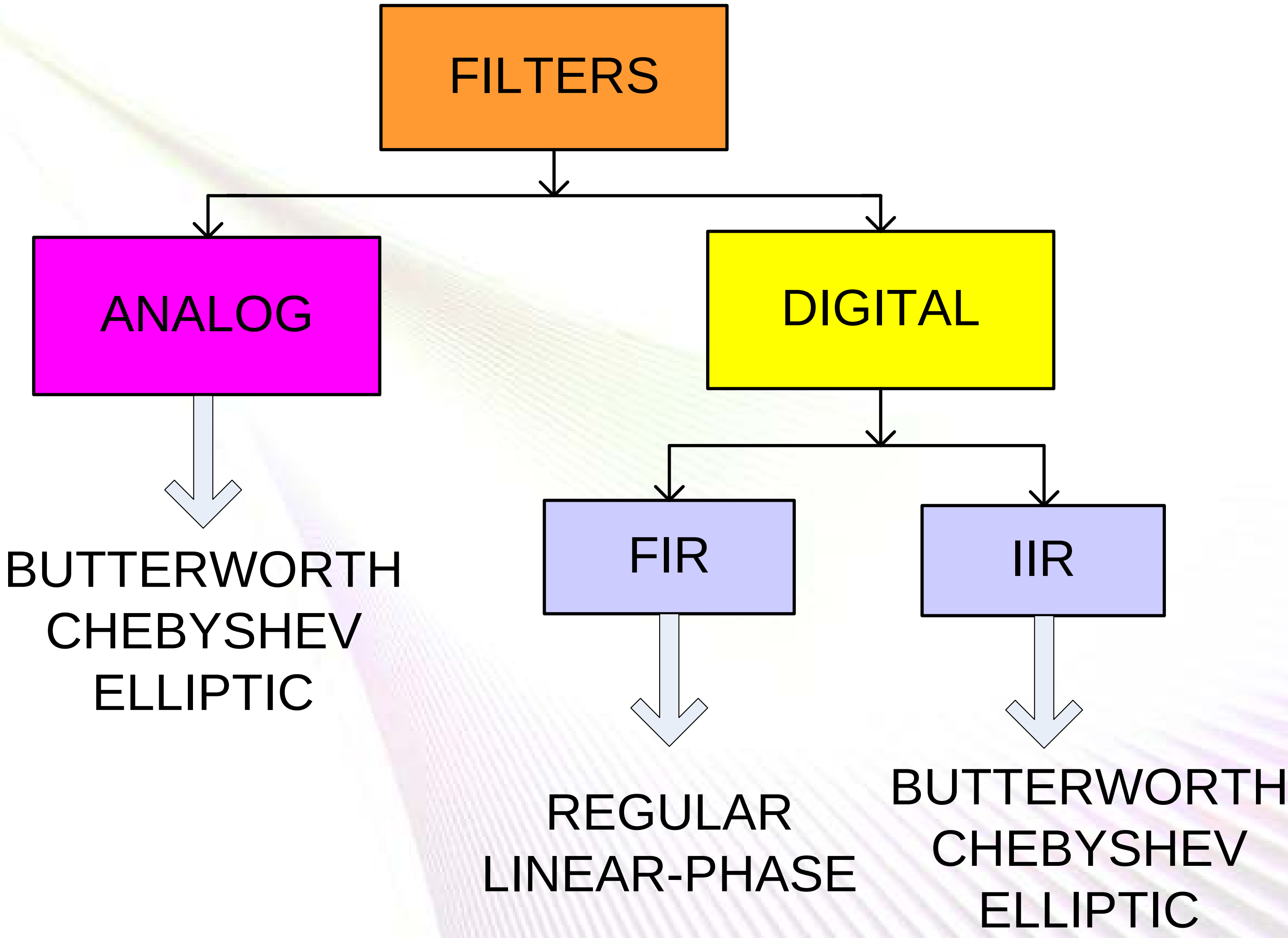
Optimized Digital Filtering for the MSP430

Kripasagar Venkat
MSP430 Applications Engineer
Texas Instruments

Agenda

- Broad classification of Filters
- Number representations
- Fast Algorithms
- Digital Filtering on the MSP430
- Performance on the MSP430

Broad classification of filters

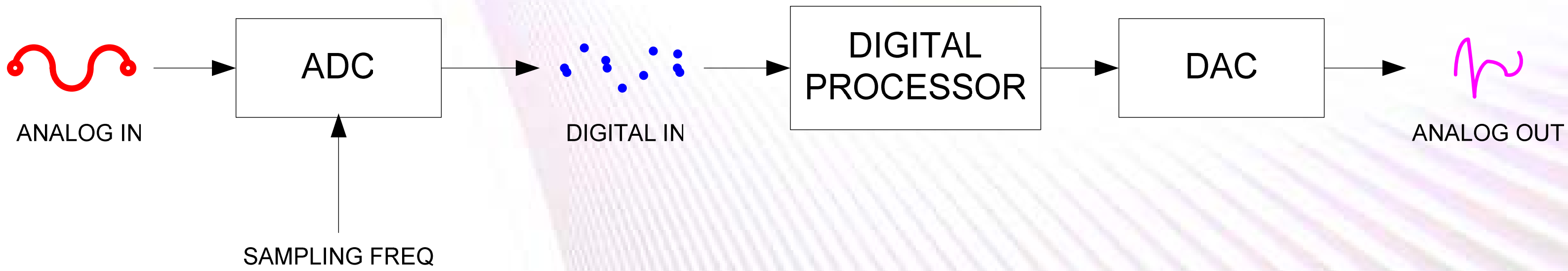


Why Digital? Analog Vs Digital filters

- **Analog filters**
  - Mature and well developed design methodologies available
  - Accuracy is limited, as they use components that are subjected to tolerances
  - Any change in filter specifications calls for a complete change in hardware with testing and verifications repeated
  - Storage and portability a cause for concern
  - Inherently expensive to improve accuracy
- **Digital filters**
  - Design is simple, borrows all concepts from its analog counterpart
  - Modifying the characteristics requires just a small change in software with no hardware changes necessary
  - With everything digital and the advent of digital microcomputers, interface is extremely simple
  - Extremely accurate→ At least a 1,000 times better accuracy when compared to its analog counterpart
  - 6dB increase in gain with every bit of increase in resolution for fixed point
  - Must consider effects of round-off, finite-word lengths and limit cycles in fixed point machines

Signal representations

- **Analog**
  - Everything in continuous domain
  - Analog in, Analog out
  - Post processing difficult
  - Frequency domain analysis difficult
- **Digital**
  - Sampling done to analog signals to convert them to digital using an Analog to Digital Converter (ADC)
  - Conversion back to analog done after processing using a Digital to Analog converter (DAC)
  - Number representations and resolution a key to performance



  - Input/Output easily captured and stored on digital media for post-processing

Agenda

- Broad classification of Filters
- Number representations
- Fast Algorithms
- Digital Filtering on the MSP430
- Performance on the MSP430

Number representations

- **Types of binary representation**
  - Unsigned binary numbers
  - Sign magnitude
  - 1's complement
  - 2's complement

- **Types of ternary representations**
  - Booth's encoding
  - Canonical Signed Digit representation

Unsigned Binary numbers

- Used to represent positive numbers only
- Full range of 0 to 2N-1 available for a N-bit binary representation
- Hassle-free number representations in the absence of sign-bits
- Sometimes used for uni-polar representations

- Example

  123=01111011b

Sign magnitude binary numbers

- Simple conversion and representation of the binary numbers
- Negative numbers included and the leftmost bit (MSB) designated as the sign-bit
- Dynamic range from $-2^{(N-1)}-1$ to $+ 2^{(N-1)}-1$ for a N-bit binary representation
- Hardware circuitry simpler
- Rarely used in practice

- **Example**

Sign bit

$123=01111011_b$

Sign bit

$-123=11111011_b$

1s complement binary numbers

- One of the widely used binary representation
- Negative numbers can be represented with the leftmost bit (MSB) as the sign-bit
- Dynamic range from $-2^{(N-1)}-1$ to $+ 2^{(N-1)}-1$
- Representation of positive integers is similar to unsigned representation
- Representation of negative integers is the complement (bitwise NOT) of their positive representations
- **Example** **Sign bit**

$123=01111011_b$

Sign bit

$-123=NOT(123=01111011_b)=10000100_b$

2s complement binary numbers

- The most commonly used binary representation among digital devices
- Negative numbers can be represented with the leftmost bit (MSB) as the sign-bit
- Dynamic range from $-2^{(N-1)}$ to $+ 2^{(N-1)}-1$
- Representation of positive integers is similar to unsigned representation
- Representation of negative integers is the 1's complement (bitwise NOT) + 1_b of their positive representations
- **Example**

Sign bit

$123=01111011_b$

Sign bit

$-123=NOT(123=01111011_b)+1_b=10000101_b$

Summary of Data representations

Number	Sign-magnitude	1s complement	2s complement
-128	-------	-------	0x80=10000000
-127	0xFF=11111111	0x80=10000000	0x81=10000001
-126	0xFE=11111110	0x81=10000001	0x82=10000010
:	:	:	:
-1	0x81=10000001	0xFE=11111110	0xFF=11111111
0	0x00/0x80=0/10000000	0x00=00000000	0x00=00000000
1	0x01=00000001	0x01=00000001	0x01=00000001
:	:	:	:
+126	0x7E=01111110	0x7E=01111110	0x7E=01111110
+127	0x7F=01111111	0x7F=01111111	0x7F=01111111

