TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

Implementing Digital Audio Algorithms on Constrained and Unconstrained Architectures

Rusty Allred, PhD, PE

MUSTANG TECHNOLOGY GROUP

Agenda

- Introduction
- Definitions
- Introducing Some Constrained Architectures
- Focus on the TAS3103
- Non-standard Methods of Implementing Standard Algorithms
  - Exploiting Filter Structures
  - Exploiting Dynamic Range Compression
- Summary

Introduction

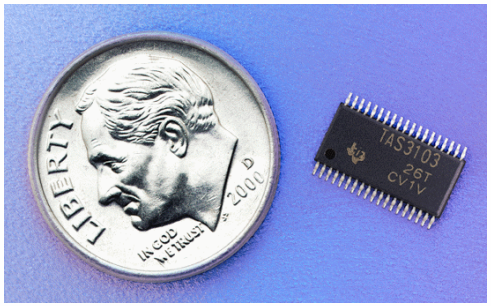
- With so many available choices of programmable architectures – microprocessors, FPGAs, Gate Arrays, and, of course DSPs – *digital processing* has come to be almost synonymous with *programmable*.
- While such devices are the mainstay of the industry, and continue to increase in capability and ease of use, the past decade has also seen the advent of a number of highly-capable, non-programmable processing resources. In some cases such devices offer quicker time to market and lower total cost than their programmable counterparts. However, getting the most of these devices requires a different mindset than is needed for implementing on programmable resources.
- The goal of this presentation is to offer the listener an array of signal processing techniques that can help get the most from non-programmable processing resources. In addition, many of these techniques can also be advantageous for implementations on programmable resources.

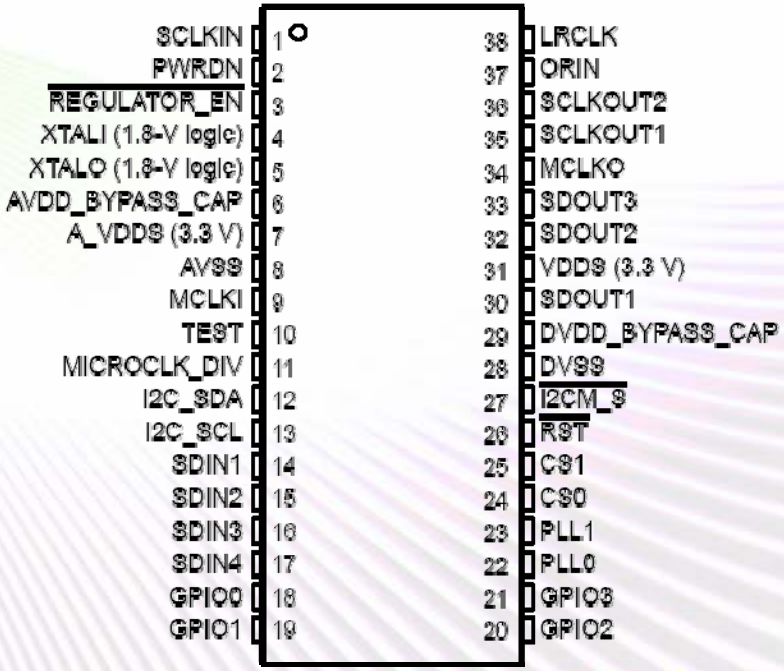
Definitions

- Programmable Architecture – DSPs, FPGAs, microprocessors, etc.
- Constrained Architecture – all devices are constrained in some ways – throughput, precision, instruction set, etc. However, for this presentation, *constrained architecture* is used to mean non-programmable.
- Configurable Architecture – many constrained architectures are configurable, meaning that, while they do not offer the near-infinite flexibility of programmable devices, they can be exploited in clever ways to offer high-performance, short time-to-market implementations.

Introducing Constrained Architectures: The TAS3103 Configurable Digital Audio Processor

- **Audio Input/Output**
  - Four Serial Audio Input Channel Pairs
  - Three Serial Audio Output Channel Pairs
  - 8-kHz to 96-kHz Sample Rates
  - 16-, 18-, 20-, 24-, and 32-Bit I/O Word Sizes Supported
- **Three Independent Monaural Processing Channels**
  - 48-bit Data Path; 28-bit Coefficients; 76-bit accumulator
  - Programmable Four Stereo Input Digital Mixer
  - 3D Effect and Reverb Structure and Filters
  - Programmable 12 Band Digital Parametric EQ
  - Programmable Digital Bass and Treble Controls
  - Programmable Digital Soft Volume Control (24 dB to -∞ dB)
  - Soft Mute
  - Programmable Loudness Compensation
  - VU Meter and Spectral Analysis I2C Output
  - Programmable Channel Delay (Up to 42 ms at 48 kHz)
  - 192-dB Dynamic Range
  - Dual Threshold Dynamic Range Compressor
- **Electrical and Physical**
  - Single 3.3-V Power Supply
  - 38-Pin TSSOP Package
  - Low Power Standby

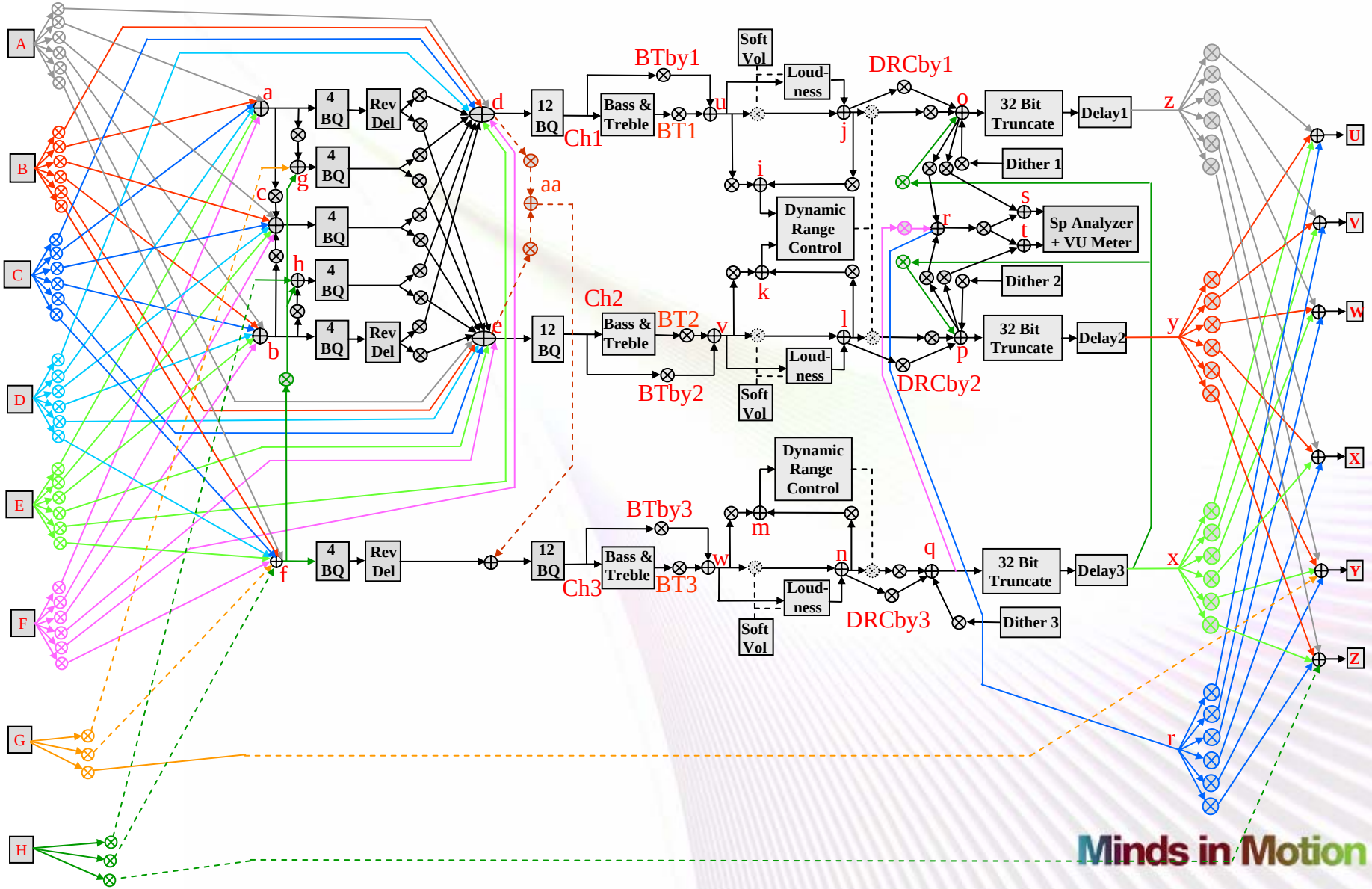




*See TI TAS3103 Data Manual

Minds in Motion

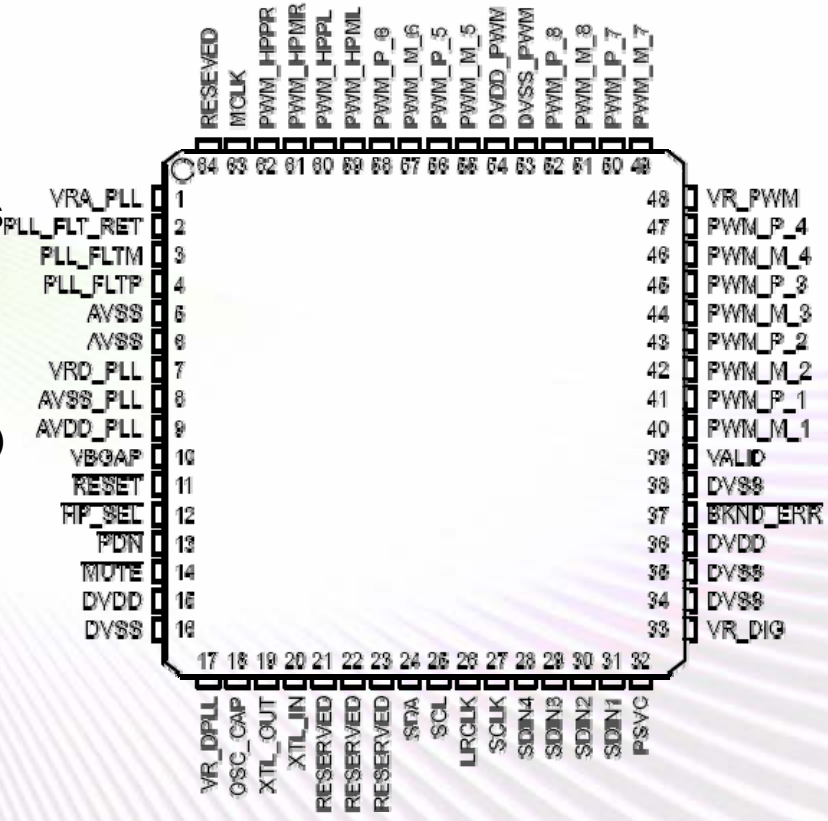
Introducing Constrained Architectures: TAS3103



Minds in Motion

Introducing Constrained Architectures: The TAS5508 Digital Audio PWM Processor

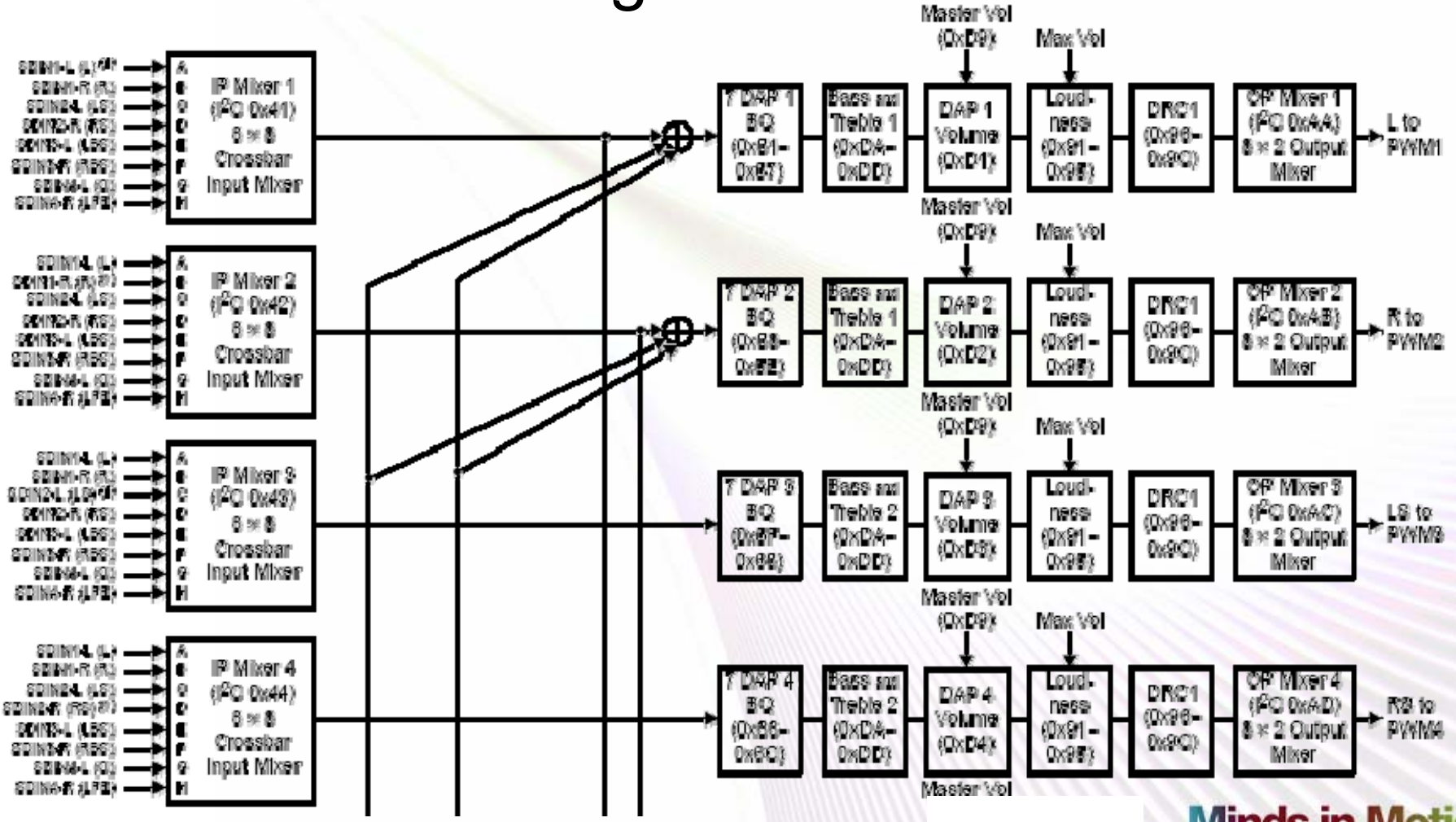
- **Audio Input/Output**
  - Four Serial Audio Input Channel Pairs
  - Eight Mono PWM Output Channels
  - 8-kHz to 96-kHz Sample Rates
  - 16-, 20-, 24-Bit I/O Word Sizes
- **Eight Independent Monaural Processing Channels**
  - 48-bit Data Path; 28-bit Coefficients; 76-bit accumulator
  - Programmable Input Digital Mixer
  - Programmable Digital Parametric EQ
  - Programmable Digital Bass and Treble Controls
  - Programmable Digital Soft Volume Control (36 dB to -109 dB)
  - Soft Mute
  - Programmable Loudness Compensation
  - 192-dB Dynamic Range
  - Dual Threshold Dynamic Range Compressors
- **Electrical and Physical**
  - Single 3.3-V Power Supply
  - 64-Pin TSSOP Package



