2015 American Control Conference
Palmer House Hilton
July 1-3, 2015. Chicago, IL, USA

The Multinotch, Part I: A Low Latency, High Numerical Fidelity Filter for Mechatronic Control Systems

Daniel Y. Abramovitch*

Abstract— **Control of lightly damped mechatronic systems is often accomplished in practice with a PID-like controller in series with a filter to limit the effects of high frequency resonances. The high frequency filtering is often limited by an inability to precisely match multiple lightly damped resonances with a digital filter, and by the extra computational delay of the such filters. The multinotch is a filter topology that addresses these issues, allowing for precise matches to many lightly damped resonances and anti-resonances, while maintaining a small and fixed computational delay [1]. Coefficient adjustments for higher precision when the resonant dynamics to be filtered span a large frequency range are described in [2].**

I. Introduction

Control of lightly damped mechatronic systems is often accomplished in practice with a PID-like controller in series with a filter to limit the effects of high frequency resonances as diagrammed in Figure 1. The high frequency filtering is often limited by an inability to precisely match multiple lightly damped resonances with a digital filter, and by the extra computational delay of the such filters. The multinotch is a filter topology that addresses these issues, allowing for precise matches to many lightly damped resonances and anti-resonances, while maintaining a small and fixed computational delay.

Two issues that must be addressed when using digital filtering in a feedback loop are numerical issues and computational latency. Numerical issues typically come from the implementation of algorithms and filters in finite word length arithmetic. In particular, high order polynomial filters lose both physical intuition and numerical sensitivity because of how physical parameters get compressed into coefficients.

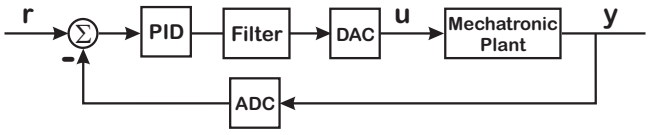

Fig. 1. A practical digital control loop for a mechatronic system. The digital controller is often implemented as a PID like controller in series with filtering to lower the effect of high frequency resonances.

The phase lag due to latency in a feedback loop erodes stability and performance characteristics. For a causal filter, the latency is in part determined by the filter length. That is, if a filter has N taps and a sample period of T_S, then the average latency through the filter will be $\frac{N}{2}T_S$. IIR filters are usually favored over FIR filters in feedback loops, as they can achieve similar bandwidth shaping with considerably smaller average delay.

*Daniel Y. Abramovitch is a system architect in the Mass Spectrometry Division, Agilent Technologies, 5301 Stevens Creek Blvd., M/S: 3U-DG, Santa Clara, CA 95051 USA, danny@agilent.com

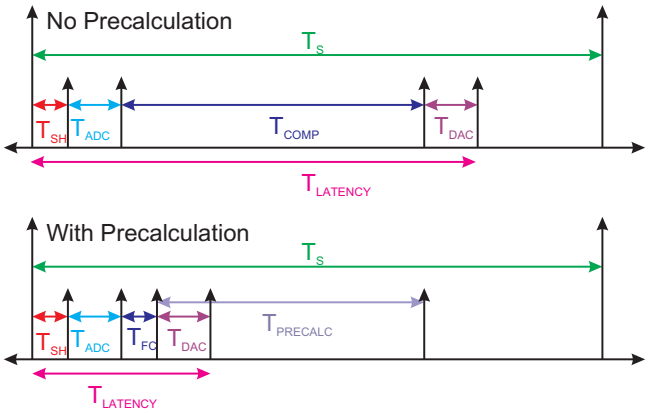

Fig. 2. Input and output timing in a digital control system. The top drawing is without precalculation; the bottom drawing is with. Note that precalculation can be started as soon as the output has been sent to the DAC and therefore is in parallel with the DAC conversion time. The computation time, T_{COMP}, of the top diagram is now split into $T_{PRECALC} + T_{FC}$ where $T_{PRECALC}$ is the computation time needed for the precalculation and T_{FC} is the time needed for the final calculation after the input sample. Modulo some small programming overhead, the split time should equal the total computation time. Here T_{SH}, T_{ADC}, and T_{DAC} represent the sample and hold, ADC conversion, and DAC conversion times, respectively.

