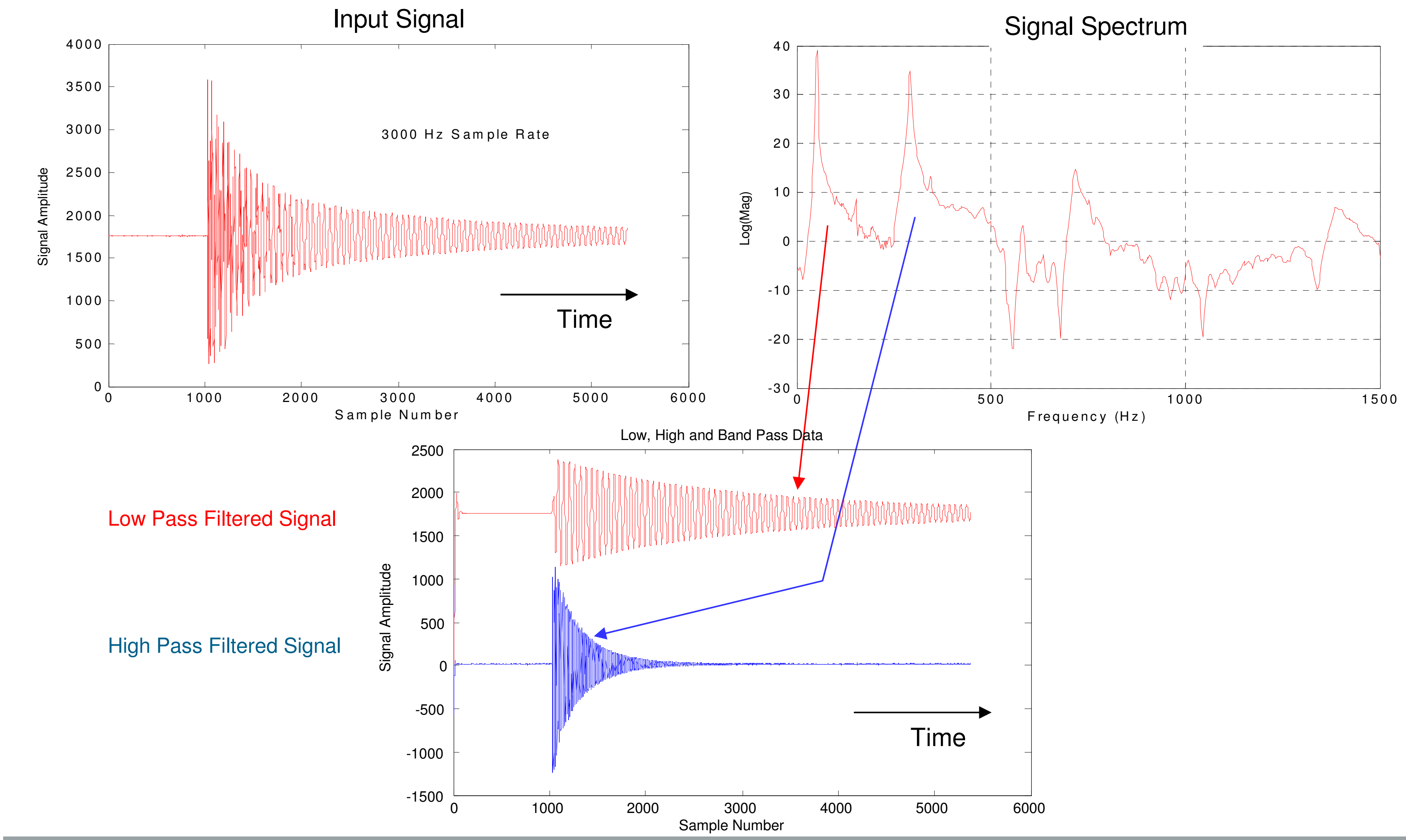
September 17, 2007

ColdFire® Technology & DSP

AMF-IND-T0094

Ms. Maureen Helm
Dr. David Hayner

Intro Demo on Your Desks



Low Pass Filtered Signal

High Pass Filtered Signal

Presenters

- **Maureen Helm**
  Systems Engineer, Sensor System Architectures

  With Freescale/Motorola for 5 years; background in advanced microprocessor design verification; current focus in sensor algorithms.

- **Dr. David Hayner**
  DMTS, Manager, Sensor System Architectures

  With Freescale/Motorola for 13 years focus on consumer products: Printers, Optical and Hard Disk Drives
  Prior to FSL: Developed video transport, distribution, processing and compression systems
  Inertial Stabilization and Pointing of large Imaging Reconnaissance Systems

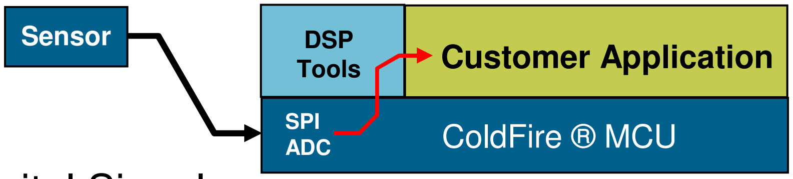
  Expertise: Multi-dimensional and Statistical Digital Signal Processing, Adaptive Control and Servo Systems

Presentation Objectives

- Review of Demo Running on Laptops
- Appreciation of the DSP Capability of ColdFire®
- Review of Basic Digital Signal Processing Concepts
- Understanding of the DSP Filtering Tools
- Successful Implementation of DSP Filtering Systems
- Summary of Our Results

Why DSP on ColdFire®

- Many applications need data from real world sensors
  - Accelerometers, Gyros, Pressure, Temperature
  - Velocity, Strain, Color, E-Field, Magnetics, . . .

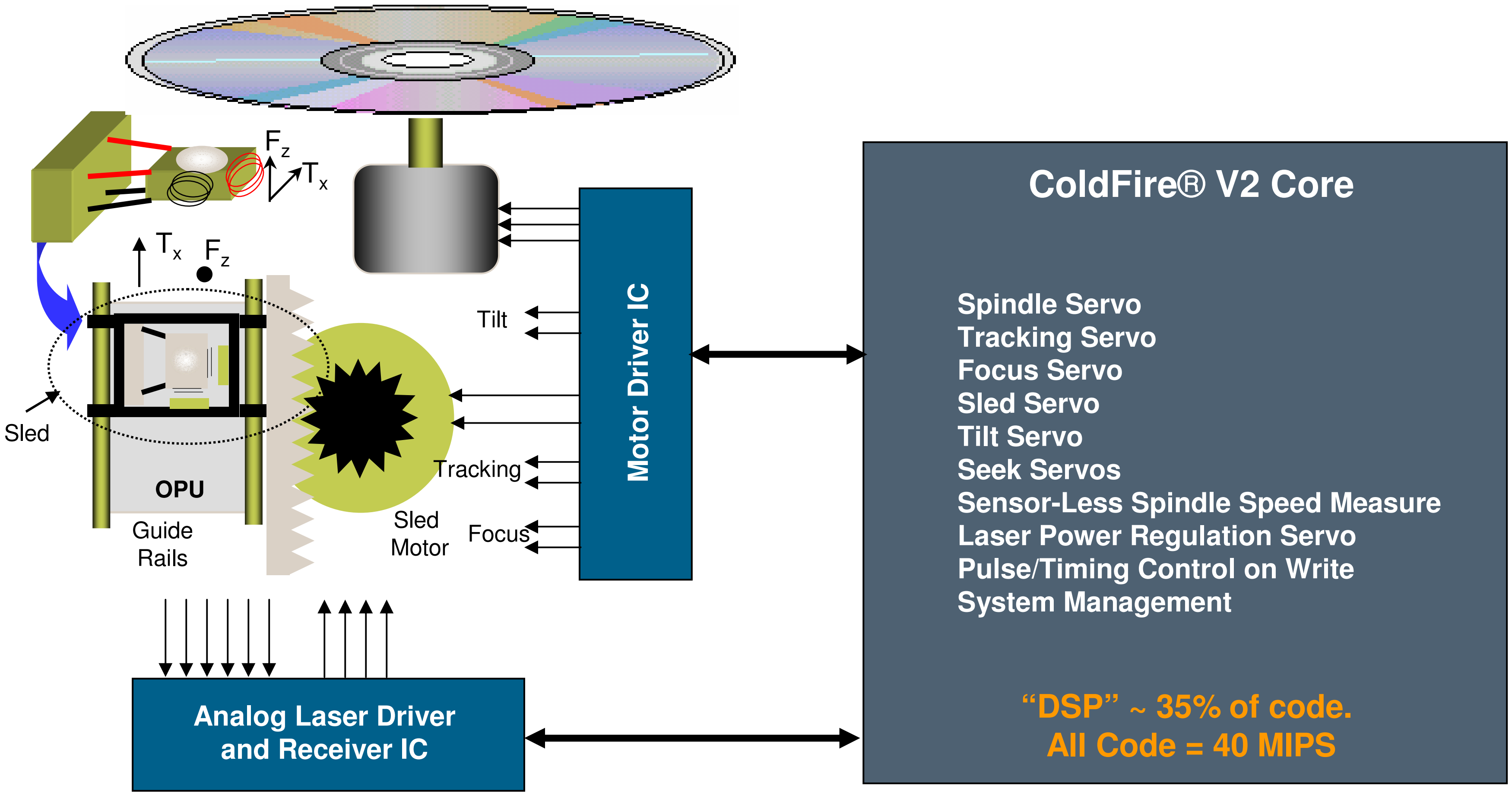


- The primary function of the application is not the Digital Signal Processing of data, but rather the intelligent use of the data.
  - FSL is providing the basic DSP tools to help extract the information

- Many apparent "High-End" DSP applications really are not
  - HDD: 10-15% DSP, 90-85% is data testing, if-then-else, comm, timing
  - Sensor-Less Motor Control: Again, about 20% DSP, 80% other.
  - A traditional "DSP" MCU may not be the best System Solution to realize 10-20% of the code.

- Freescale's DSP blocks and tools will enable our ColdFire® customers to cost effectively and QUICKLY integrate DSP functionality into their applications and products.
  - Providing very sophisticated DSP on ColdFire® for 2-3% of processor BW
  - Allowing our customers to focus on apps and markets, not DSP.

Why DSP on ColdFire® : Optical Disk Drive Example

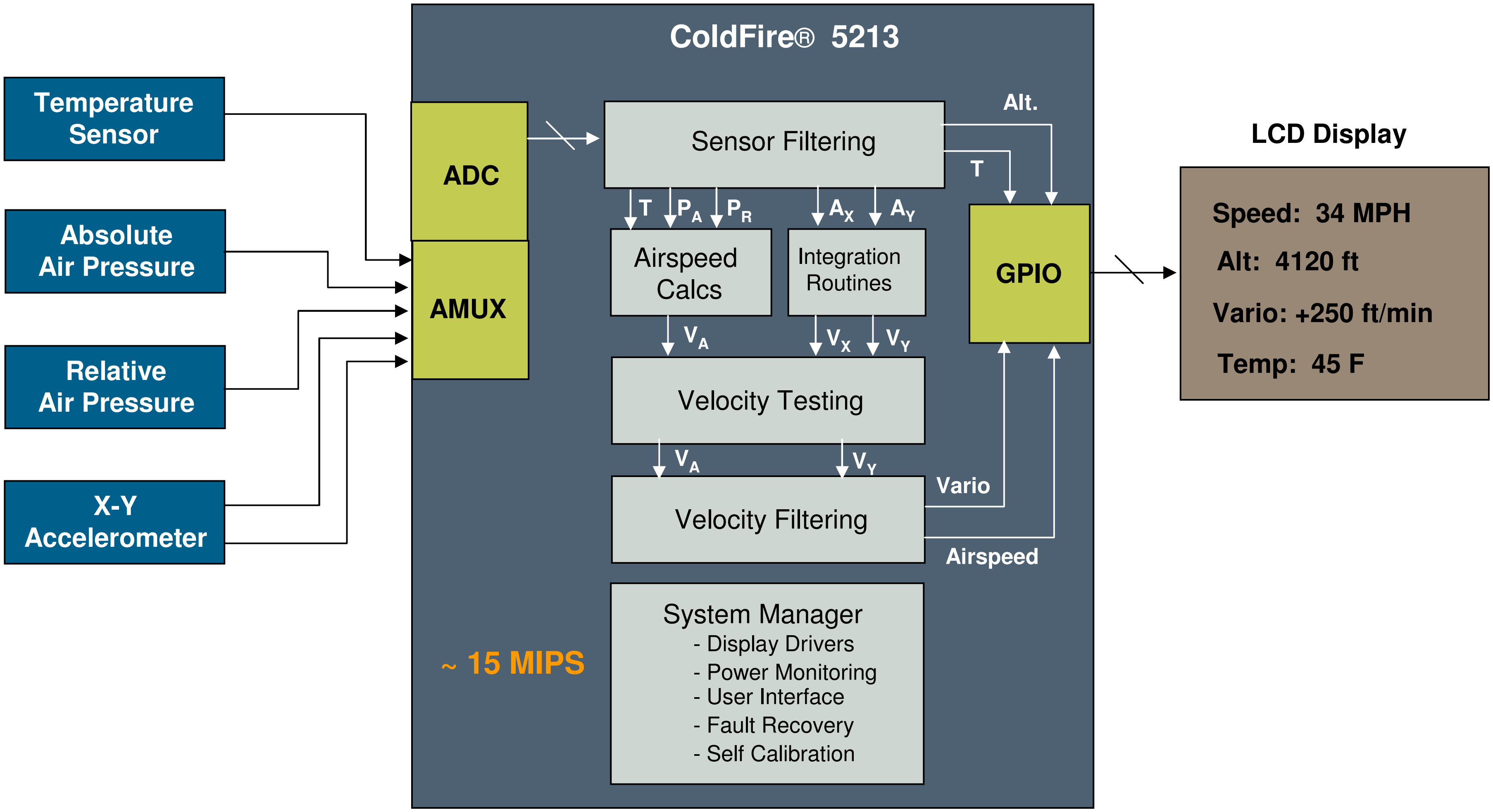


ColdFire® V2 Core

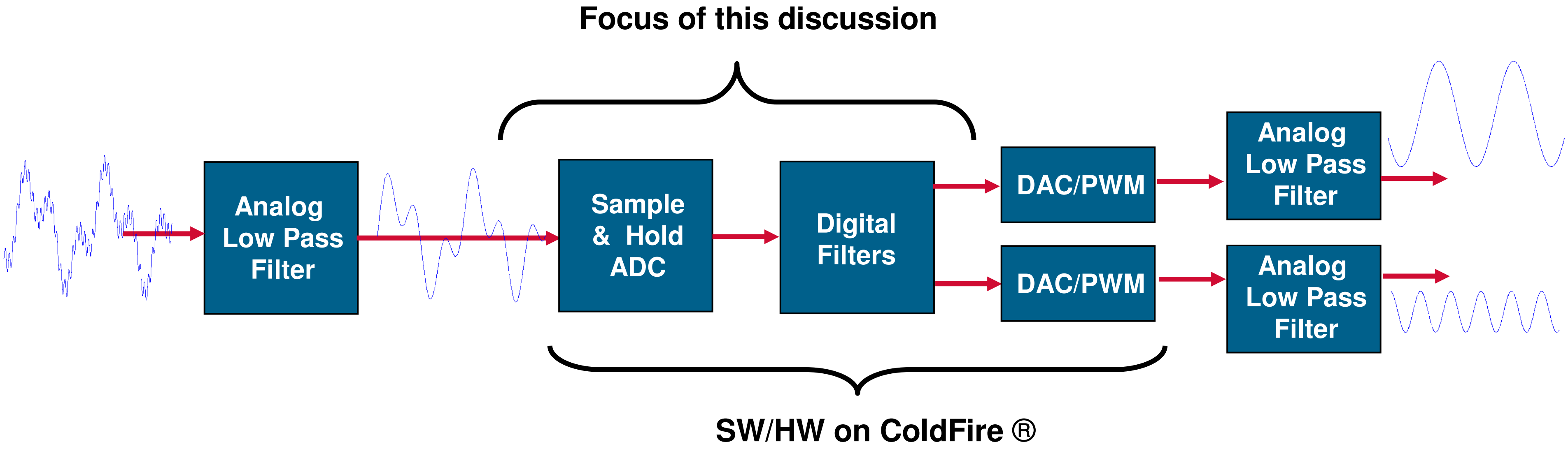
- Spindle Servo
- Tracking Servo
- Focus Servo
- Sled Servo
- Tilt Servo
- Seek Servos
- Sensor-Less Spindle Speed Measure
- Laser Power Regulation Servo
- Pulse/Timing Control on Write
- System Management

"DSP" ~ 35% of code.
All Code = 40 MIPS

Why DSP on ColdFire® :
True Airspeed Sensor, Altimeter and Variometer



Typical DSP Chain



Topics:

