[I%+J%]
230   NEXT J%
240   Y[I%] = Y[I%]/101 'Complete the average by dividing
250 NEXT I%
260 '
270 END
```

Moving Average Filter – Frequency Response

- Defined as the Discrete Fourier transform of a rectangular pulse

$$H(f) = \frac{\sin(\pi f M)}{M \sin(\pi f)}$$

- Good as a smoothing filter, bad as a low-pass filter



Moving Average – Recursive Implementation

- One of the main advantages of the moving average filter is that it can be implemented very fast

  $y[53] = (x[47] + x[48] + x[49] + x[50] + x[51] + x[52] + x[53])/7$

  $y[54] = (x[48] + x[49] + x[50] + x[51] + x[52] + x[53] + x[54])/7$

  $y[54] = y[53] + (x[54] – x[47])/7$

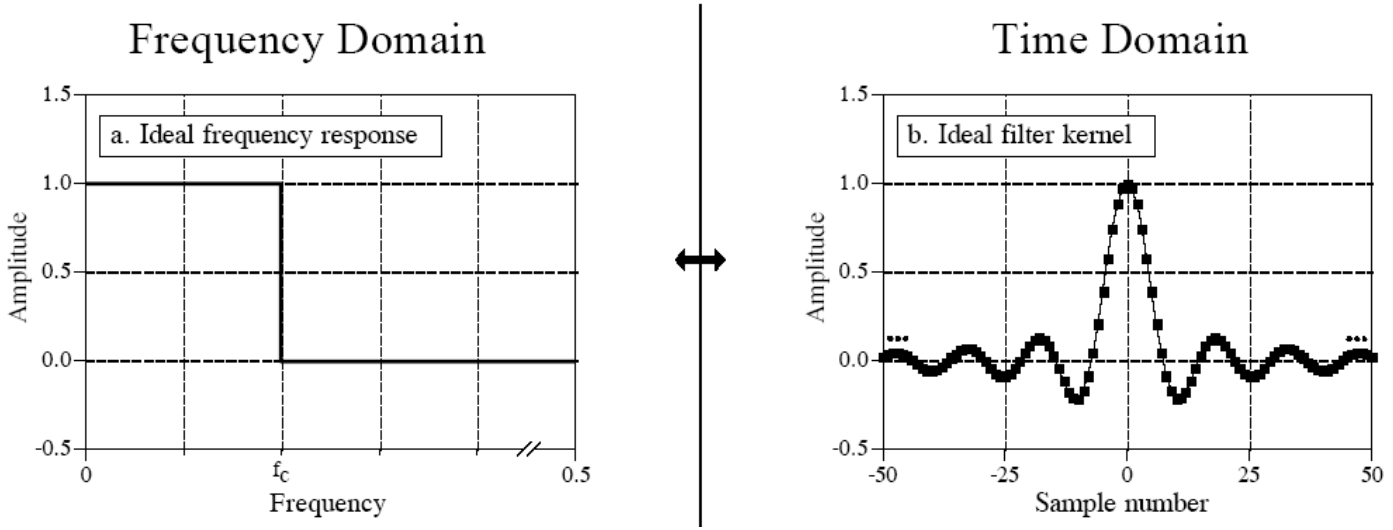
- So:

  $y[i] = y[i-1] + (x[i] – x[i-M])/M$

  where M = number of samples

- Called a recursive equation
- Not the same as in IIR filter that we will discuss later

Finite Impulse Response (FIR) – Windowed-Sinc

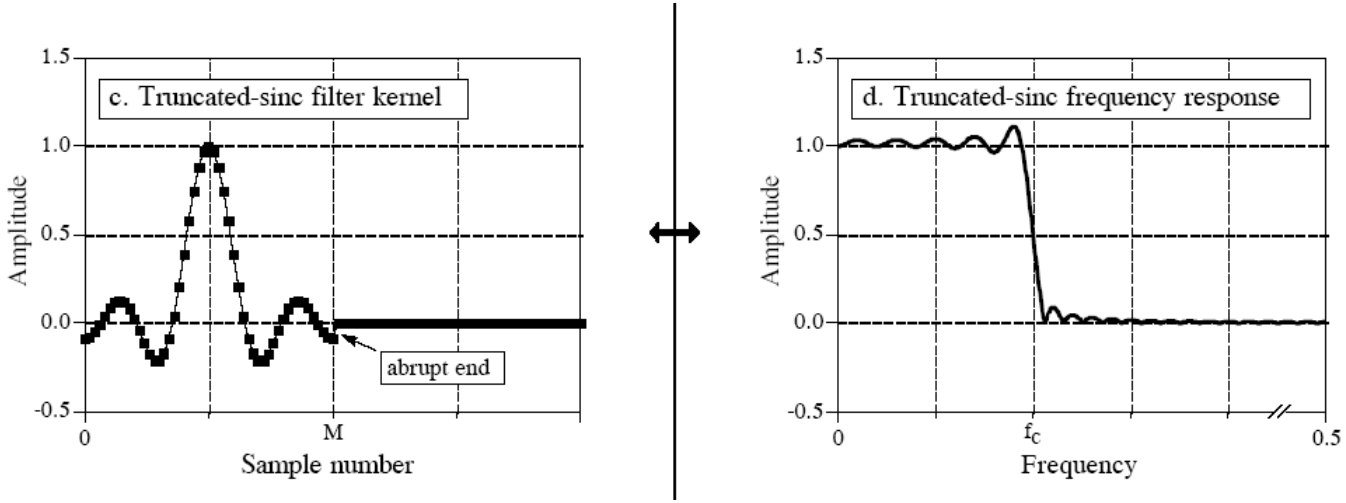


- The Inverse Fourier Transform of the 'ideal frequency response' produces a **sinc function**