Booth's encoding [5]

- Done to increase the speed of execution of many algorithms
- "-1" added to the existing binary set thereby converting it to a ternary set
- Algorithm groups pairs of adjacent bits in the binary representation resulting in a ternary set
- $t_i = b_{i-1} - b_i$ for i = 0 to N -1 (N-bit representation)
- **Example** $\mathbf{123 = 01111011_b \Rightarrow}$ **Binary format**

$$\mathbf{123 = 64 + 32 + 16 + 8 + 2 + 1}$$

$$\mathbf{123 = \underbrace{01111011}_{t_i = b_{i-1} - b_i}0 \hookleftarrow \text{Implied zero at bit position -1}}$$

$$\mathbf{123 = 1000\tilde{1}10\tilde{1}_t = \text{Ternary format, } \tilde{1} \Rightarrow -1}$$

$$\mathbf{123 = 128 - 8 + 4 - 1 = 123}$$

Canonical signed digit representation [2]

- Similar to Booth's encoding: It increases the speed of execution
- "-1" added to the existing binary set thereby converting it to a ternary set
- Algorithm: Reducing groups of adjacent 1s and representing them using a ternary set
- Leaves the 0s unchanged
- **Example**

$$\mathbf{123 = 01111011_b \Rightarrow \text{Binary format}}$$

$$\mathbf{123 = 011110\underbrace{11}_{\text{grouped}} = 0\underbrace{11111}_{\text{grouped}}0\tilde{1}_t = 10000\tilde{1}0\tilde{1}_t}$$

$$\mathbf{123 = 10000\tilde{1}0\tilde{1}_t \Rightarrow \text{Ternary format, } \tilde{1} \Rightarrow -1}$$

$$\mathbf{123 = 128 - 4 - 1 = 123}$$

Ternary representation of fractions

- Fractions can also be represented in a ternary form
- **Booth encoding example**

$$0.12345 = 0.000111111001_b$$

$$0.12345 = 0.000111111001\ \underline{0}_{\text{Implied zero}}$$

$$0.12345 = 0.00100000\tilde{1}01\tilde{1}\ _{\text{BOOTH}}$$

- **CSD encoding example**

$$0.12345 = 0.000111111001_b$$

$$0.12345 = 0.000\underbrace{111111}_{\text{grouped}}001_b$$

$$0.12345 = 0.00100000\tilde{1}001_{\text{CSD}}$$

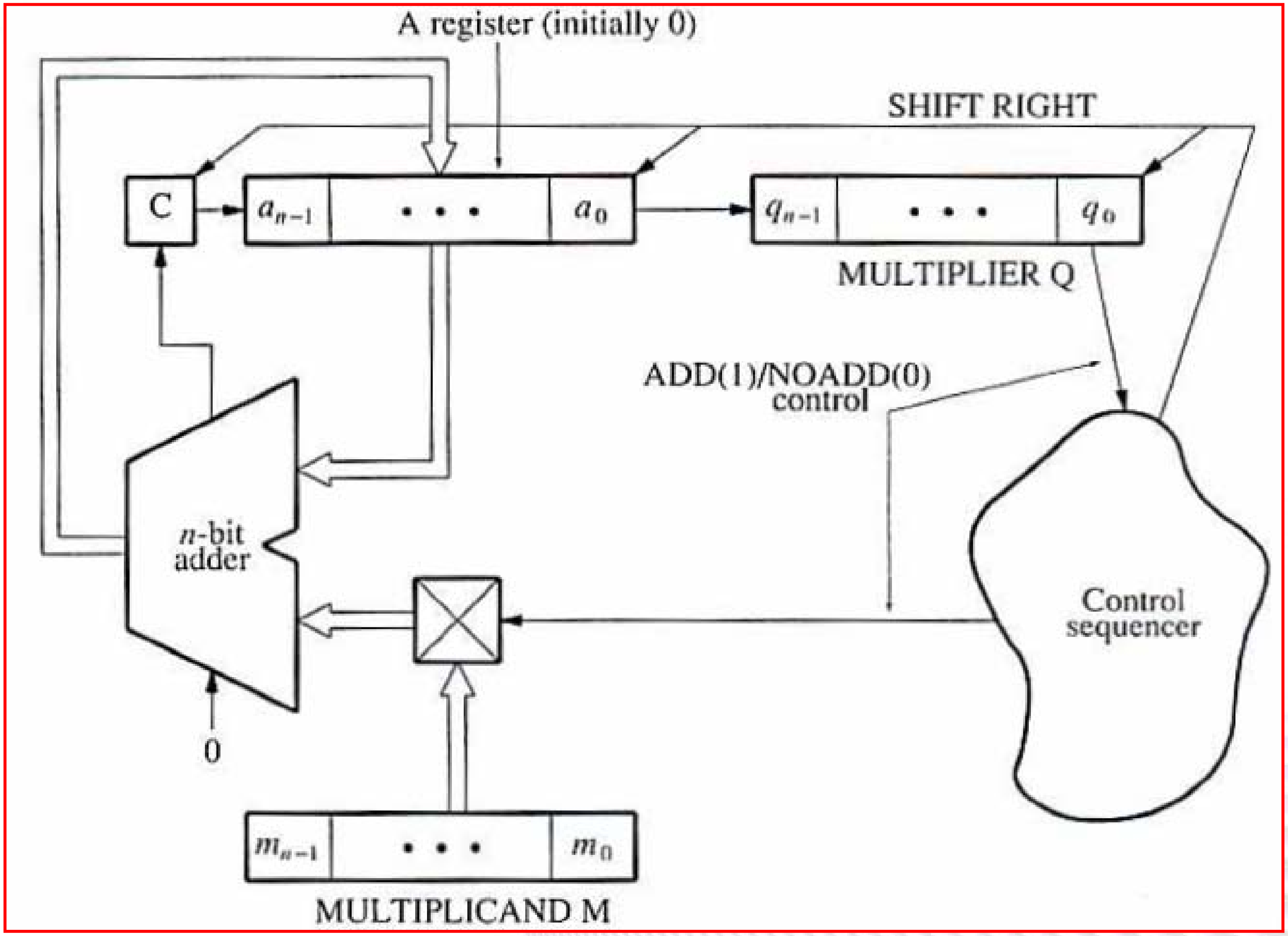
Agenda

- Broad classification of Filters
- Number representations
- Fast Algorithms
- Digital Filtering on the MSP430
- Performance on the MSP430

Existing Fast Algorithms

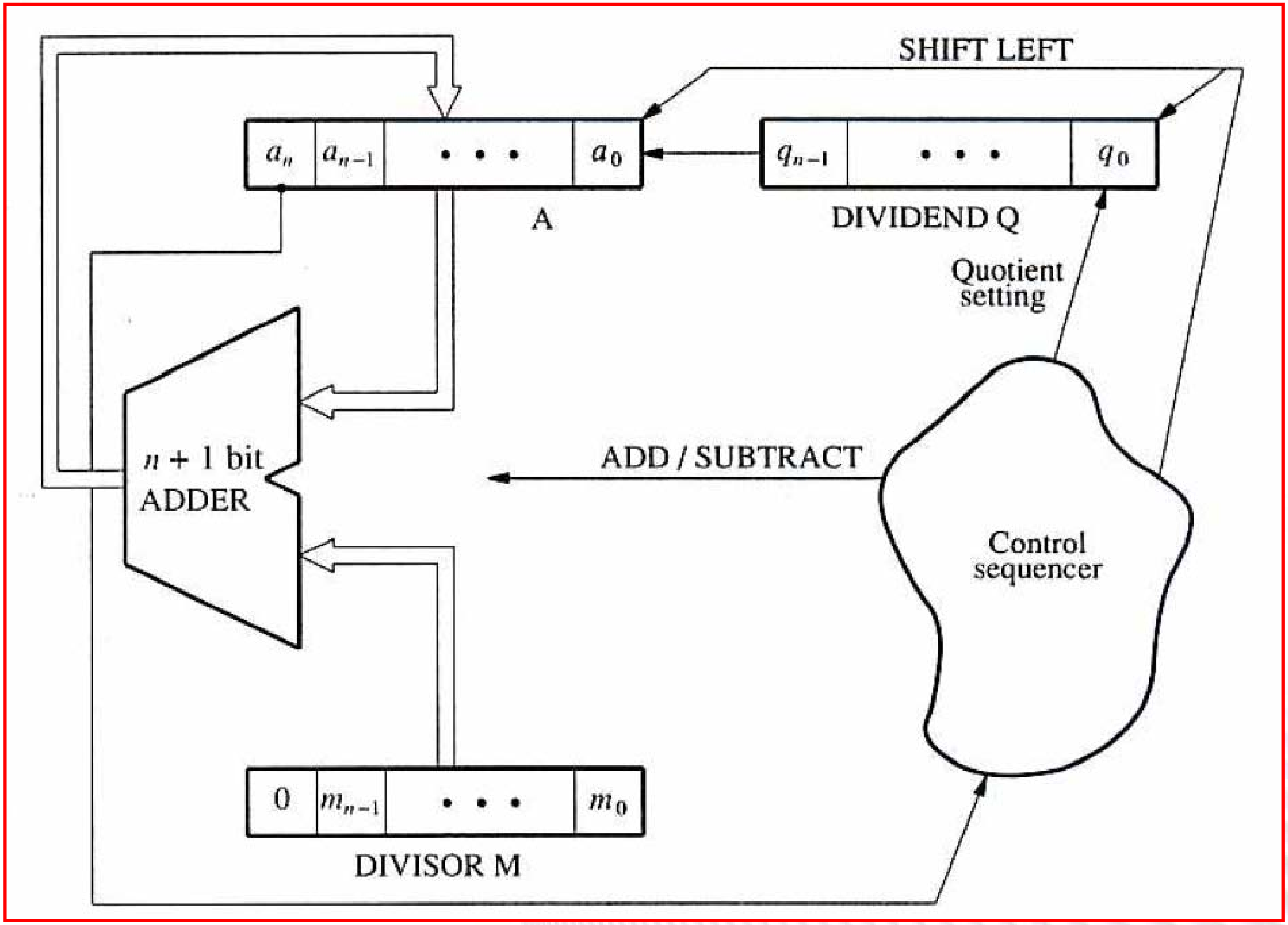
- **Fast Multiplication**
  - Based on shift and add arithmetic
  - Tailor-made for micro-controllers in the absence of a hardware multiplier
  - Limited to integer-integer multiply
- **Fast division**
  - Based on shift and add arithmetic
  - Limited to integer-integer division
- **Horner's scheme**
  - Also based on shift and add arithmetic
  - Tailor-made for micro-controllers in the absence of a hardware multiplier
  - Exhibits better accuracy for the same register-width limitations
  - Supports integer-float multiplication and division
  - Faster than the existing algorithms when used with CSD format