The second source of latency is simply the time required to compute the filter equations between the time that a new sample comes into the filter and the filter produces its output. This delay is generally - but not necessarily - less than one sample period, but it is complicated by the fact that it can change with the number of taps in a filter. That is, a 2nd order filter obviously takes fewer computation steps than a 10th order filter. This variable latency can cause unexpected problems with the control loop. A standard technique to minimize this variable latency is to compute everything that does not depend upon the most recent sample ahead of time in a precalculation [3], diagrammed in Figure 2. Once the most recent sample arrives, the last few calculations are performed and the filter output is produced. This has the benefit of not only minimizing the computational latency, but also of making it independent of filter length.

However, doing a precalculation usually involves using a form of the digital filter that can have numerical issues with finite word length. The structure of the multinotch allows it to minimize the computational latency using precalculation, while still preserving the numerical properties needed for finite word length arithmetic.

II. Digital Filter Equations and Biquads

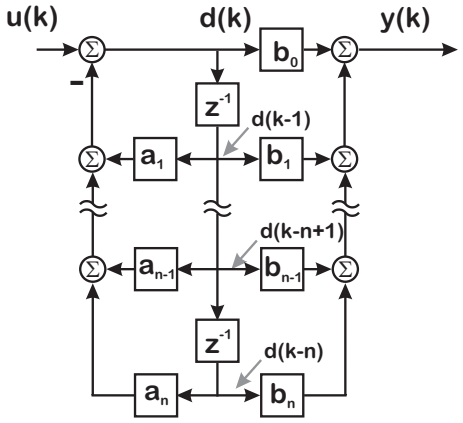

Fig. 3. An nth order polynomial filter in Direct Form II configuration [4].

Single-Input, Single-Output (SISO) digital filters, as shown in Figure 3, can be represented as transfer functions in the unit delay operator, z^{-1} which lends itself readily to real-time implementation in assembly [5] or a high level language:

$$\frac{Y(z^{-1})}{U(z^{-1})} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}} \quad (1)$$

The transfer function in(1) is not unique but has an advantage in that the coefficient of the current output term, $y(k)$, is 1, so the filter implementation is:

$$\begin{aligned} y(k) &= -a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n) \\ &\quad + b_0 u(k) + b_1 u(k-1) + \ldots + b_n u(k-n). \quad (2) \end{aligned}$$

Looking at (2), we see that $y(k)$ depends mostly on previous inputs and outputs. The only current value needed is $u(k)$ and this is only multiplied by b_0. So we can break this up into [6]:

$$y(k) = b_0 u(k) + \text{prec}(k), \text{ where} \quad (3)$$

$$\begin{aligned} \text{prec}(k) &= -a_1 y(k-1) - \ldots - a_n y(k-n) \\ &\quad + b_1 u(k-1) + \ldots + b_n u(k-n), \quad (4) \end{aligned}$$

and prec(k) depends only on previous values of $y(k)$ and $u(k)$. This means that prec(k) can be computed for step k immediately after the filter has produced the output for time index, $k-1$ [3]. When the input at time step k, $u(k)$, comes into the filter, it needs merely be multiplied by b_0 and added to prec(k) to produce the filter output. Thus, the delay between the input of $u(k)$ and the output of $y(k)$ is small and independent of the filter length.

The problem with polynomial filters is that the elements that generate discrete filter coefficients can become extremely sensitive, particularly when they relate to high Q elements near the unit circle. As these get multiplied together to form the polynomial coefficients, not only is physical intuition completely lost, but the coefficients become even more susceptible. This is particularly true with fixed point arithmetic used in many DSP and FPGA (Field Programmable Gate Array) implementations. Thus, what we would like to do is implement the filter of (1) as a series of second order filters, known as biquads, but still maintain the ability to do final calculations in the form of (3) [5].

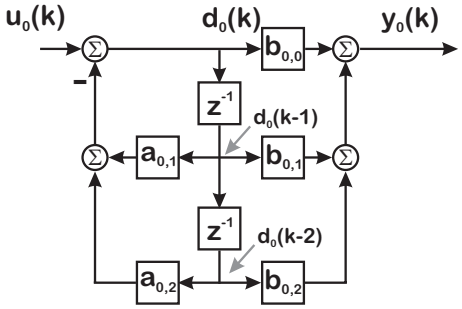

Fig. 4. A second order polynomial filter in Direct Form II, known as a digital biquad filter.