*See TI TAS5508 Data Manual

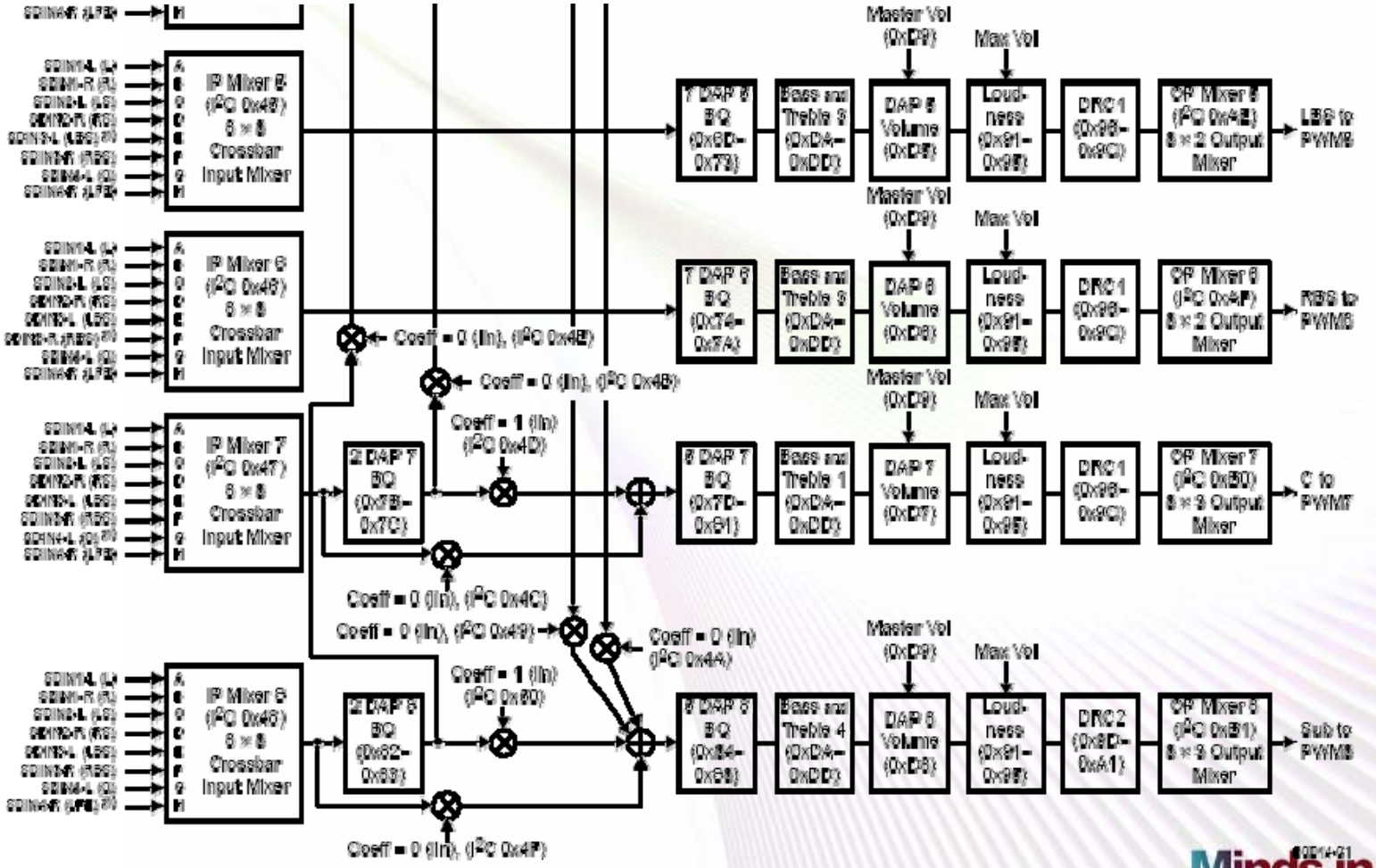
Minds in Motion

Introducing Constrained Architectures: The TAS5508 Digital Audio PWM Processor



Minds in Motion

Introducing Constrained Architectures: The TAS5508 Digital Audio PWM Processor



Minds in Motion

Introducing Constrained Architectures:
The TLV320AIC3111 Low Power Audio Codec

Product Preview

- **Audio Input/Output**
  - – Stereo Differential and Single Ended Inputs
  - – Electrolet MIC Input with Hardware AGC available
  - – Stereo HeadPhone Amp & 1W Class-D BTL Speaker Driver
  - – 8-bit SAR ADC
  - – Flexible I/O Structure

- **Control Interface**
  - – I²C Interface

- **Electrical and Physical**
  - – 1.8V Digital Power Supply
  - – 3.3-V Power Supply
  - – 32-Pin QFN Package
  - – Low Power Standby

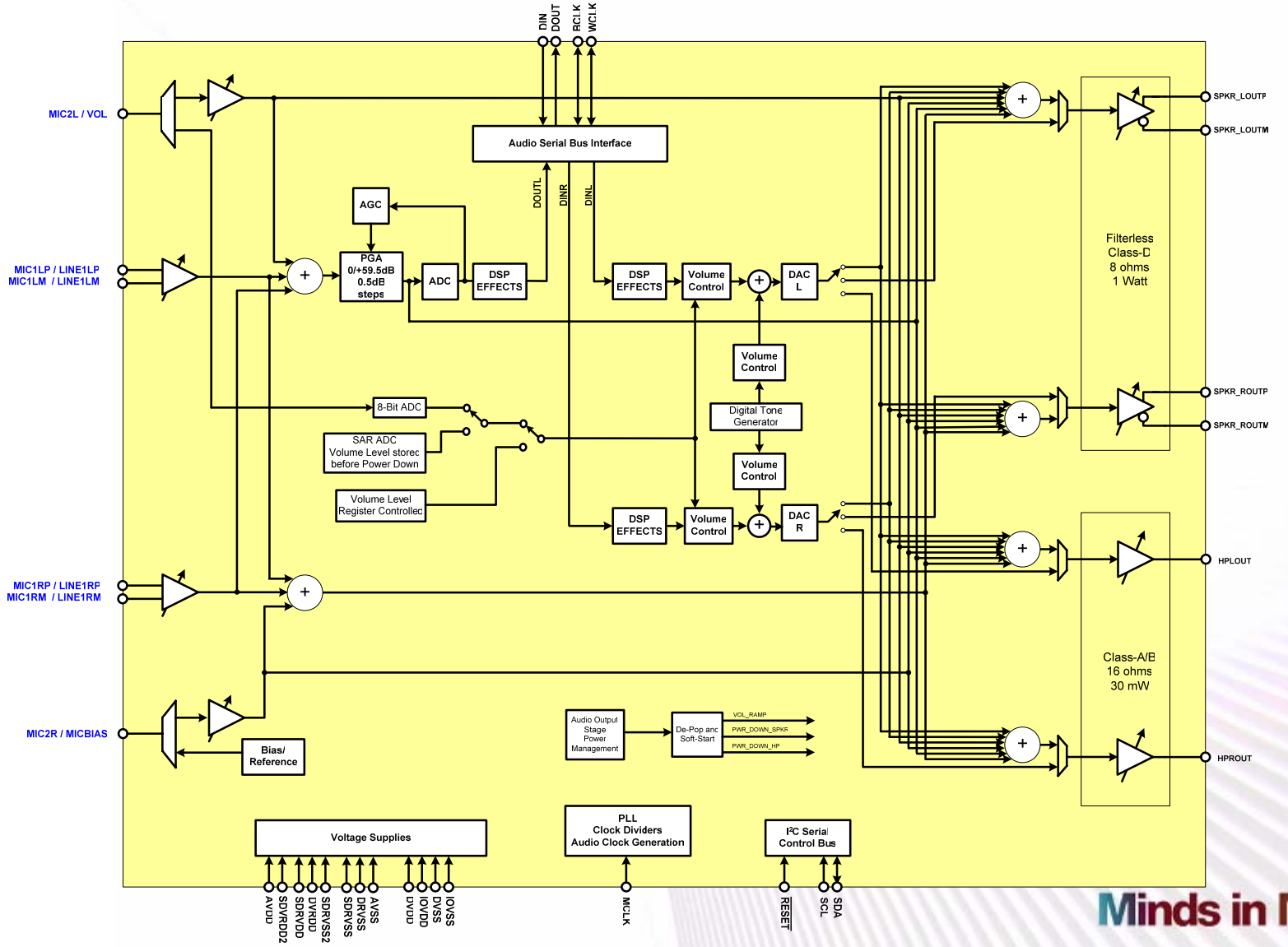
Bi-Quad Coefficient or Instruction Programmable:

- **Programmable Bi-Quad Coefficient mode**
  - – 24-bit Data Path; 16-bit Coefficients
  - – Programmable Stereo Input Digital Mixer
  - – Programmable Digital Bass and Treble Controls, 3D Effects
  - – Programmable 6 Band Digital Parametric EQ
  - – Programmable Digital Soft Volume Control (24 dB to -∞ dB), Soft Mute
  - – DAC Volume – can also be controlled by external Pin
  - – Digital Tone Generator with independent volume control
  - – Dynamic Range Compressor

- **Programmable DSP Instruction mode**
  - – Up to 80 Bi-Quads are available
  - – Acoustic Echo Cancellation
  - – A-Law / μ-Law Encoding and Decoding
  - – Adaptive algorithms possible

Product Preview

Introducing Constrained Architectures:
The TLV320AIC3111 Low Power Audio Codec



Product Preview

Product Preview

Minds in Motion

Additional Notes on Constrained Architectures

- See also:
  - TAS3108, TAS3208 – programmable digital audio processors
  - TAS5504 – configurable PWM processor
  - TAS3001, TAS3002, TAS3004 – first generation digital audio processors
- There also exist other constrained architecture devices from TI and other manufacturers.

Focus on the TAS3103

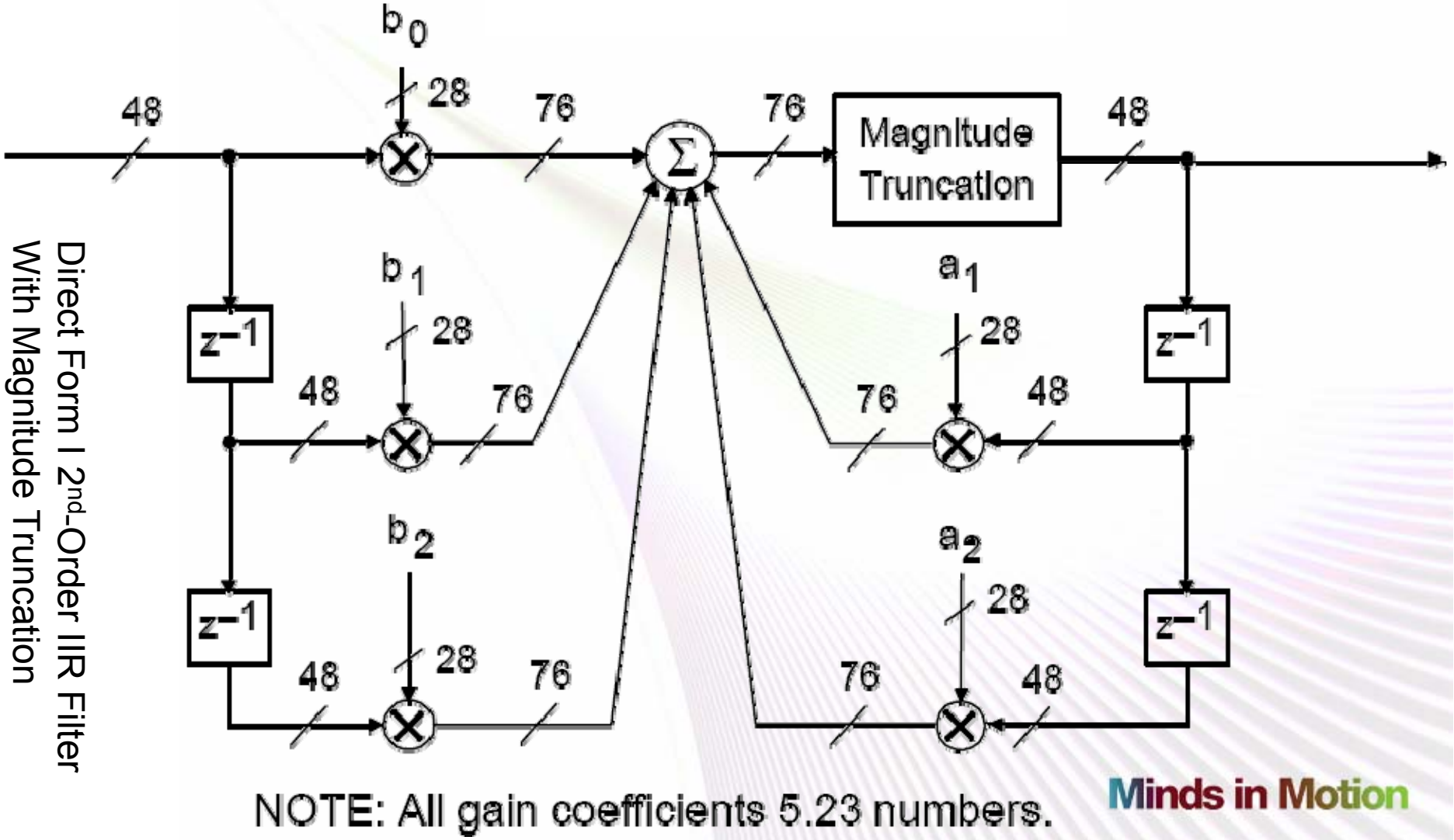
- The TAS3103 is the most versatile constrained-architecture digital audio processor, and is, therefore, an excellent case study.