$$h[i] = \frac{\sin(2\pi f_c i)}{i\pi}$$

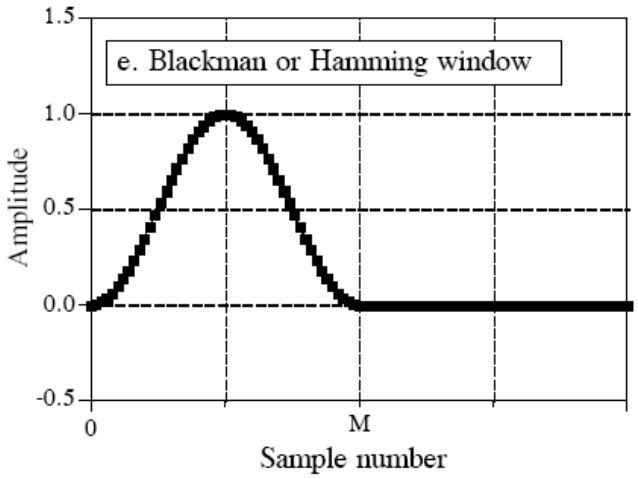
- The function is infinite in time
  - Not so hard for math
  - Hard for computers

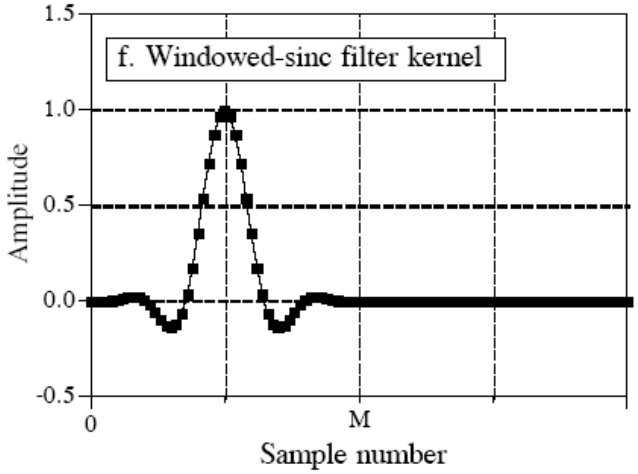
Windowed-Sinc – Modifications from the ideal

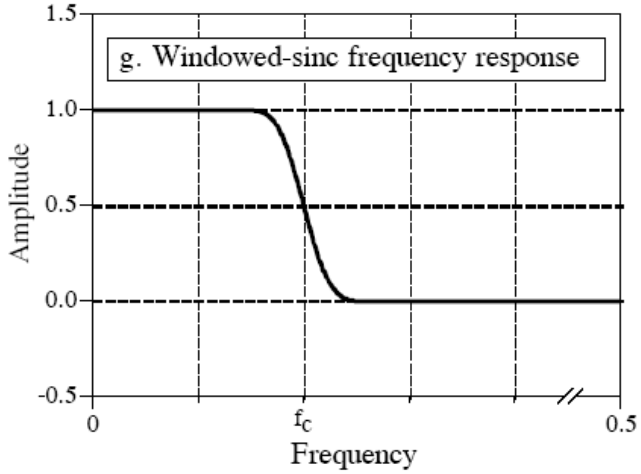


- 2 modifications from ideal
  - Truncate to M+1 points
    - Symmetrically chosen around the main lobe
    - M is even
  - The entire sequence is shifted to the right so that it runs from 0 to M
    - Makes it easier to program
- The truncation causes problems in the frequency domain
  - Excessive ripple in the passband
  - Poor attenuation in the stop band
  - Does not matter how long M is, these problems still exist

Windowed-Sinc – Windowing







- These problems can be dealt with by multiplying the *sinc* function by a windowing function such as the **Hamming** window or the **Blackman** window.
- The Blackman window is given by:

$$w[i] = 0.42 - 0.5\cos(2\pi i/M) + 0.08\cos(4\pi i/M)$$