Biquads are second order sections that typically are numerically well behaved. For example, if we implement a 10^{th} order filter using 5 biquads, coefficient quantization issues that we might get with a 10^{th} order polynomial filter are localized to each biquad and therefore limited. The latter might be something that we would do if we had a floating point DSP chip, but would not be advisable for fixed point calculations such as the ones that we do on the FPGA. So, the biquad of Figure 4 would look like:

$$\frac{Y_0(z)}{U_0(z)} = N_0(z) = \frac{b_{0,0} + b_{0,1} z^{-1} + b_{0,2} z^{-2}}{1 + a_{0,1} z^{-1} + a_{0,2} z^{-2}} \quad (5)$$

which gets implemented in the time domain as:

$$\begin{aligned} y_0(k) &= -a_{0,1} y_0(k-1) - a_{0,2} y_0(k-2) + b_{0,0} u_0(k) \\ &\quad + b_{0,1} u_0(k-1) + b_{0,2} u_0(k-2) \quad (6) \end{aligned}$$

It turns out that it is easier to implement this using the delay format [5] which resembles a controller canonical form [7] in control or a direct form II IIR filter [4], [8], [9]:

$$d_0(k) = -a_{0,1} d_0(k-1) - a_{0,2} d_0(k-2) + u_0(k) \quad (7)$$

$$y_0(k) = b_{0,0} d_0(k) + b_{0,1} d_0(k-1) + b_{0,2} d_0(k-2) \quad (8)$$

Biquads are nice because the growth in values can be limited by the short nature of the filter. Thus, finite word length problems are minimized as the sums from the numerator and denominator can balance each other out for a well designed filter [5].

As with (2), only one part of the filter actually depends upon the current input, $u(k)$, and that is the calculation in (6). Likewise, in (7) only one part of the sum is due to $u(k)$, and there is only one multiply involving $d(k)$ in (8). Everything else can be calculated ahead of time. So, we can generate the precalculation for this form of the biquad as:

$$d_0(k) = \text{prec}_{0,1}(k) + u_0(k) \text{ and} \quad (9)$$

$$y_0(k) = b_{0,0} d_0(k) + \text{prec}_{0,2}(k) \text{ where} \quad (10)$$

$$\text{prec}_{0,1}(k) = -a_{0,1}d_0(k-1) - a_{0,2}d_0(k-2) \text{ and} \quad (11)$$
$$\text{prec}_{0,2}(k) = b_{0,1}d_0(k-1) + b_{0,2}d_0(k-2). \quad (12)$$

As before, we see that $\text{prec}_{0,1}(k)$ and $\text{prec}_{0,2}(k)$ depend only on prior values of the filter input and output and can be computed ahead of time.

III. Higher Order Filters as a Series of Biquads

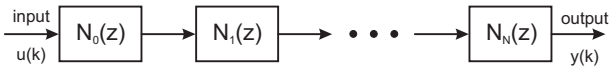

Fig. 5. Series connection of multiple filters. In our case, these filters are each a second order digital transfer function (biquad).

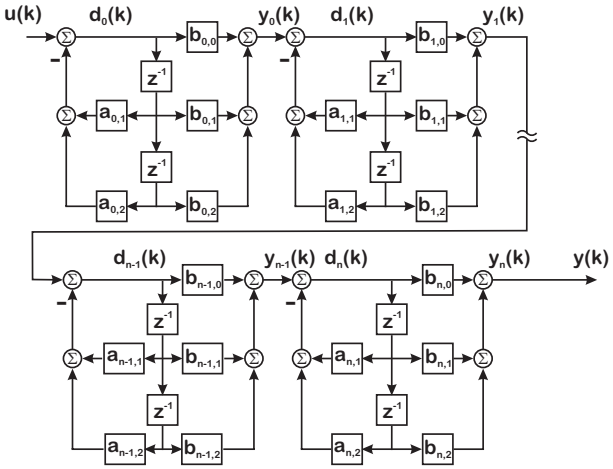

Fig. 6. An expanded realization view of the serial biquad chain from Figure 5.

If we wish to stack individual blocks of biquads together to filter out multiple resonances and/or anti-resonances, we can simply do a serial cascade of blocks as shown in Figure 5, which could be represented internally as shown in Figure 6. The problem here is that in this form it is not possible to do precalculation of anything but the first block, because the downstream blocks all depend upon the output of the previous blocks. Consider the example of the two biquad case, where in we augment Equations 5–8 with:

$$\frac{Y_1(z)}{U_1(z)} = N_1(z) = \frac{b_{1,0} + b_{1,1}z^{-1} + b_{1,2}z^{-2}}{1 + a_{1,1}z^{-1} + a_{1,2}z^{-2}} \quad (13)$$

which gets implemented in the time domain as:

$$y_1(k) = -a_{1,1}y_1(k-1) - a_{1,2}y_1(k-2) + b_{1,0}u_1(k) + b_{1,1}u_1(k-1) + b_{1,2}u_1(k-2) \quad (14)$$