Existing multiplication algorithm [5]



Directly taken from Reference 5

Existing Division algorithm [5]



Directly taken from Reference 5

Horner's algorithm for multiplication [2]

- Uses only shift and add instructions
- Based on the difference in the bit positions of 1s in the multiplier
- Exhibits better accuracy compared to the existing methods
- Finite word-length effects does not affect the multiplier
- Scaling of multipliers not needed and easily accommodates floating point arithmetic
- Increases code size

Horner's algorithm-Description

- Representation of multipliers

Fraction

$$0.12345 = 0.000111111001_b$$

Design Equations

$$X_1 = X \cdot 2^{-3} + X$$

$$X_2 = X_1 \cdot 2^{-1} + X$$

$$X_3 = X_2 \cdot 2^{-1} + X$$

$$X_4 = X_3 \cdot 2^{-1} + X$$

$$X_5 = X_4 \cdot 2^{-1} + X$$

$$X_6 = X_5 \cdot 2^{-1} + X$$

$$\textit{Final result} = X_6 \cdot 2^{-4}$$

Integer

$$441 = 0110111001_b$$

Design Equations

$$X_1 = X \cdot 2^{1} + X$$

$$X_2 = X_1 \cdot 2^{2} + X$$

$$X_3 = X_2 \cdot 2^{1} + X$$

$$X_4 = X_3 \cdot 2^{1} + X$$

$$X_5 = X_4 \cdot 2^{3} + X$$

$$\textit{Final result} = X_5 \cdot 2^{0}$$

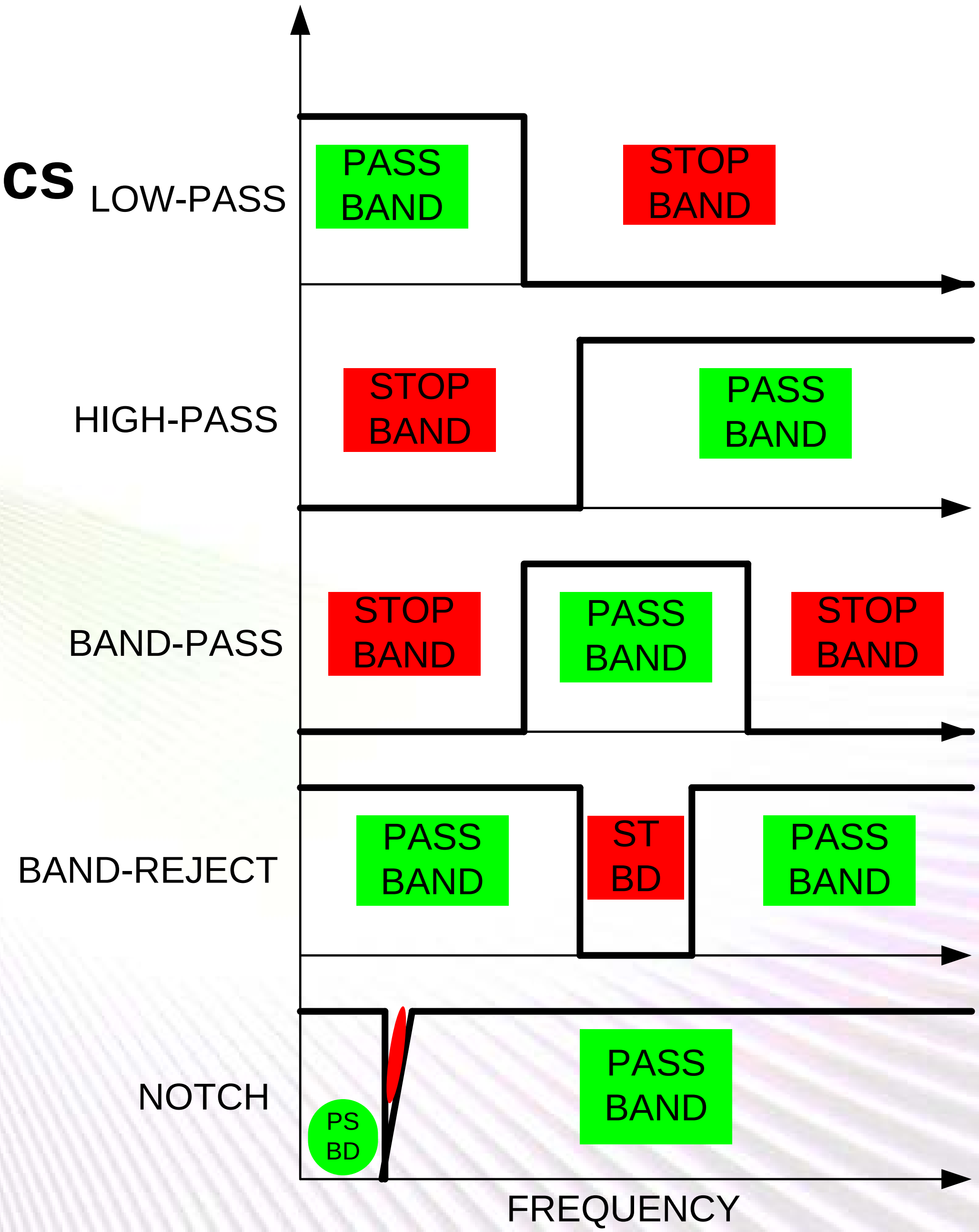
Agenda

- Broad classification of Filters
- Number representations
- Fast Algorithms
- Digital Filtering on the MSP430
- Performance on the MSP430

Digital Filtering

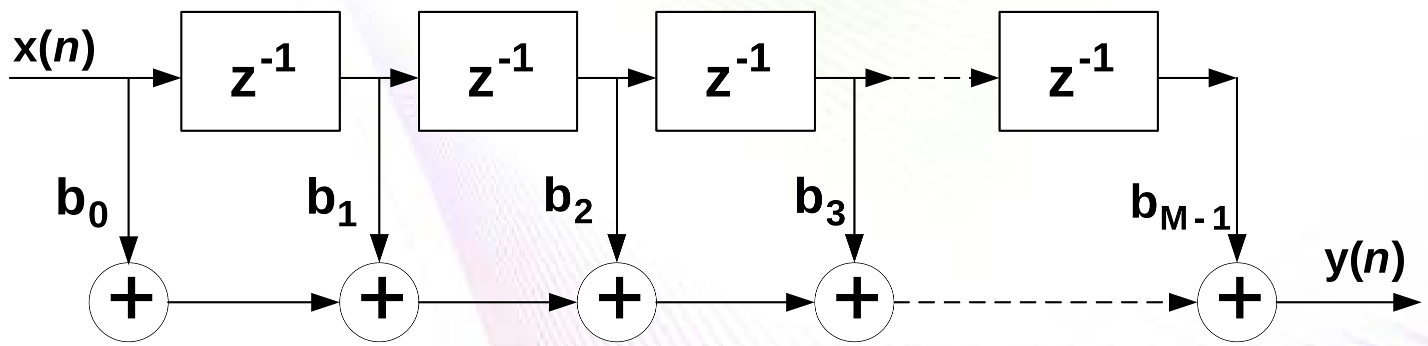
- **Frequency characteristics**
  - Low-pass
  - High-pass
  - Band-pass
  - Band-reject
  - Notch