1) Review of Analog Filters

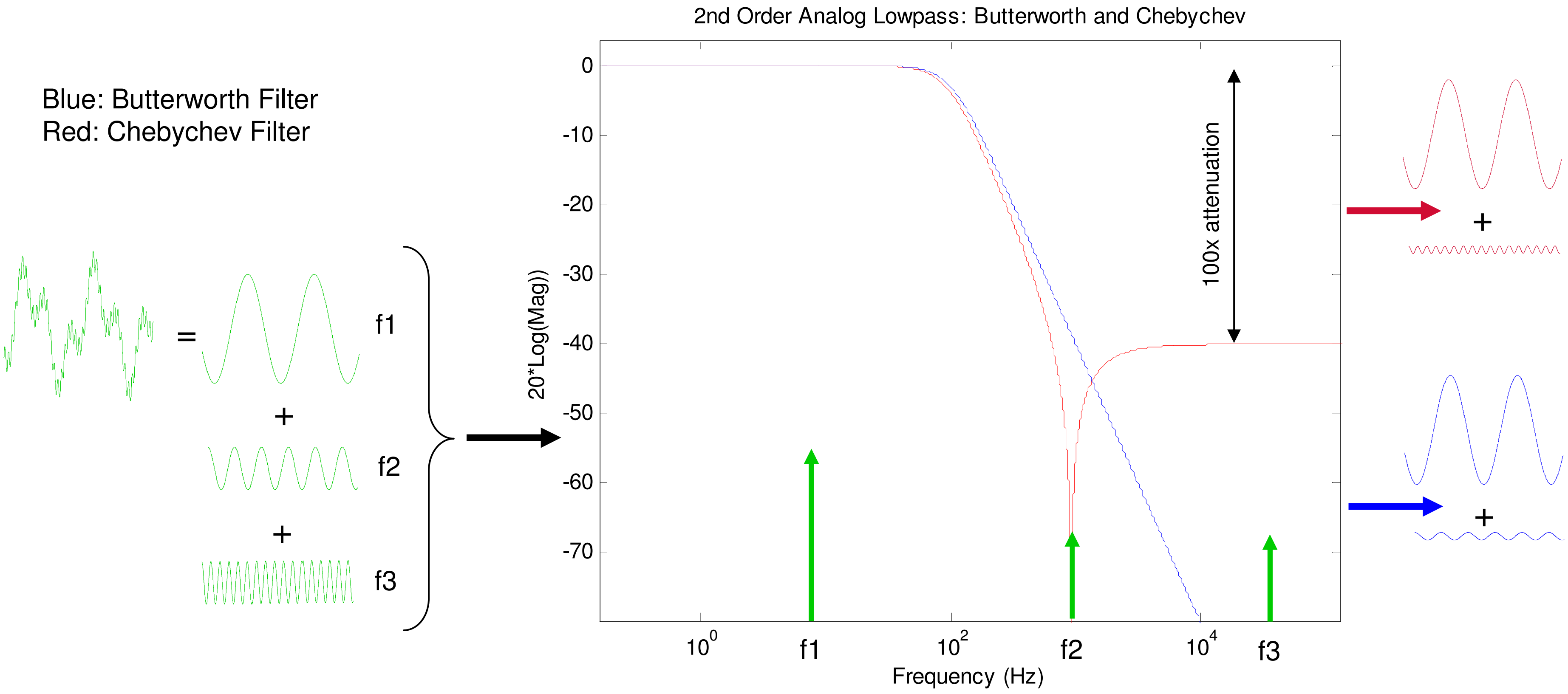
2) Sampling, Aliasing

3) Digital Filtering Examples

4) Digital to Analog

2nd Order Analog Low Pass Filters

Blue: Butterworth Filter
Red: Chebychev Filter

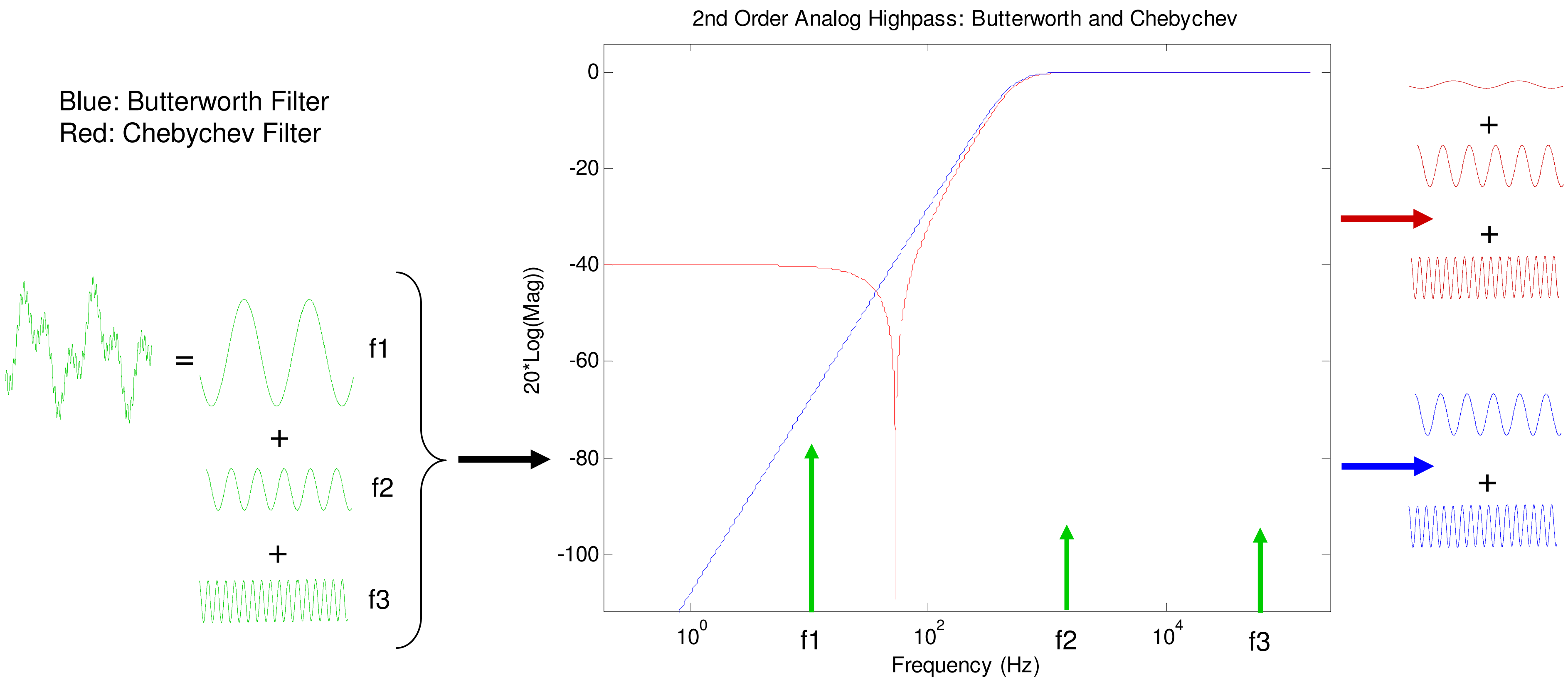


Generic 2nd Order Butterworth LP Transfer Function

$$H(s) = \frac{w_c^2}{s^2 + ks + w_c^2}$$

2nd Order Analog High Pass Filters

Blue: Butterworth Filter
Red: Chebychev Filter

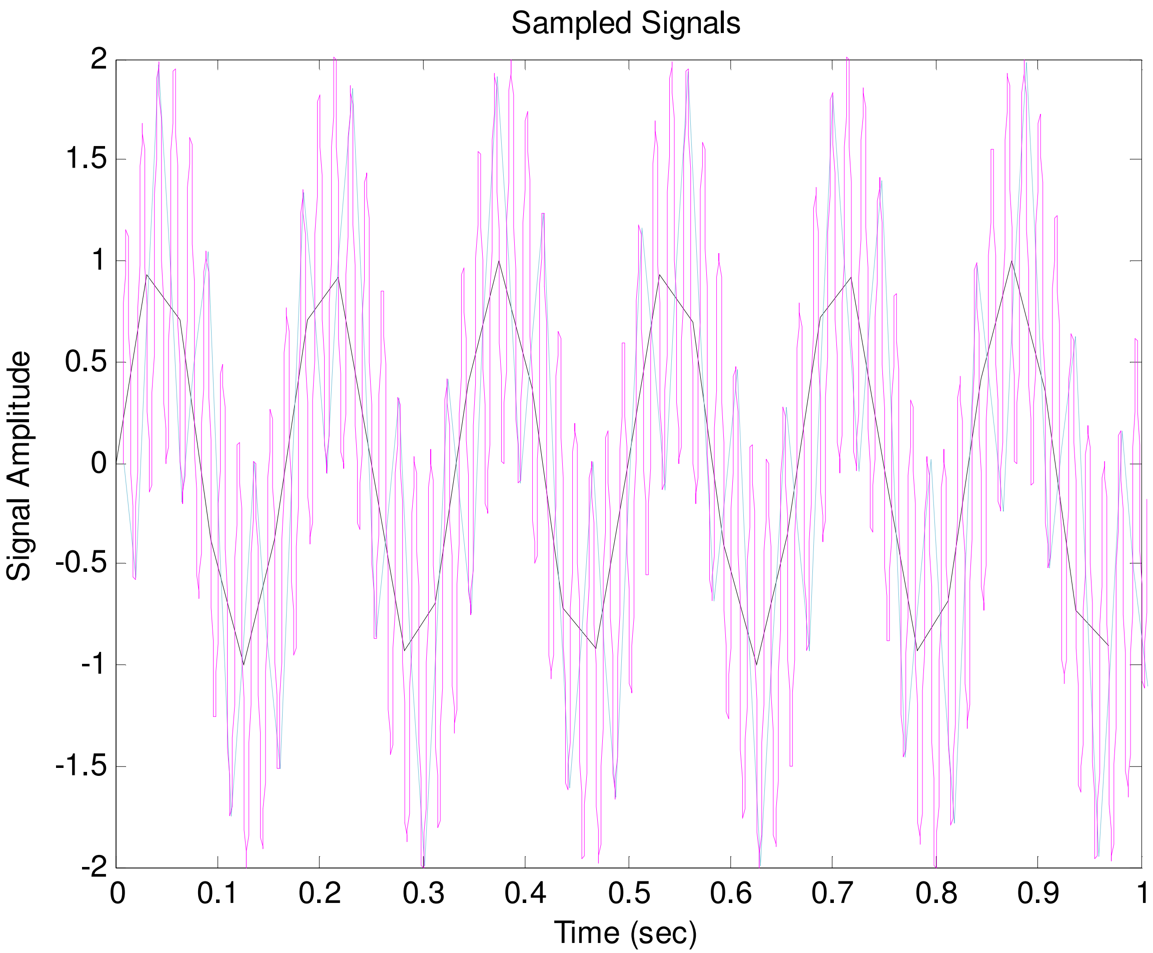


Generic 2nd Order HP Butterworth Transfer Function

$$H(s) = \frac{s^2}{s^2 + ks + w_c^2}$$

Nyquist and Aliasing

Magenta: SR = 256 Hz

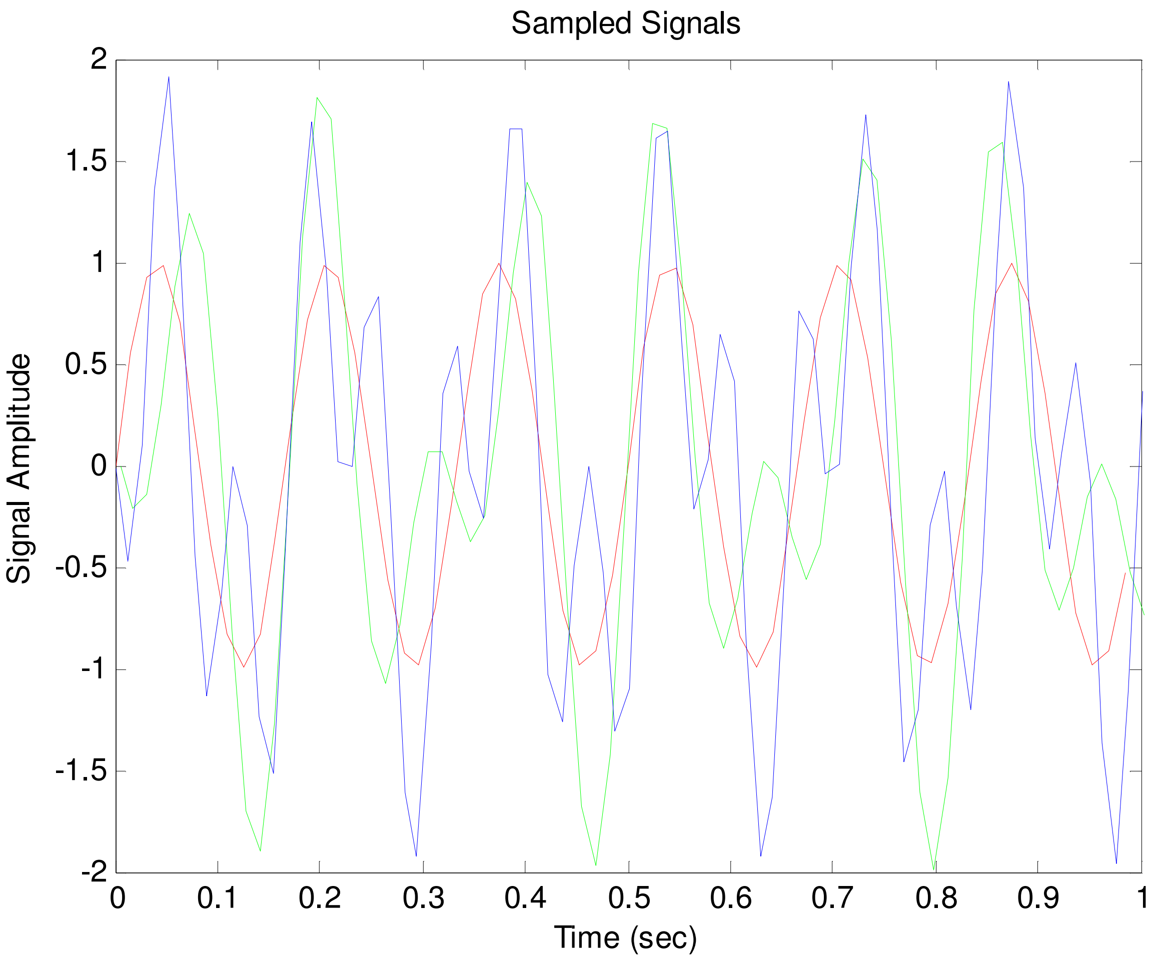


Nyquist and Aliasing

Red: SR = 64 Hz

Green: SR = 73 Hz

Blue: SR = 79 Hz



Nyquist and Aliasing

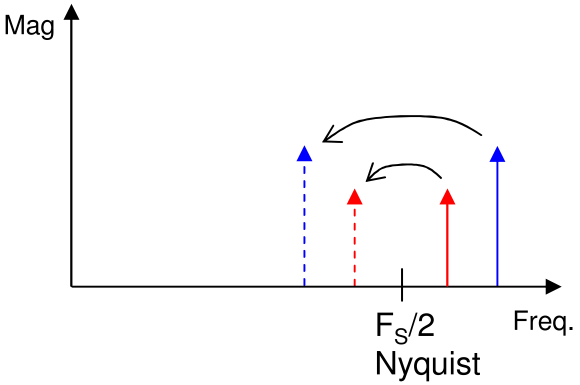
Depending on the Sample Rate, higher frequency signals become lower frequency signals.

Once this "aliasing" occurs, it CANNOT be undone.

Must process the signal BEFORE sampling so that it is NOT aliased by sampling.

What will be aliased?

All frequency content at or above Sample Rate/2. (Fs/2)



How to prevent aliasing?

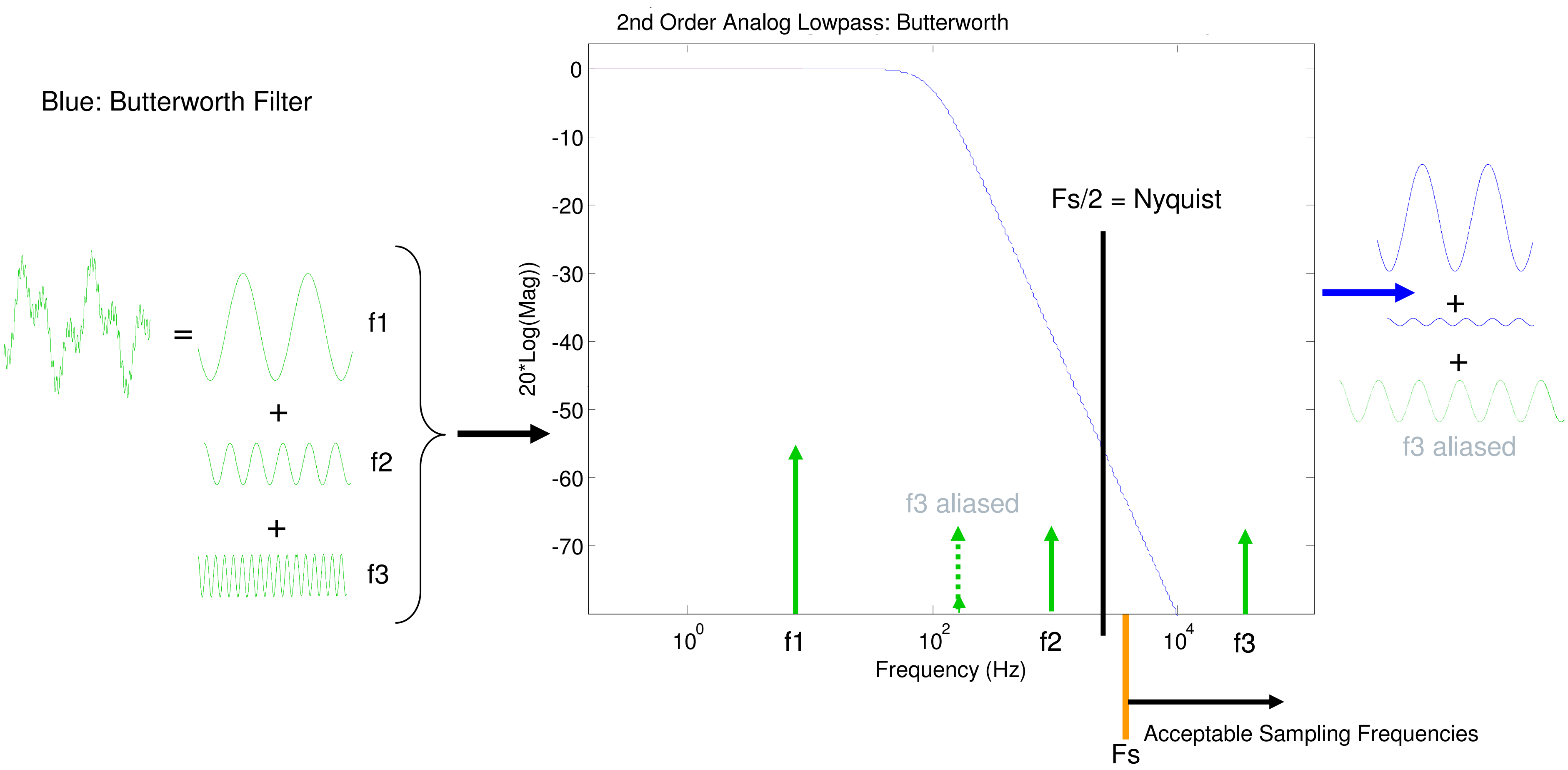
1) Sample very fast, at least 2x higher than any possible signal frequency.

2) Remove or reduce signals with frequencies greater than 0.5 x Fs.

Since we typically have limits on processor speed, sampling speed, etc., option two is almost always selected.

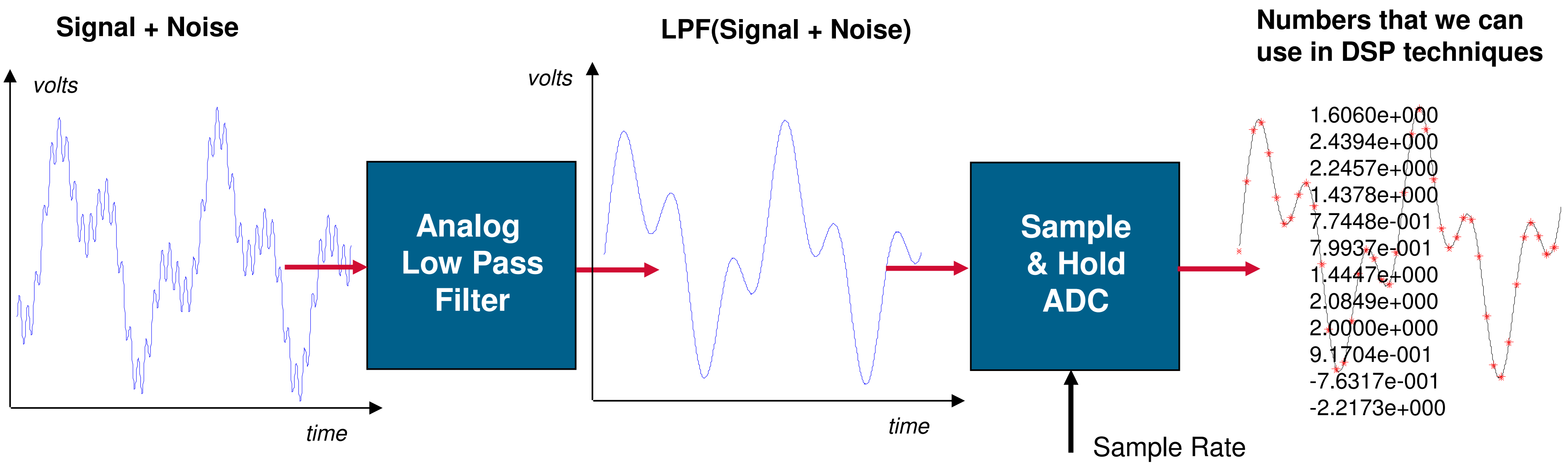
Low Pass Filter for Anti-Aliasing

Blue: Butterworth Filter



Acceptable Sampling Frequencies

Anti-Aliasing Filter and Sampling



Signal + Noise

LPF(Signal + Noise)

Numbers that we can use in DSP techniques

Analog Low Pass Filter

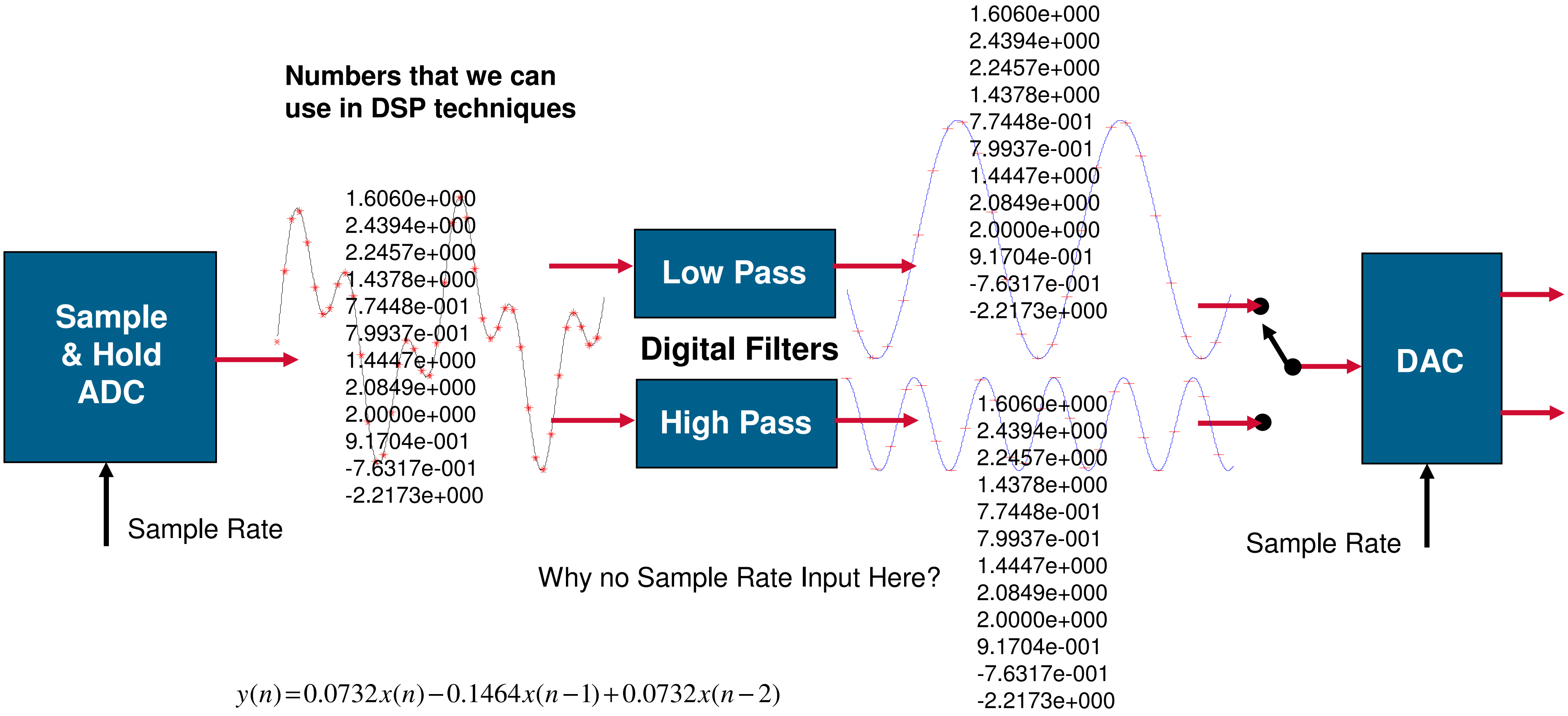
Sample & Hold ADC

Sample Rate

1.6060e+000
2.4394e+000
2.2457e+000
1.4378e+000
7.7448e-001
7.9937e-001
1.4447e+000
2.0849e+000
2.0000e+000
9.1704e-001
-7.6317e-001
-2.2173e+000

The Low Pass filter eliminates, or reduces the amplitude of, signals that could (will) be aliased.