- TI provides tools for configuring the device, including filter design tools, and tools for designing and setting all of the system coefficients.

Focus on the TAS3103

- But, even more functionality is available by fully understanding the features and functionality of the device.
- In particular, the device can be used to implement a wide array of standard algorithms.
- Furthermore, the techniques used to get the most from the TAS3103 are transferable to other devices, and can even be used to get the most from programmable devices.

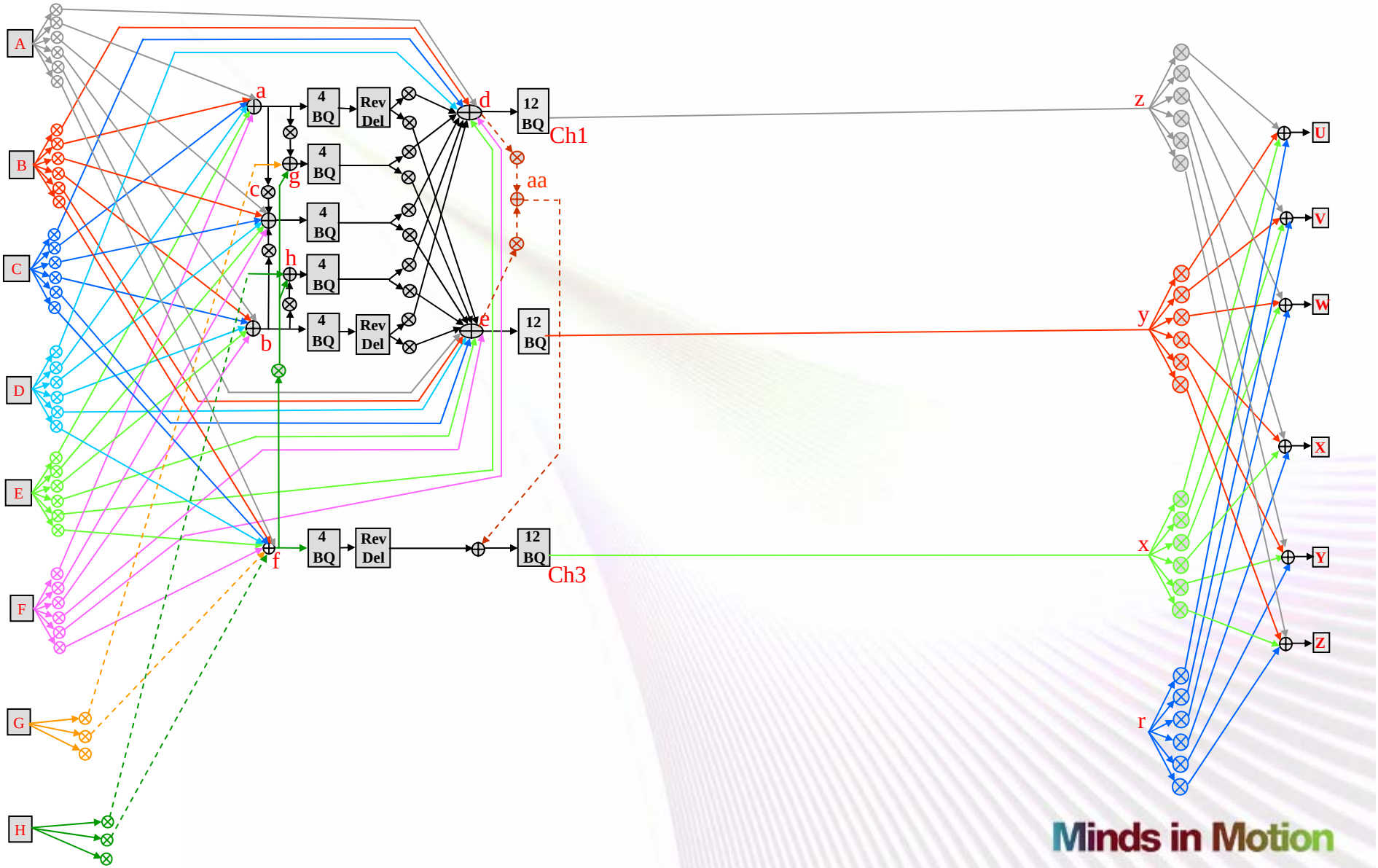
Focus on the TAS3103: The Biquad Filter



Direct Form I 2nd-Order IIR Filter
With Magnitude Truncation

NOTE: All gain coefficients 5.23 numbers.

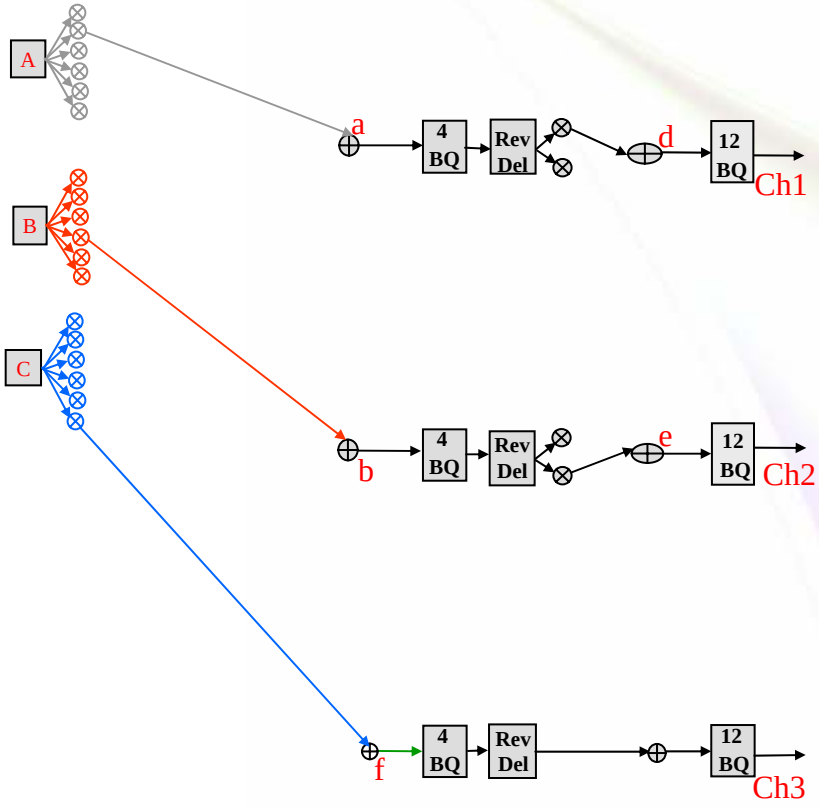
Focus on the TAS3103: The Biquad Filter



Minds in Motion

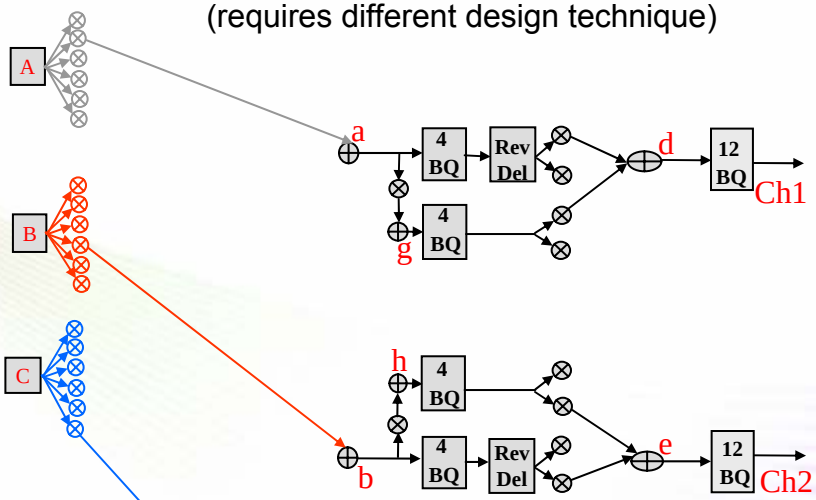
Focus on the TAS3103: The Biquad Filter

16 Biquads Per Channel

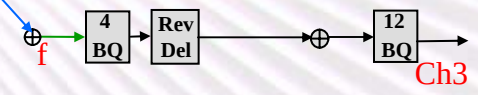


20 Biquads Per Channel (1, 2)

(requires different design technique)



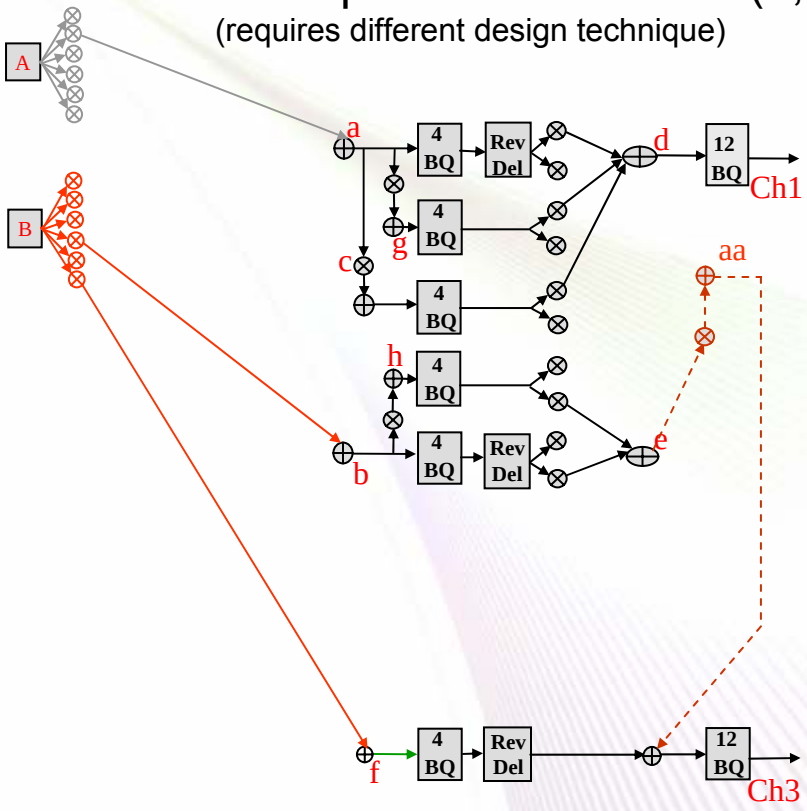
16 Biquads Per Channel (3)



Focus on the TAS3103: The Biquad Filter

24 Biquads Per Channel (1, 3)

(requires different design technique)