- **Basic types**
  - FIR
  - IIR



FIR filters

- Finite Impulse response filters
- Simplest to design
- Inherently stable
- Can exhibit linear phase across all frequencies

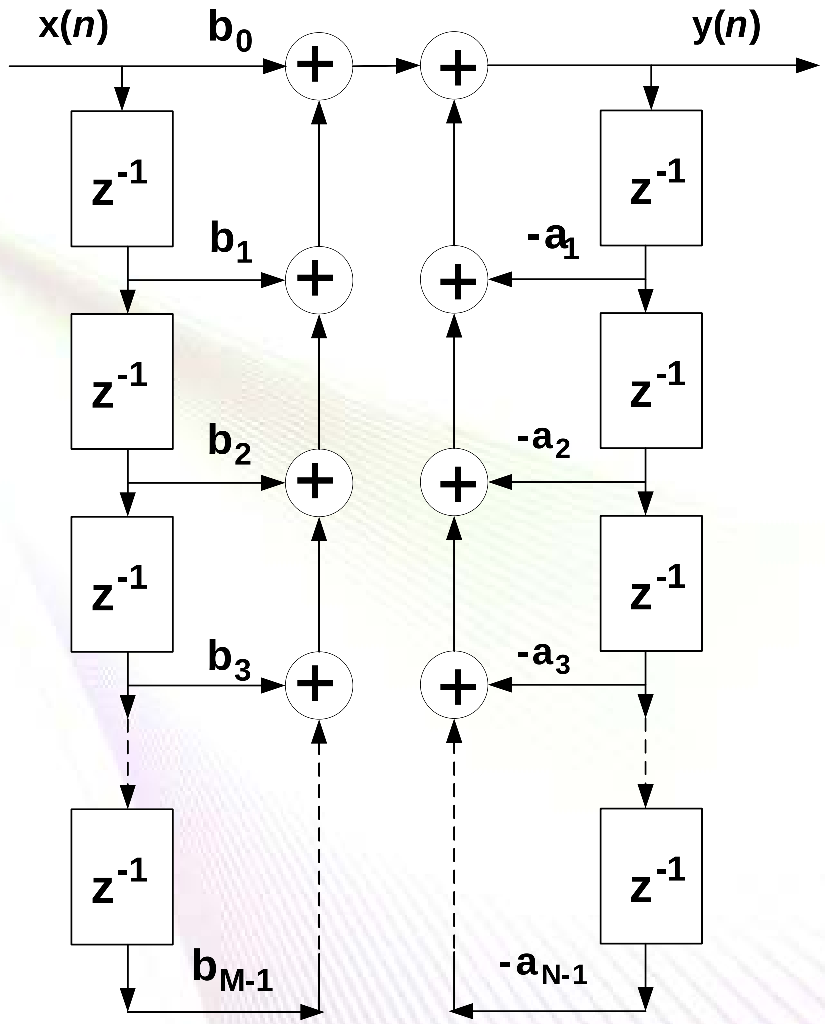


$$y(n) = \sum_{i=0}^{M-1} b(i) \cdot x(n-i)$$

IIR filters

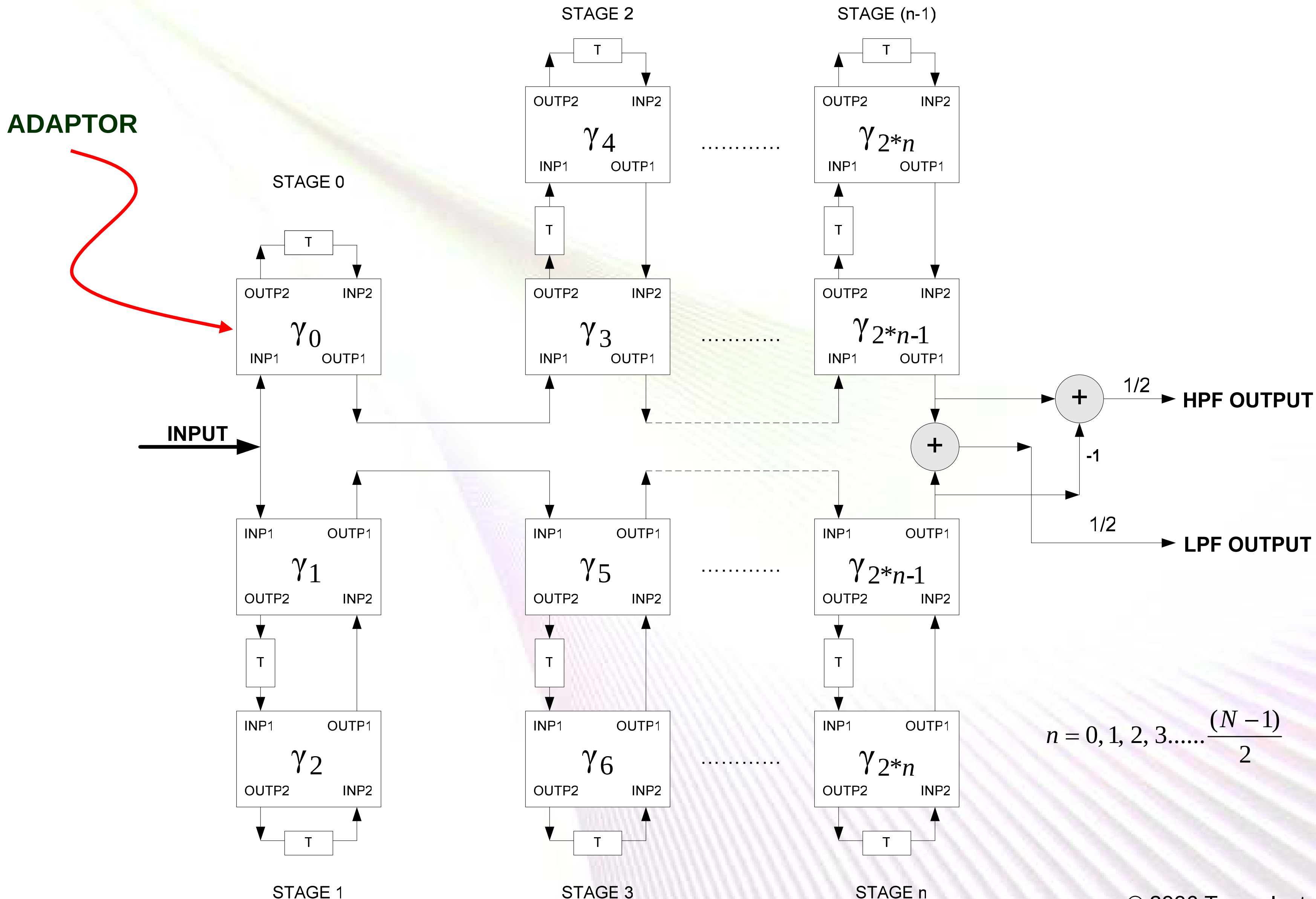
- **Conventional**
  - Designed directly from Analog filter counterparts
  - Perform better than the FIR filter for the same order
  - Recursive in both input and output samples
  - Extremely sensitive to filter coefficients
  - Performance is below par due to register-width limitations in fixed point machines
- **Wave Digital Filters [3,4]**
  - Answer to all the problems faced by conventional IIR filters
  - Tailor-made for Fixed point low-end micro-controllers
  - Extremely stable over non-linear operating conditions
  - The coefficients have excellent dynamic range
  - Little effect from register-width limitations
  - Perform as well as the Conventional IIR filters
  - Lattice structure most widely used

Conventional IIR filter signal flow



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

LWDF Signal Flow diagram



$$n = 0, 1, 2, 3 \ldots\ldots \frac{(N-1)}{2}$$

LWDF-Adaptor types

- The coefficients (γ) of the LWDF is always between -1 and 1
- To improve the amplitude scaling performance the entire range [-1,1] is divided into sub-ranges and different structures are used inside their respective adaptor