Digital Filtering



Numbers that we can use in DSP techniques

Sample & Hold ADC

Sample Rate

1.6060e+000
2.4394e+000
2.2457e+000
1.4378e+000
7.7448e-001
7.9937e-001
1.4447e+000
2.0849e+000
2.0000e+000
9.1704e-001
-7.6317e-001
-2.2173e+000

Low Pass

Digital Filters

High Pass

Why no Sample Rate Input Here?

1.6060e+000
2.4394e+000
2.2457e+000
1.4378e+000
7.7448e-001
7.9937e-001
1.4447e+000
2.0849e+000
2.0000e+000
9.1704e-001
-7.6317e-001
-2.2173e+000

1.6060e+000
2.4394e+000
2.2457e+000
1.4378e+000
7.7448e-001
7.9937e-001
1.4447e+000
2.0849e+000
2.0000e+000
9.1704e-001
-7.6317e-001
-2.2173e+000

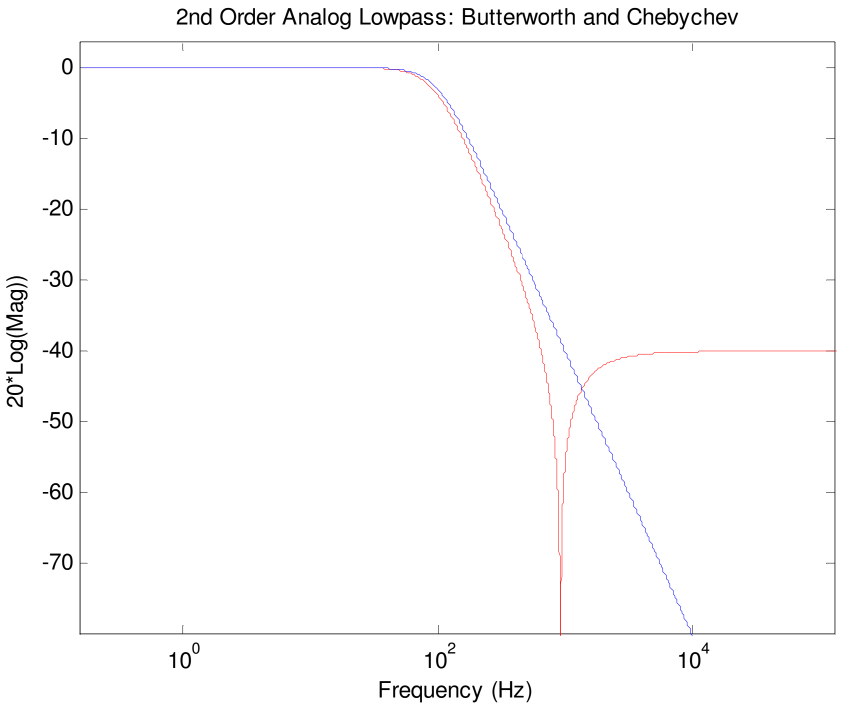
DAC

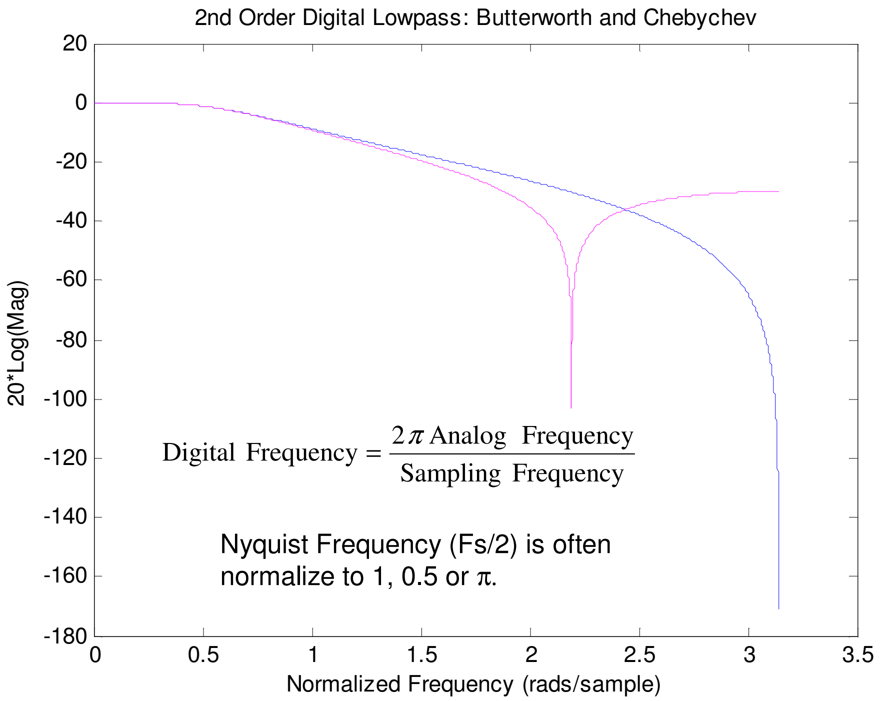
Sample Rate

$$y(n) = 0.0732x(n) - 0.1464x(n-1) + 0.0732x(n-2) \\ + 1.099y(n-1) - 0.3984y(n-2)$$

Just a sequence of numbers

2nd Order Analog vs. Digital Lowpass Filters





$$\text{Digital Frequency} = \frac{2\pi \text{ Analog Frequency}}{\text{Sampling Frequency}}$$

Nyquist Frequency (Fs/2) is often normalize to 1, 0.5 or π.

$$H_B^a(s) = \frac{3.948e5}{s^2 + 8.886e2s + 3.948e5}$$

$$H_B^d(z) = \frac{0.0732 - 0.1464z^{-1} + 0.0732z^{-2}}{1 - 1.103z^{-1} + 0.3953z^{-2}}$$

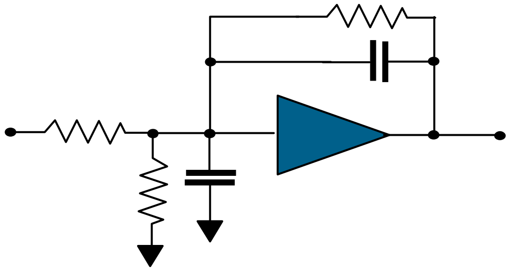
$$H_C^a(s) = \frac{0.01s^2 + 3.336e5}{s^2 + 888.6s + 3.948e5}$$

$$H_C^d(z) = \frac{0.0946 - 0.1101z^{-1} + 0.0946z^{-2}}{1 - 1.099z^{-1} + 0.3984z^{-2}}$$

Analog vs. Digital Filters

Analog Filter

$$H_B^a(s)=\frac{3.948e5}{s^2+8.886e2s+3.948e5}$$



Realized with resistors, capacitors, inductors and amplifiers.

Time/frequency is absolute

Network response is a function of parameter variations, temperature, phase of the moon.

Tuning requires changing of network values.

Digital IIR Filter

$$H_B^d(z)=\frac{0.0732-0.1464z^{-1}+0.0732z^{-2}}{1-1.103z^{-1}+0.3953z^{-2}}$$

$$\begin{aligned}y(n)=&0.0732x(n)-0.1464x(n-1)+0.0732x(n-2)\\&+1.099y(n-1)-0.3984y(n-2)\end{aligned}$$

$$y(n)=\sum_{i=0}^{N}a(i)x(n-i)+\sum_{j=1}^{M}b(j)\,y(n-j),\quad M\geq N$$

Realized with digital multiplication, adds, shifts and data moves.

Time/frequency is relative to sampling rate.

Network response is a function of coefficient quantization and timing variations.

Tuning requires changing register values.

Digital FIR vs. IIR Filters

Digital FIR Filter

Finite Impulse Response

$$y(n) = 0.085x(n)+0.083x(n-1)+0.079x(n-2)+0.073x(n-3)+$$
$$0.069x(n-4)+0.053x(n-5)+0.044x(n-6)+ \ . \ . \ .$$

$$y(n)=\sum_{i=0}^{N-1} a(i)x(n-i)$$

Can implement non-realizable analog functions

Many more Multiplies, Adds and Data Moves

Digital IIR Filter

Infinite Impulse Response

$$y(n)=0.0732x(n)-0.1464x(n-1)+0.0732x(n-2)$$
$$+1.099\,y(n-1)-0.3984\,y(n-2)$$

$$y(n) = \sum_{i=0}^{N} a(i)x(n-i)+\sum_{j=1}^{M} b(j)\,y(n-j), \quad M \geq N$$

Numerator Terms Denominator Terms

Digital imitation of analog filters

Generally the fewest operations – often 10x more efficient

Realized with digital multiplication, adds, shifts and data moves.

Time/frequency is relative to sampling rate.

Network response is a function of coefficient quantization and timing variations.

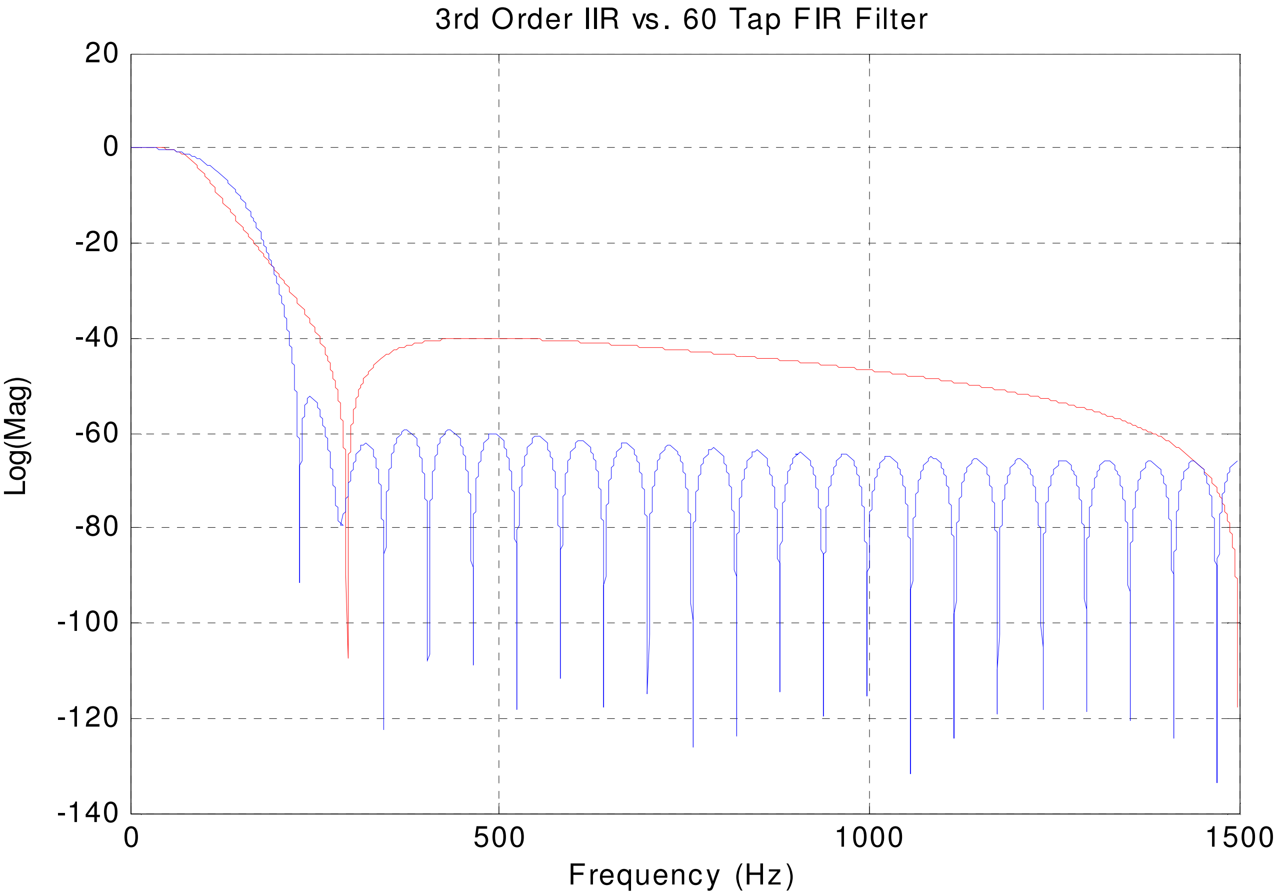
Tuning requires changing register values.

Digital FIR vs. IIR Filters

½ FIR Coefficients,
a(0) to a(N/2 -1)

3.8828e-004
1.3847e-004
-1.8346e-004
-6.4280e-004
-1.2924e-003
-2.1528e-003
-3.1951e-003
-4.3278e-003
-5.3930e-003
-6.1707e-003
-6.3948e-003
-5.7769e-003
-4.0380e-003
-9.4382e-004
3.6603e-003
9.8193e-003
1.7447e-002
2.6319e-002
3.6073e-002
4.6234e-002
5.6245e-002
6.5510e-002
7.3447e-002
7.9534e-002
8.3363e-002
8.4669e-002

FIR = Blue



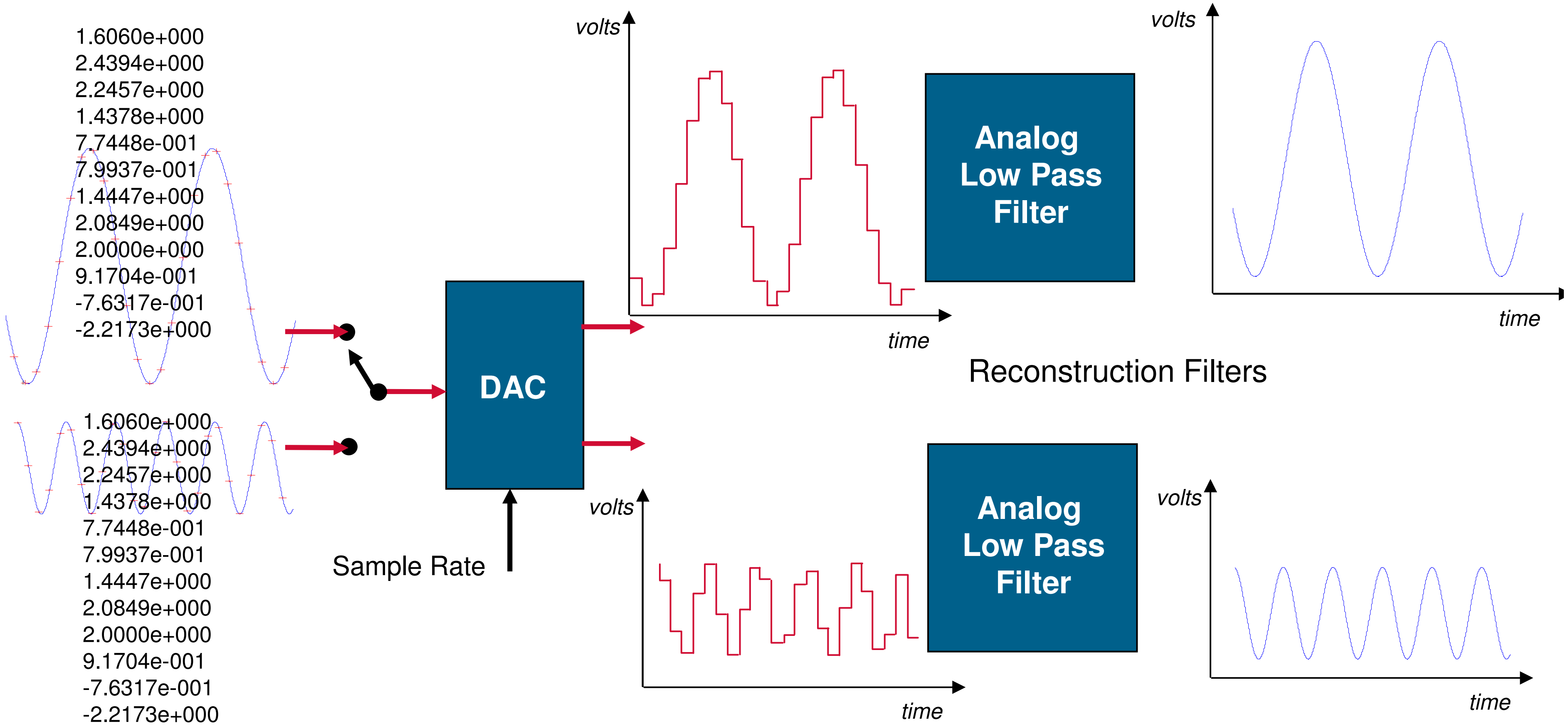
IIR Coefficients

7.5535e-003
-4.7813e-003
-4.7813e-003
7.5535e-003

-2.6301e+000
2.3258e+000
-6.9009e-001

IIR = Red

Analog Reconstruction



1.6060e+000
2.4394e+000
2.2457e+000
1.4378e+000
7.7448e-001
7.9937e-001
1.4447e+000
2.0849e+000
2.0000e+000
9.1704e-001
-7.6317e-001
-2.2173e+000

1.6060e+000
2.4394e+000
2.2457e+000
1.4378e+000
7.7448e-001
7.9937e-001
1.4447e+000
2.0849e+000
2.0000e+000
9.1704e-001
-7.6317e-001
-2.2173e+000

DAC

Sample Rate

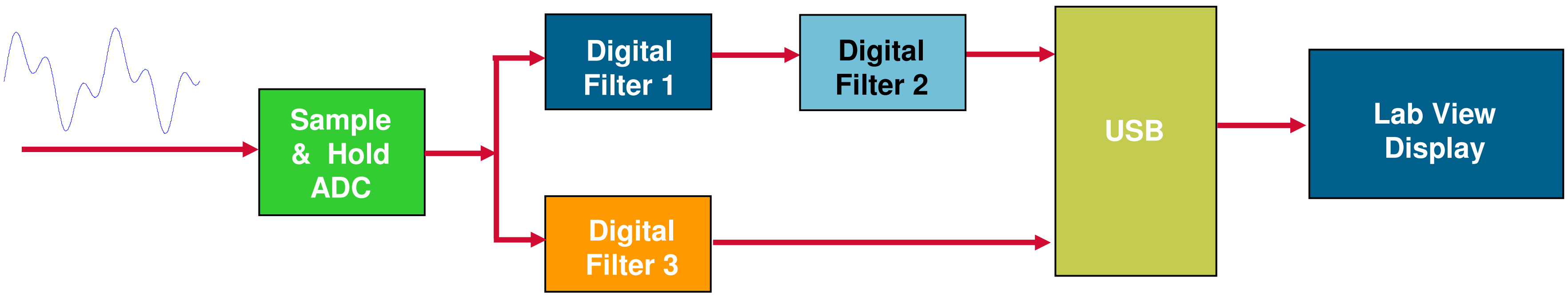
Analog Low Pass Filter

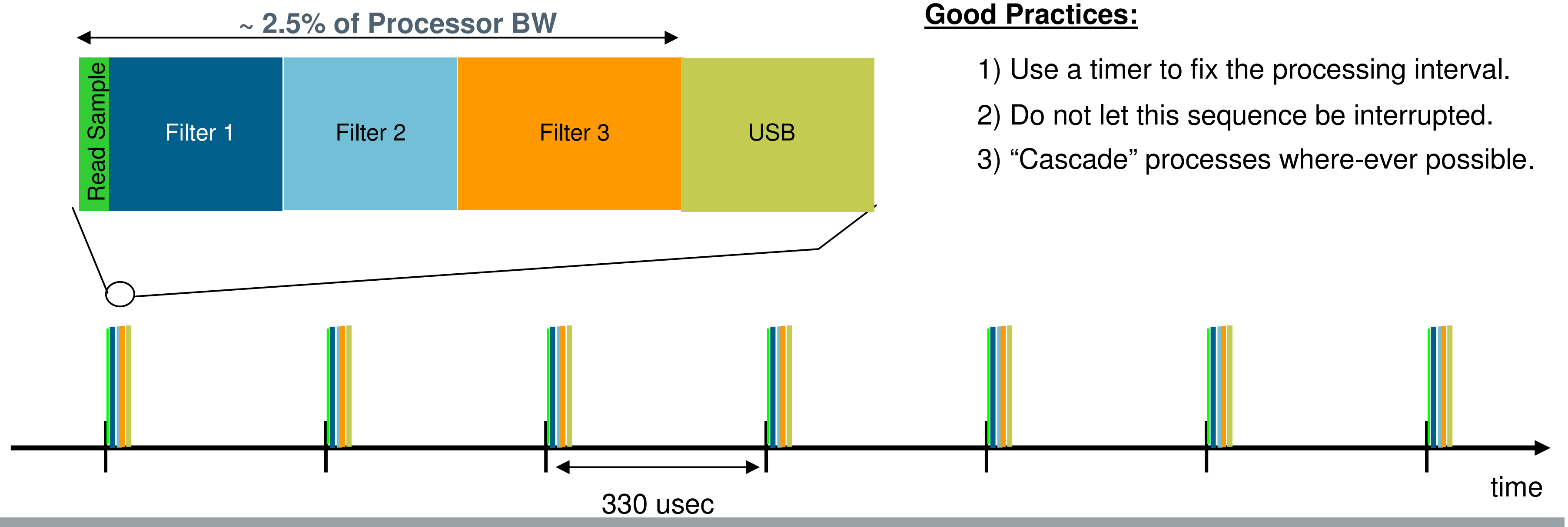
Analog Low Pass Filter

Reconstruction Filters

Just a sequence of numbers

Realization





Good Practices:

1) Use a timer to fix the processing interval.
2) Do not let this sequence be interrupted.
3) “Cascade” processes where-ever possible.