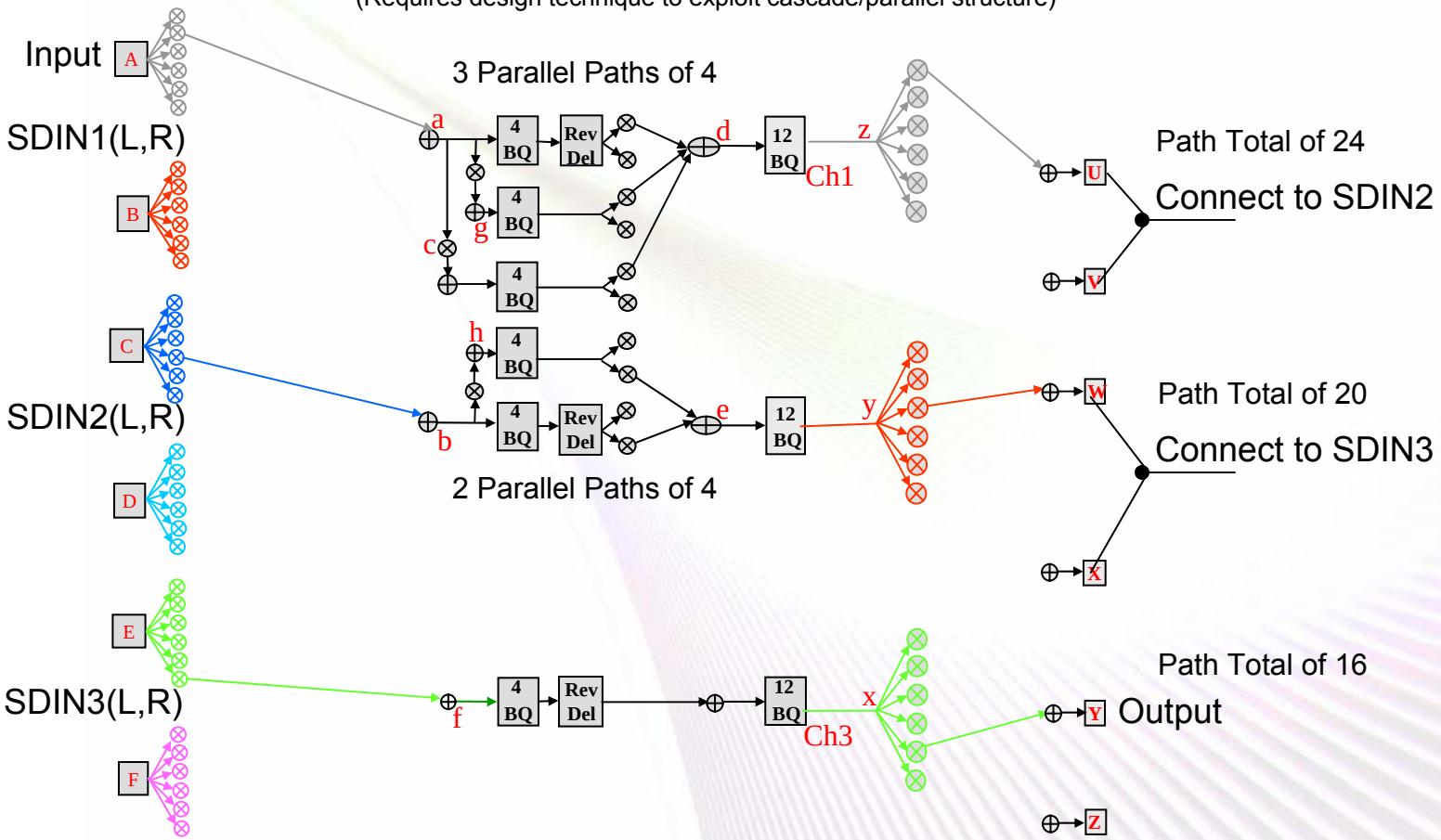


Channel 2 Processing Not Used

Focus on the TAS3103: The Biquad Filter

Utilizing all 60 Biquads for Single-Channel 120th-Order IIR or 120-Tap FIR Filter

(Requires design technique to exploit cascade/parallel structure)



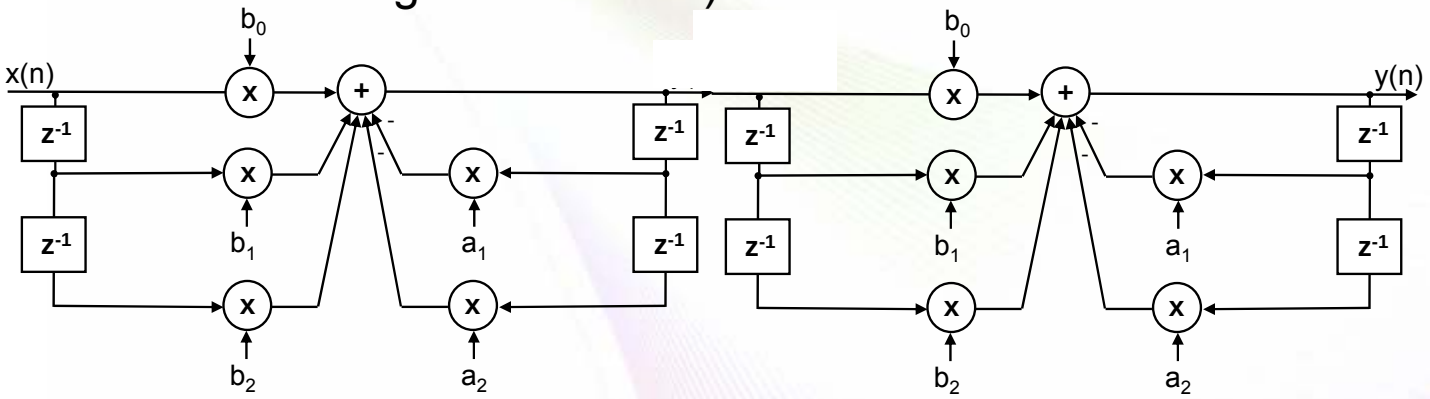
Minds in Motion

The Biquad Filter

The biquad is the most general filter because it can be used to make any other basic filter:

Zero the a coefficients to make an FIR filter.
Zero b_2 and a_2 to make a first order filter.

Cascade as many as needed to make any filter order (combine with one first order filter to get odd orders).



	Biquad 1					Biquad 2					Order	Type
Case 1	$b_0 \neq 0$	$b_1 \neq 0$	$b_2 \neq 0$	$a_1 = 0$	$a_2 = 0$	$b_0 \neq 0$	$b_1 \neq 0$	$b_2 \neq 0$	$a_1 = 0$	$a_2 = 0$	4^{th}	FIR
Case 2	$b_0 \neq 0$	$b_1 \neq 0$	$b_2 \neq 0$	$a_1 \neq 0$	$a_2 \neq 0$	$b_0 \neq 0$	$b_1 \neq 0$	$b_2 = 0$	$a_1 \neq 0$	$a_2 = 0$	3^{rd}	IIR

The Biquad Filter

Factoring higher-order filters for implementation as cascaded biquads.
The transfer function of an n[th]-order filter (n=2 for a biquad):

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_n z^{-n}}{1 + a\ z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}}$$

This can be rewritten in the following form:

$$H(z) = b_0 \cdot \frac{(z - Z_{1R} - Z_{1I} i)(z - Z_{1R} + Z_{1I} i)}{(z - P_{1R} - P_{1I} i)(z - P_{1R} + P_{1I} i)} \cdot$$

$$\frac{(z - Z_{2R} - Z_{2I} i)(z - Z_{2R} + Z_{2I} i)\cdots}{(z - P_{2R} - P_{2I} i)(z - P_{2R} + P_{2I} i)\cdots} \cdot \frac{(z - Z_{nR} - Z_{nI} i)(z - Z_{nR} + Z_{nI} i)}{(z - P_{nR} - P_{nI} i)(z - P_{nR} + P_{nI} i)}$$

The roots of the numerator equation are called zeros, while the roots of the denominator equation are called poles. In general these are complex.

The Biquad Filter

In the following equation, we have called the zeros $Z_{iR} \pm Z_{il}$ and poles $P_{iR} \pm P_{il}$

$$H(z) = b_0 \cdot \frac{(z - Z_{1R} - Z_{1I}i)(z - Z_{1R} + Z_{1I}i)}{(z - P_{1R} - P_{1I}i)(z - P_{1R} + P_{1I}i)} \cdot$$
$$\frac{(z - Z_{2R} - Z_{2I}i)(z - Z_{2R} + Z_{2I}i)\cdots}{(z - P_{2R} - P_{2I}i)(z - P_{2R} + P_{2I}i)\cdots} \cdot \frac{(z - Z_{nR} - Z_{nI}i)(z - Z_{nR} + Z_{nI}i)}{(z - P_{nR} - P_{nI}i)(z - P_{nR} + P_{nI}i)}$$

where the subscript R denotes the real part, and I denotes the imaginary part of the complex value. Also, note that complex poles and zeros occur only in *complex conjugate pairs*. This means that the complex zero (or pole) $Z_{iR}+Z_{il}$ will never occur unless $Z_{iR}-Z_{il}$ also occurs in the same filter polynomial.

The sole exception is the case where the imaginary portion is identically 0, that is, when the zero (or pole) is real. Therefore, odd-order filters will always contain at least one real zero and one real pole. It is possible that there will be others as well.

The Biquad Filter: Odd Orders

An odd-order filter, therefore, will have the following form:

$$H(z) = b_0 \cdot \frac{(z - Z_{1R})}{(z - P_{1R})} \cdot \frac{(z - Z_{2R} - Z_{2I}i)(z - Z_{2R} + Z_{2I}i)\cdots}{(z - P_{2R} - P_{2I}i)(z - P_{2R} + P_{2I}i)\cdots} \cdot \frac{(z - Z_{nR} - Z_{nI}i)(z - Z_{nR} + Z_{nI}i)}{(z - P_{nR} - P_{nI}i)(z - P_{nR} + P_{nI}i)}$$

Although a full biquad is not required to implement the first-order portion of this filter, a biquad can be used by zeroing the b_2 and a_2 coefficients:

$$H(z) = \frac{(z - Z_{1R})}{(z - P_{1R})} = \frac{1 - Z_{1R}z^{-1}}{1 - P_{1R}z^{-1}}$$

The coefficients for a biquad implementing this single filter would be:

$b_{01} = 1^{*}$; $b_{11} = -Z_{1R}$; $b_{21} = 0$; $a_{11} = -P_{1R}$; $a_{21} = 0$

*Note that this is the b_0 of the new filter, not the b_0 from the equation above. However, the b_0 from the equation above does need to be implemented somewhere in the cascade of biquads. Usually this is done by applying it where the scaling works best, or by applying roots of it in multiple locations. To apply it to this first-order filter, multiply the entire numerator by b_0, changing the numerator coefficients to: $b_{01} = b_0$; $b_{11} = -Z_{1R} * b_0$.

The Biquad Filter: Real Roots

When there are two real roots, the equation changes in the following manner:

$$H(z) = b_0 \cdot \frac{(z - Z_{1R})(z - Z_{2R})}{(z - P_{1R})(z - P_{2R})} \cdot$$

$$\frac{(z - Z_{3R} - Z_{3I}i)(z - Z_{3R} + Z_{3I}i)\cdots}{(z - P_{3R} - P_{3I}i)(z - P_{3R} + P_{3I}i)\cdots} \cdot \frac{(z - Z_{nR} - Z_{nI}i)(z - Z_{nR} + Z_{nI}i)}{(z - P_{nR} - P_{nI}i)(z - P_{nR} + P_{nI}i)}$$

Note that there can be multiple real roots for either odd- or even-order filters. There is no requirement that a pair of real poles will occur along with a pair of real zeros. The only guarantees are that odd-order filters will have at least one real zero and one real pole, and that all complex roots will occur in complex conjugate pairs. If there are two real zeros and no real poles, the zeros can be combined as shown below, and implemented along with a pair of complex poles.

$$H(z) = b_0 \cdot \frac{(z - Z_{1R})(z - Z_{2R})}{(z - P_{1R})(z - P_{2R})} = \frac{b_0 - b_0(Z_{1R} + Z_{2R})z^{-1} + b_0 Z_{1R} Z_{2R} z^{-2}}{1 - (P_{1R} + P_{2R})z^{-1} + P_{1R} P_{2R} z^{-2}}$$

The Biquad Filter: Complex Conjugate Pairs

The remaining coefficients will be complex conjugate pairs:

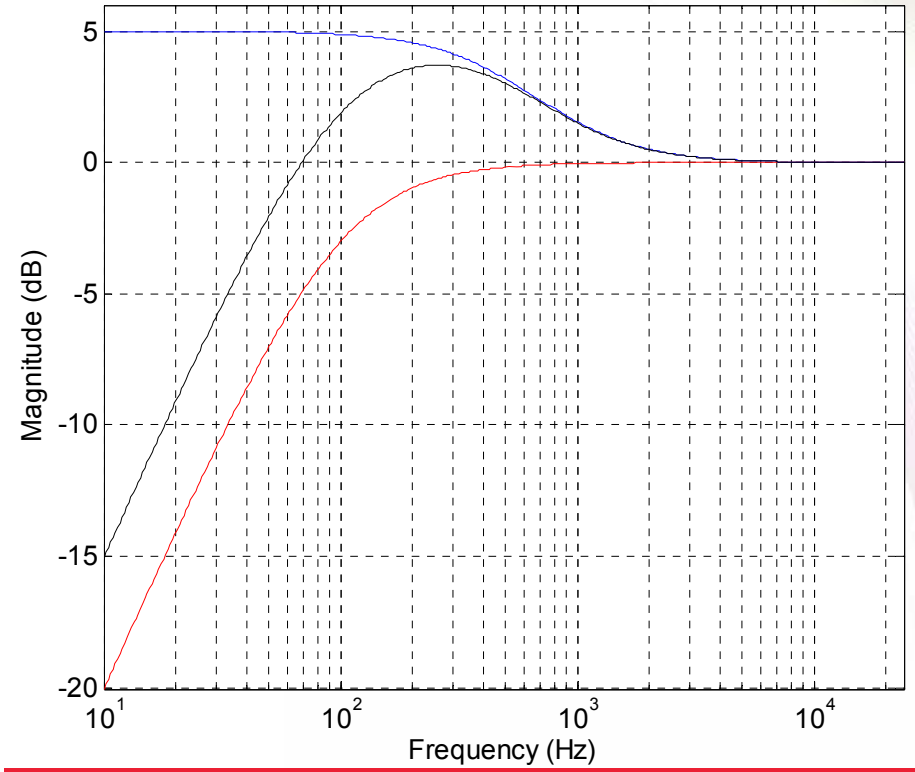
$$b_0 \cdot \frac{(z - Z_{1R} - Z_{1I}i)(z - Z_{1R} + Z_{1I}i)}{(z - P_{1R} - P_{1I}i)(z - P_{1R} + P_{1I}i)} = \frac{b_0 - 2b_0 Z_{1R} z^{-1} + b_0 (Z_{1R}^2 + Z_{1I}^2) z^{-2}}{1 - 2P_{1R} z^{-1} + (P_{1R}^2 + P_{1I}^2) z^{-2}}$$

In essence, filters of any order can be implemented by factoring the filter and combining the poles and zeros into second-order sections, each of which is implemented as a biquad. Cascading biquads results in multiplication of the associated transfer functions.

The Biquad Filter: Examples – First Order

Consider a first-order Butterworth high-pass filter with a 100 Hz cut-off and designed for a 48 kHz sample rate. The plot is shown in red.

Consider also a 5 dB first-order shelf filter with a cut-off at 500 Hz, also designed for a 48 kHz sample rate. This filter is shown in blue in the plot.



Now, if the two are combined as described previously, the resulting second-order filter is shown in black in the plot.