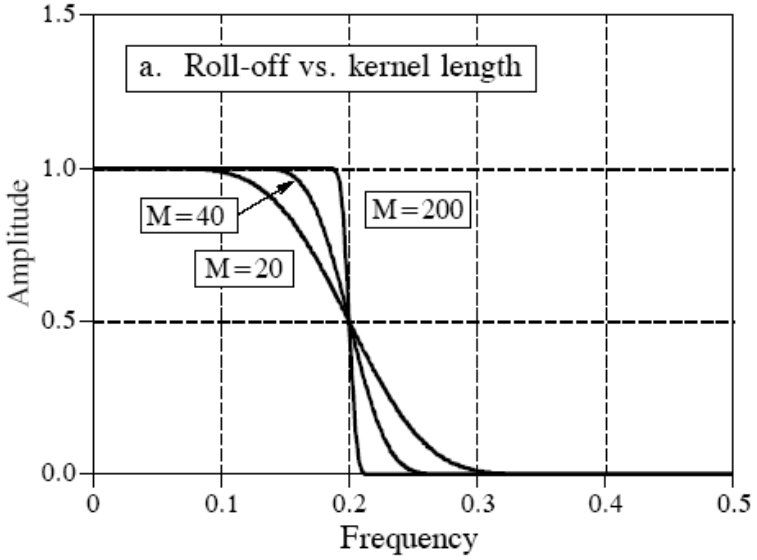
Windowed-Sinc – Blackman window

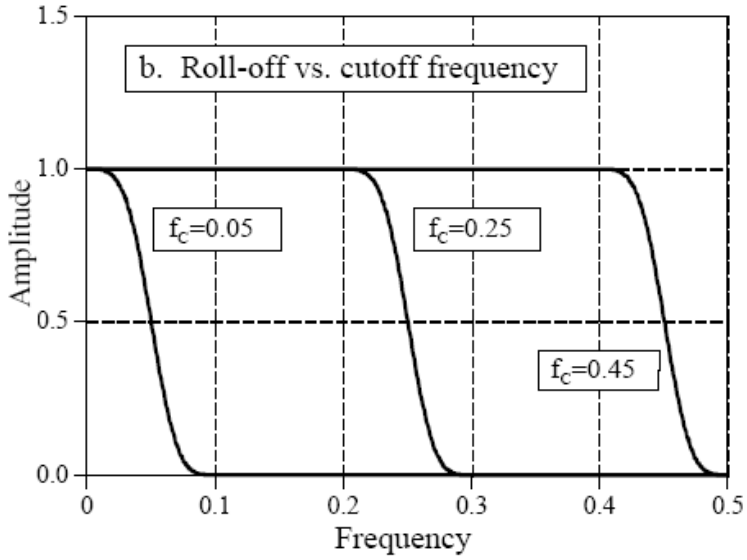
- Two parameters must be selected:
  - The cutoff frequency, f_c
    - Expressed as a fraction of the sampling rate
  - The length of the filter kernel, M
    - BW is the width of the transition band (Expressed as a fraction of the sampling rate)
    - Must be an even number

$$0 < f_c < 0.5$$

$$M \approx \frac{4}{BW}$$

$$0 < BW < 0.5$$





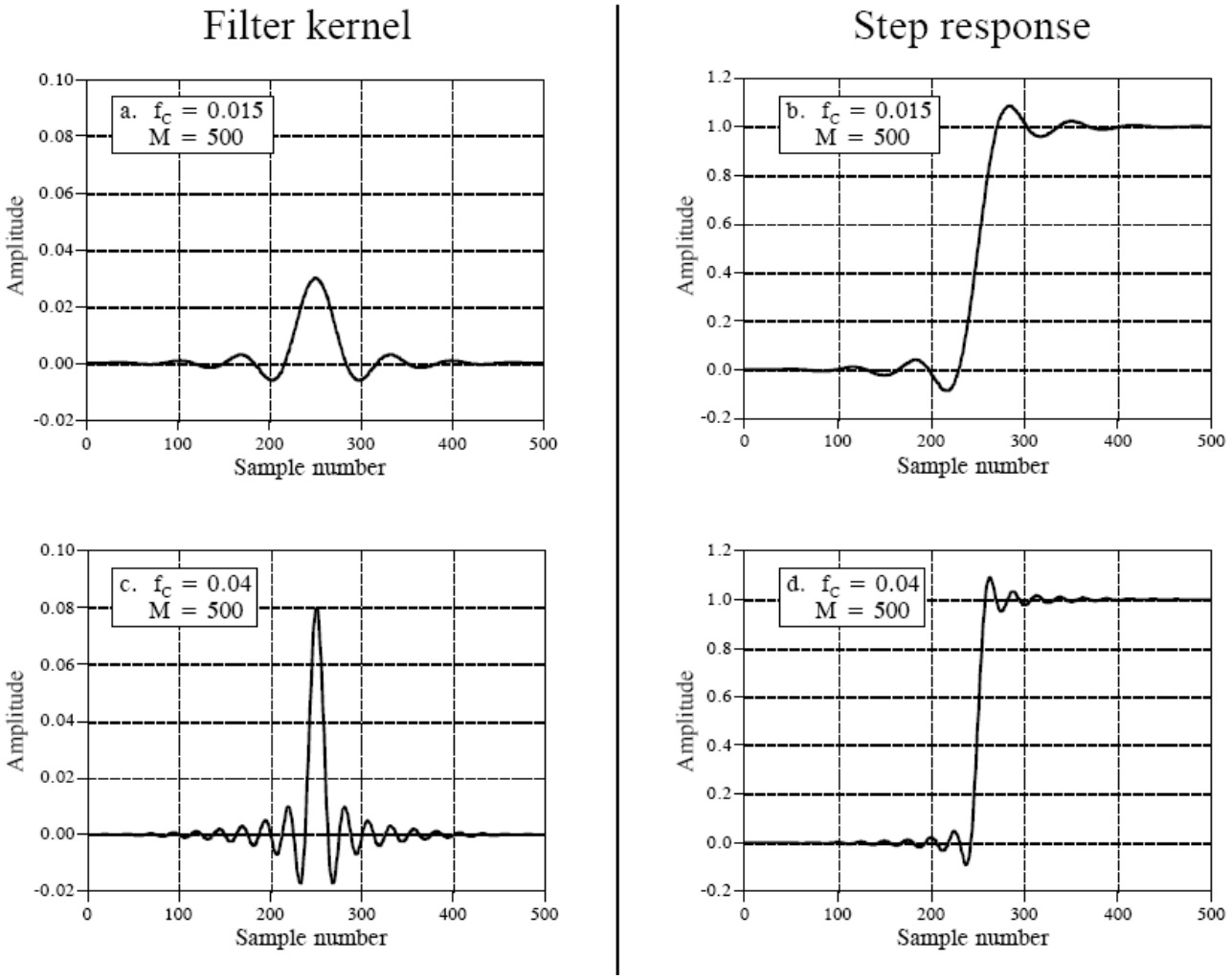
Windowed-Sinc – Blackman window

- After f_c and M have been selected, the filter kernel is calculated from the following:

$$h[i] = \begin{cases} K \dfrac{\sin\left(2\pi f_c \left(i - M/2\right)\right)}{i - M/2}\left[0.42 - 0.5\cos\left(\dfrac{2\pi i}{M}\right) + 0.08\cos\left(\dfrac{4\pi i}{M}\right)\right] & i \neq \dfrac{M}{2} \\ 2\pi f_c K & i = \dfrac{M}{2} \end{cases}$$

- The second equation is to avoid a divide by 0 when i=M/2
- K is a constant chosen to provide unity gain at zero frequency