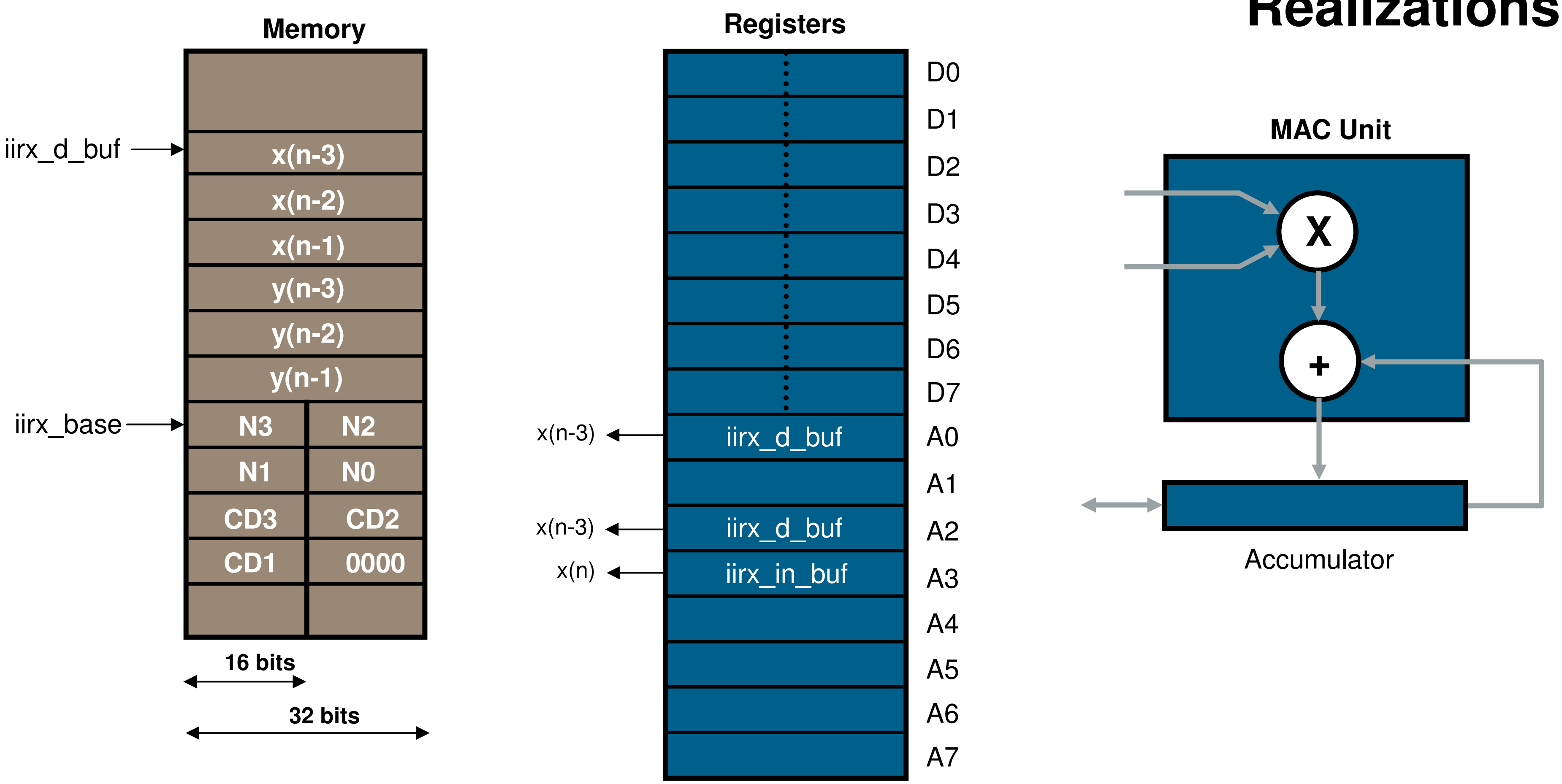
Realizations

Numerator Terms

$$y(n) = \underbrace{\sum_{i=0}^{N} a(i)x(n-i)} + \sum_{j=1}^{M} b(j)\, y(n-j)$$

```
lea.l      iirx_in_buf,A3        ; load pointer to data
lea.l      iirx_d_buf,A0         ; load pointer to  intermediate data buffer
move.l     A0,A2                 ; copy this for use by movem
lea.l      iirx_base,A1          ; load start address for numerator coefs.
clr.l      D0
move.l     D0,ACC                ; clear the acc
move.l     (A1)+,D5              ; load N3,N2 into D5, A1 points to N1,N0
move.l     (A0)+,D1              ; load x(n-3) into D1 (part of a trick)
mac.w      D1.l,D5.u,(A0)+,D0    ; x(n-3)*N3 -> acc, x(n-2) into D0 (more trick)
mac.w      D0.l,D5.l,(A1)+,D6    ; Acc+x(n-2)*N2 -> acc, N1,0 into D6
move.l     (A0)+,D1              ; move x(n-1) into D1
mac.w      D1.l,D6.u             ; Acc+x(n-1)*N1 -> acc.
move.l     D7,D2                 ; x(n) into D2
mac.w      D2.l,D6.l             ; acc+x(n)*N0 -> acc.
movem.l    D0-D2,(A2)            ; and move the data back, shifted down.
move.l     ACC,D7                ; move the acc into D7
clr.l      D6                    ; clear this reg.
move.b     Nshift,D6             ; load Nshift value
asr.l      D6,D7                 ; shift by difference between num and den
clr.l      D6                    ; clear D6 again
addx.l     D6,D7                 ; round
move.l     D7,ACC                ; move back to acc.
```

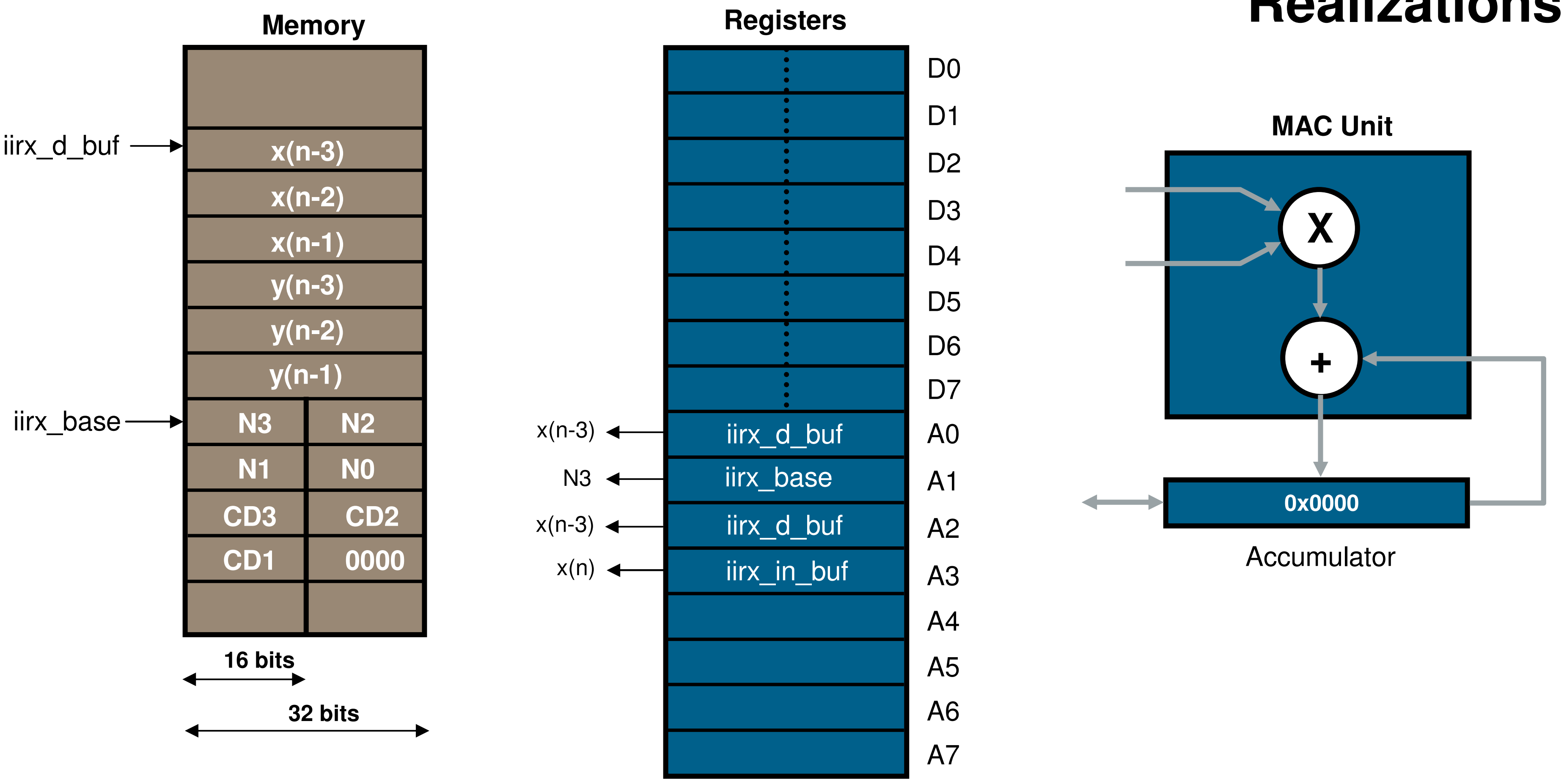
Realizations



Initialize:

```
lea.l   iirx_in_buf,A3          ; load pointer to data
lea.l   iirx_d_buf,A0           ; load pointer to  intermediate data buffer
move.l  A0,A2                   ; copy this for use by movem
```

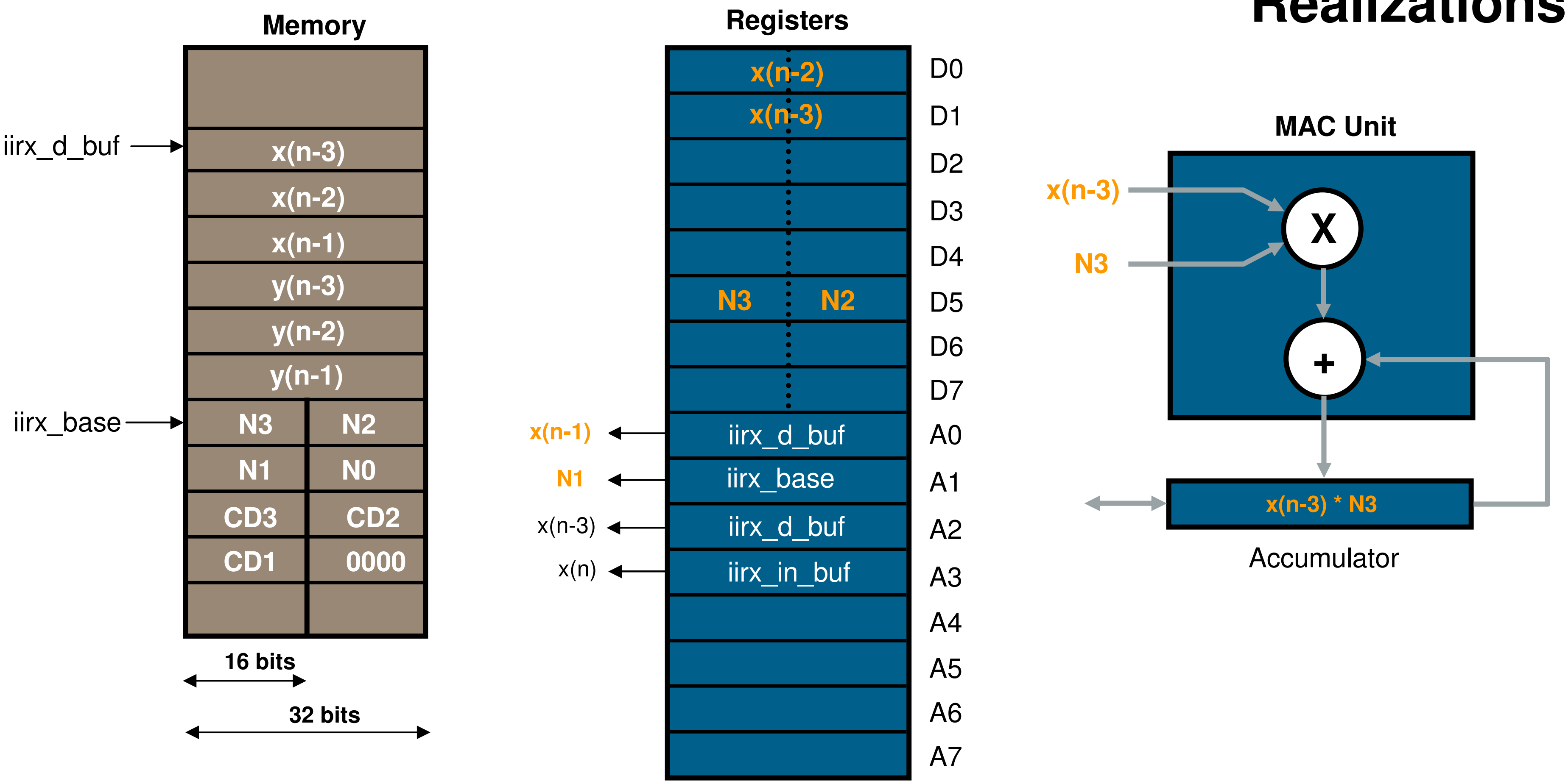
Realizations



More initialization:

```
move.l      A0,A2                 ; copy this for use by movem
lea.l       iirx_base,A1          ; load start address for numerator coefs.
clr.l       D0
move.l      D0,ACC                ; clear the acc
```

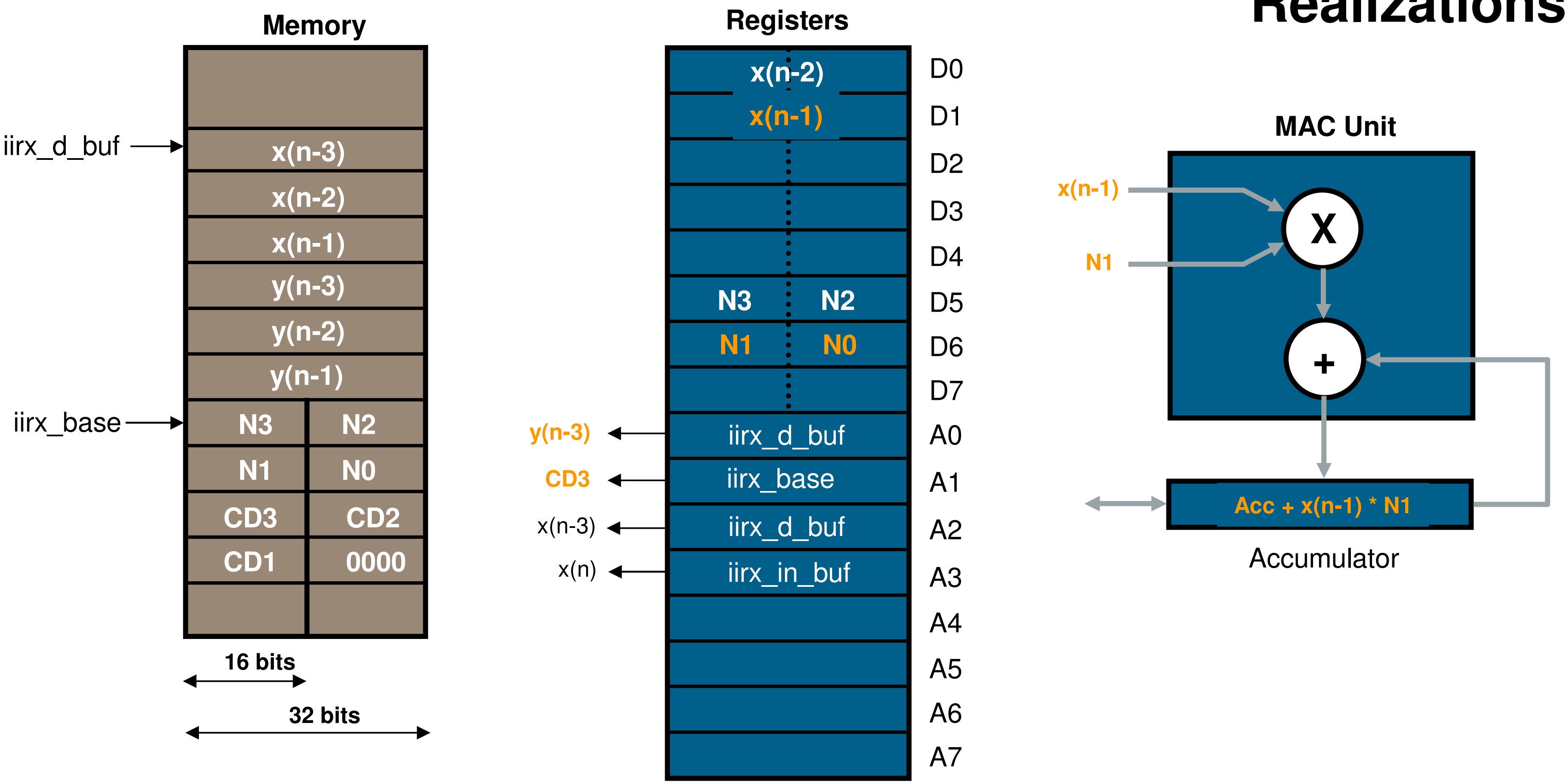
Realizations



Load Data and Start:

```
move.l    D0,ACC                 ; clear the acc
move.l    (A1)+,D5               ; load N3,N2 into D5, A1 points to N1,N0
move.l    (A0)+,D1               ; load x(n-3) into D1 (part of a trick)
mac.w     D1.l,D5.u,(A0)+,D0     ; x(n-3)*N3 -> acc, x(n-2) into D0 (more trick)
```