B = [0.99349748134078 -0.99349748134078]
A = [1.0 -0.98699496268155]

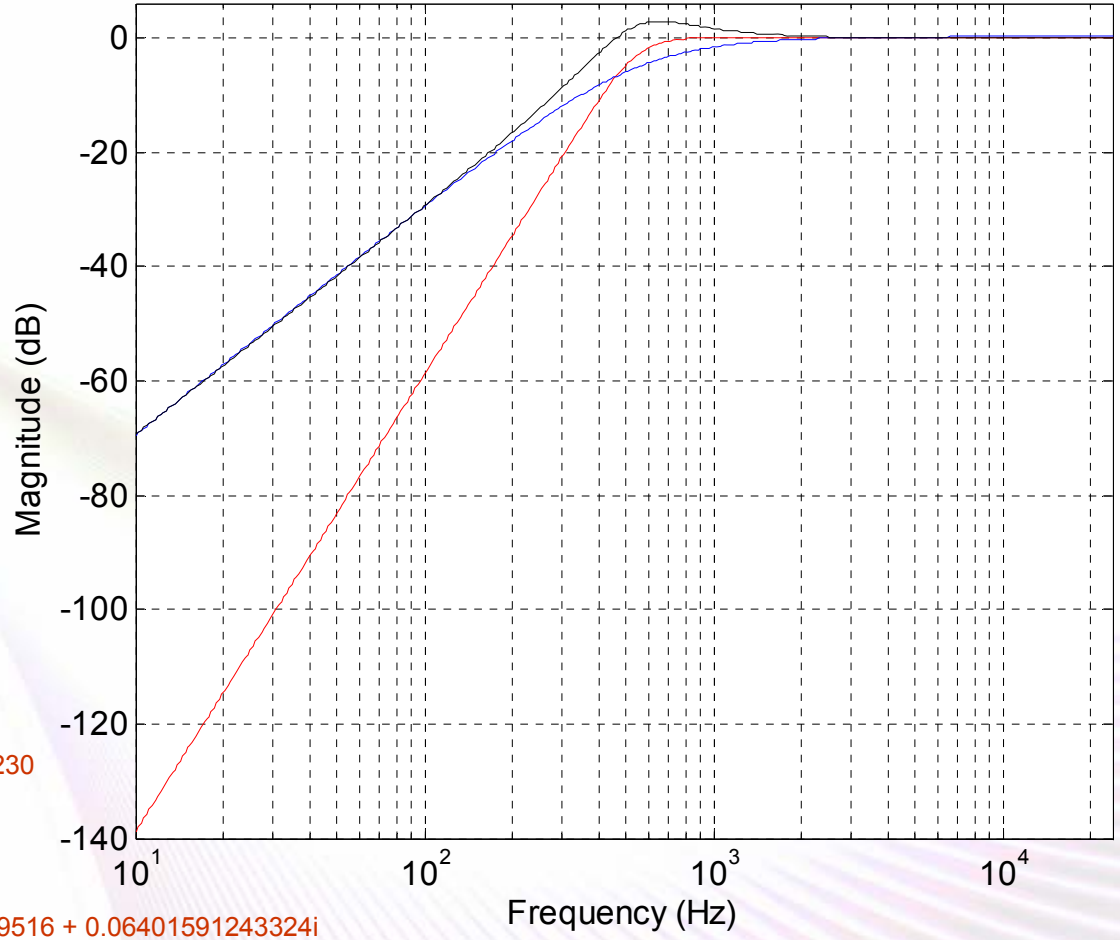
B = [1.02467059808085 -0.91193160991121]
A = [1.0 -0.93660220799206]

B = [1.01800765839728 -1.92400941599910 0.90600175760182]
A = [1.0 -1.92359717067361 0.92442166132458]

Minds in Motion

The Biquad Filter: Examples – Fourth Order

Consider a fourth-order Butterworth high-pass filter with a 500 Hz cut-off designed for a 44.1 kHz sample rate. It can be factored into two biquads. The original magnitude response is shown in red; the two resulting biquads in blue and black.



B = [0.91110246841372 -3.64440987365487 5.46661481048230
-3.64440987365487 0.91110246841372]
A = [1.0 -3.81386538359704 5.45872379150560
-3.47494261156512 0.83010770795173]

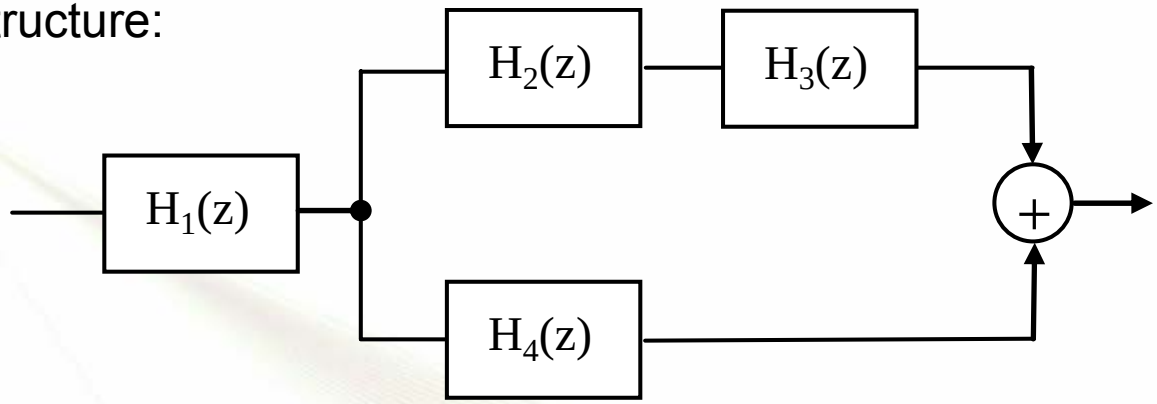
Zeros:	Poles:
1.00022566526664	0.97101464699516 + 0.06401591243324i
0.99977439304055	0.97101464699516 - 0.06401591243324
0.99999997084641 - 0.00022563610571i	0.93591804480336 + 0.02555784867173i
0.99999997084641 + 0.00022563610571i	0.93591804480336 - 0.02555784867173i

Resulting biquads:
B1 = [0.95451687696641 -1.90903369827763 0.95451686990725]
A1 = [1.00000000000000 -1.87183608960671 0.87659579021726]
B2 = [0.95451687696641 -1.90903380958802 0.95451688402558]
A2 = [1.00000000000000 -1.94202929399032 0.94696748172380]

Minds in Motion

The Biquad Filter: Other Structures

Other filter structures can also be converted to cascaded biquads. For example, consider the following structure:



This structure has the following overall transfer function: $H_1(z)\left[H_2(z)H_3(z)+H_4(z)\right]$

Assuming H1(z) and H4(z) are first order and H2(z) and H3(z) are second order, this transfer function can be rewritten as follows:

$$\frac{B_{10}+B_{11}z^{-1}}{1+A_{11}z^{-1}}\cdot\left[\frac{B_{20}+B_{21}z^{-1}+B_{22}z^{-2}}{1+A_{21}z^{-1}+A_{22}z^{-2}}\cdot\frac{B_{30}+B_{31}z^{-1}+B_{32}z^{-2}}{1+A_{31}z^{-1}+A_{32}z^{-2}}+\frac{B_{40}+B_{41}z^{-1}}{1+A_{41}z^{-1}}\right]$$

Other Structures: Example

H1(z): first-order Butterworth high-pass filter with 80 Hz cut-off. Coefficients:
B1 = [0.99479123765938 -0.99479123765938]; A1 = [1.0 -0.98958247531875]

H2(z): second-order Butterworth high-pass, 300 Hz cut-off. Coefficients:
B2 = [0.97261389849984 -1.94522779699969 0.97261389849984]; A2 = [1.0 -1.94447765776709 0.94597793623228]

H3(z): second-order Butterworth low-pass, 3 kHz cut-off. Coefficients:
B3 = [0.02995458220809 0.05990916441618 0.02995458220809]; A3 = [1.0 -1.45424358625159 0.57406191508395]

H4(z): first-order Butterworth low-pass, 5 kHz cut-off. Coefficients:
B4 = [0.25342728698435 0.25342728698435]; A4 = [1.0 -0.49314542603130]

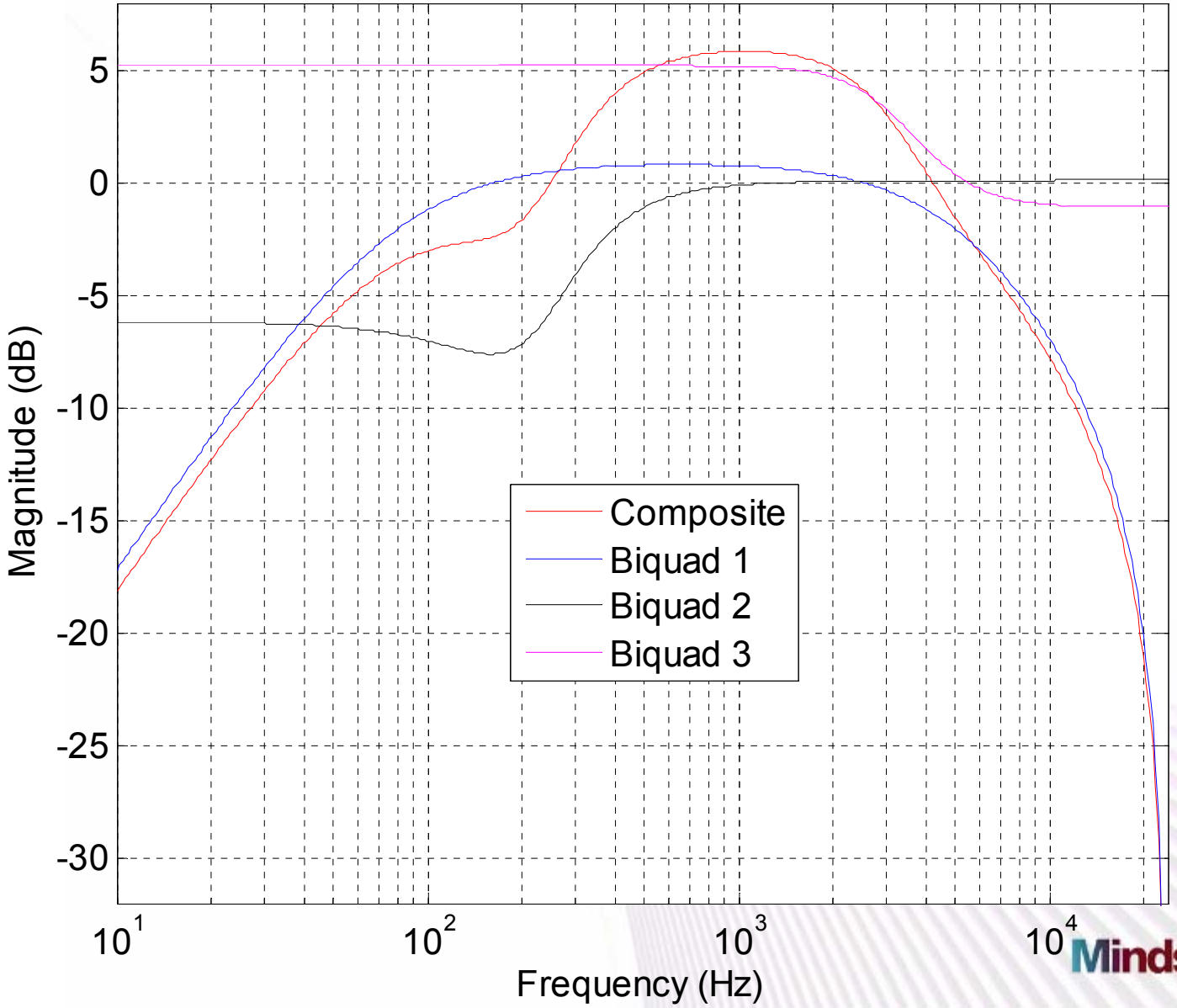
With substitution and some algebra we find the overall coefficients of this filter:
B=[0.28108973410751 -0.90011731939953 0.80032820573604 0.31515805072053 -0.95880370670137 0.58495926867901 -0.12261423314218]
A=[1.0 -4.88144914536874 9.87517090253539 -10.59711698022950 6.35966205774405 -2.02127862265964 0.26501273714634]

Factoring, we find the poles and zeros:

Zeros:	Poles:
-1.00000000000000	0.98958247534274
0.99999999999696	0.49314542603125
0.98594396121424 + 0.02327116735616i	0.97223882887180 + 0.02701103189409i
0.98594396121424 - 0.02327116735616i	0.97223882887180 - 0.02701103189409i
0.61517698021458 + 0.26465822467728i	0.72712179312558 + 0.21296904245800i
0.61517698021458 - 0.26465822467728i	0.72712179312558 - 0.21296904245800i

Combining into second order sections, and applying the original b_0 to the first filter
B1 = [0.28108973410751 0.00000000000085 -0.28108973410666]
A1 = [1.0 -1.48272790137399 0.48800807139595]
B2 = [1.0 -1.97188792242849 0.97262704188495]
A2 = [1.0 -1.94447765774359 0.94597793620998]
B3 = [1.0 -1.23035396042916 0.44848669287526]
A3 = [1.0 -1.45424358625116 0.57406191508364]

Other Structures: Example



Other Biquad Tricks: FIR

We have already mentioned that FIR filters can be implemented on biquads by zeroing the denominator coefficients. Cascading enough of these, such as the 60 available in the TAS3103, will allow implementing modest-sized FIR filters. For higher orders, order reduction techniques can also be used to implement IIR filters that retain the characteristics of FIR filters (magnitude response, linear phase response, etc.).

The next few charts follow V Sreeram and P Agathoklis, "Design of Linear-Phase IIR Filters via Impulse-Response Grammians," IEEE Transactions on Signal Processing,Vol. 40, No. 2, February 1992, pp 389 - 394.