  - Chosen such that the sum of all the samples is equal to one
  - K is typically ignored during the calculation of the filter kernel, all the samples are afterwards **normalized** as needed

Windowed-Sinc – Blackman window



Windowed-Sinc – Blackman window pseudo code

```
100 'LOW-PASS WINDOWED-SINC FILTER
110 'This program filters 5000 samples with a 101 point windowed-sinc filter,
120 'resulting in 4900 samples of filtered data.
130 '
140 DIM X[4999] 'X[ ] holds the input signal
150 DIM Y[4999] 'Y[ ] holds the output signal
160 DIM H[100] 'H[ ] holds the filter kernel
170 '
180 PI = 3.14159265
190 FC = .14 'Set the cutoff frequency (between 0 and 0.5)
200 M% = 100 'Set filter length (101 points)
210 '
220 GOSUB XXXX 'Mythical subroutine to load X[ ]
230 '
240 ' 'Calculate the low-pass filter kernel from slide 22
250 FOR I% = 0 TO 100
260   IF (I%-M%/2) = 0 THEN H[I%] = 2*PI*FC
270   IF (I%-M%/2) <> 0 THEN H[I%] = SIN(2*PI*FC * (I%-M%/2)) / (I%-M%/2)
280   H[I%] = H[I%] * (0.42 - 0.5*COS(2*PI*I%/M%) + 0.08*COS(4*PI*I%/M%) )