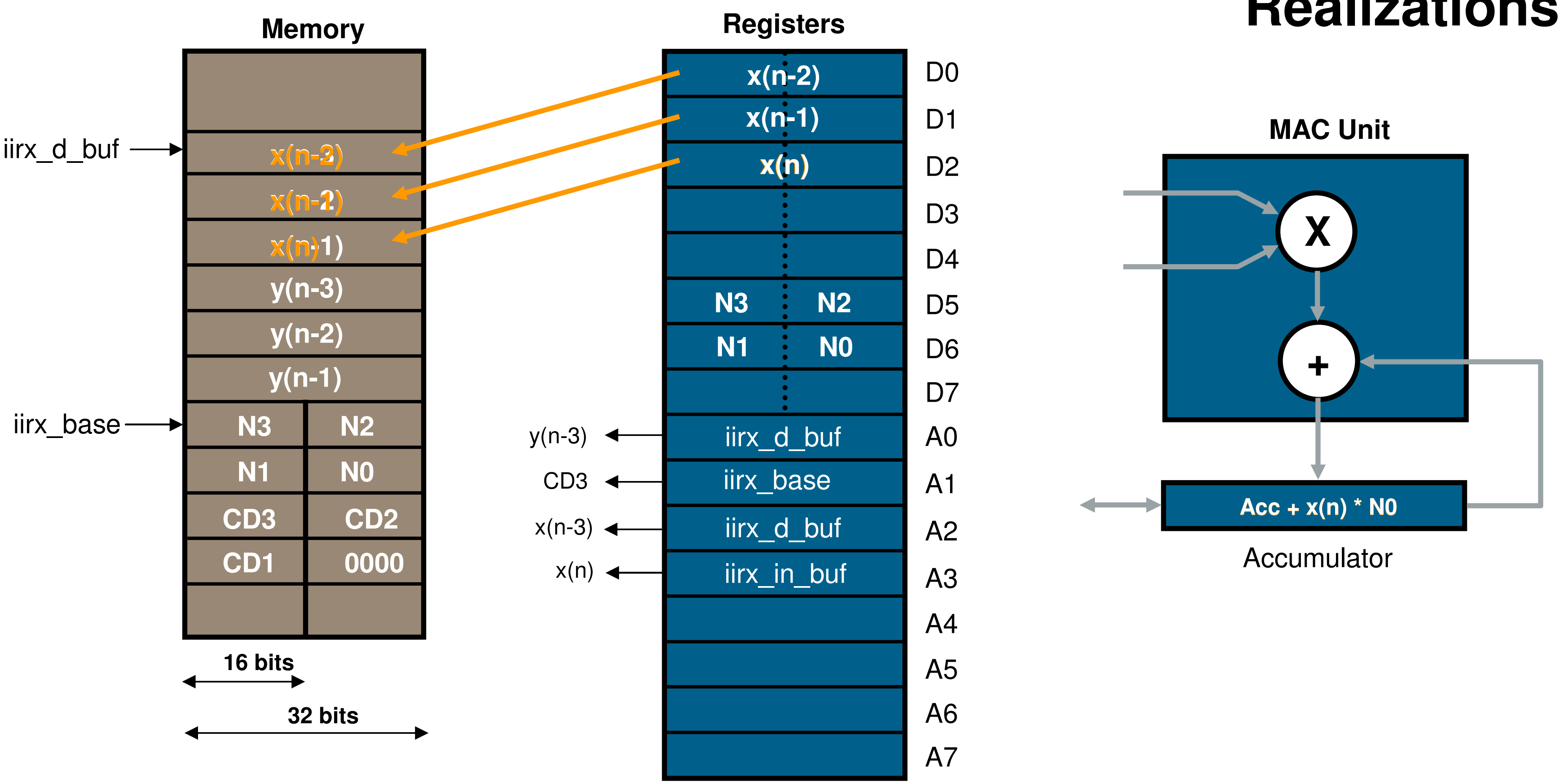
Realizations



More Inner Product:

```
mac.w       D1.l,D5.u,(A0)+,D0      ; x(n-3)*N3 -> acc, x(n-2) into D0 (more trick)
mac.w       D0.l,D5.l,(A1)+,D6      ; Acc+x(n-2)*N2 -> acc, N1,0 into D6
move.l      (A0)+,D1                ; move x(n-1) into D1
mac.w       D1.l,D6.u               ; Acc+x(n-1)*N1 -> acc.
```

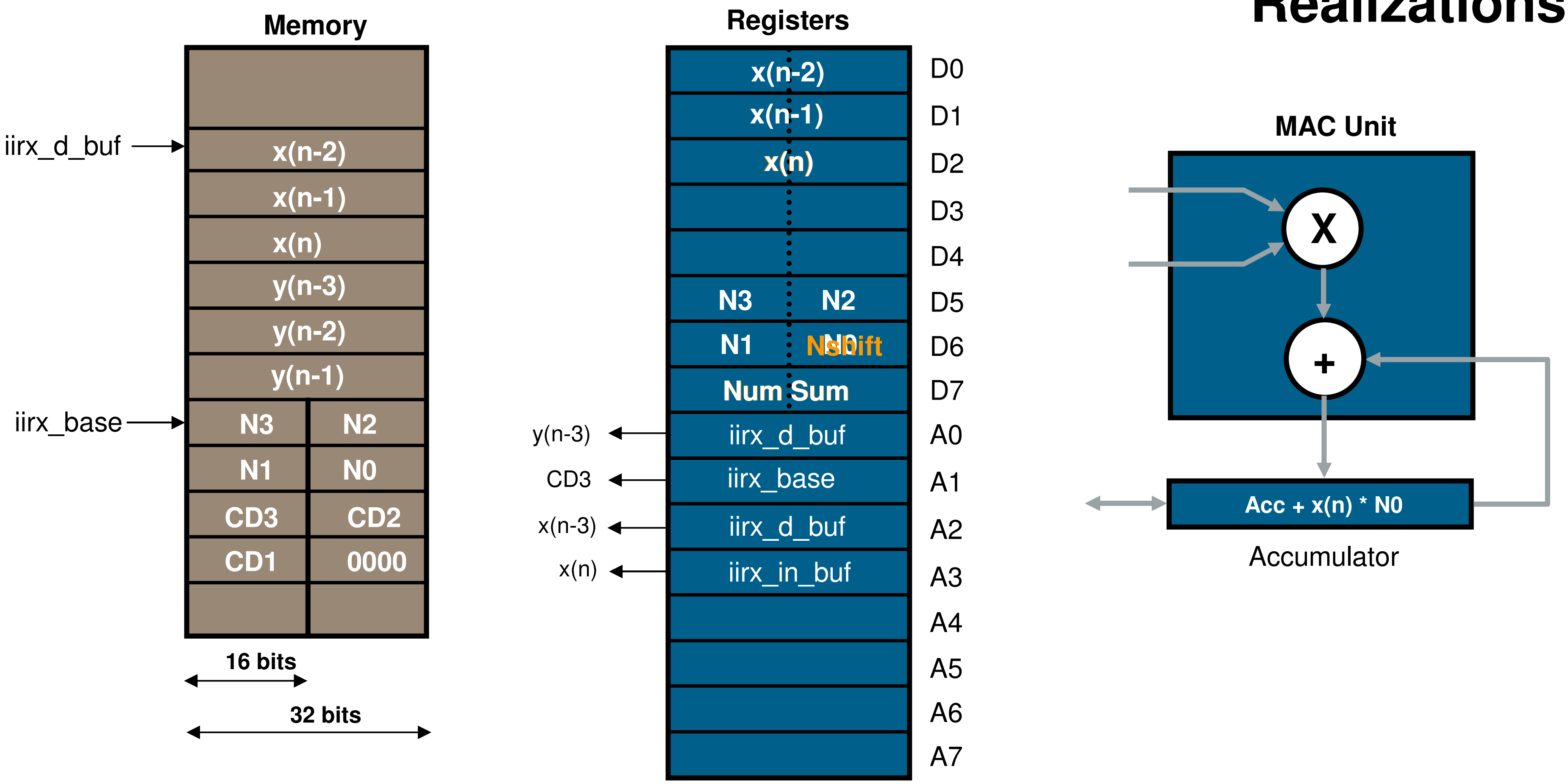
Realizations



Finish Num. and Rotate Buffer:

```
mac.w      D1.l,D6.u           ; Acc+x(n-1)*N1 -> acc.
move.l     (A3),D2             ; move x(n) into D2
mac.w      D2.l,D6.l           ; acc+x(n)*N0 -> acc.
movem.l    D0-D2,(A2)          ; and move the data back, shifted down
```

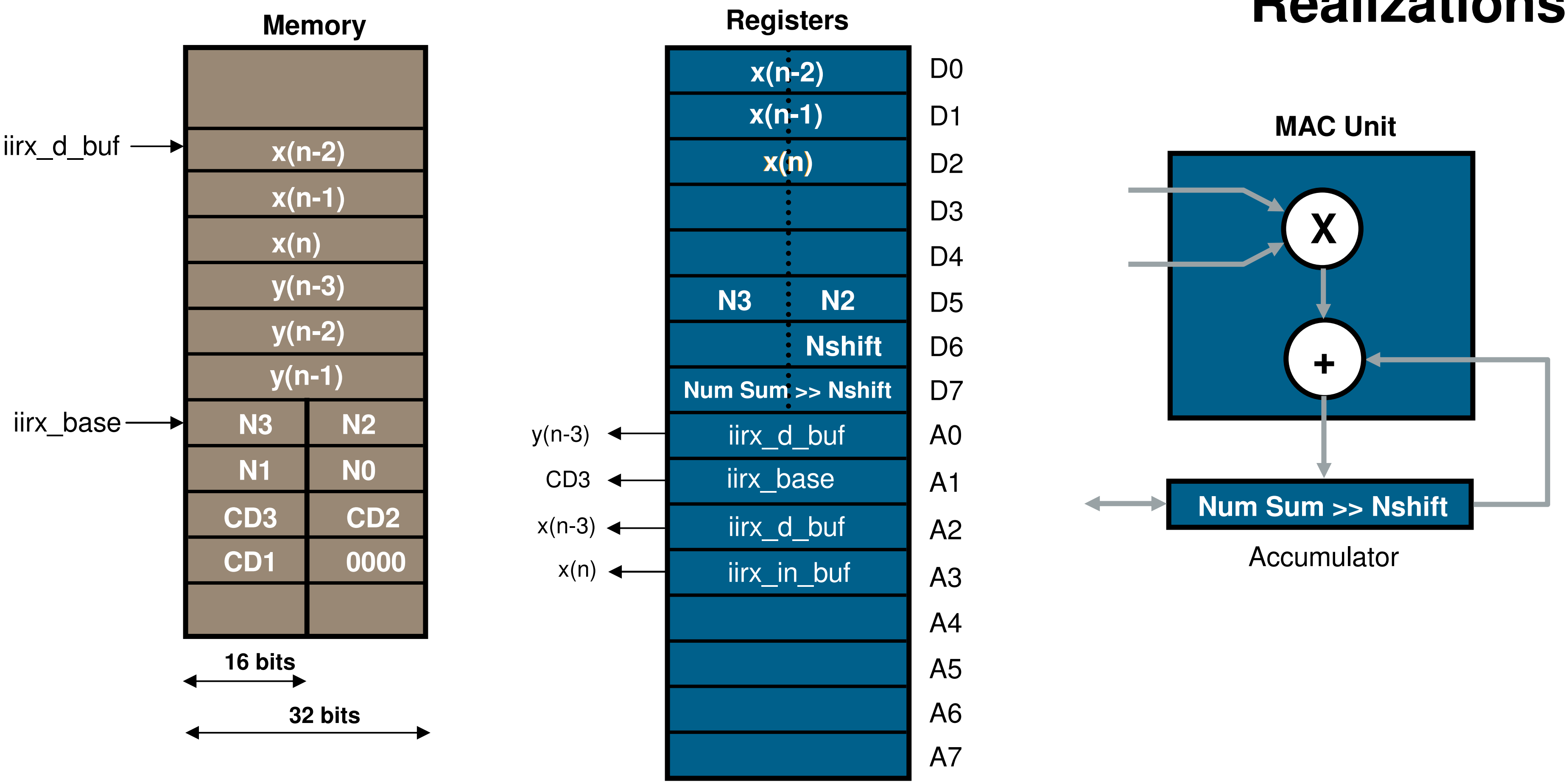
Realizations



Scale Num.:

```
movem.l    D0-D2,(A2)          ; and move the data back, shifted down.
move.l     ACC,D7              ; move the acc into D7
clr.l      D6                  ; clear this reg.
move.b     Nshift,D6           ; load Nshift value
```

Realizations



Scale Num. and round:

```
asr.l      D6,D7     ; shift by difference between num and den
clr.l      D6        ; clear D6 again
addx.l     D6,D7     ; round
move.l     D7,ACC    ; move back to acc.
```

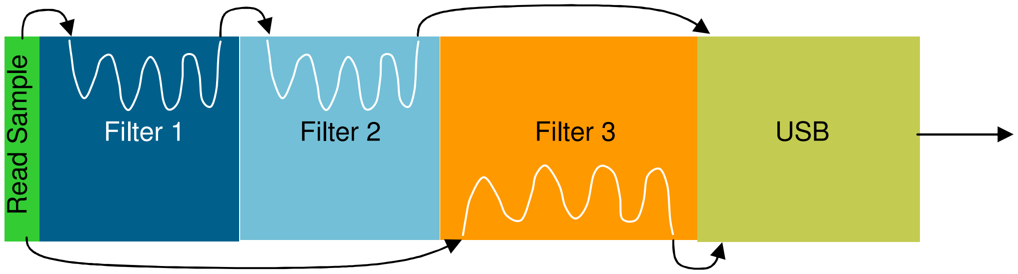
Realizations

$$y(n) = \sum_{i=0}^{N} a(i)x(n-i) + \underbrace{\sum_{j=1}^{M} b(j)\, y(n-j)}_{}$$

Denominator or Feedback Terms

```
  move.l    A0,A2               ; save this location to be used by movem
  move.l    (A1)+,D5            ; load CD3,CD2 into D5, A1 pts to CD1
  move.l    (A0)+,D1            ; load y(n-3) into D1
  mac.w     D1.l,D5.u,(A0)+,D0  ; y(n-3)*CD3 -> acc, y(n-2) into D0 (more trick)
  mac.w     D0.l,D5.l,(A1)+,D6  ; Acc+y(n-2)*CD2 -> acc. CD1 into D6
  move.l    (A0),D1             ; y(n-1) into D1
* mac.w     D1.l,D6.u           ; acc+y(n-1)*CD1 -> acc.  Done
* move.l    ACC,D7              ; move the acc into D7
  clr.l     D6                  ; clear D6
  move.b    Dshift,D6           ; load shift value
* asr.l     D6,D7               ; shift denom.
  clr.l     D6                  ; clear this again
* addx.l    D6,D7               ; round
* move.w    D7,iirx_out_buf     ; write data out
  move.l    D7,D2               ; get the output into the feedback
  movem.l   D0-D2,(A2)          ; and then move the data back, shifted down.
```

Realizations



Ok for most signal processing applications

However, for servo (feedback control applications), latency = phase loss → potential instability, yield loss

A slightly different structure: - Want to minimize the time from sampling to output.

"Normal" Calculation Sequence

$$y(n)=0.0732x(n)-0.1464x(n-1)+0.0732x(n-2)+1.099y(n-1)-0.3984y(n-2)$$

Write y(n)
to DAC

Lower Latency
Calculation Sequence:

This will cost about 10% more cycles,
but can be worth the effort.

Load 0.0732
Clr Dx,
Load Dshift
Load y-(n)

Busy Wait Loop
to Read ADC

$$y(n)=0.0732x(n)+y^{-}(n)$$

Write y(n)
to DAC

$$y^{-}(n) = -0.1464x(n)+0.0732x(n-1)$$
$$+1.099y(n)-0.3984y(n-1)$$

Update
Buffers

DSP Summary

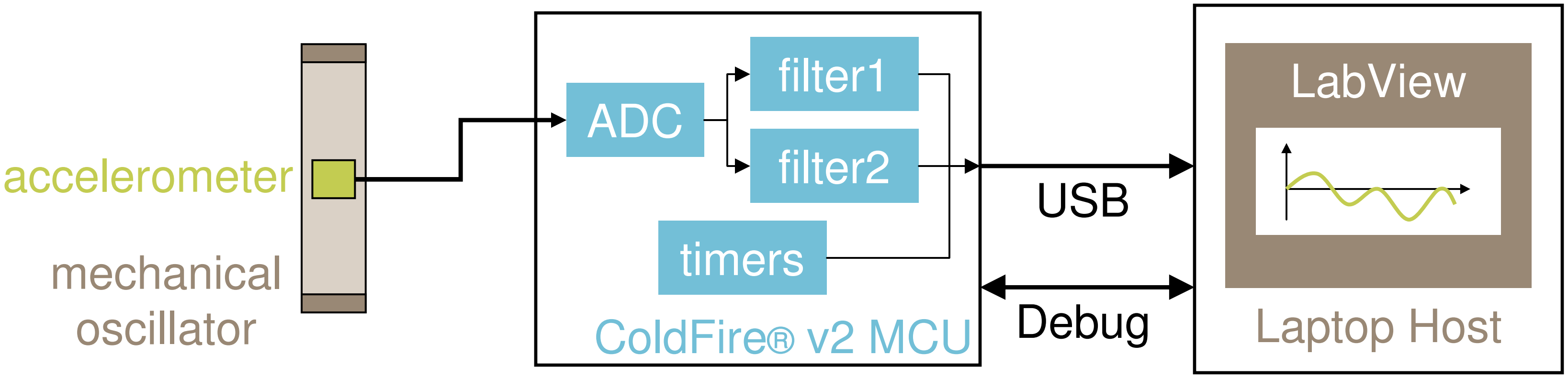
- Summary of DSP Review
  - Motivation for using ColdFire® in low-moderate intensity DSP apps.
  - Review of Aliasing and Sampling → Sample Rate is King
  - Overview of Analog and Digital Filtering
  - Some practical realization considerations
- Next: How to use the tools and libraries

Agenda

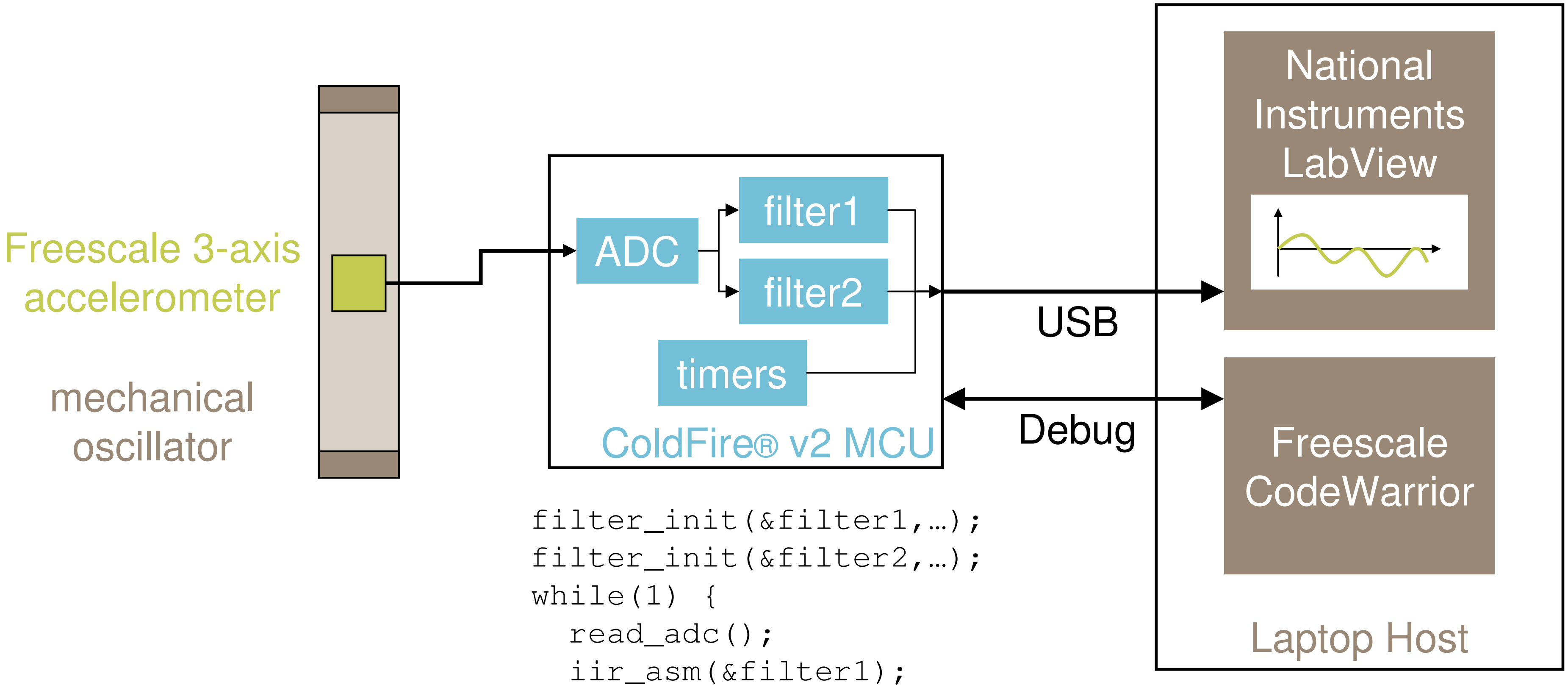
- ►Understand key concepts in digital signal processing (DSP)
  - Filter effects, sample rate, computational complexity
- ►Learn DSP-enabling features of ColdFire® architecture
  - Multiply-accumulate (MAC) unit
- ►Experience real-world sensor signals
  - Freescale 3-axis accelerometer, signal generator
- ►Implement filters from optimized library
  - C-callable assembly filters, predefined coefficients
- ►Realize performance gains
  - Compare assembly and compiled C filter performance
- ►Incorporate DSP functions into your next ColdFire® application