Other Biquad Tricks: FIR

An FIR filter with coefficients $\begin{bmatrix} h_0 & h_1 & \cdots & h_N \end{bmatrix}$ can be represented in state space by the following formulation:

$$x(k+1) = Ax + bu(k)$$
$$y(k) = cx(k) + du(k)$$

where

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$$
$$c = \begin{bmatrix} h_1 & h_2 & \cdots & h_N \end{bmatrix}$$
$$d = h_0$$

To reduce this system, and implement it as an IIR, perform the following steps:

1) Compute the impulse response grammian (notice that it is constructed from the FIR coefficients)

$$G = \sum_{k=0}^{\infty} \begin{bmatrix} h_{k+1}^2 & h_{k+1}h_{k+2} & \cdots & h_{k+1}h_{k+N} \\ h_{k+1}h_{k+2} & h_{k+2}^2 & \vdots & h_{k+2}h_{k+N} \\ \vdots & \vdots & \vdots & \vdots \\ h_{k+1}h_{k+N} & h_{k+2}h_{k+N} & \cdots & h_{k+N}^2 \end{bmatrix}$$

Other Biquad Tricks: FIR

2) Compute the eigenvalues, V, and the eigenvectors, T, of G. Order the eigenvectors in decreasing eigenvalue order. (TTGT is a diagonal matrix of the eigenvalues with the eigenvalues sorted largest on top, smallest on bottom.)

3) Compute the following entities: $A_i = T^{-1}AT$ $\quad c_i = cL$

$b_i = L^{-1}b$ $\quad d_i = h_0$

4) Partition the above system according to the desired order of the new system:

$$A_r = \begin{bmatrix} A_{i11} & A_{i12} \\ A_{i21} & A_{i22} \end{bmatrix} \qquad b_r = \begin{bmatrix} b_{i1} \\ b_{i2} \end{bmatrix}$$

$$d_r = h_0 \qquad c_r = \begin{bmatrix} c_{i1} & c_{i2} \end{bmatrix}$$

5) Now use the following equation to convert this to an IIR filter:

$$H(z) = c_r\left(zI - A_r\right)^{-1}b_r + d_r$$

Other Biquad Tricks: FIR Example

Caution: Most values shown in reduced precision to save space

FIR Filter: [-0.079522992311770 -0.099883450674512 -0.148074492628409 -0.154866077916989 0.524276046519238
0.524276046519238 -0.154866077916989 -0.148074492628409 -0.099883450674512 -0.079522992311770]

A =

0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0

b' = [1 0 0 0 0 0 0 0 0]

c = [-0.0999 -0.1481 -0.1549 0.5243 0.5243 -0.1549 -0.1481 -0.0999 -0.0795]

d = -0.0795

Reduce Order, 9 to 6:

G =

0.6678	0.1959	-0.2749	-0.2237	-0.1006	0.0112	0.0419	0.0218	0.0079
0.1959	0.6579	0.1811	-0.2904	-0.1713	-0.0482	-0.0043	0.0271	0.0118
-0.2749	0.1811	0.6359	0.1581	-0.2128	-0.0937	-0.0711	-0.0262	0.0123
-0.2237	-0.2904	0.1581	0.6119	0.2393	-0.1316	-0.1177	-0.0941	-0.0417
-0.1006	-0.1713	-0.2128	0.2393	0.3371	-0.0355	-0.0504	-0.0401	-0.0417
0.0112	-0.0482	-0.0937	-0.1316	-0.0355	0.0622	0.0457	0.0272	0.0123
0.0419	-0.0043	-0.0711	-0.1177	-0.0504	0.0457	0.0382	0.0227	0.0118
0.0218	0.0271	-0.0262	-0.0941	-0.0401	0.0272	0.0227	0.0163	0.0079
0.0079	0.0118	0.0123	-0.0417	-0.0417	0.0123	0.0118	0.0079	0.0063

T =

0.0102	0.0087	0.0161	-0.1279	-0.2336	-0.4621	-0.5838	-0.3082	0.5284
-0.0099	-0.0410	-0.0515	-0.1188	0.2812	0.5823	-0.3506	0.4535	0.4844
-0.0086	0.0272	0.1002	-0.2250	-0.4821	-0.2977	-0.0804	0.7616	-0.1758
0.0082	0.0388	-0.1002	-0.2268	0.4816	-0.1353	-0.5762	-0.0015	-0.5955
0.0935	0.0364	0.1596	-0.2704	-0.5673	0.5746	-0.2405	-0.3246	-0.2760
-0.1139	-0.3851	-0.4507	-0.7304	0.0255	-0.0463	0.2891	-0.0972	0.0791
-0.1891	0.2314	0.7625	-0.4565	0.2605	-0.0608	0.1884	-0.0613	0.1005
0.2649	0.8554	-0.3631	-0.2083	0.0057	0.0003	0.1270	-0.0054	0.0814
0.9338	-0.2471	0.1876	-0.0953	0.1079	-0.0659	0.0684	0.0250	0.0375

V =

[0.000000756419523;
0.000007811070486;
0.000046768042865;
0.01137487482936;
0.029633955777272;
0.311186325996652;
0.460681649954508;
0.963653049925580;
1.257094671901085]

Other Biquad Tricks: FIR Example

A_i = [0.2089 0.9329 -0.0782 -0.1673 0.1750 -0.0881 0.0955 0.0393 0.0526
-0.2903 -0.1155 0.5744 -0.4189 0.4619 -0.2659 0.2796 0.1058 0.1534
-0.0100 -0.2347 -0.7926 -0.3273 0.3470 -0.1818 0.1952 0.0783 0.1074
-0.0014 0.0110 -0.0210 0.8002 0.2915 -0.3579 0.3146 0.0570 0.1660
-0.0009 0.0075 -0.0138 -0.1806 -0.7124 -0.4751 0.3769 0.0278 0.1911
-0.0001 0.0013 -0.0022 -0.0684 0.1466 -0.5038 -0.5706 -0.3971 -0.1523
-0.0001 0.0012 -0.0020 -0.0494 0.0956 0.4690 0.4354 -0.4872 -0.0263
-0.0000 0.0003 -0.0005 -0.0062 0.0049 0.2257 -0.3369 0.2427 0.8256
0.0000 -0.0004 0.0007 0.0158 -0.0293 -0.0758 0.0159 -0.7229 0.4372]

b'_i = [0.0102 0.0087 0.0161 -0.1279 -0.2336 -0.462 -0.5838 -0.3082 0.5284]

c_i = [-0.000008882332383 0.000024266904975 -0.000110314423131 0.013635893853407
-0.040219421632148 0.257784610034901 -0.396273137768735 -0.30258580833140 -0.592435773698846]

d_i = -0.0795

A_r = [0.8002 0.2915 -0.3579 0.3146 0.0570 0.1660
-0.1806 -0.7124 -0.4751 0.3769 0.0278 0.1911
-0.0684 0.1466 -0.5038 -0.5706 -0.3971 -0.1523
-0.0494 0.0956 0.4690 0.4354 -0.4872 -0.0263
-0.0062 0.0049 0.2257 -0.3369 0.2427 0.8256
0.0158 -0.0293 -0.0758 0.0159 -0.7229 0.4372]

b'_r = [-0.1279 -0.2336 -0.4621 -0.5838 -0.3082 0.5284]

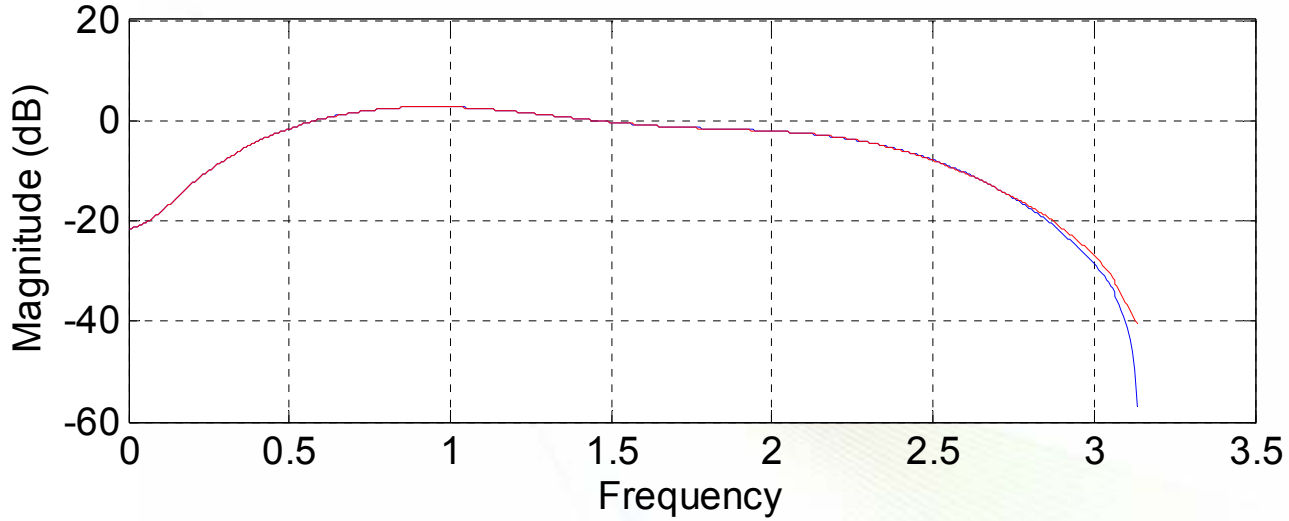
c_r = [0.0136 -0.0402 0.2578 -0.3963 -0.3026 -0.5924]

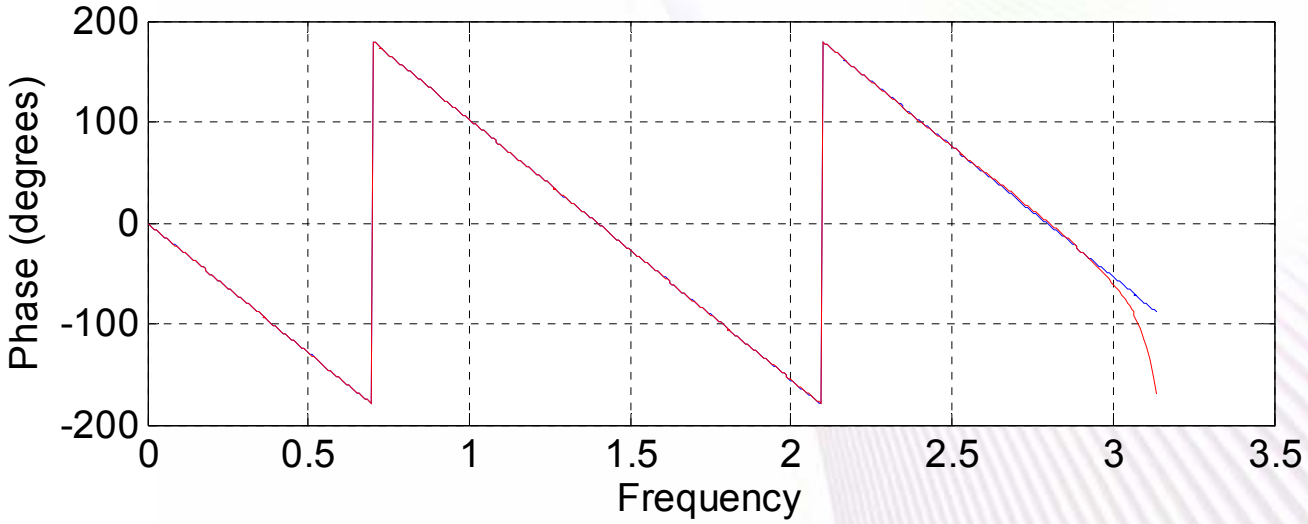
d_r = -0.0795

B_r = [-0.07952299231177 -0.04427480376513 -0.09280660599397 -0.06242920911555 0.60468644078407 0.13661347075775 -0.42368481820613]

A_r = [1.0 -0.69925672835387 0.18318319691378 -0.09007923518236 0.12184385725213 -0.07564371164443 0.01851704394719]