The delay forms are:

$$d_1(k) = -a_{1,1}d_1(k-1) - a_{1,2}d_1(k-2) + u_1(k) \quad (15)$$

$$y_1(k) = b_{1,0}d_1(k) + b_{1,1}d_1(k-1) + b_{1,2}d_1(k-2) \quad (16)$$

with precalcs implemented as:

$$d_1(k) = \text{prec}_{1,1}(k) + u_1(k) \text{ and} \quad (17)$$

$$y_1(k) = b_{1,0}d_1(k) + \text{prec}_{1,2}(k) \text{ where} \quad (18)$$

$$\text{prec}_{1,1}(k) = -a_{1,1}d_1(k-1) - a_{1,2}d_1(k-2) \text{ and} \quad (19)$$
$$\text{prec}_{1,2}(k) = b_{1,1}d_1(k-1) + b_{1,2}d_1(k-2). \quad (20)$$

Furthermore, they are stitched in series by letting the overall input go into biquad 0, the overall output come from biquad 1 and using the output of biquad 0 as the input to biquad 1. That is

$$u_0(k) = u(k), y(k) = y_1(k), \text{ and } u_1(k) = y_0(k). \quad (21)$$

We see that $y_1(k)$ depends on $b_{1,0}d_1(k)$, which depends on $u_1(k) = y_0(k)$, which depends on $b_{0,0}d_0(k)$, which depends upon $u_0(k)$. This is obvious if we try to chain these together:

$$d_1(k) = prec_{1,1}(k) + u_1(k) = prec_{1,1}(k) + y_0(k) \quad (22)$$
$$d_1(k) = prec_{1,1}(k) + prec_{0,2}(k) + b_{0,0}\left[prec_{0,1}(k)\right] + b_{0,0}u_0(k) \quad (23)$$

The output, $y_1(k)$ is obtained via:

$$y_1(k) = b_{1,0}d_1(k) + prec_{1,2}(k) \quad (24)$$
$$y_1(k) = prec_{1,2}(k) + b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k)\right] + b_{1,0}b_{0,0}prec_{0,1}(k) + b_{1,0}b_{0,0}u_0(k) \quad (25)$$

We can add in a third biquad:

$$d_2(k) = prec_{2,1}(k) + u_2(k) = prec_{2,1}(k) + y_1(k) \quad (26)$$
$$= prec_{2,1}(k) + prec_{1,2}(k) + b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k)\right] + b_{1,0}b_{0,0}\left[prec_{0,1}(k)\right] + b_{1,0}b_{0,0}u(k) \quad (27)$$

and likewise we get $y_2(k)$ via

$$y_2(k) = b_{2,0}d_2(k) + prec_{2,2}(k) \quad (28)$$
$$y_2(k) = prec_{2,2}(k) + b_{2,0}\left[prec_{2,1}(k) + prec_{1,2}(k)\right] + b_{2,0}b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k)\right] + b_{2,0}b_{1,0}b_{0,0}prec_{0,1}(k) + b_{2,0}b_{1,0}b_{0,0}u_0(k). \quad (29)$$

Several things start to emerge about this recursion. First, there is a definite pattern. Next, we see that we can still precompute our original precalc sections, but each one is scaled by some concatenation of $b_{i,0}$ terms to update the most recent $d_i(k)$ and $y_i(k)$ values. This means that the more biquad sections we add, the more multiplications are required to generate the current d_i and y_i terms once the $u(k)$ input is available. The further down the chain we go, the more multiplies need to be computed once the current For the i^{th} biquad, counting from 0, $d_i(k)$ requires i multiplies and $y_i(k)$ requires $i+1$ multiples (after $u(k)$ is available). This means that the computational latency grows with filter length.