Real-Time Demo with LabView

- ▶ Running on ColdFire® Demo Board (M52221DEMO)
  - Sample analog accelerometer data with ADC (3 kHz)
  - Execute two parallel digital filters
  - Send via USB: raw and filtered data, timestamp, filter execution cycles (downsampled 3:1)
- ▶ Running on LabView
  - Receive and parse USB data (1 kHz)
  - Plot multiple waveforms, zoomable axes
  - Display ColdFire® processor usage for filter execution



Block Diagram Detail



```
filter_init(&filter1,…);
filter_init(&filter2,…);
while(1) {
  read_adc();
  iir_asm(&filter1);
  iir_asm(&filter2);
  output_usb();
  pause(0.0003);}
```

Signal Generation

Signal Processing

Board Programming
Signal Display

DSP-Enabling Architecture

- ColdFire® MAC architecture enables DSP algorithms
- IIR and FIR filters gain performance with MAC instructions
- Single instruction: multiply-accumulate with load
  - Multiply two 16-bit word or 32-bit longword operands
  - Add 32-bit product to 32-bit accumulator (ACC) register
  - Load 32-bit longword for next instruction and increment address register (ptr)
- Enables efficient and concise filter code

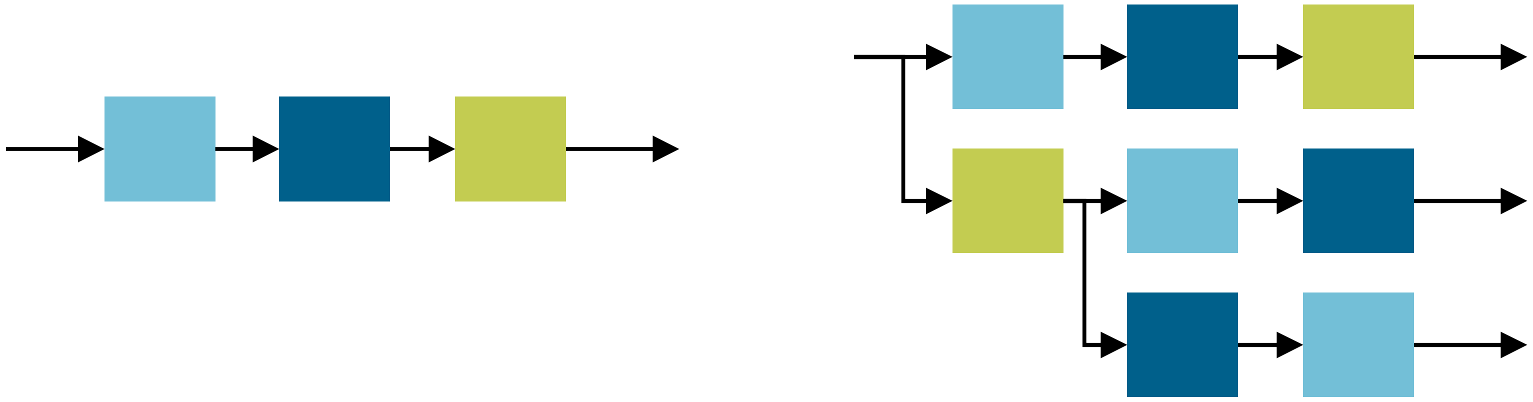
$$y(n)=\sum_{i=0}^{N}\overbrace{a(i)x(n-i)}^{\text{multiply}}$$

accumulate

FIR

ColdFire® DSP Library

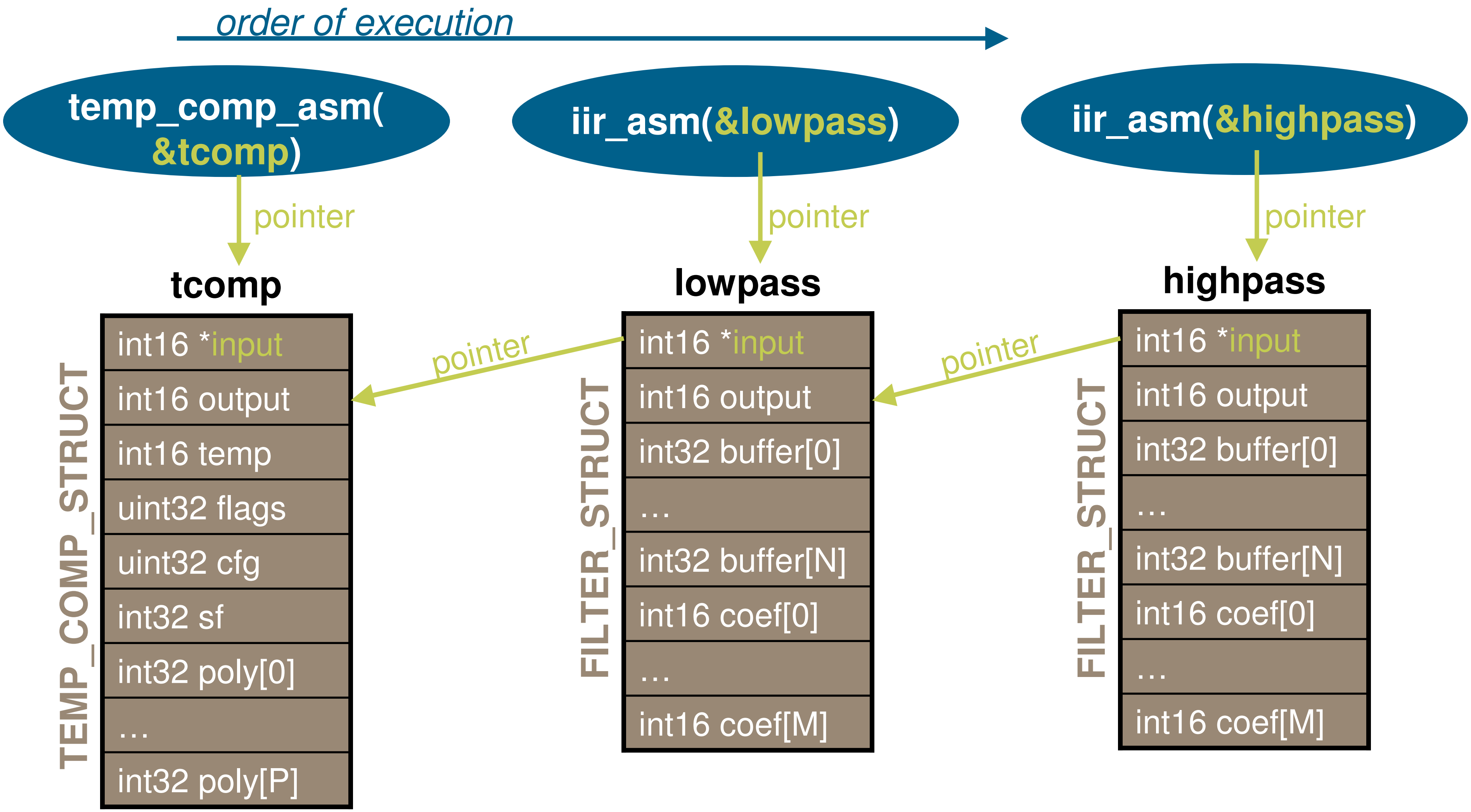
- Key DSP algorithms implemented in ColdFire® ISA_A+ assembly
- Optimized for computational performance
  - Extensive usage of multiply-accumulate (MAC) unit
- C-callable functions create simple user interface
- Configurable filter coefficients enable many different applications
  - Same code, different coefficients = different filter!
- Easy to daisy-chain blocks together
  - Customize datapath for application
  - Combine series and parallel configurations



Assembly Function Classes

- **2nd-6th Order IIR Filter,** FIR Filter
  - **Configurable filter coefficients define filter transfer function**
- Temperature Compensation
  - Evaluate polynomial in temperature, voltage, etc. to calculate scale factor, and multiply by input acceleration data
- Nonlinear Data Sanity Block
  - Determine if input data exceeds upper or lower bounds determined by windowed mean and variance
- Sample Rate Converter
  - Decimation-interpolation algorithm to reduce sample rate
- Output Data Formatter
  - Convert native two-complement into one-complement, offset binary, or single-precision floating point
- User-Defined

Daisy-Chained Data Structures



order of execution

temp_comp_asm(&tcomp)

iir_asm(&lowpass)

iir_asm(&highpass)

pointer

pointer

pointer

tcomp

TEMP_COMP_STRUCT

- int16 *input
- int16 output
- int16 temp
- uint32 flags
- uint32 cfg
- int32 sf
- int32 poly[0]
- …
- int32 poly[P]

pointer

lowpass

FILTER_STRUCT

- int16 *input
- int16 output
- int32 buffer[0]
- …
- int32 buffer[N]
- int16 coef[0]
- …
- int16 coef[M]

pointer

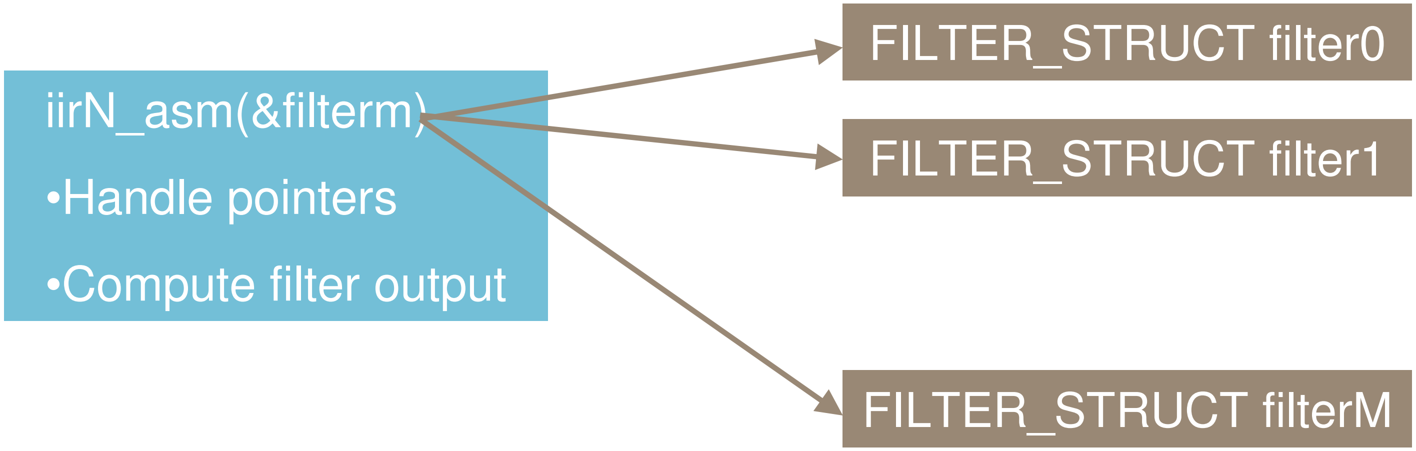
highpass

FILTER_STRUCT

- int16 *input
- int16 output
- int32 buffer[0]
- …
- int32 buffer[N]
- int16 coef[0]
- …
- int16 coef[M]

C-Callable Assembly Functions

- Pointer to data structure is sole argument
- Assembly function parses data structure elements
- Single copy of function code in memory
- Multiple instances of data structure, one for each function call



Data Structures and Initialization

- Each assembly function has an associated typedef data structure and initialization function
- Data structures define input location (pointer), contain filter coefficients, and maintain buffers required for the algorithm
- Every function call requires a separate instance of its associated data structure

Assembly Function	*Typedef Structure*	*Initialization Function*
temp_comp_asm	TEMP_COMP_STRUCT	temp_comp_init
variance_asm	VARIANCE_STRUCT	variance_init
iir_asm	FILTER_STRUCT	filter_init
src1_asm	SRC_STRUCT	src_init
data_format_asm	FORMATTER_STRUCT	formatter_init

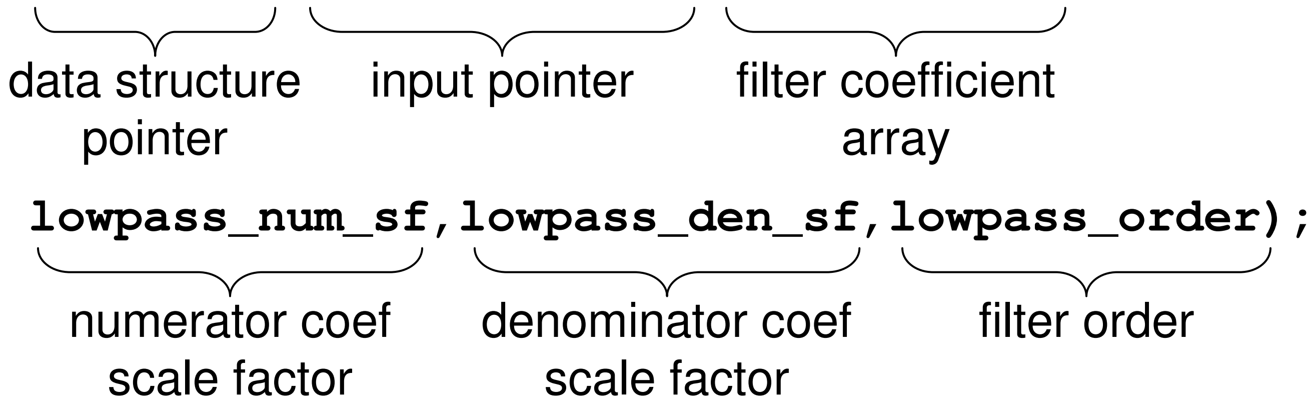
Code Example: Single Filter

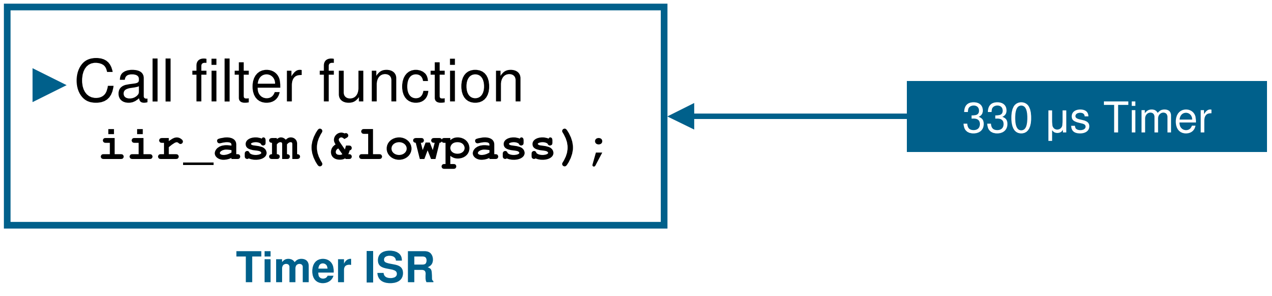
- Declare data structure

```
FILTER_STRUCT lowpass;
```

- Initialize data structure

```
filter_init(&lowpass,&sensor.output,lowpass_coef,
            lowpass_num_sf,lowpass_den_sf,lowpass_order);
```





- Call filter function

```
iir_asm(&lowpass);
```

Code Example: Daisy-Chaining

- ▶ Declare multiple data structures

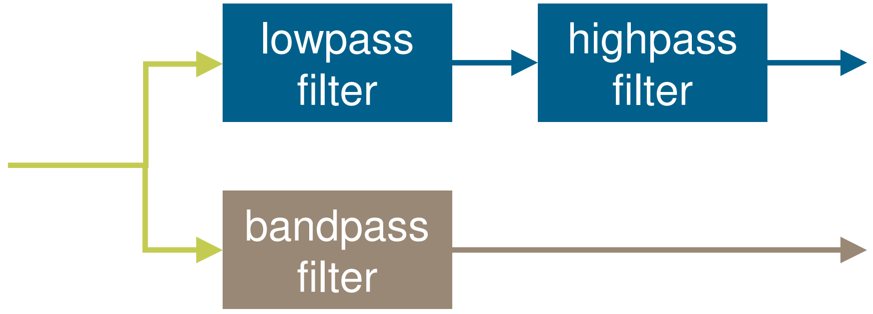
```
FILTER_STRUCT lowpass,highpass,bandpass;
```

- ▶ Initialize multiple data structures

```
/* upper path */
filter_init(&lowpass,&sensor.output,lowpass_coef,...);
filter_init(&highpass,&lowpass.output,highpass_coef,...);

/* lower path */
filter_init(&bandpass,&sensor.output,bandpass_coef,...);
```

- ▶ Call filter function

```
/* upper path - note order */
iir_asm(&lowpass);
iir_asm(&highpass);

/* lower path */
iir_asm(&bandpass);
```



lowpass filter

highpass filter

bandpass filter

Timer ISR

330 µs Timer

Data Type: int16

- 16-bit signed twos-complement data type
- Twos-complement format
  - Other formats such as unsigned, floating point, or offset binary will not work
- 16-bit length
  - Longer input data (such as int32) will not work
  - Shorter input data (such as int8) will work but must be sign-extended to 16-bits (cast to int16)
- Range = [0x8000:0x7FFF] = [-32,768:32,767]
- Fixed-point scaling preserved through filters (linear system)
  - Maintained by data structures

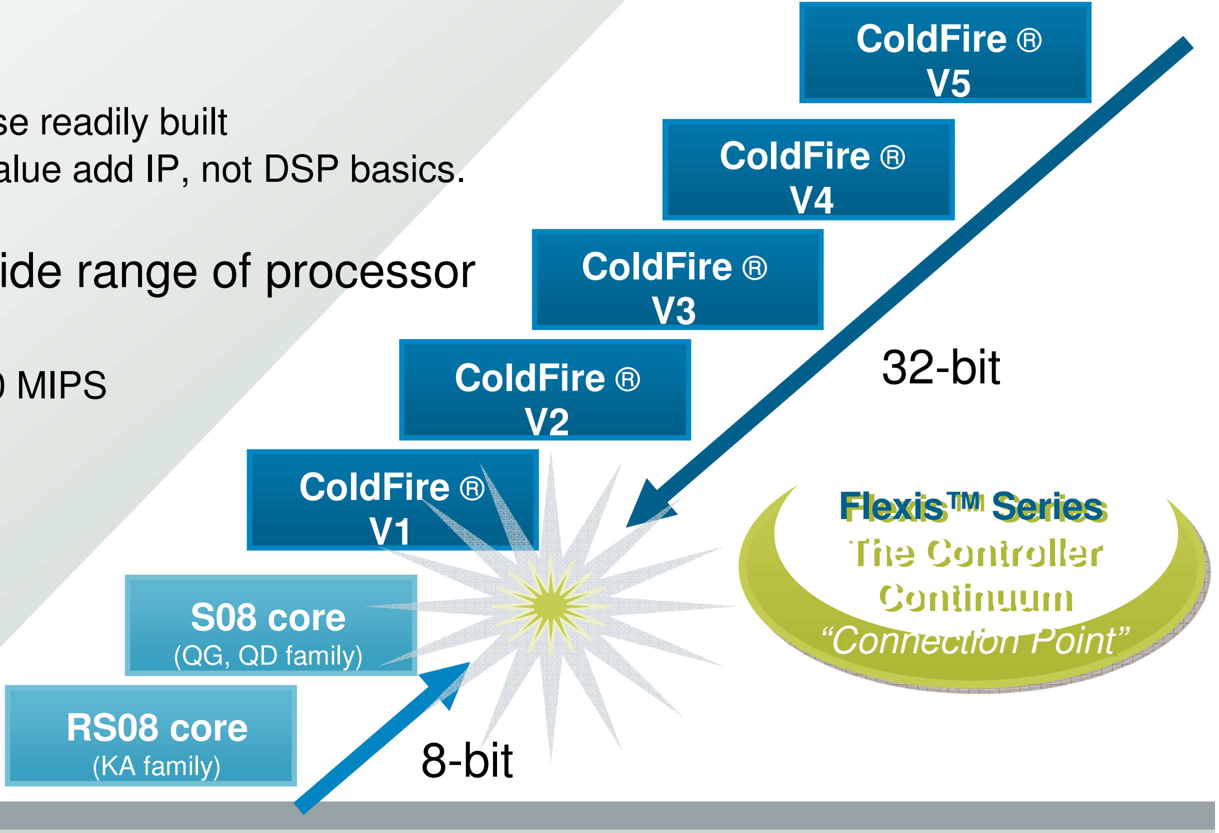
Performance and Memory

Filter Order	Compiled C		Assembly		Improvement	
	Time	Size	Time	Size	Time	Size
3	404 cycles	224 bytes	145 cycles	126 bytes	**2.79x**	**1.78x**
4	487 cycles	224 bytes	159 cycles	136 bytes	**3.06x**	**1.65x**
5	570 cycles	224 bytes	167 cycles	146 bytes	**3.41x**	**1.53x**

- Assembly is significantly smaller and faster than compiled C
- Compiler does not use MAC instructions effectively
- Cycle time increases with filter order in both implementations
- Code size increases with filter order in assembly only
  - Different assembly code for each filter order
- Cycle time improvement increases with filter order

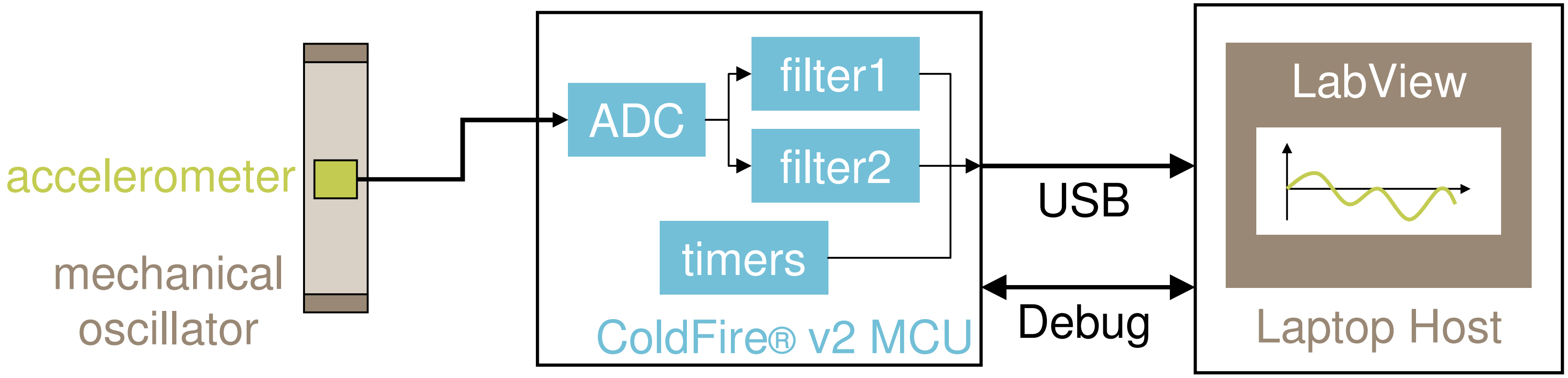
Summary

- ColdFire® + MAC architecture: Highly efficient for basic DSP ops.
  - 3 x 4th order IIR filters running at 3Khz consumed ~3% of MCU BW (at 60Mhz)
  - V2 at 60 Mhz could run 2x 4th order filters at ~200Khz SR.
- Tools and Libraries enable our customers to easily incorporate DSP
  - Use less expensive sensors
  - Provide higher value systems
  - Realize capabilities not otherwise readily built
  - Customers can focus on their value add IP, not DSP basics.
- Code is portable across a wide range of processor performance
  - V1 @ 10's of MIPs to V5 @ 600 MIPS
  - Extensive Tool Support
- Thank you. Questions?