290 NEXT I%
```

Windowed-Sinc – Blackman window pseudo code

```
300 '
310 SUM = 0 'Normalize the low-pass filter kernel for
320 FOR I% = 0 TO 100 'unity gain at DC
330   SUM = SUM + H[I%]
340 NEXT I%
350 '
360 FOR I% = 0 TO 100
370   H[I%] = H[I%] / SUM
380 NEXT I%
390 '
400 FOR J% = 100 TO 4999 'Convolve the input signal & filter kernel
410   Y[J%] = 0
420   FOR I% = 0 TO 100
430     Y[J%] = Y[J%] + X[J%-I%] * H[I%]
440   NEXT I%
450 NEXT J%
460 '
470 END
```

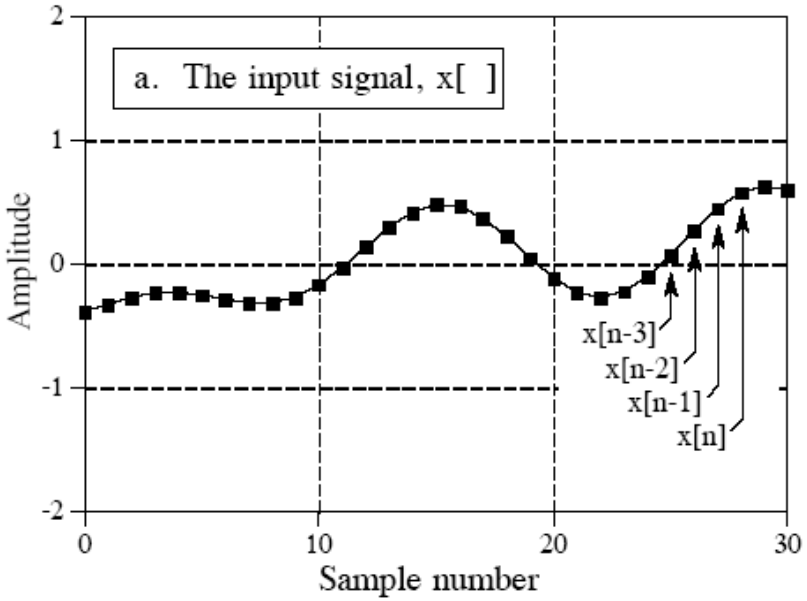
Infinite Impulse Response (IIR)

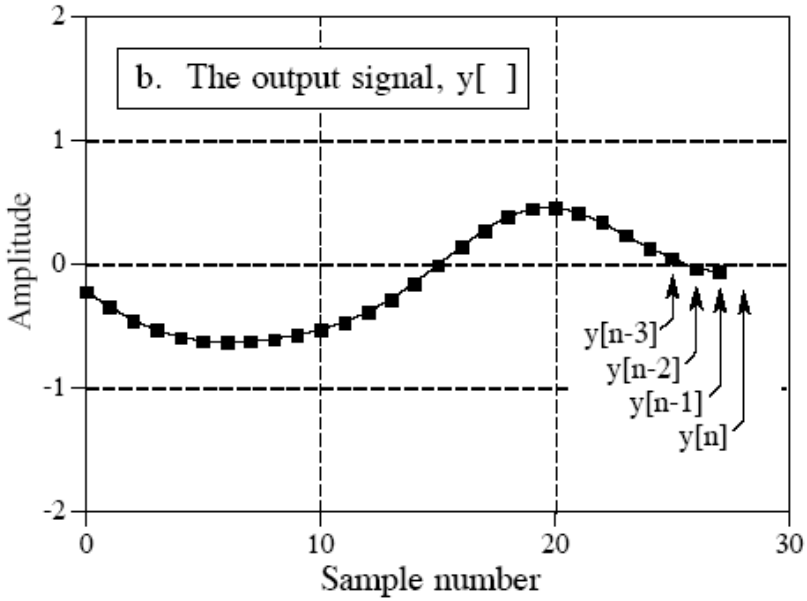
- Efficient way of achieving a long impulse response without having to perform a long convolution
- Execute rapidly, but have less performance and flexibility than other digital filters
- **Recursion equation**:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \ldots$$
$$+ b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \ldots$$

  - a_i and b_i are **recursion coefficients**