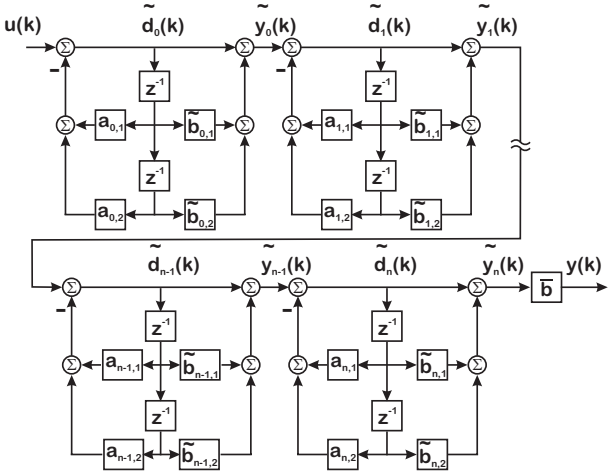

Fig. 7. The updated biquad cascade, with factored out b_0 terms.

IV. An Improved Structure

A look at Equations 27 and 29 reveals that the problem lies with the $b_{i,0}$ terms. If these terms were only 1, then our problem would be solved. Remembering third grade math, we note that multiplication is associative, and thus we can factor out the $b_{i,0}$ terms from our filters, ending up with an equivalent cascade of biquads:

$$\frac{Y(z)}{U(z)} = N_n(z)N_{n-1}(z)\cdots N_1(z)N_0(z) \quad (30)$$

$$= b_{n,0}\cdots b_{1,0}b_{0,0}\tilde{N}_n(z)\cdots\tilde{N}_1(z)\tilde{N}_0(z) \quad (31)$$

where

$$\tilde{N}_i(z) = \frac{1+\tilde{b}_{i,1}z^{-1}+\tilde{b}_{i,2}z^{-2}}{1+a_{i,1}z^{-1}+a_{i,2}z^{-2}} \quad (32)$$

$$\tilde{b}_{i,1} = \frac{b_{i,1}}{b_{i,0}}, \text{ and } \tilde{b}_{i,2} = \frac{b_{i,2}}{b_{i,0}}. \quad (33)$$

The direct feedthrough gains are concatenated together as:

$$\bar{b} = b_{n,0}b_{n-1,0}\cdots b_{1,0}b_{0,0}. \quad (34)$$

This structure, shown in Figure 7 has the advantage that:

- It retains the biquad form with the same poles and zeros as the original filter of Figure 6.
- $\tilde{d}_i(k)$ and $\tilde{y}_i(k)$ can be computed from summing precalculated terms with the current input, $u(k)$.
- Once $u(k)$ is available, the computation of $y(k)$ involves two additions of precalculated terms and one multiplication by $\bar{b}$. This means that the computational latency between the measurement input and the filter output is very small and independent of filter length.
- The coefficients of individual biquad sections are computed independently, retaining most of the physical intuition in the filter, even after discretization.
- Picking complex pole-zero pairs that are close to each other generally limits the signal growth in any biquad block, which is helpful for fixed point math.
- Such a structure can be implemented in such a way that different biquad sections can be turned on or off. In this case only the aggregate direct feedthrough gain, $\bar{b}$ needs to be adjusted when a biquad block is activated or deactivated in a real time controller.

Looking at the structure, we see that:

$$\tilde{d}_0(k) = D_{P,0}(k-1) + u(k), \quad (35)$$

$$\tilde{d}_i(k) = \sum_{j=0}^{i} D_{P,j}(k-1) + \sum_{j=0}^{i-1} F_{P,j}(k-1) + u(k), \quad (36)$$

for $i \geq 1$, and

$$\tilde{y}_i(k) = \sum_{j=0}^{i} D_{P,j}(k-1) + \sum_{j=0}^{i} F_{P,j}(k-1) + u(k), \quad (37)$$

for $i \geq 0$, where the delay precalculations are

$$D_{P,j}(k-1) = -a_{j,1}\tilde{d}_j(k-1) - a_{j,2}\tilde{d}_j(k-2) \quad (38)$$

and the output precalculations are

$$F_{P,j}(k-1) = \tilde{b}_{j,1}\tilde{d}_j(k-1) + \tilde{b}_{j,2}\tilde{d}_j(k-2). \quad (39)$$

Note that as the signal moves through biquad stages, sums get added to the precalculation. However, because these are all scaled the same, the sums that have already been computed can be reused. The final output, $y(k)$ is obtained from the last biquad output using $\bar{b}$ from Equation 34 as

$$y(k) = \bar{b}\tilde{y}_n(k). \quad (40)$$

V. Generating Coefficients

$f_{N,i}$	Center frequency of numerator (Hz)
$\omega_{N,i}$	Center frequency of numerator (rad/s)
$Q_{N,i}$	Quality factor of numerator
$\zeta_{N,i} = \frac{1}{Q_{N,i}}$	Damping factor of numerator
$f_{D,i}$	Center frequency of denominator (Hz)
$\omega_{D,i}$	Center frequency of denominator (rad/s)
$Q_{D,i}$	Quality factor of denominator
$\zeta_{D,i} = \frac{1}{Q_{D,i}}$	Damping factor of denominator

TABLE I
Physical coefficients used to specify a biquad section.