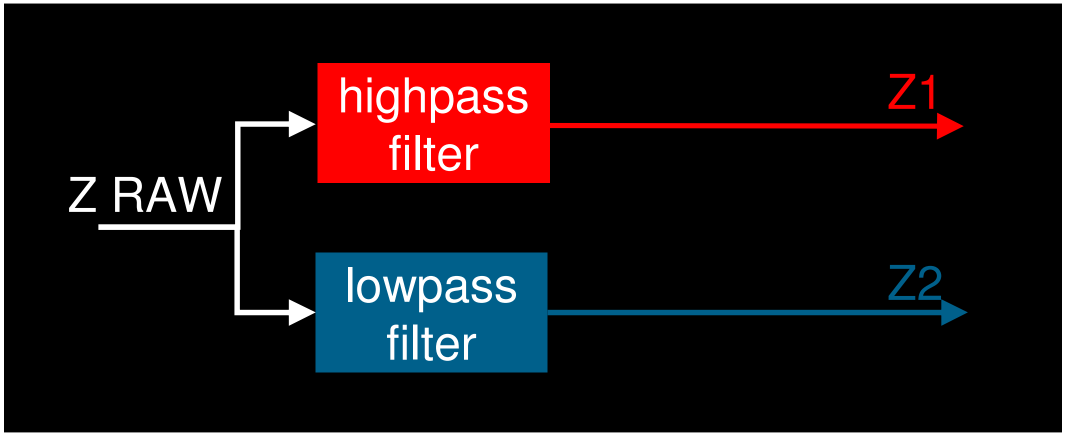
Hands-On Experiments

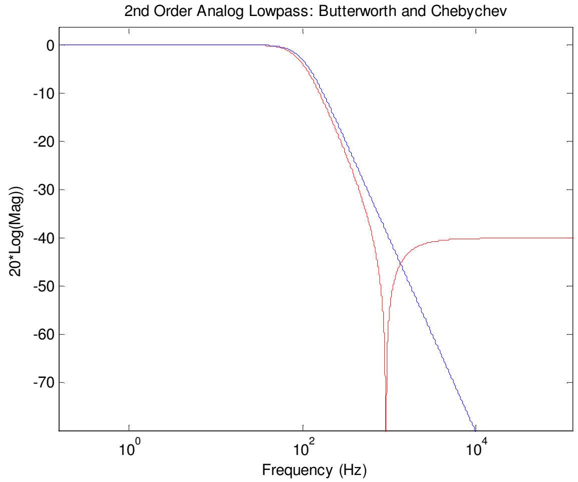
1. Walk-in Demo – Comparing Highpass and Lowpass Filters
2. Highpass Filter Design – Comparing Cutoff Frequencies
3. Cascaded Filter Design – Daisy-Chained Filters
4. Interactive Filter Design – Predicting Response
5. Signal Aliasing – Reducing Sample Rate

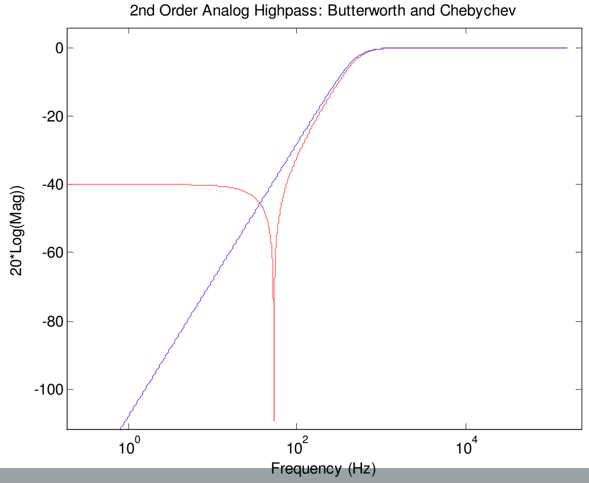


Experiment 1: Walk-In Demo

- Strike beam
- Compare response of highpass and lowpass filters
- Examine computational performance
- View aliasing effects

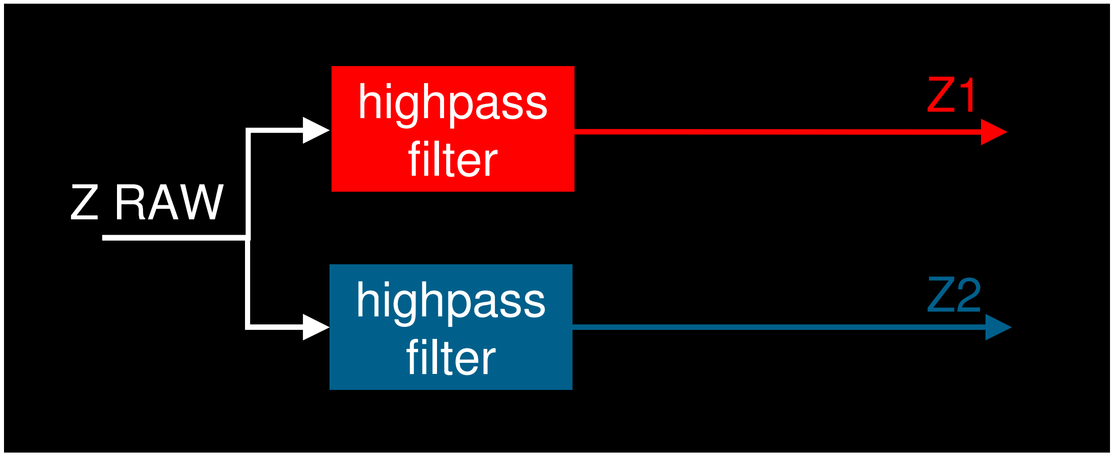


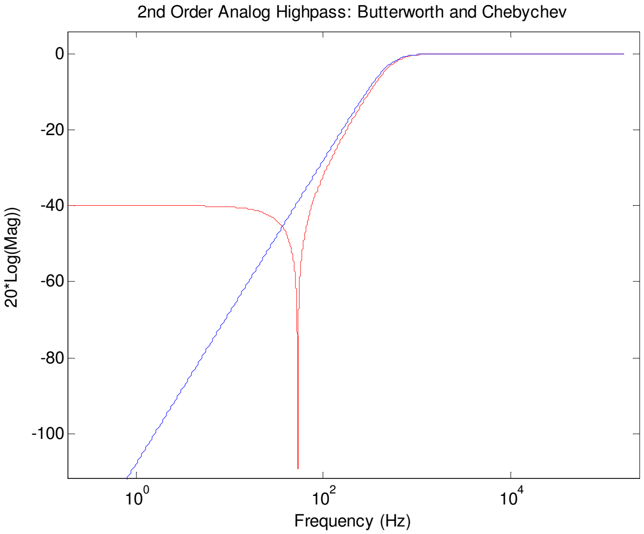




Experiment 2: Highpass Filter Design

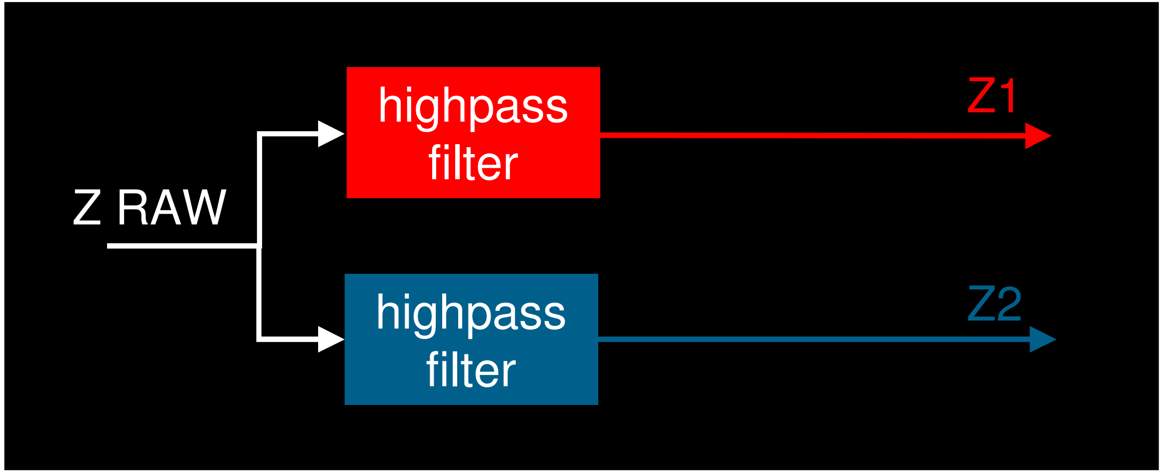
- ▶ Change filter coefficients in upper filter
  - Two highpass filters with different cutoff frequencies
- ▶ Compile code
- ▶ Program FLASH
- ▶ Strike beam
- ▶ Compare frequency responses of two filters



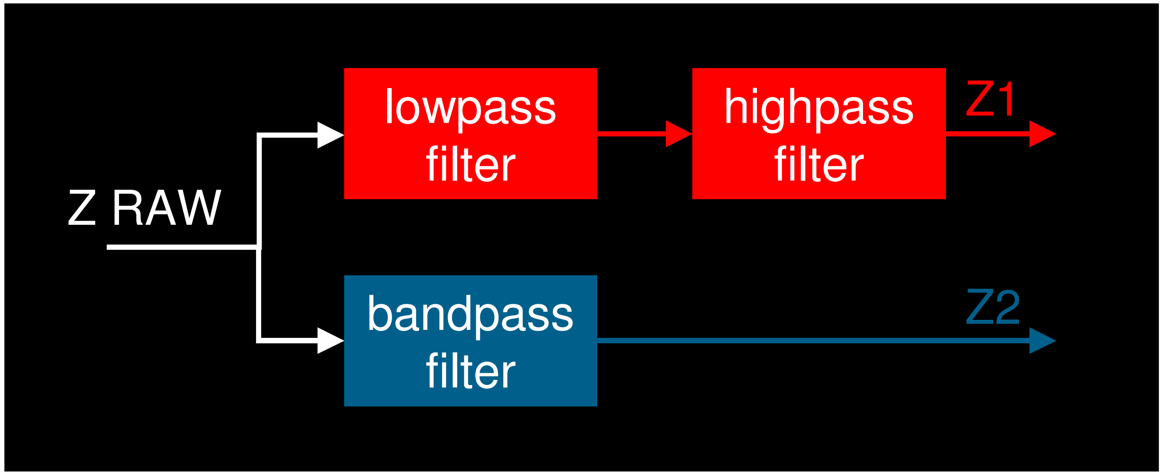

2nd Order Analog Highpass: Butterworth and Chebychev

Experiment 3: Cascaded Filter Design

- Insert new filter into lower path by daisy-chaining



Was This Structure



Now This Structure

Code Example: Daisy-Chaining

- ▶ Declare multiple data structures

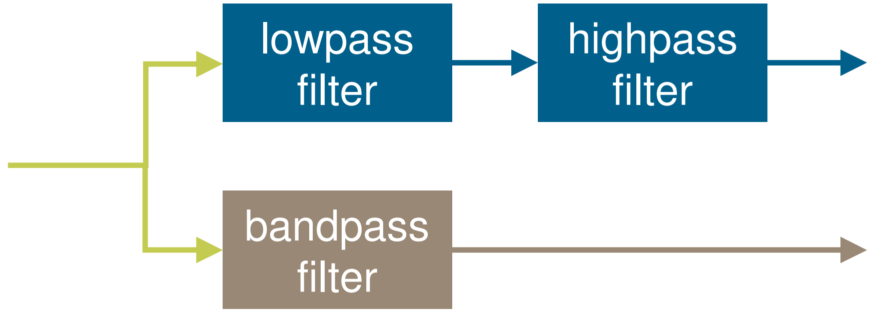
```
FILTER_STRUCT lowpass,highpass,bandpass;
```

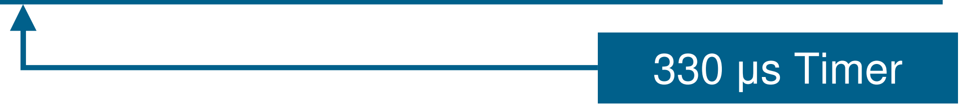
- ▶ Initialize multiple data structures

```
/* upper path */
filter_init(&lowpass,&sensor.output,lowpass_coef,...);
filter_init(&highpass,&lowpass.output,highpass_coef,...);

/* lower path */
filter_init(&bandpass,&sensor.output,bandpass_coef,...);
```

- ▶ Call filter function

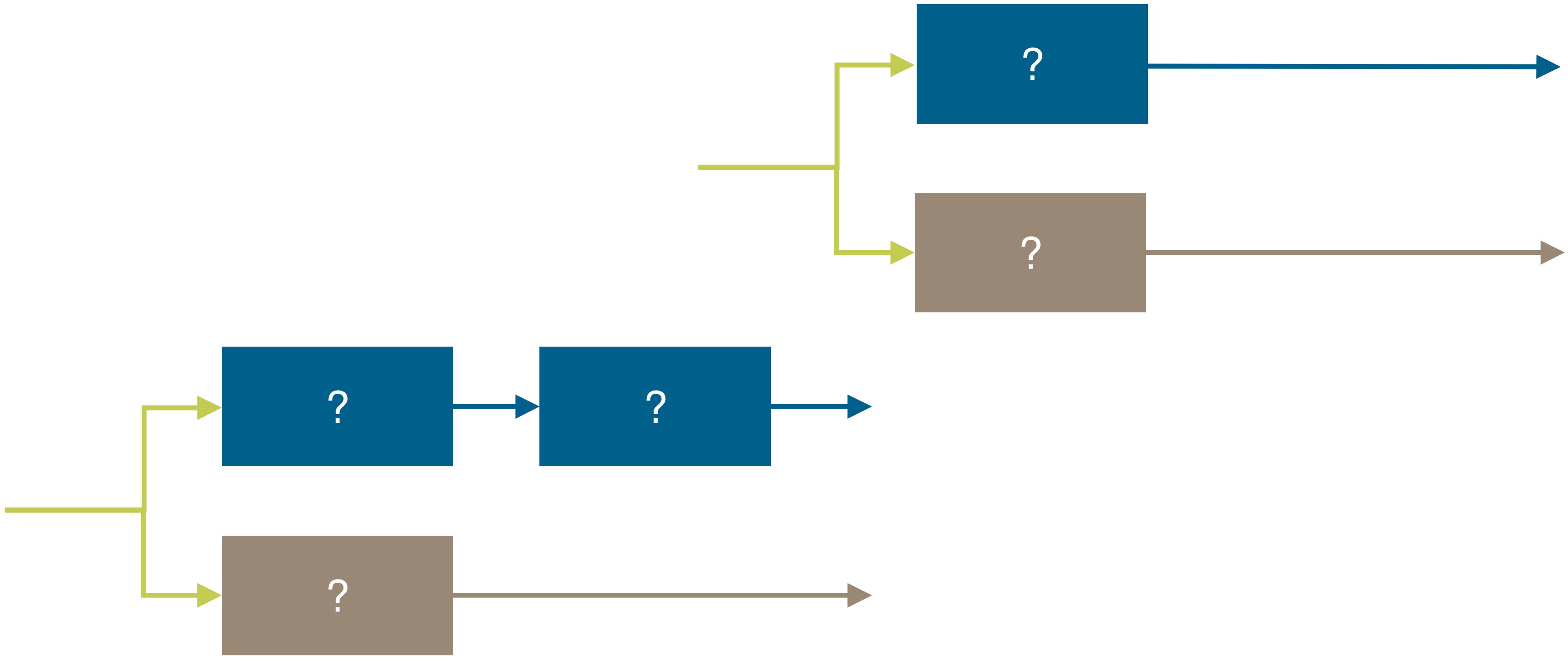
```
/* upper path – note order */
iir_asm(&lowpass);
iir_asm(&highpass);

/* lower path */
iir_asm(&bandpass);
```





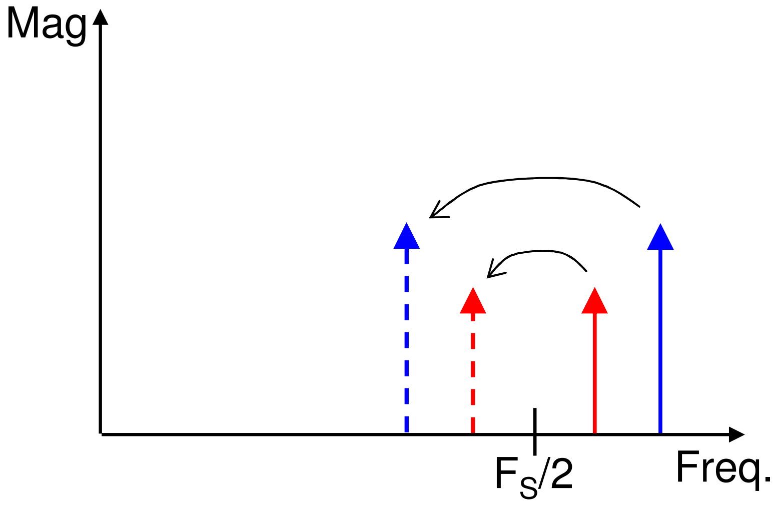
Experiment 4: Interactive Filter Design

- Choose your own filters, order, structure
- Predict frequency response
- Predict computational performance



Experiment 5: Sample Rate and Aliasing

- Decrease sample rate
- Change filter coefficients (maintain same continuous time cutoff frequency)
- View aliasing effects



freescale™
semiconductor

Filter Pseudo-code

```
acc = 0;
for (i=0;i<=N;i++) {
  acc = acc + a(i)*x(n-i);
}
for (j=1;j<=M;j++) {
  acc = acc + b(j)*y(n-j);
}
y(n) = acc;
```

$$y(n) = \underset{\text{accumulate}}{\sum_{i=0}^{N}} \overbrace{a(i)x(n-i)}^{\text{multiply}} + \underset{\text{accumulate}}{\sum_{j=1}^{M}} \overbrace{b(j)y(n-j)}^{\text{multiply}}$$

Fixed Point Filter Implementation

$$y(n)=\underset{\substack{\uparrow\\ \text{numerator coefficients}\\ \text{scale factor}}}{2^{-a\_sf}}\sum_{i=0}^{N}a(i)\overbrace{x(n-i)}^{\substack{\text{current and}\\ \text{previous inputs}}}+\underset{\substack{\uparrow\\ \text{denominator coefficients}\\ \text{scale factor}}}{2^{-b\_sf}}\sum_{j=1}^{M}b(j)\overbrace{y(n-j)}^{\text{previous outputs}}$$

- Compute numerator sum (multiply-accumulate)
- Scale by difference in coefficient scale factors (arithmetic bitshift)
  - This is initial value for denominator sum
  - Assumes larger denominator scale factor (smaller real value)
- Compute denominator sum (multiply-accumulate)
- Scale by denominator coefficient scale factor (arithmetic bitshift)
- Update input and output buffers
  - x[n] becomes x[n-1] for next iteration, etc.