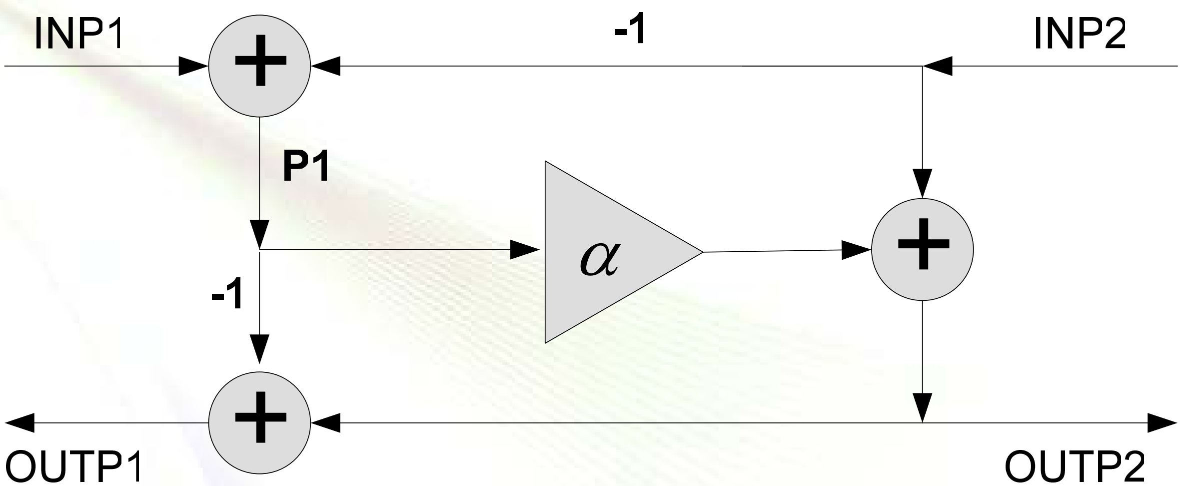
$$Type\ \ 1 \qquad 0.5 < \gamma < 1, \qquad \alpha = 1 - \gamma$$

$$Type\ \ 2 \qquad 0 < \gamma \le 0.5, \qquad \alpha = \gamma$$

$$Type\ \ 3 \qquad -0.5 \le \gamma < 0, \qquad \alpha = |\gamma|$$

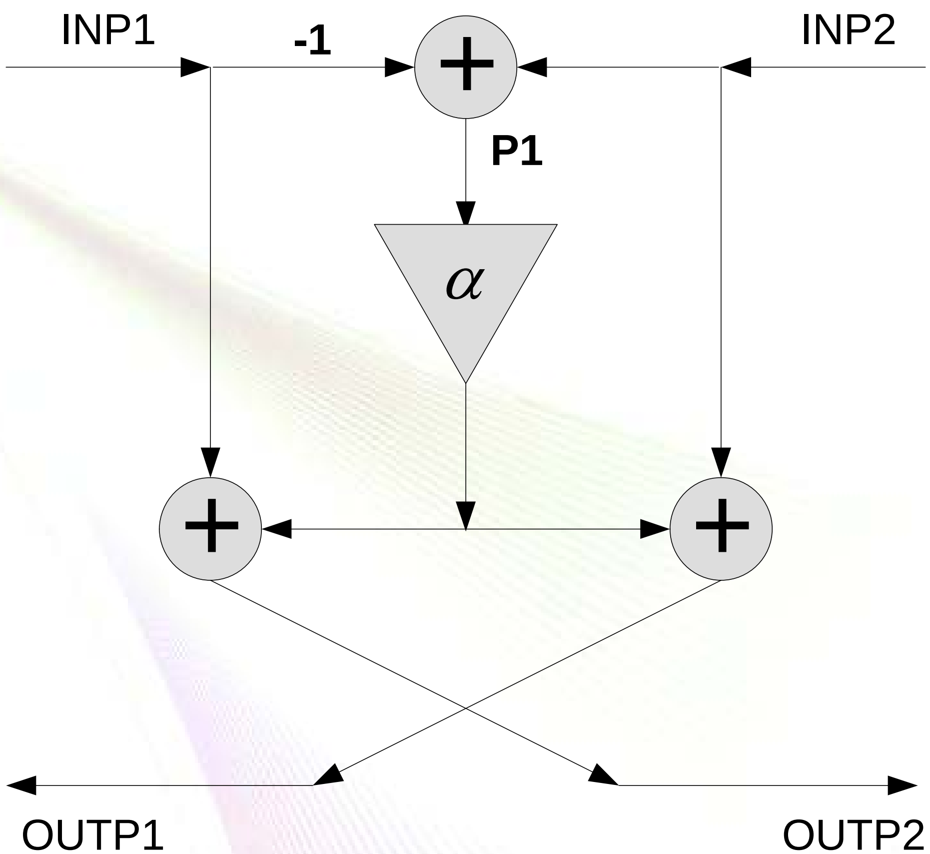
$$Type\ \ 4 \qquad -1 < \gamma < -0.5, \qquad \alpha = 1 + \gamma$$

Type 1 Adaptor structure



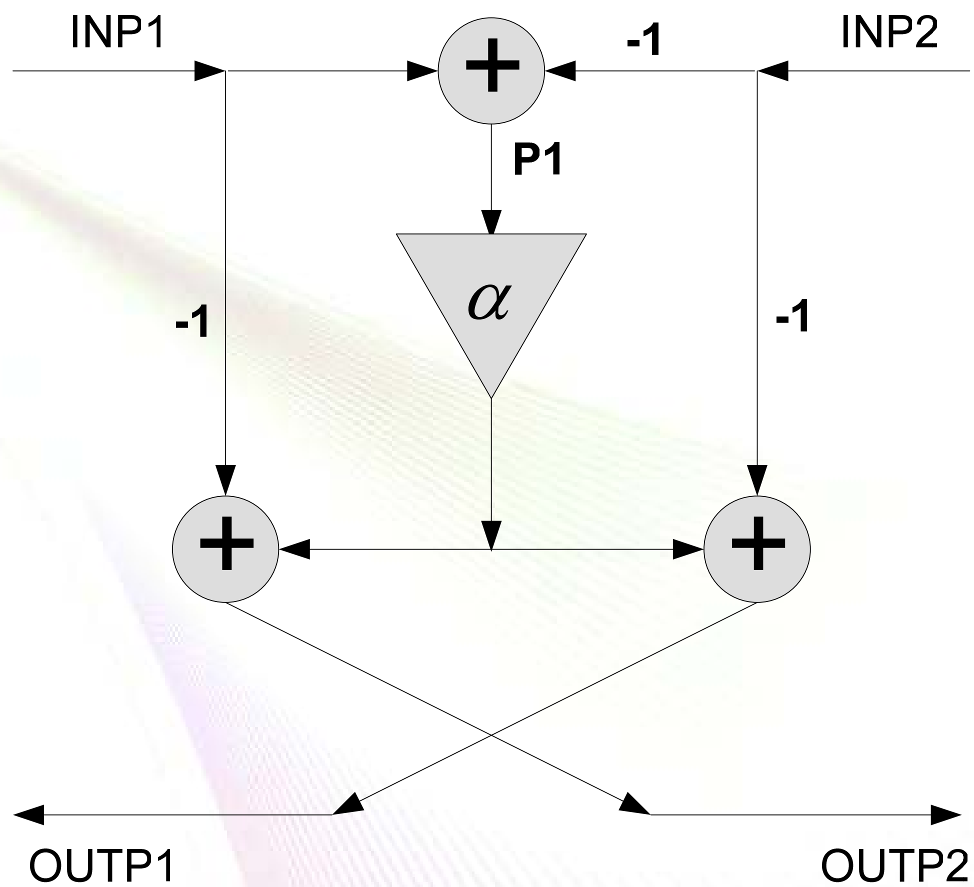
P1 = INP1 - INP2
OUTP2 = α * P1 + INP2
OUTP1 = OUTP2 - P1

Type 2 Adaptor structure



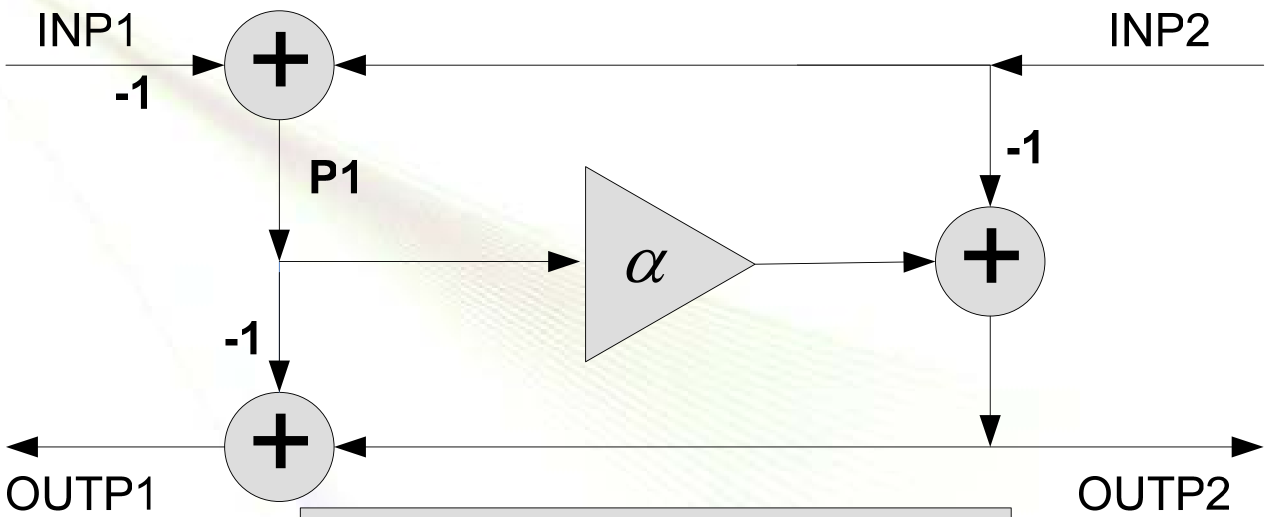
P1 = INP2 - INP1
OUTP2 = α * P1 + INP1
OUTP1 = α * P1 + INP2