Because this filter is a digital biquad, there are no excess poles or zeros. Furthermore, using this form where we have factored the $b_{i,0}$ gain out of each numerator means that all the biquads will have a uniform structure. Taking our design from an analog response of a ratio of a second order numerator and denominator, we can discretize the poles and zeros using matched pole-zero mapping [6]. This allows us to parametrize each biquad section using very physical parameters, as shown in Table I. The factored out gain, $b_{i,0}$, can be used as is, or can be altered so that, for example, the DC gain of the biquad section will be 1.

Assuming a complex pair of poles (or zeros), mapping via $z = e^{sT_S}$, and recombining the results yields some straightforward formulas. For $a_{i,2}$ and $\tilde{b}_{i,2}$ we have

$$a_{i,2} = e^{-2\omega_{D,i}T_S\zeta_{D,i}} \quad (41)$$

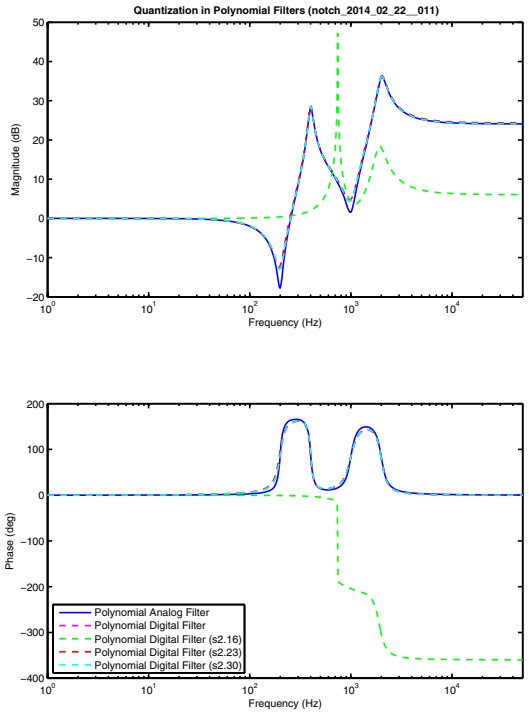

Fig. 8. Effects of quantization on polynomial filter in the first example of Table II. The effect is only pronounced for the s2.16 quantization.

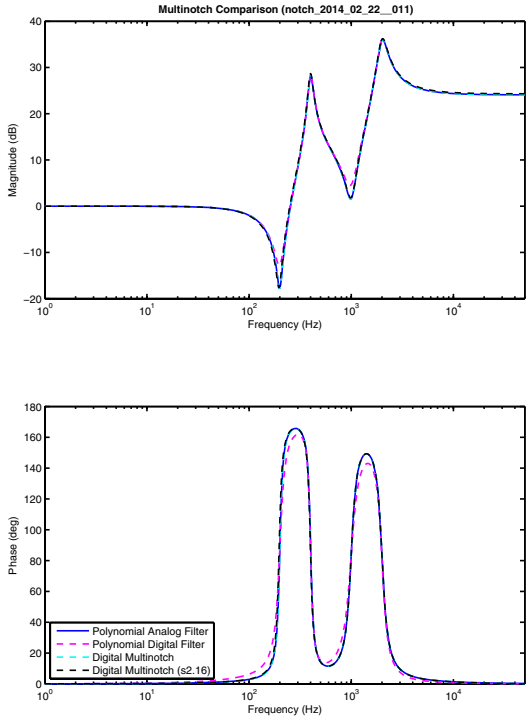

Fig. 9. A comparison of the quantized multinotch to unquantized filters for the first example of Table II. The curves are practically identical.

$$\tilde{b}_{i,2} = e^{-2\omega_{N,i}T_S\zeta_{N,i}} \tag{42}$$

Whether the poles (or zeros) are a complex pair depends upon $|\zeta_{D,i}|$ ($|\zeta_{N,i}|$). For $|\zeta_{D,i}| < 1$ we have a complex pair of poles and so

$$a_{i,1} = -2e^{-\omega_{D,i}T_S\zeta_{D,i}} \cos\left(\omega_{D,i}T_S\sqrt{1-\zeta_{D,i}^2}\right). \tag{43}$$