- Typical impulse response will be a sinusoidal oscillation that exponentially decays and is infinitely long

Infinite Impulse Response (IIR)





Infinite Impulse Response (IIR) – Pseudo code

```
100 'RECURSIVE FILTER
110 '
120 DIM X[499] 'holds the input signal
130 DIM Y[499] 'holds the filtered output signal
140 '
150 GOSUB XXXX 'mythical subroutine to calculate the recursion
160 ' 'coefficients: A0, A1, A2, B1, B2
170 '
180 GOSUB XXXX 'mythical subroutine to load X[ ] with the input data
190 '
200 FOR I% = 2 TO 499
210   Y[I%] = A0*X[I%] + A1*X[I%-1] + A2*X[I%-2] + B1*Y[I%-1] + B2*Y[I%-2]
220 NEXT I%
230 '
240 END
```

Infinite Impulse Response (IIR) – Single pole, low pass

- Use only 2 coefficients a_0 and b_1

$$y[n] = a_0 x[n] + b_1 y[n-1]$$

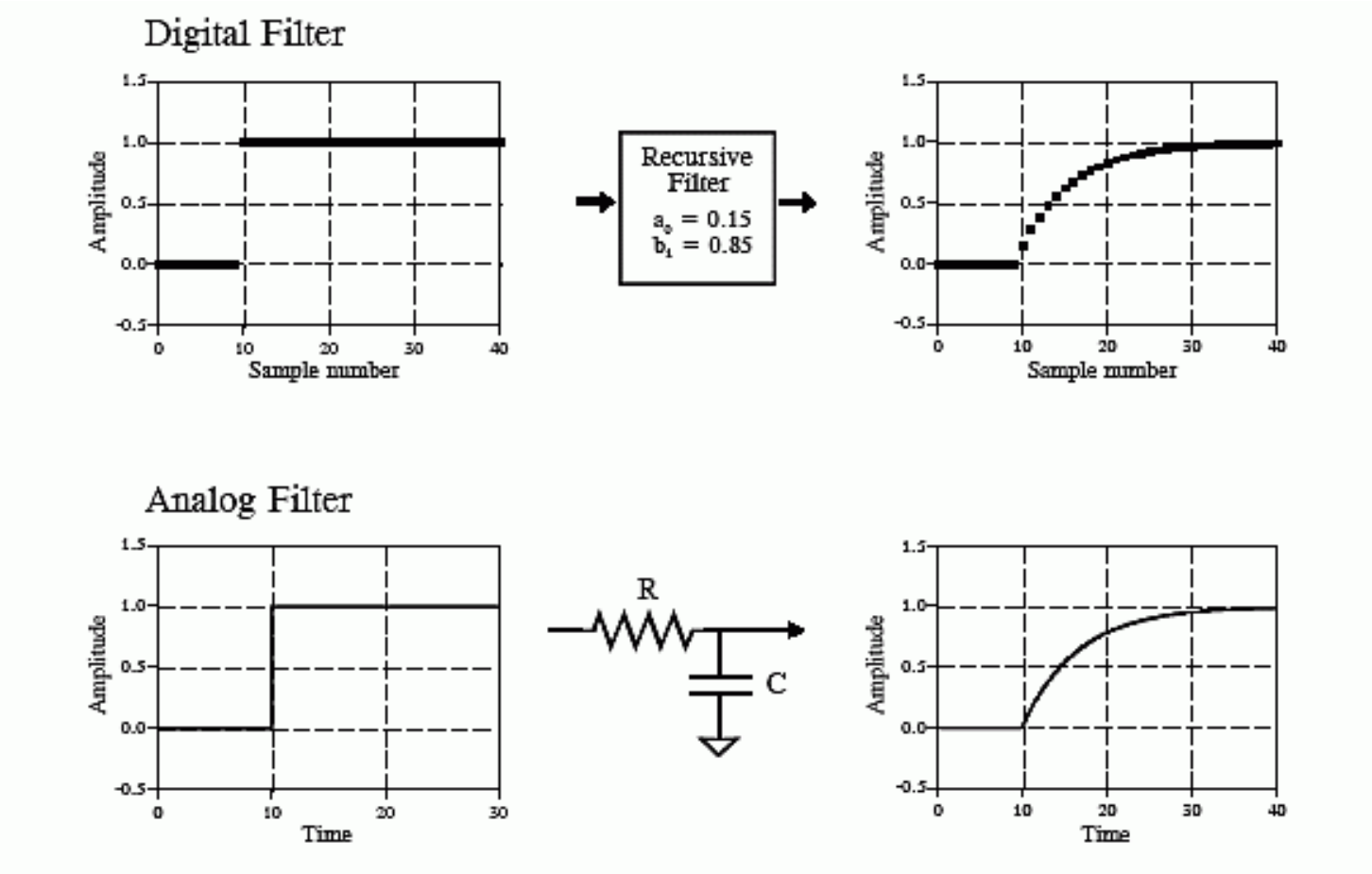
- Good for when you would use an RC network such as:
  - DC removal,
  - High-frequency noise suppression,
  - Wave shaping,
  - Smoothing,
  - etc
- Equations:

$$a_0 = 1 - e^{-2\pi f_c}$$

$$b_1 = e^{-2\pi f_c}$$

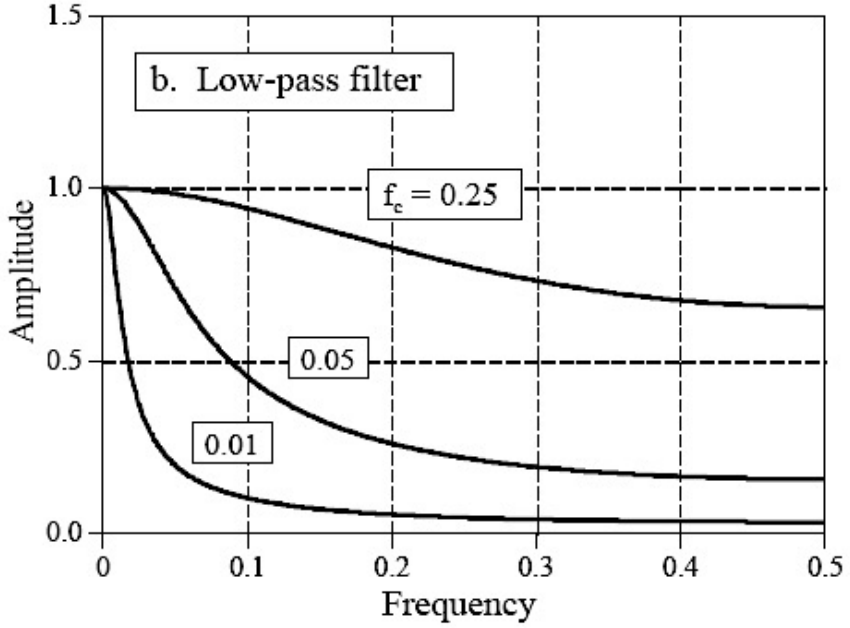