Type 3 Adaptor structure



P1 = INP1 - INP2
OUTP2 = α * P1 - INP1
OUTP1 = α * P1 - INP2

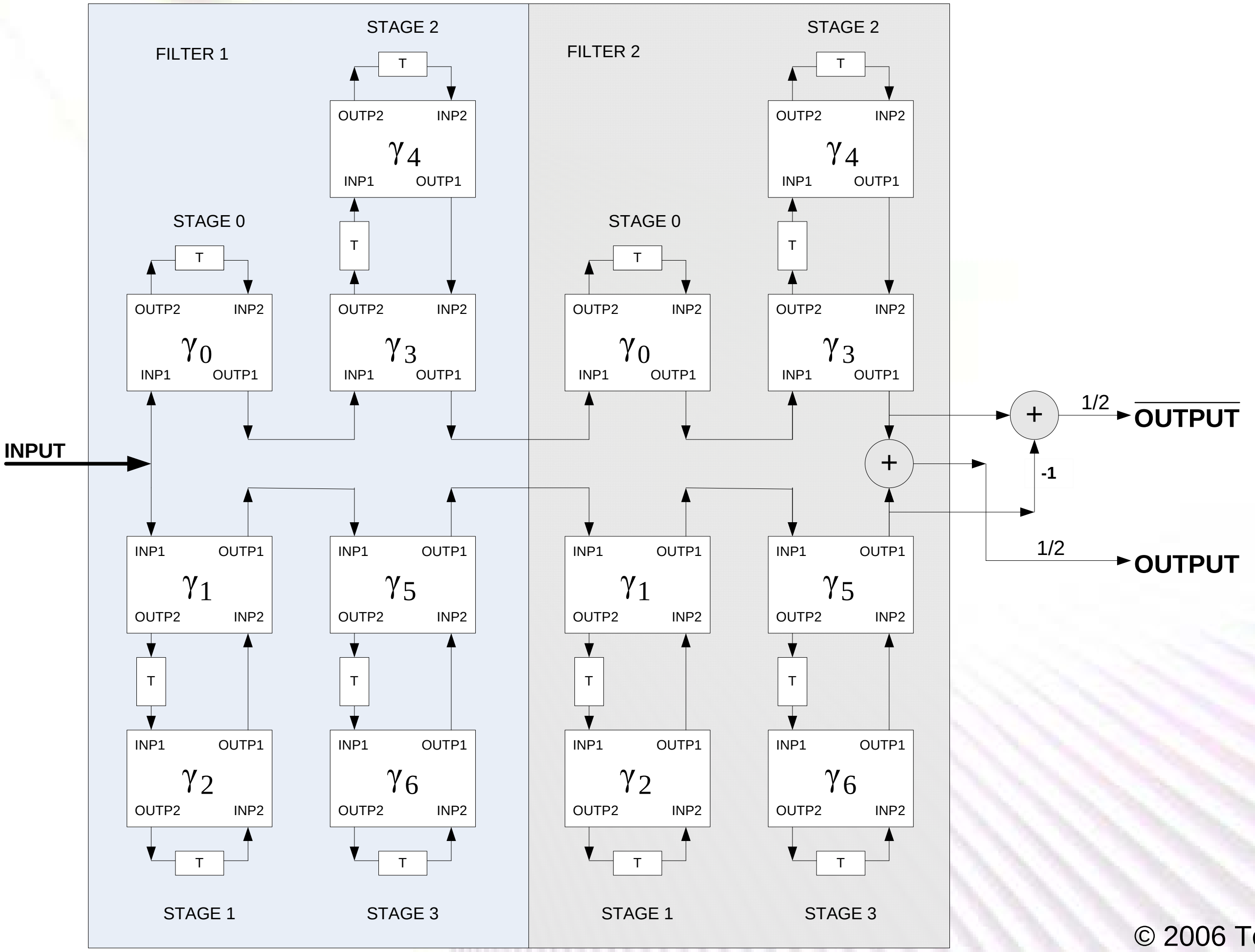
Type 4 Adaptor structure



$$P1 = INP2 - INP1$$
$$OUTP2 = \alpha * P1 - INP2$$
$$OUTP1 = OUTP2 - P1$$

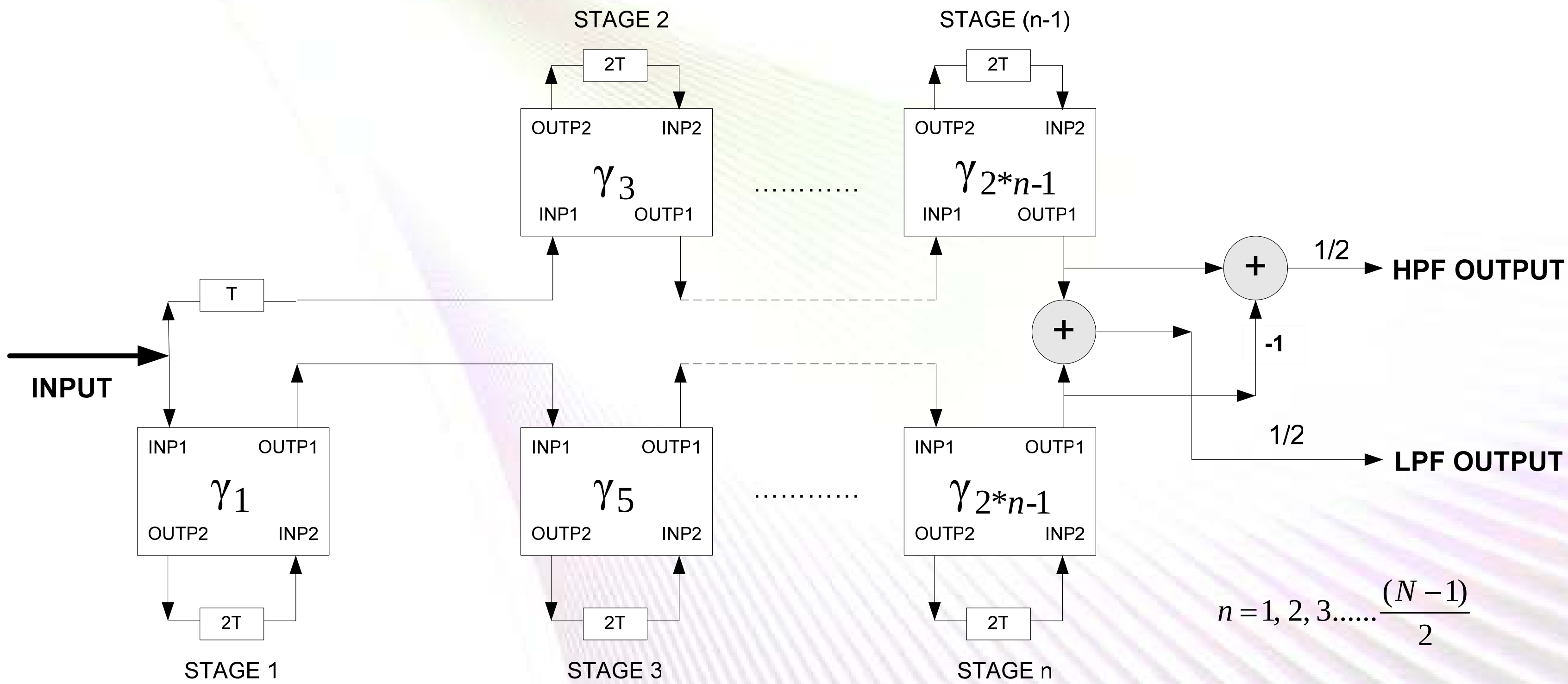
Special types of LWDF

- **Cascade of LWDF**
  - Similar to cascade of Conventional IIR filters
  - Useful when band-pass or band reject filters are desired



Special types of LWDF

- **Bi-reciprocal LWDF**
  - Easier to design
  - Lower order compared to conventional LWDF
  - Automatically gives a cut-off at ¼ the sampling frequency



$$n = 1, 2, 3 \ldots\ldots \frac{(N-1)}{2}$$

Horner's algorithm with CSD

- Reduces the number of add operations in each multiply resulting in less instruction cycles and smaller code size
- Faster execution maintaining the same level of accuracy