Other Biquad Tricks: FIR Example

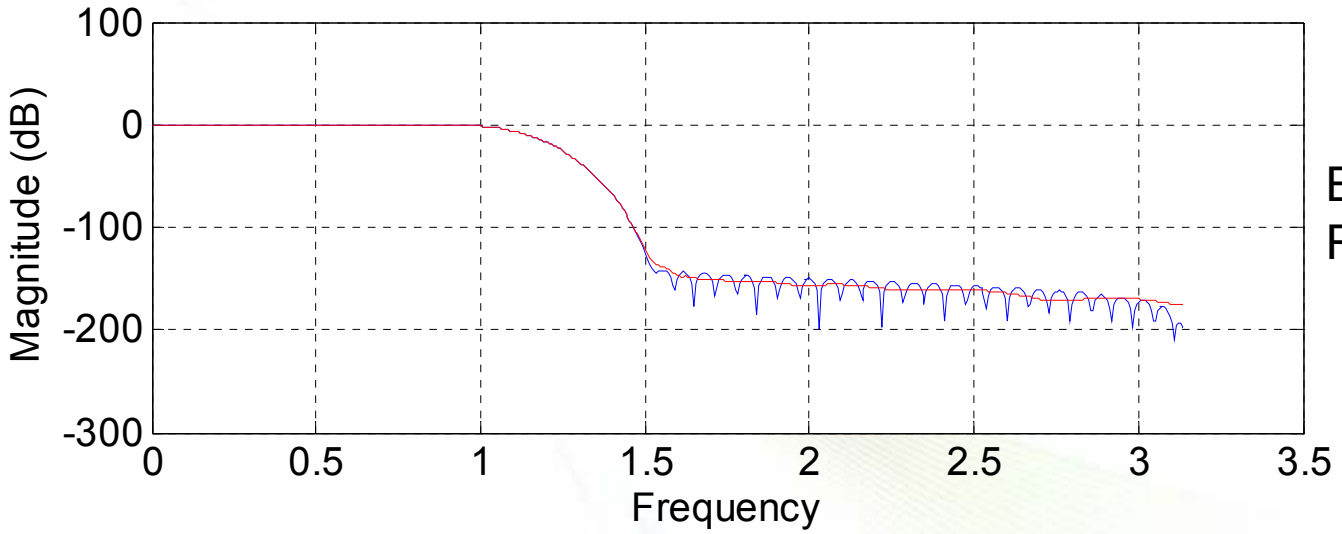


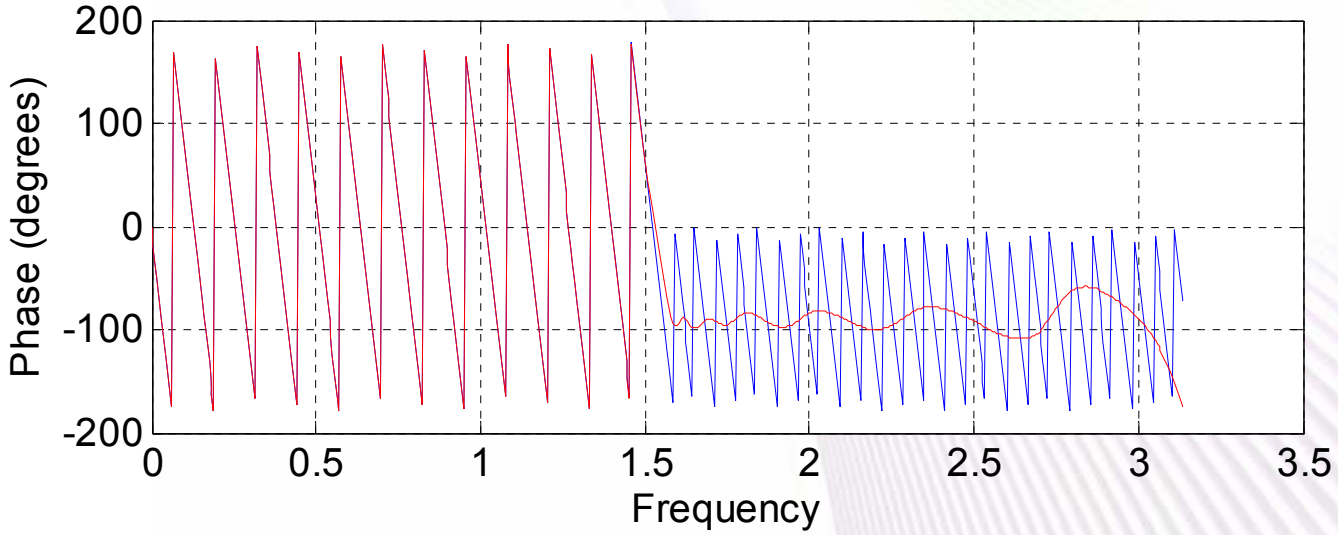


Blue: 9th-order FIR
Red: 6th-order IIR

Minds in Motion

Other Biquad Tricks: FIR Example

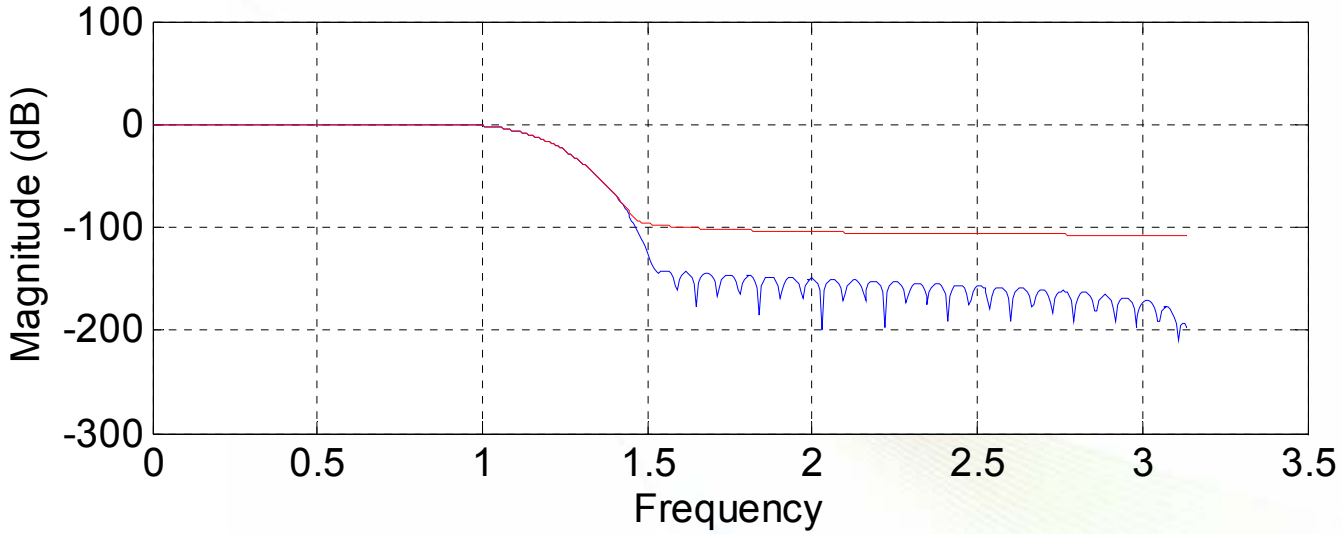




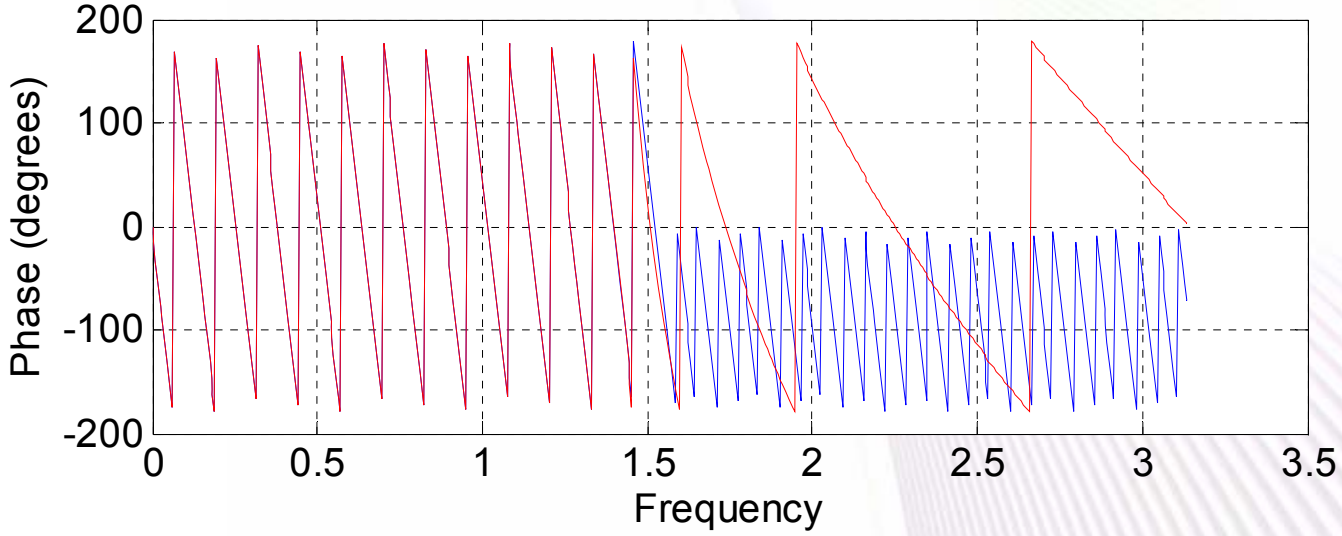
Blue: 100th-order FIR
Red: 40th-order IIR

Minds in Motion

Other Biquad Tricks: FIR Example



Blue: 100th-order FIR
Red: 30th-order IIR



Other Biquad Tricks: Oscillator

Consider the transfer function of a biquad:

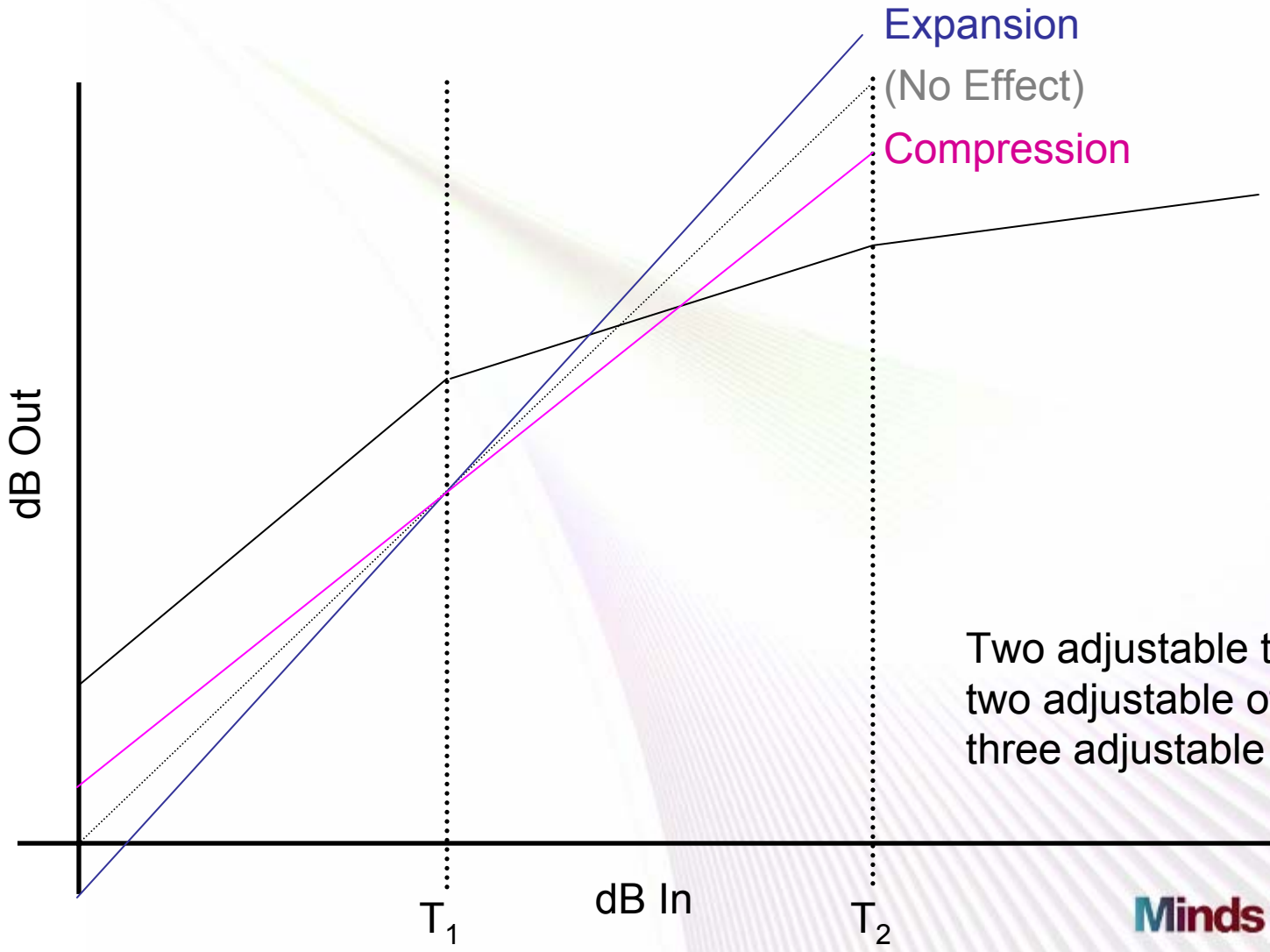
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a\ z^{-1} + a_2 z^{-2}}$$

Initialize b_0 to 1, $b_1 = b_2 = 0$, $a_1 = -2\cos(2\pi f/Fs)$, $a_2 = 2$.
Apply energy to the input (any signal).

This filter is unstable, which will drive the output to saturation. After a short instant, change the value of a_2 to 1. Now the filter is marginally stable and will oscillate at full scale. The input can now be removed, or, equivalently, the value of b_0 set to 0, and the oscillation will continue indefinitely.