If $|\zeta_{N,i}| < 1$ we have a complex pair of zeros and so

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i}T_S\zeta_{N,i}} \cos\left(\omega_{N,i}T_S\sqrt{1-\zeta_{N,i}^2}\right). \tag{44}$$

While these two cases represent cases when the desired filters have very sharp peaks or notches (for example to equalize a response with very sharp notches or peaks), there are other possibilities. For example setting $|\zeta_{D,i}| = 1$ ($|\zeta_{N,i}| = 1$) means that the poles (zeros) are real and equal, so $a_{i,1}$ ($\tilde{b}_{i,1}$) are given by:

$$a_{i,1} = -2e^{-\omega_{D,i}T_S\zeta_{D,i}} \;\; (\tilde{b}_{i,1} = -2e^{-\omega_{N,i}T_S\zeta_{N,i}}). \tag{45}$$

Finally, if $|\zeta_{D,i}| > 1$ ($|\zeta_{N,i}| > 1$) means that the poles (zeros) are real and distinct, so $a_{i,1}$ ($\tilde{b}_{i,1}$ are given by using the cosh relation:

$$a_{i,1} = -2e^{-\omega_{D,i}T_S\zeta_{D,i}} \cosh\left(\omega_{D,i}T_S\sqrt{\zeta_{D,i}^2-1}\right) \tag{46}$$

and

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i}T_S\zeta_{D,i}} \cosh\left(\omega_{N,i}T_S\sqrt{\zeta_{N,i}^2-1}\right). \tag{47}$$

The entire conversion routine, which turns the physical parameters of Table I into discrete filter coefficients can be implemented in a short Matlab or Octave function.

Example 1				
Biquad #	$f_{N,n}$ **(Hz)**	Q_n	$f_{N,d}$ **(Hz)**	Q_d
1	200	10	400	10
2	1000	5	2000	5
Example 2				
Biquad #	$f_{N,n}$ **(Hz)**	Q_n	$f_{N,d}$ **(Hz)**	Q_d
1	200	10	400	10
2	1000	5	2000	5
3	10,000	10	20,000	10
4	8000	10	4000	10

TABLE II
FILTER PARAMETERS FOR BOTH EXAMPLES.

VI. EXAMPLES

In order to compare filters, a set of filter parameters was chosen in the form of sets analog biquad parameters, such as those in Table II. These parameters were then translated into an analog polynomial filter, which was discretized via Matlab's c2d function. The coefficients of the digital filter were then scaled up by a quantization factor, say $2^{16}-1$ for an s2.16 quantization. The scaled up coefficients were then fixed (fractional portion removed) and scaled down by the same quantization factor. Thus, floating point numbers were made to represent fixed point coefficients. Frequency responses were computed for the analog, digital polynomial, and various quantized digital polynomial filters.

To compare against the multinotch, the same analog biquad parameters were converted into individual digital biquads as described in Section V. Since the biquads act serially on data, the frequency responses of each digital

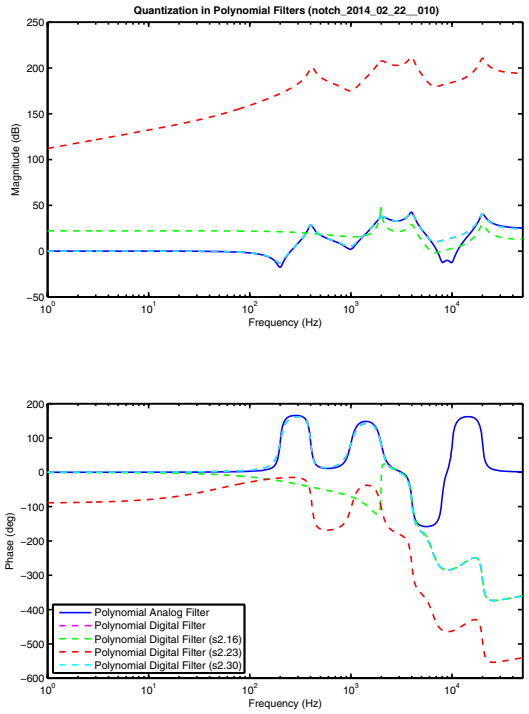

Fig. 10. Quantization in the polynomial filters of the second example of Table II shows severe effects for the s2.16 and s2.23 cases. For this example, the s2.30 quantization is able to represent the unquantized digital polynomial filter, but this still does not match the analog response.