Multiplier

$$0.12345 = 0.000111111001_b = 0.00100000\bar{1}001_{CSD}$$

Design Equations

$$X_1 = X \cdot 2^{-3} + X$$

$$X_2 = X_1 \cdot 2^{-1} + X$$

$$X_3 = X_2 \cdot 2^{-1} + X$$

$$X_4 = X_3 \cdot 2^{-1} + X$$

$$X_5 = X_4 \cdot 2^{-1} + X$$

$$X_6 = X_5 \cdot 2^{-1} + X$$

$$\textit{Final result} = X_6 \cdot 2^{-4}$$

6 add and 12 shift instructions

With CSD

Design Equations

$$X_1 = X \cdot 2^{-3} - X$$

$$X_2 = X_1 \cdot 2^{-6} + X$$

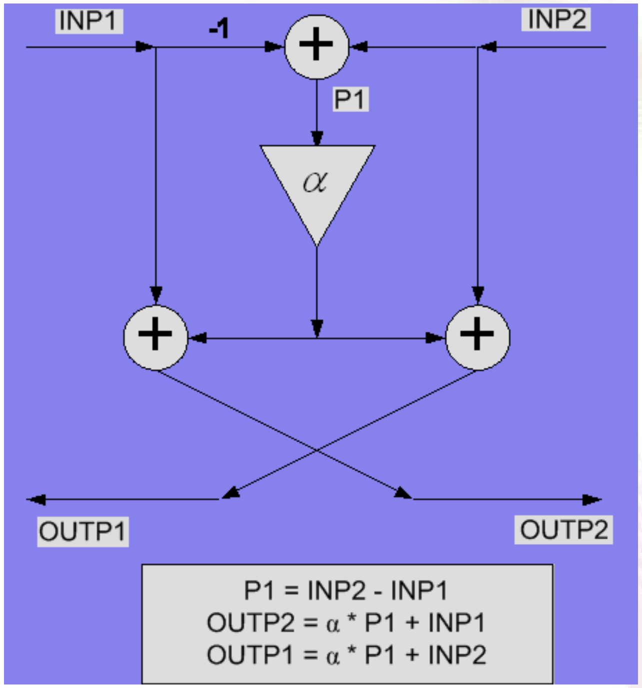
$$\textit{Final result} = X_2 \cdot 2^{-3}$$

2 add and 12 shift instructions

Reduction of 4 cycles per multiply for this multiplier!!

Horner’s algorithm for LWDF

- With Horner’s method used for multiplication the entire LWDF can be done with just shift and add operations




```
; Adaptor 0      Type 1        Alpha0=0.33228647

mov.w    R12, R11           ; R11=Input sample
sub.w    &delay0,R12        ; R12=P10
mov.w    R12,R13
rra.w    R13
rra.w    R13
rra.w    R13
rra.w    R13
add.w    R12,R13            ; X1=X*2^-4+X
rra.w    R13
rra.w    R13
add.w    R12,R13            ; X2=X1*2^-2+X
rra.w    R13
rra.w    R13
add.w    R12,R13            ; X3=X2*2^-2+X
rra.w    R13
rra.w    R13
add.w    R12,R13            ; X4=X3*2^-2+X
rra.w    R13
rra.w    R13                ; Final output=X4*2^-2
add.w    &delay0,R13        ; Design equation implementation
mov.w    R13, &delay0       ; delay0 updated with OUTP20
sub.w    R12,R13            ; R13=OUTP10
mov.w    R13,R15            ; R15=OUTP10
```

30 cycles / 54 bytes of memory

Implementing LWDF on the MSP430

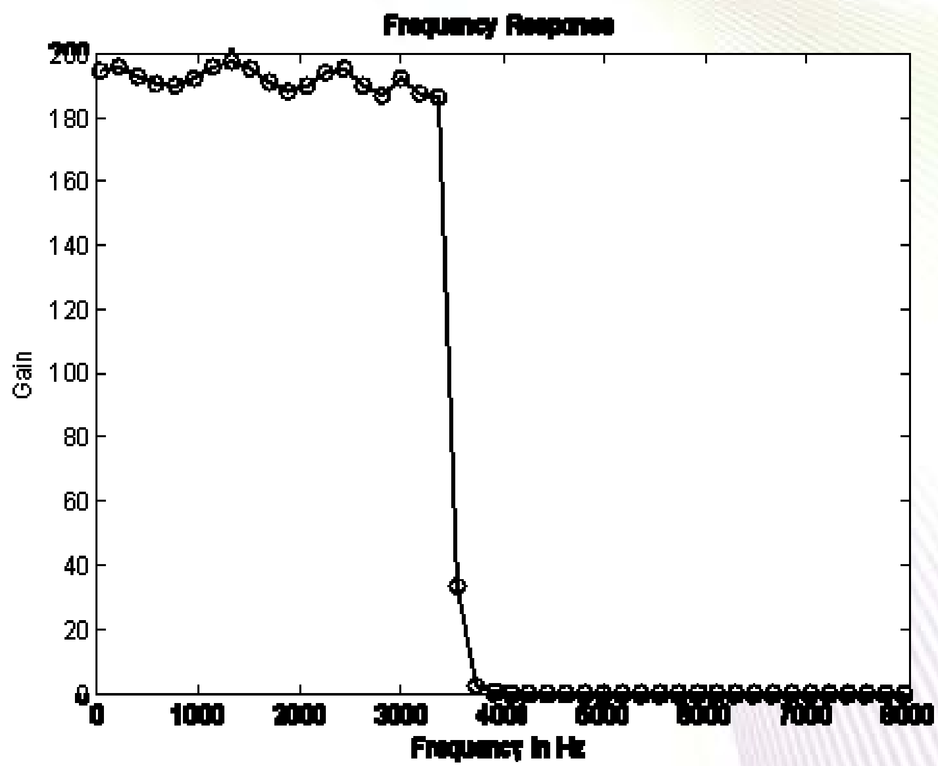
- The MSP430 supports a single cycle add/subtract and a single cycle shift
- Approximately 30-35 cycles with every increase in the order of the LWDF
- Good amount of accuracy when compared to a floating point implementation
- Exhaustive documentation to implement these filters on the MSP430 CPU
- Good performance at speech/audio sampling rates
- Real-time operation possible

Agenda

- Broad classification of Filters
- Number representations
- Fast Algorithms
- Digital Filtering on the MSP430
- Performance on the MSP430

Example 1-Implementation of LPF

Sampling frequency	=16000 Hz
Pass-band edge frequency	= 3400 Hz
Stop-band edge frequency	= 4500 Hz
Pass-band ripple	= 0.5 dB
Stop-band attenuation	= 50 dB
Filter type	= Chebyshev
Order	= 9

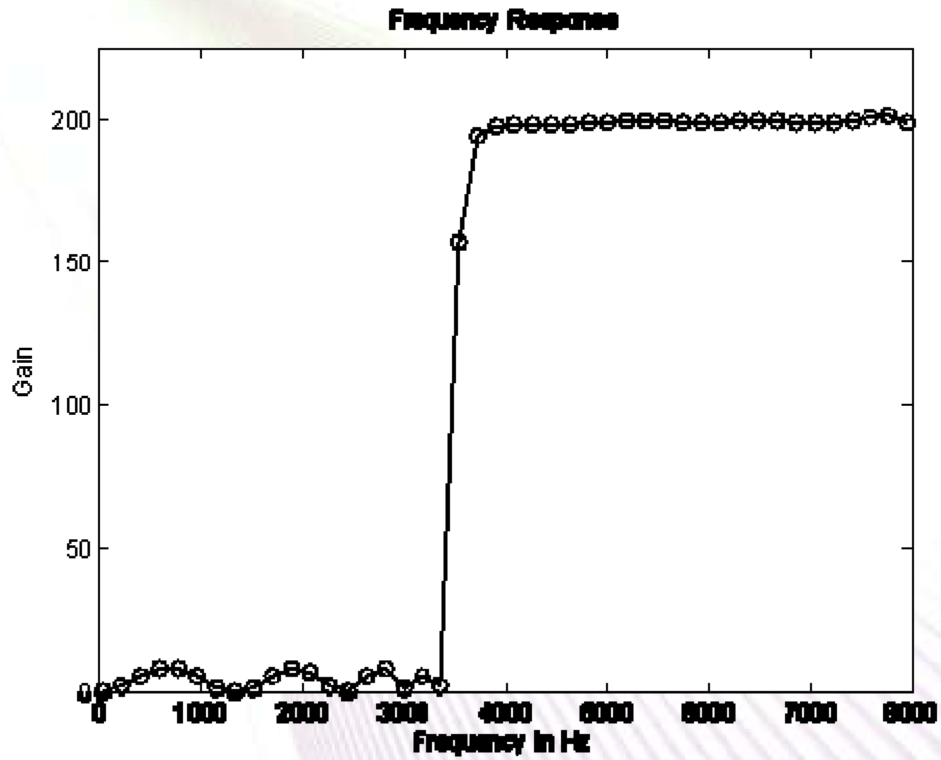


MSP430 Performance

CPU frequency	= 8 MHz
Cycles available between samples	= 500
Filter execution cycles	= 320
% CPU Utilization	= 64 %