  - f_c is the -3dB **cutoff frequency**

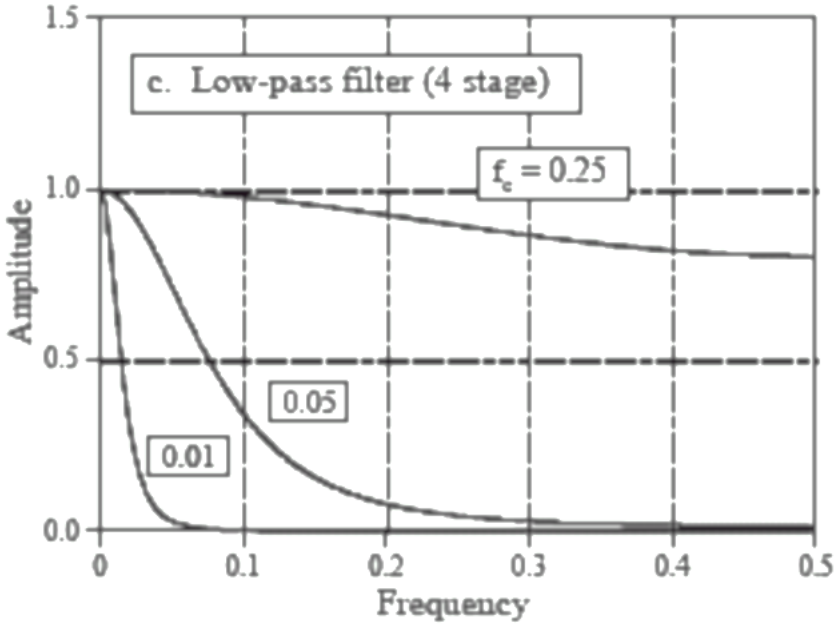
Infinite Impulse Response (IIR) – Single pole, low pass



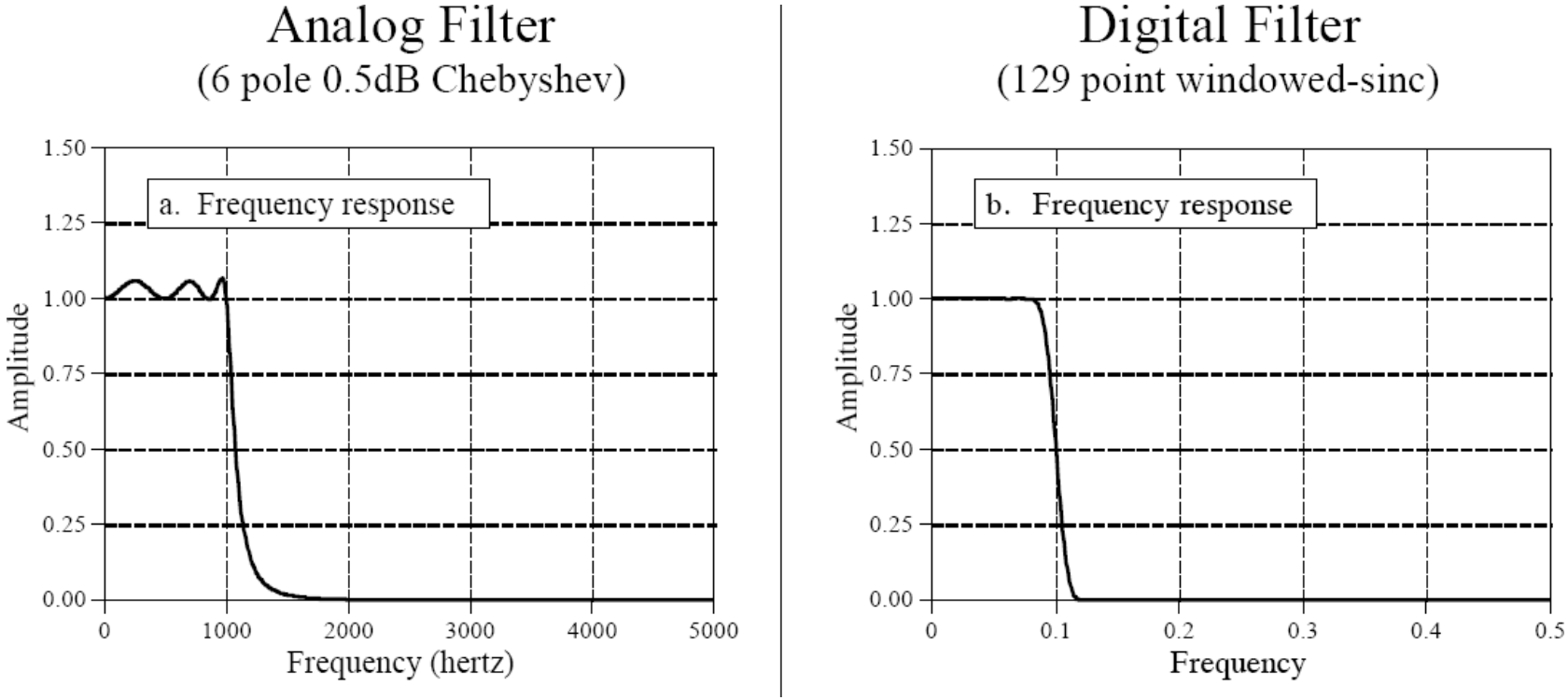
Infinite Impulse Response (IIR) – Frequency Response

- Single pole IIR filters are not good at separating one band of frequencies from another
- Good in time, poor in frequency
- Can be improved by cascading filters



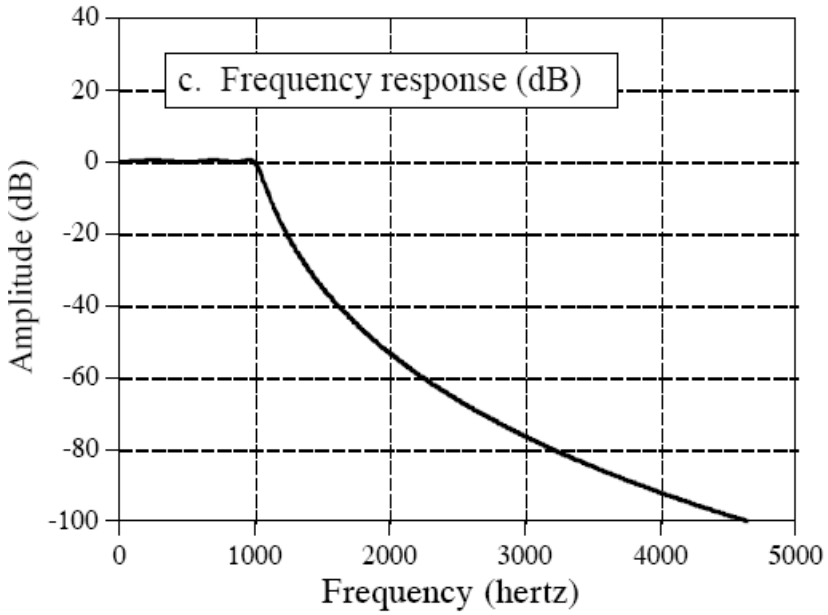


Analog vs. Digital

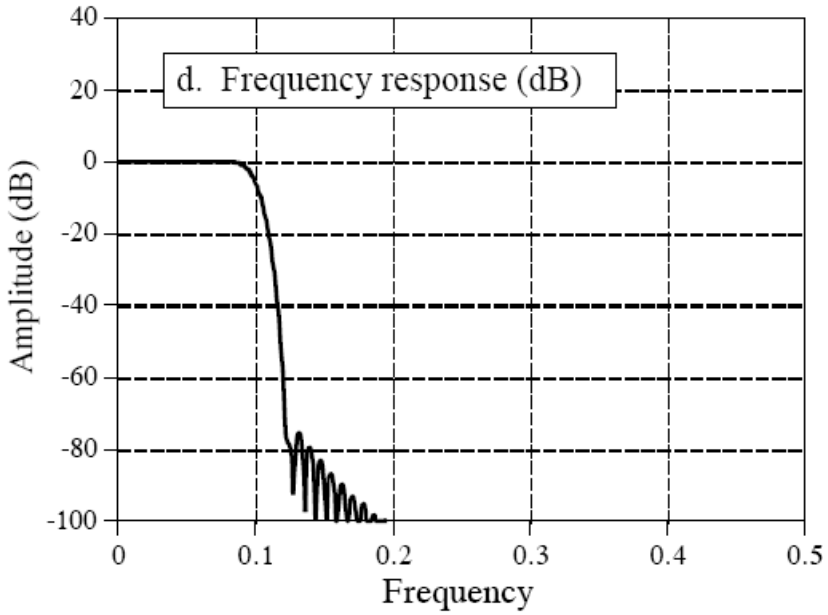


Analog vs. Digital

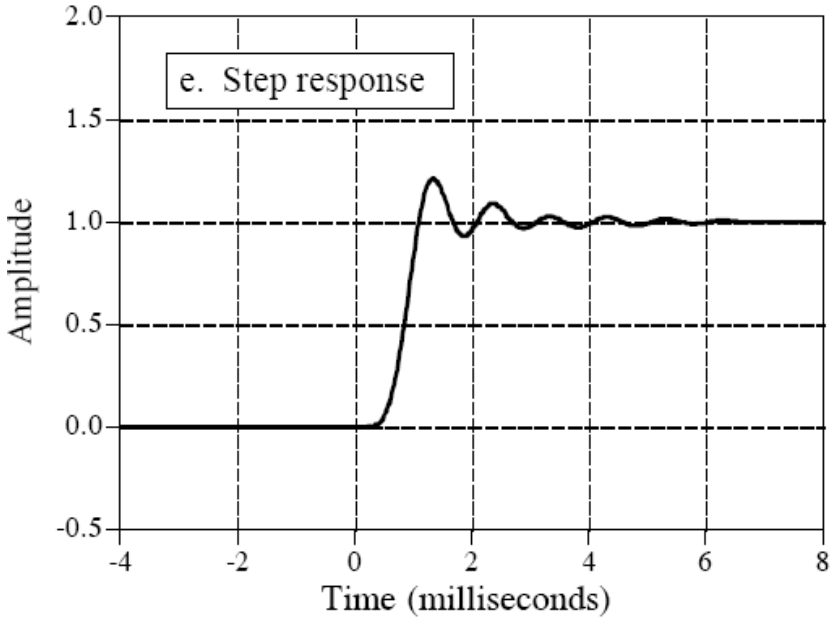
Analog Filter

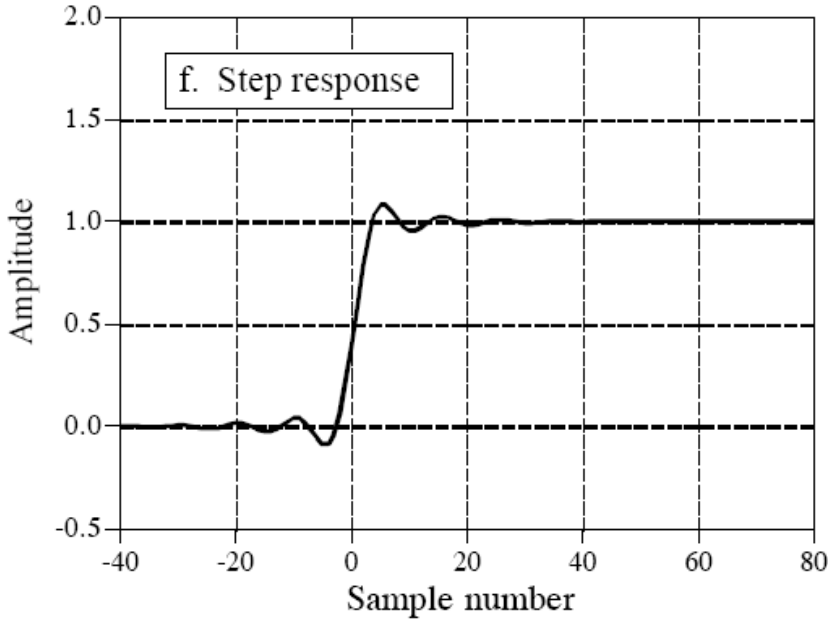


Digital Filter



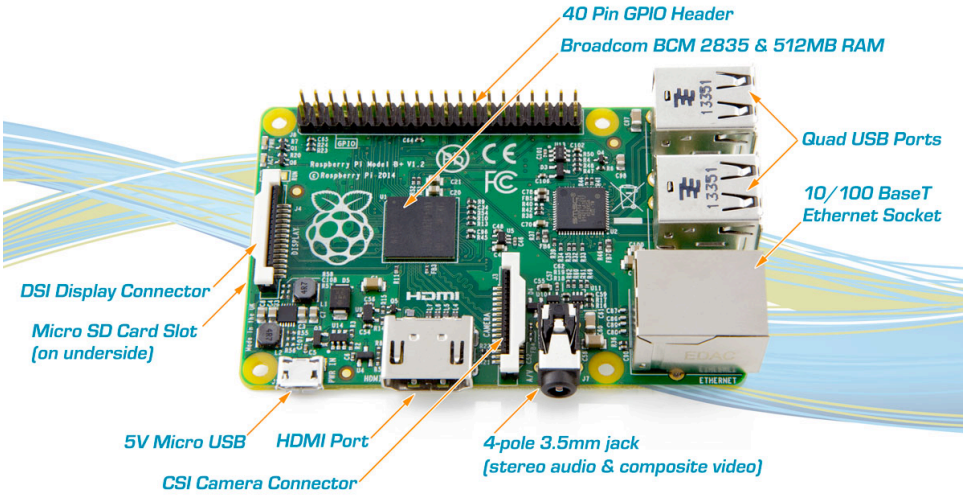
Analog vs. Digital



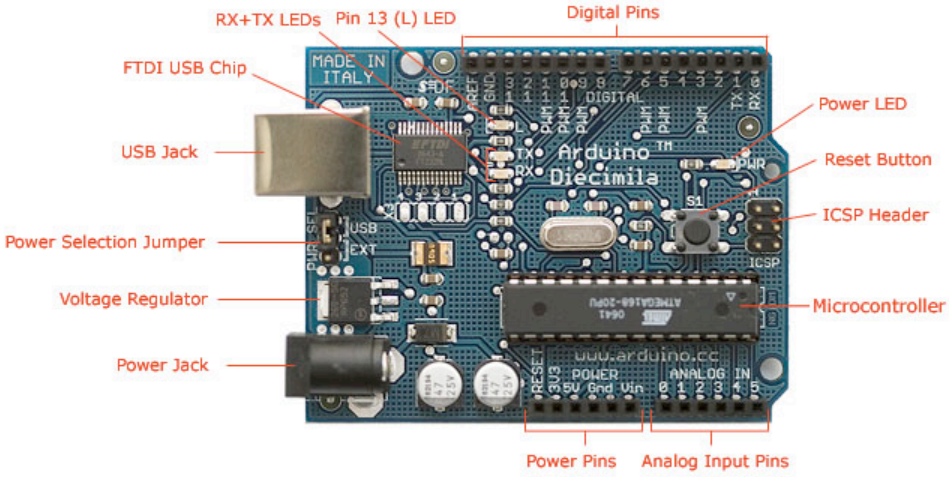


Digital Processing Hardware

- DSP (Digital Signal Processor)
- FPGA (Field Programmable Gate Array)
- MCU/MPU (Microcontroller/Microprocessor Unit)
- GPU (Graphics Processing Unit)

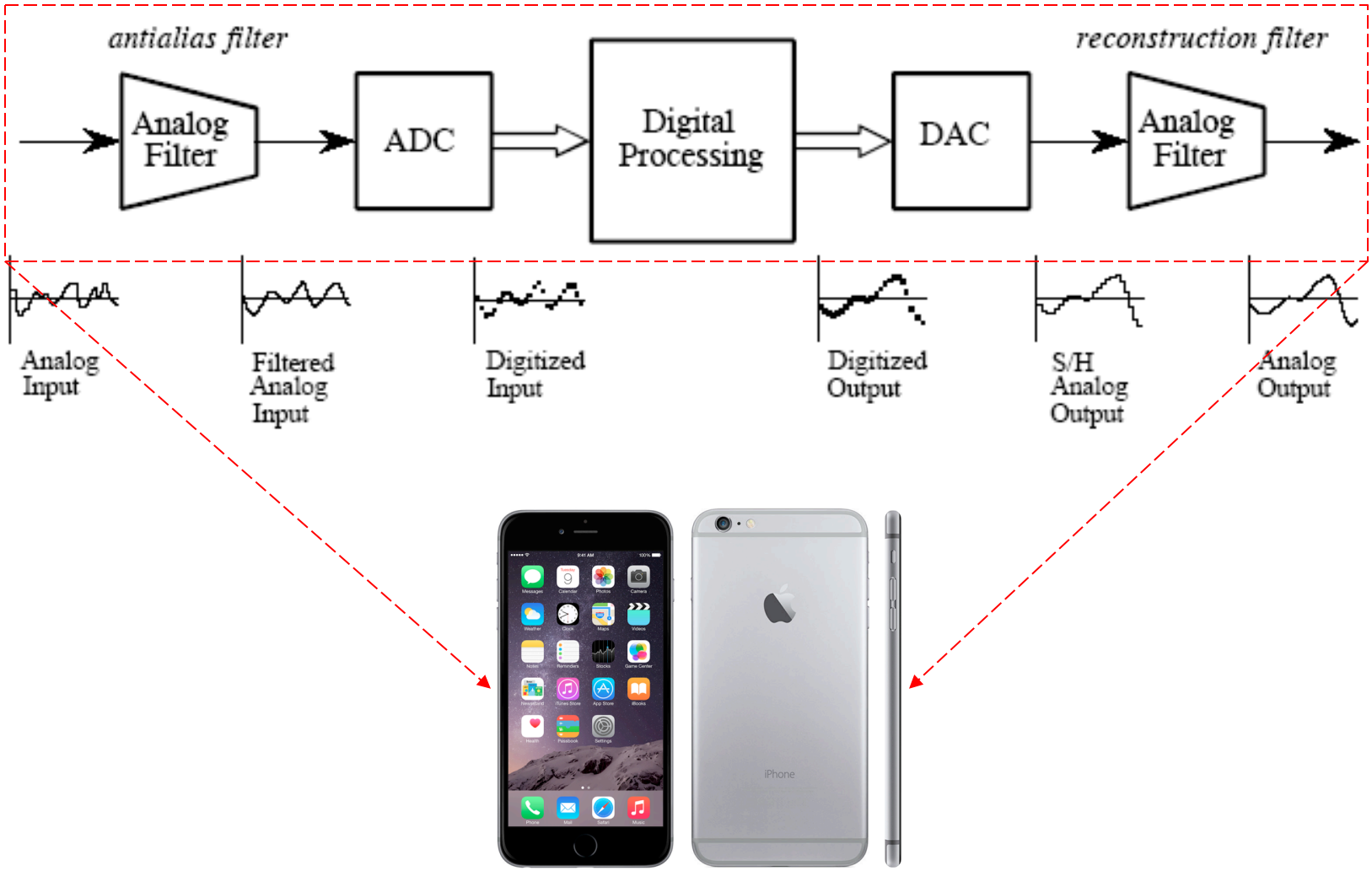


Raspberry Pi B+



Arduino Diecimilia