Coefficient Fixed Point Scaling

- Compare magnitude between x and y coefficients
  - x coefficients smaller by orders of magnitude
- Use different fixed point scaling

$$y(n) = 1.77*10^{-4}x(n) + 7.06*10^{-4}x(n-1) + 1.06*10^{-3}x(n-2) + 7.06*10^{-4}x(n-3) + 1.77*10^{-4}x(n-4)$$
$$+3.35*y(n-1) - 4.25*y(n-2) + 2.42*y(n-3) - 0.52*y(n-4)$$

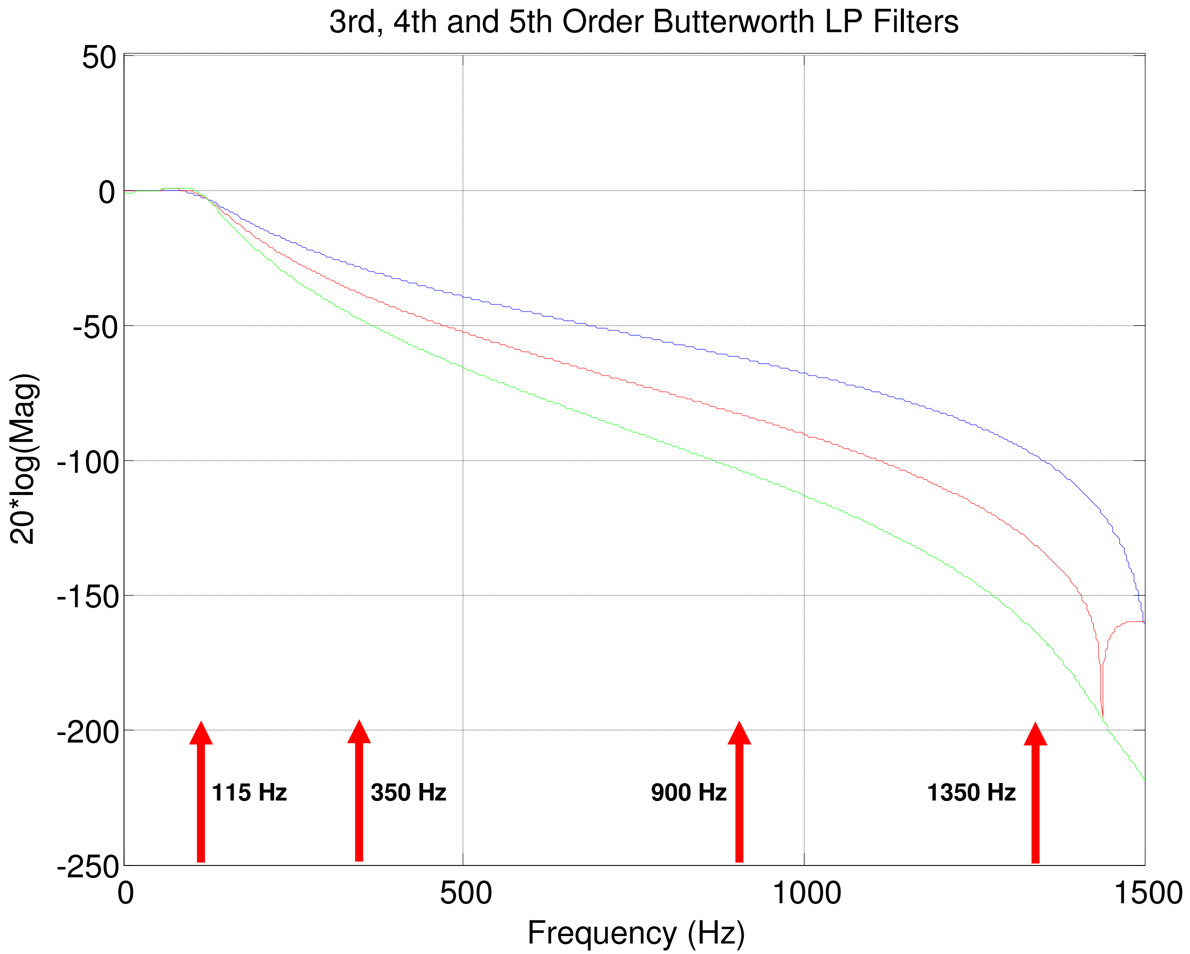
$$a = \{1.77*10^{-4}, 7.06*10^{-4}, 1.06*10^{-3}, 7.06*10^{-4}, 1.77*10^{-4}\}$$
$$b = \{3.35, -4.25, 2.42, -0.52\}$$

Sample Filters for Demos

Blue: 3^{rd} LP, 120 Hz

Red: 4^{th} LP, 120 Hz

Green: 5th LP, 120Hz



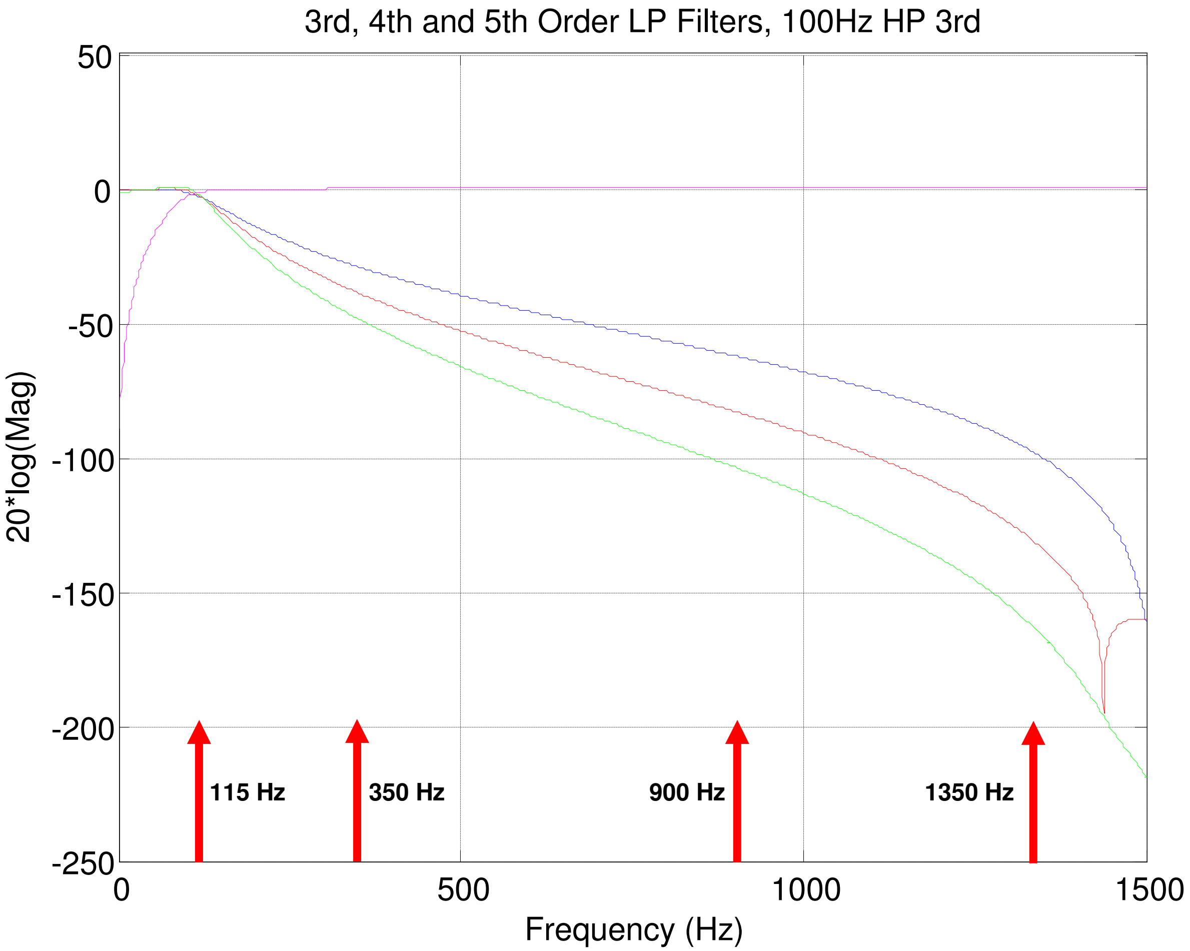
Sample Filters for Demos

Magenta: 3rd HP, 100 Hz

Blue: 3rd LP, 120 Hz

Red: 4th LP, 120 Hz

Green: 5th LP, 120Hz



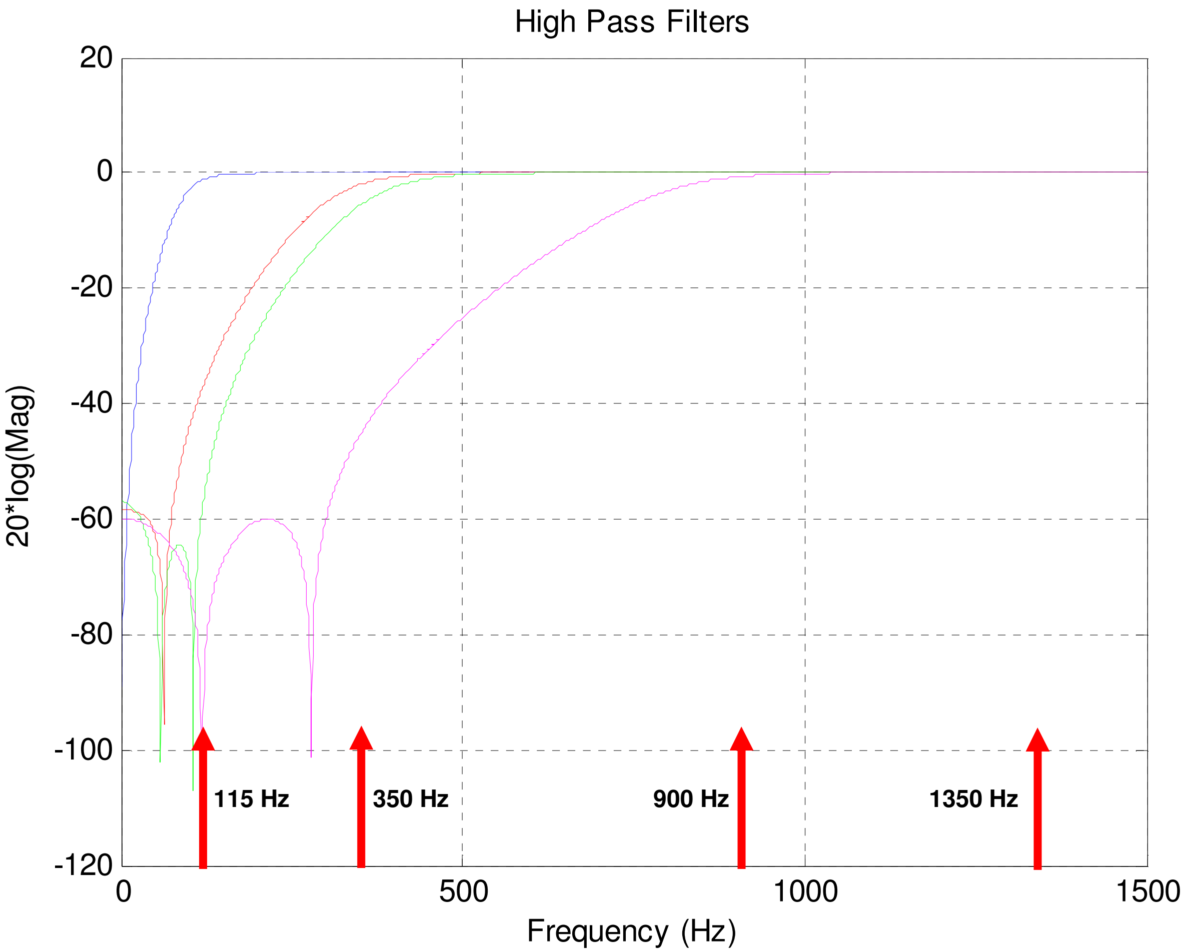
Sample Filters for Demos

Blue = 3rd Butterworth 100 Hz HP

Red = 4th Butterworth 350 Hz HP

Green = 4th Chebychev 350 Hz HP

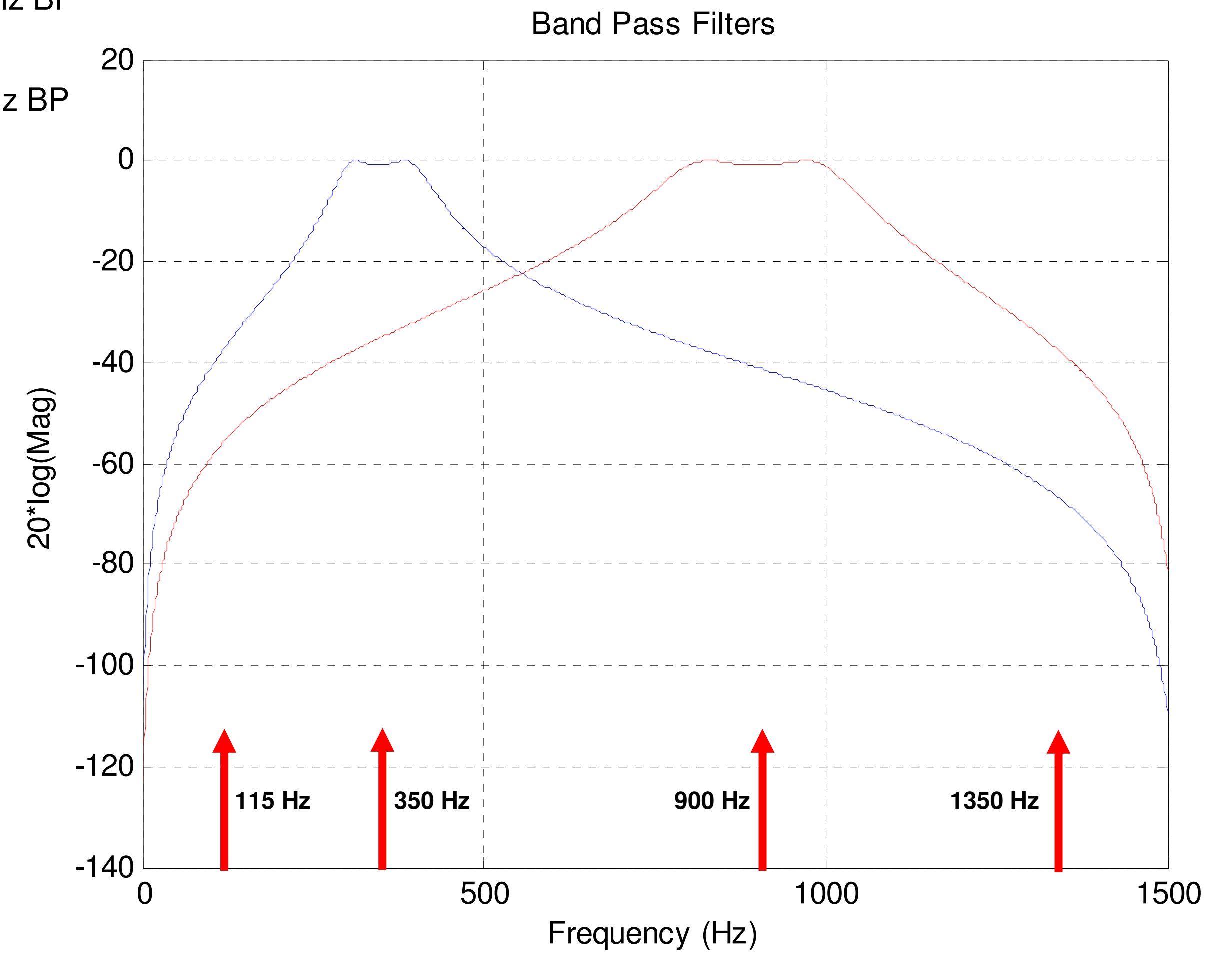
Mag. = 4th Chebychev 900 Hz HP



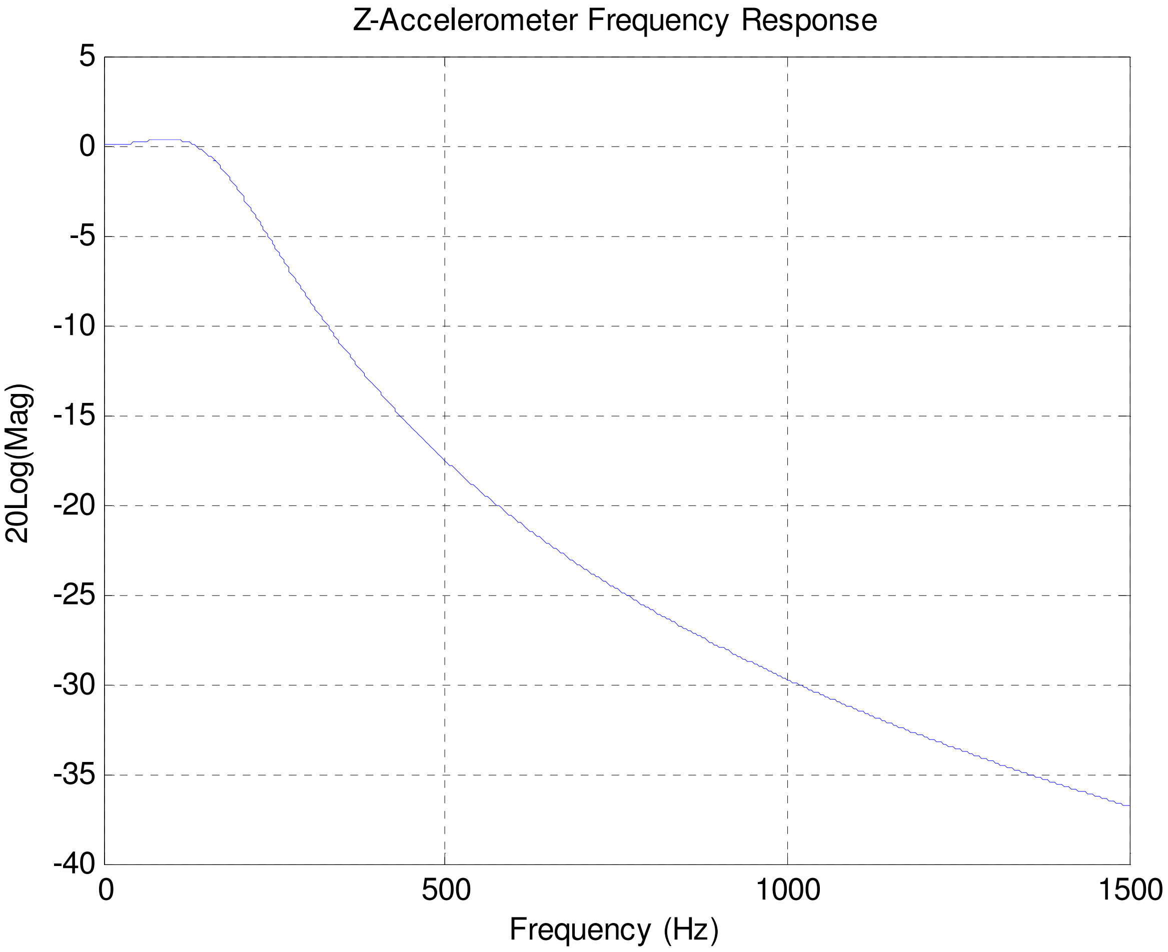
Sample Filters for Demos

Blue = 4th Order Chebychev 350 Hz BP

Red = 4th Order Chebychev 900 Hz BP



Accelerometer Frequency Response



Related Session Resources

Sessions (Please limit to 3)

Session ID	Title
AZ304	Hands-On Workshop: Coldfire Technology and Digital Signal Processing

Demos (Please limit to 3)

Pedestal ID	Demo Title

Meet the FSL Experts (Please limit to 3)

Title	Time	Location
Controller Continuum	2-4, June26	

freescale™
semiconductor