Complementary output of the LPF

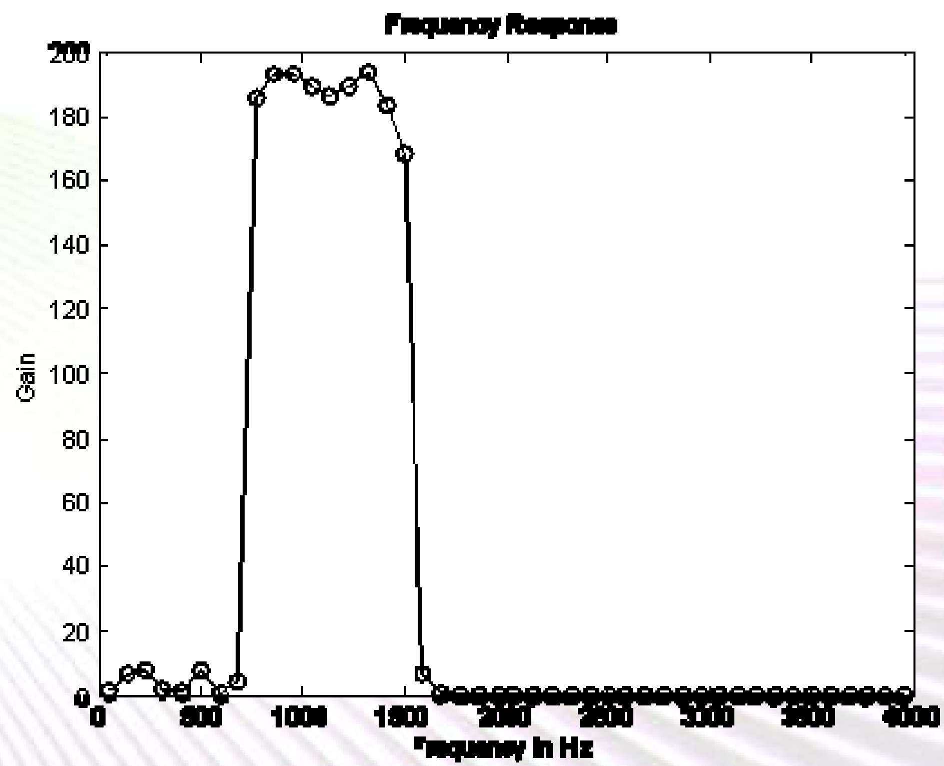
- Do you need a High pass response at the same time?
- Complementary output available with no overhead in design with just one extra instruction cycle



Example 2-Implementation of BPF

- High pass filter cascaded with a Low pass filter
- Complementary band reject output available with no overhead in design with just one extra instruction cycle

Sampling frequency	**= 8000 Hz**
Lower stop-band edge frequency	**= 700 Hz**
Lower pass-band edge frequency	**= 950 Hz**
Lower pass-band ripple	**= 0.5 dB**
Lower stop-band attenuation	**= 50 dB**
Higher pass-band edge frequency	**= 1500 Hz**
Higher stop-band edge frequency	**= 1850 Hz**
Higher pass-band ripple	**= 0.5 dB**
Higher stop-band attenuation	**= 50 dB**
Filter type	**= Elliptical**
Order	**= 14**



MSP430 Performance

CPU frequency	**= 8 MHz**
Cycles available between samples	**= 1000**
Filter execution cycles	**= 501**
% CPU Utilization	**= 50.1 %**

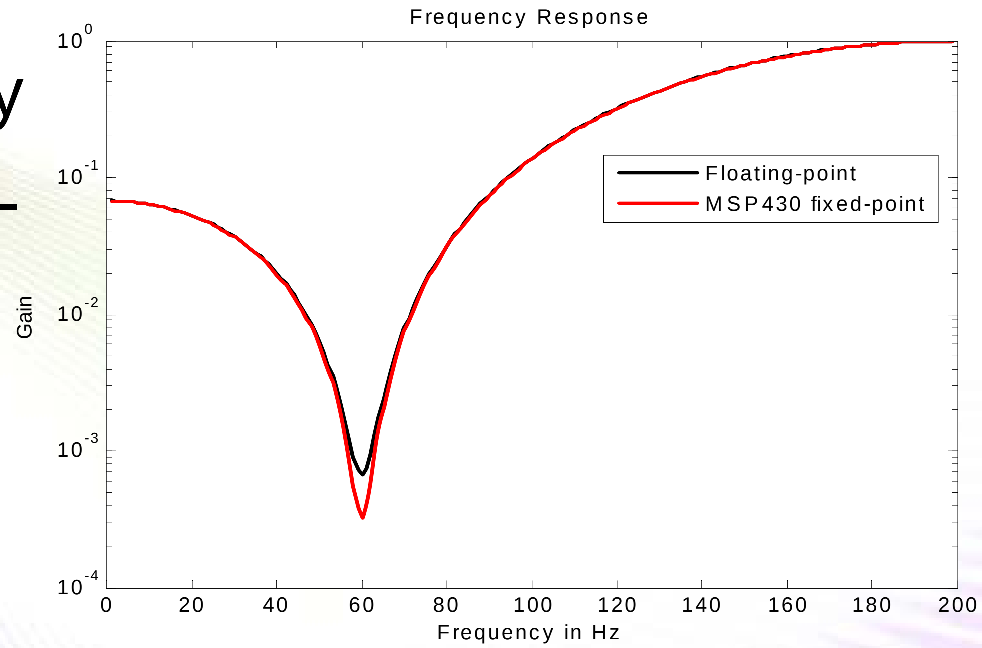
MSP430 implementation of FIR and IIR

- **Design methodology**
  - Difference equation implemented as usual
  - Use Horner's method along with CSD for all multiply operations
  - Integer-Float multiplication with Horner's method extremely accurate
  - Filter should be stable even with fixed register-widths for the coefficients

- **Accuracy and execution time efficiency**
  - Horner's method provides good accuracy
  - Each multiply takes approximately 25-30 cycles for 16-bit resolution for coefficients
  - Order chosen depending on the availability of cycles
  - At least 10-times faster than a C – library implementation

Example 3- Notch FIR filter

- Remove the 60Hz hum coming from the power lines
- A simple FIR Notch filter at 60Hz



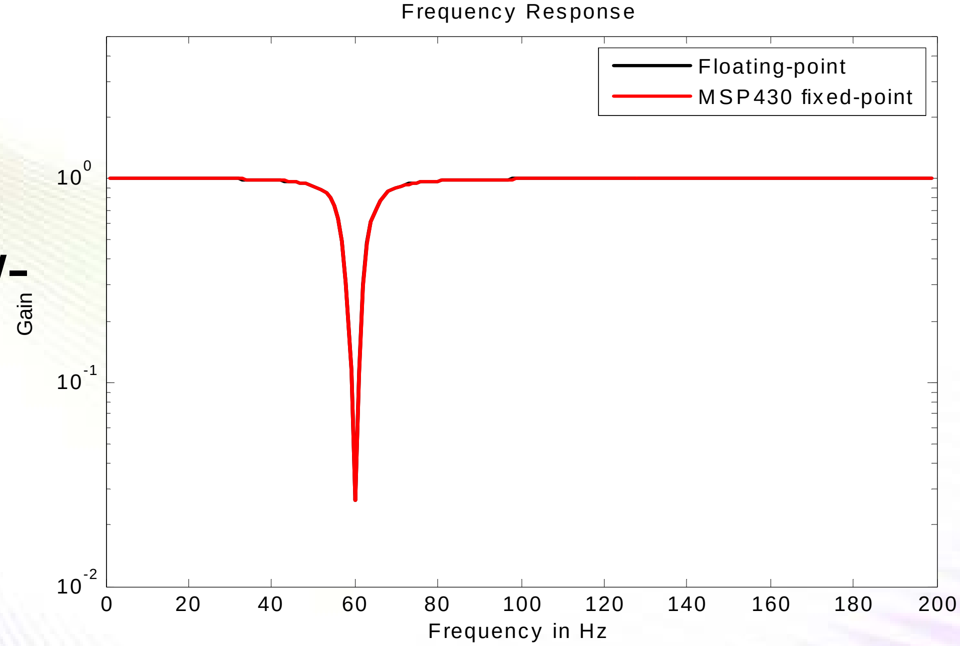
- Extremely good accuracy
- Simple solution at a Low-CPU clock

MSP430 Performance

Sampling frequency = 400Hz
CPU frequency = 32768Hz
Cycles available between samples = 82
Filter execution cycles = 52
% CPU Utilization = 63.4 %

Example 4- Notch IIR filter

- Do you need a higher roll-off? Use the IIR filter instead!!
- A stable IIR Notch filter at 60Hz with a narrow band
- As accurate as infinite precision
- Simple solution at a Low-CPU clock



MSP430 Performance

Sampling frequency = 400Hz
CPU frequency = 1.048576MHz
Cycles available between samples = 2621
Filter execution cycles = 131
% CPU Utilization = 5 %

Summary

- **Filtering on MSP430**
  - Extremely simple and efficient
  - LWDF eliminates the possibility of instability of IIR filters
  - Performance close to Floating point implementation
  - Code size is large when Horner's algorithm is used
  - Efficient MSP430 RISC architecture to boost your performance and reduce power consumption

- **Choice of Digital Filters over Analog filters**
  - Digital filters can make your design simpler and flexible
  - Better performance in addition to lower cost
  - Final cost is reduced with no external circuitry needed

References

1. Texas Instruments, MSP430 family user guides
2. Venkat, Kripasagar, *Efficient Multiplication and Division Using MSP430*, literature number SLAA329
3. Kaiser, Ulrich, "Wave Digital Filtering for TI's Sensor Signal Processor MSP430", *Texas Instruments*
4. Venkat, Kripasagar, *Wave Digital Filtering Using the MSP430*, literature number SLAA331
5. *Computer Organization*, Carl Hamacher, Zvonko Vranesic, and Safawat Zaky, 3rd Edition, McGraw Hill Publication, 1990

Thank you

IMPORTANT NOTICE

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265