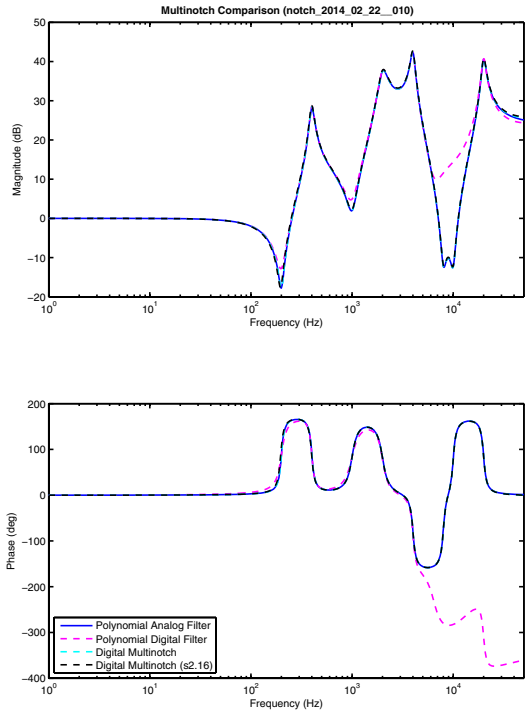

Fig. 11. A comparison of the quantized multinotch to unquantized filters in the second example of Table II. Note that even without quantization, the discrete polynomial filter no longer matches the analog response. The multinotch, with even the coarsest quantization considered in these tests, s2.16, still matches the analog response.

biquad was multiplied with those that were previously computed to form the overall response of the filter. In the case where quantization was applied, the individual biquad parameters were scaled up by the quantization factor, fixed, and scaled back down. As the direct feedthrough gain is applied afterwords, this was separately scaled up, fixed, and scaled down to compute the individual quantized biquad complex responses. Again, these responses were multiplied to that of the previous sections to give an overall biquad frequency response.

The filter parameters were set according to Table II. Both examples use a sample rate of 100 kHz for discretization. The results for the first example are in Figures 8 and 9. In this simple fourth order filter, the effects of quantization are only evident for the polynomial digital filter quantized at s2.16, although we can see that the discretized polynomial filter never fully matches the analog response. In the second example, there are far more biquads and they have higher Q values. The results in Figures 10 show that the polynomial digital filter fails to match the analog response even without quantization. With quantization, only the highest number of bits, s2.30, matches the unquantized digital polynomial filter. On the other hand, the multinotch in Figure 11 matches the analog response, even with the coarsest s2.16 quantization.

VII. Conclusions

The multinotch represents a new way of structuring digital filters so as combine physical intuition, numerical stability, and low, fixed latency into a single architecture. It lends itself extremely well to real time implementation, and is especially effective in fixed point environments, such as FPGA or DSP code. Furthermore, the regular, recursive structure of the filter allows it to be programmed easily. It combines for the first time two extremely desirable features in a real time filter: small, fixed computational latency and high numerical fidelity. It also preserves the physical intuition of the analog parameters in the digital filter implementation, which is extremely helpful in debugging physical systems.

References

[1] D. Y. Abramovitch and C. R. Moon, "Cascaded digital filters with reduced latency," International Application Published Under the Patent Cooperation Treaty WO 2012/118483, World Intellectual Property Organization, September 9 2012.

[2] D. Y. Abramovitch, "The Multinotch, Part II: Extra precision via Δ coefficients," in *Submitted to the 2015 American Control Conference*, (Chicago, IL), AACC, IEEE, July 2015.

[3] K. J. Åström and B. Wittenmark, *Computer Controlled Systems, Theory and Design*. Englewood Cliffs, N.J. 07632: Prentice Hall, second ed., 1990.

[4] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, N. J.: Prentice Hall, 1975.

[5] Texas Instruments, *TMS320C4x User's Guide*, 1993.

[6] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*. Menlo Park, California: Addison Wesley Longman, third ed., 1998.

[7] T. Kailath, *Linear Systems*. Englewood Cliffs, N.J. 07632: Prentice-Hall, 1980.

[8] J. O. Smith, *Introduction to Digital Filters with Audio Applications*. http://www.w3k.org/books/: W3K Publishing, 2007.

[9] P. M. Embree, *C Algorithms for Real-Time DSP*. Upper Saddle River, NJ 07458: Prentice Hall PTR, 1995.