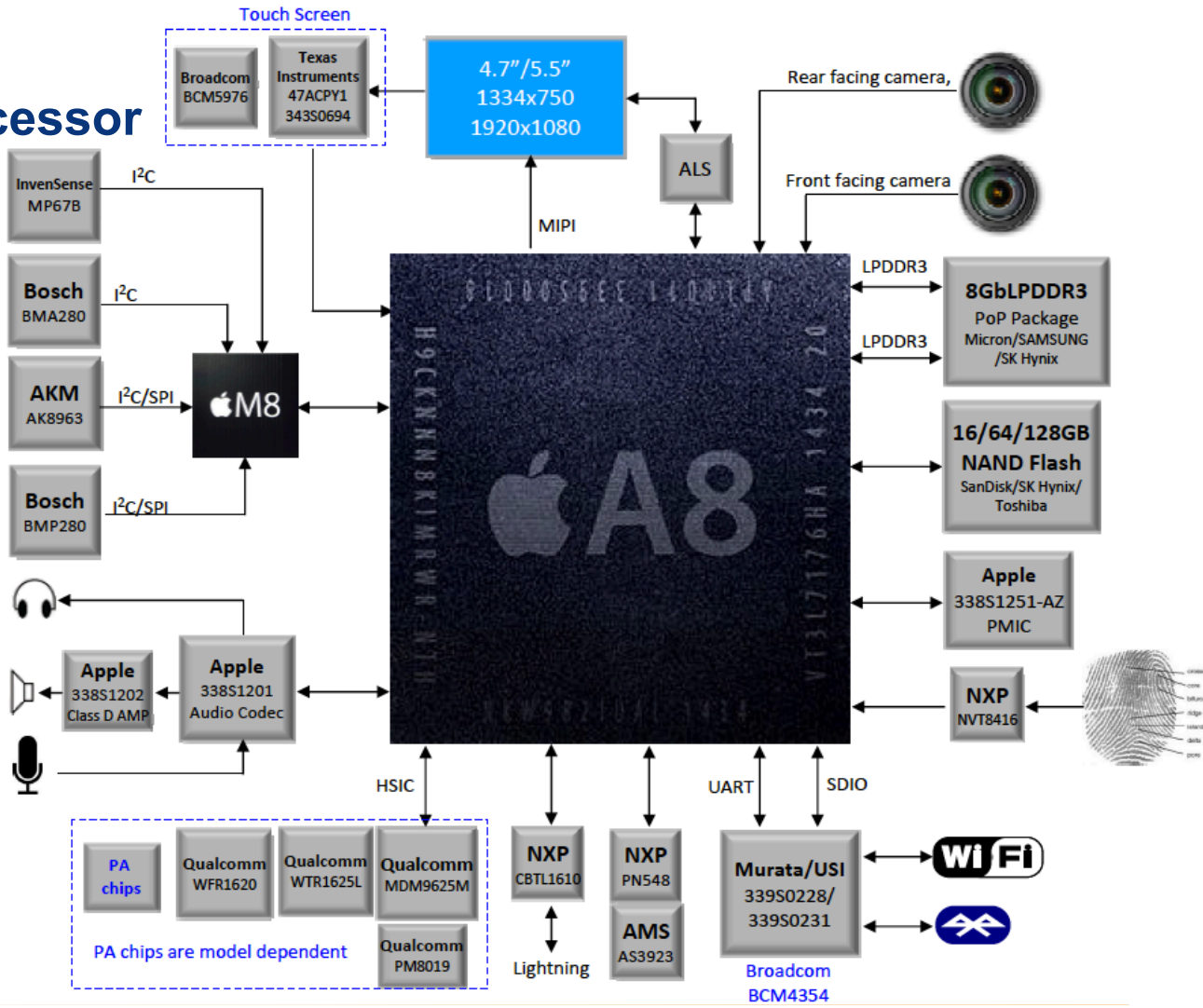
Digital Processing Hardware



Digital Processing Hardware

- **A8 processor**
- **M8 motion co-processor**

"collect, process, and store sensor data even if the device is asleep, and applications can retrieve data when the device is powered up again. This reduces power draw of the device and saves battery life"

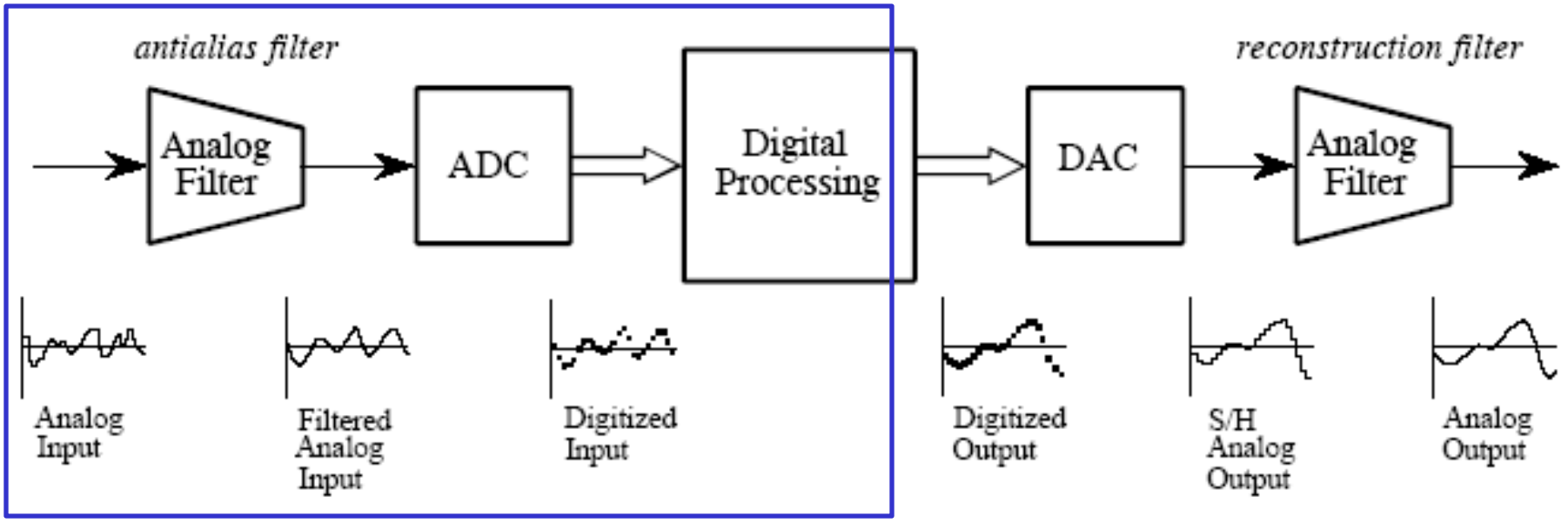


Conclusions

- Digital Filters
  - Background
  - Finite Impulse Response (FIR)
    - Moving average
    - Windowed-sinc
  - Infinite Impulse Response (IIR)
    - Single pole
  - Comparison with Analog Filters

The lab this week

- Analog Filtering (1 week)
  - Basics of generating waveforms using Arduino and waveform analysis using oscilloscope will be introduced.
  - Analog filters and circuits will be implemented using discrete electronic components



Questions?

References

[1] Smith, Steven W., “The Scientist and Engineer’s Guide to Digital Signal Processing.” California Technical Publishing, San Diego, CA 1997-1999.