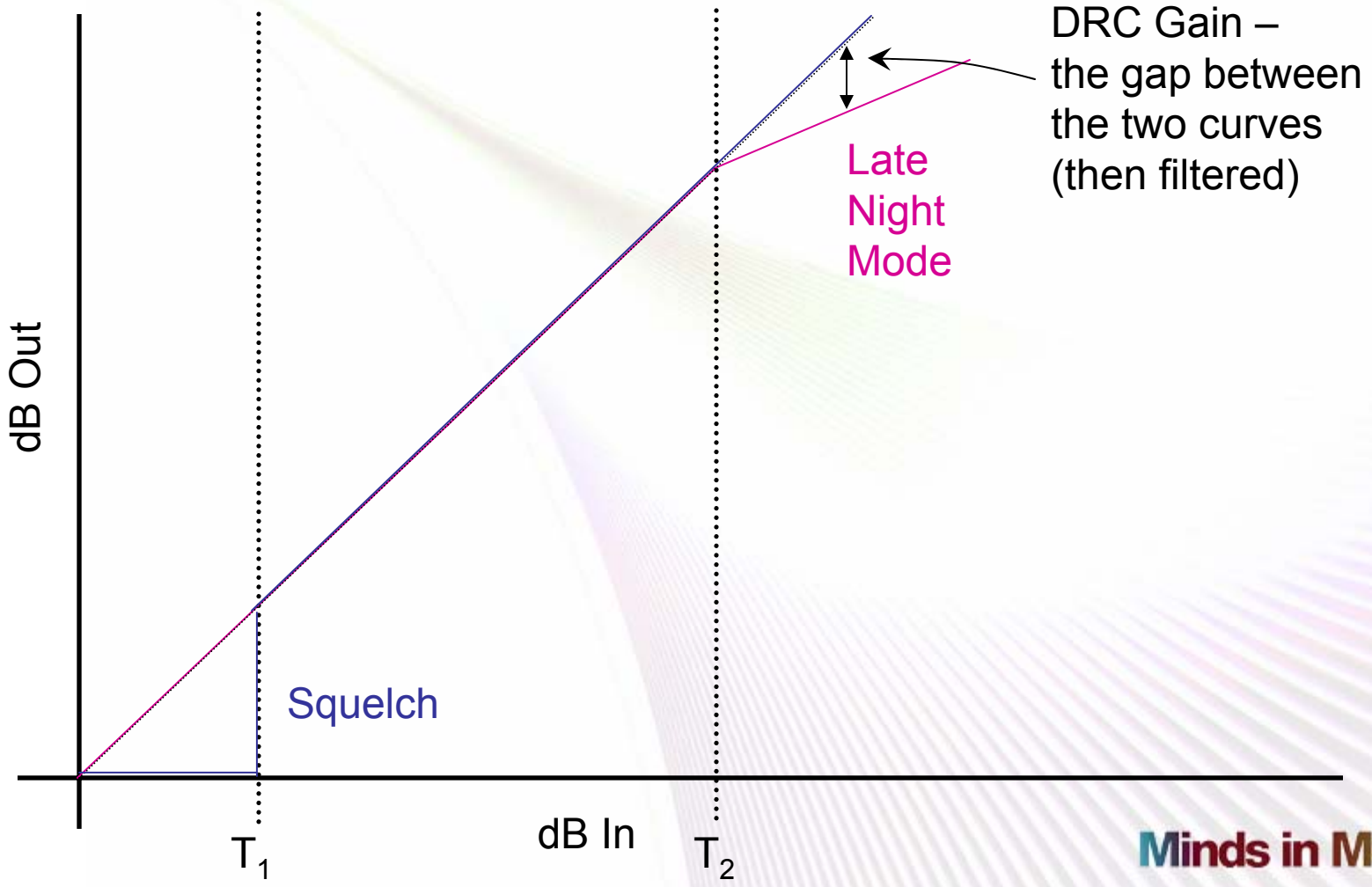
Congratulations! You have just built an oscillator.

Dynamic Range Compression and Expansion

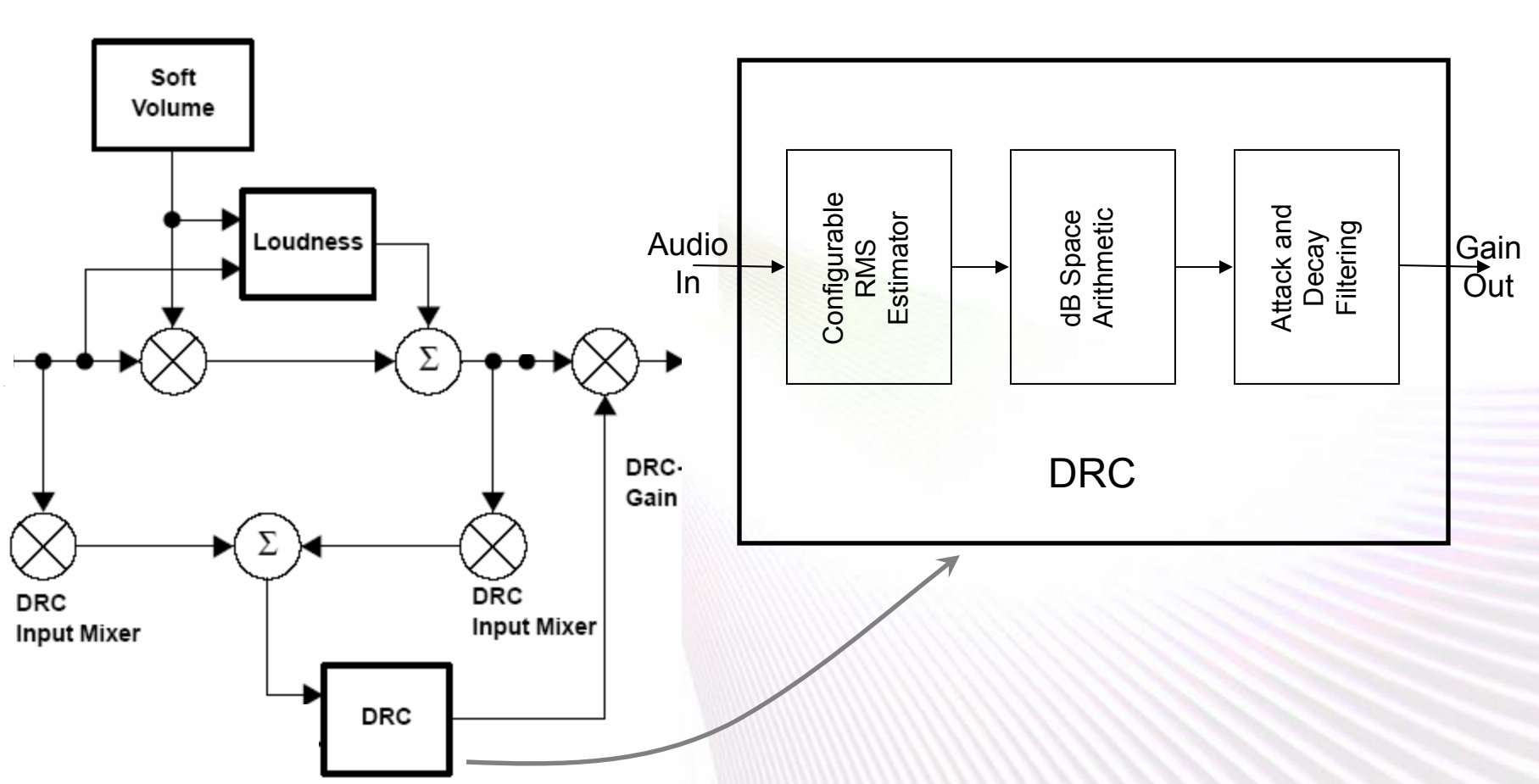


Two adjustable thresholds,
two adjustable offsets,
three adjustable slopes

Dynamic Range Compression and Expansion



Dynamic Range Compression

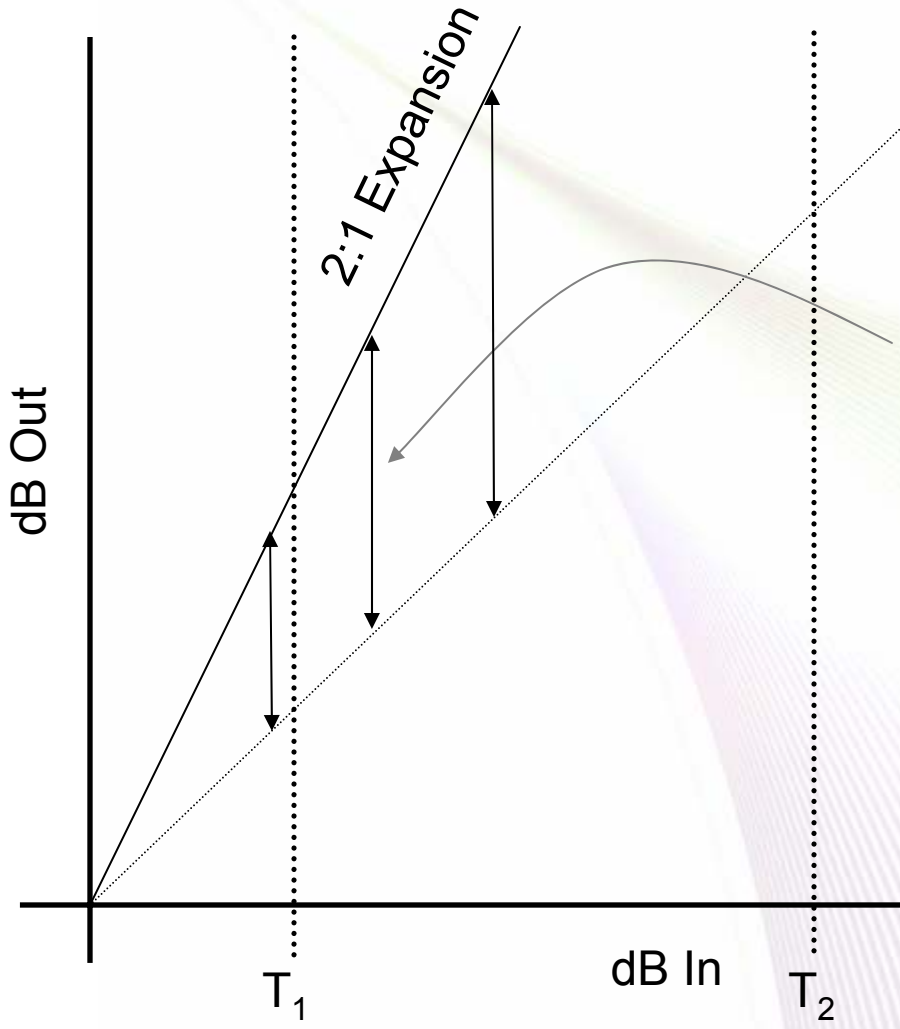


The Challenge

- Now, imagine that an algorithm requires the square of the signal. The constrained architecture has no such capability, but it does have a DRC…

- What we need is an RMS estimate that is proportional to the input sample, and then settings that provide an output gain proportional to that RMS estimate. Set the RMS estimation window to a single sample, and then…

Dynamic Range Compression and Expansion

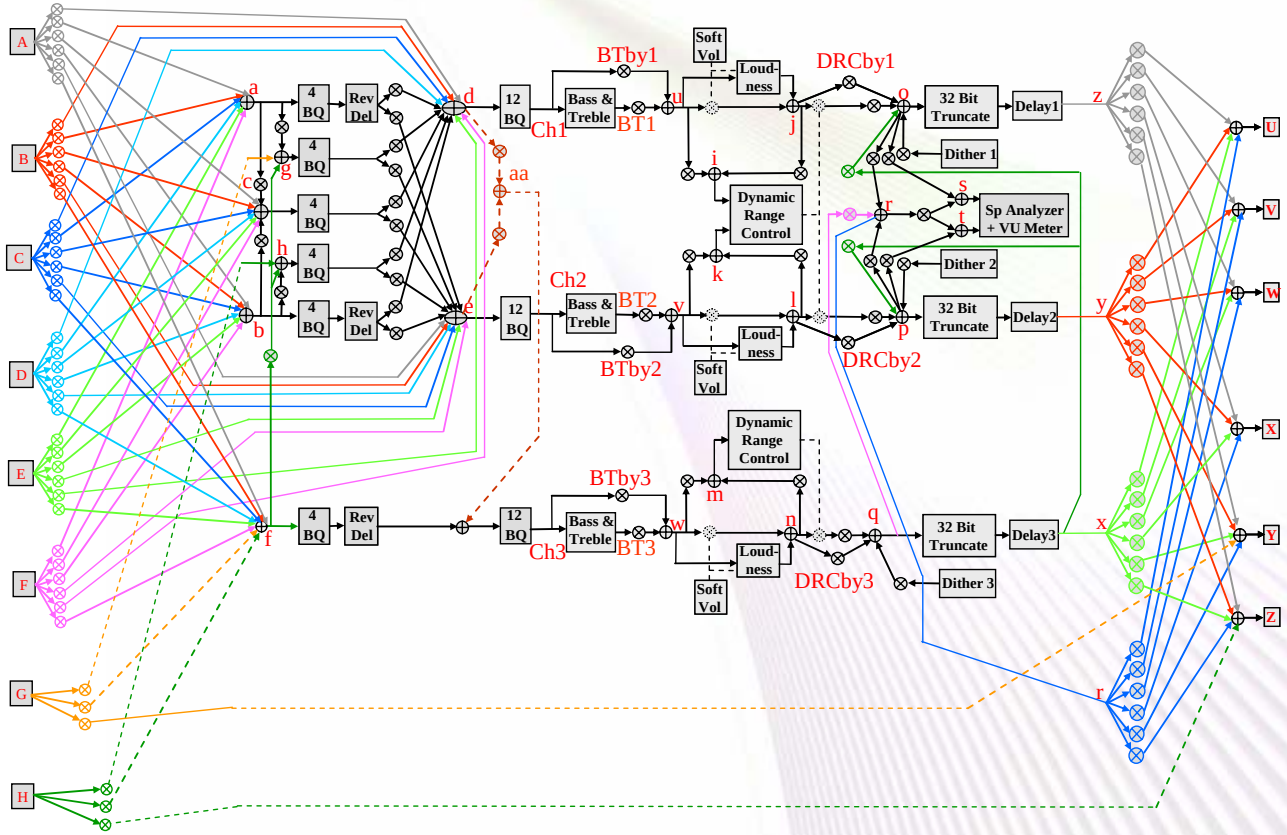


Gain is proportional to input RMS level. If RMS estimator is set to a single sample, this gain is proportional to the level of each sample. (Attack and decay filtering can be defeated for a raw gain output, but they will typically be employed to prevent audible artifacts.)

This is not quite signal^2, rather it is $\alpha \cdot signal \cdot |signal|$, which has proven adequate in application.

A Word on Efficiency

- The ordinary way of thinking about efficiency does not necessarily apply to non-programmable architectures. Typically, there is no extra computational burden for using an element of the architecture that's already there since they run all of the time anyway.



Minds in Motion

Conclusions

- Your constrained architecture might not be exactly like the TAS3103 (although it could be), but any such device becomes much more flexible when its capabilities are fully understood and when signal processing principles are creatively applied.

- The creative application of signal processing principles can also be beneficial for finding optimal implementations for non-constrained architectures.

TI Developer Conference
March 7-9, 2007 • Dallas, TX

Implementing Digital Audio Algorithms on Constrained and Unconstrained Architectures

Rusty Allred, PhD, PE
Mustang Technology Group, LP
400 W Bethany, Suite 110
Allen, TX 75013
rusty@mustangtechnology.com
(972) 396-4432 (direct)

Minds in Motion