



WP330 (v1.2) August 10, 2009

Infinite Impulse Response Filter Structures in Xilinx FPGAs

By: Michael Francis

A large percentage of filters implemented in the digital domain are Finite Impulse Response (FIR) filters. These filters are used over a wide range of sample rates and are well supported in terms of tools, software, and IP cores. Another type of digital filter is the Infinite Impulse Response (IIR) filter, which tends not to be so well supported and is generally used in the lower sample rates, that is, less than 200 kHz. The IIR filter, known as a recursive filter, uses feedback to compute outputs.

This white paper covers the different kinds of IIR filters and structures, and, with the use of The MathWorks® tools, shows how these structures can be mapped to the Xilinx® FPGA architecture. A final consideration is how to pipeline IIR filters to support higher sample rates.

Introduction

The FIR filter is well understood and used within Digital Signal Processing (DSP). The FIR filter takes in input samples, processes them, and outputs the samples. The advantages of FIR filters are that they have linear phase that is stable, they are insensitive to quantization effects, easy to design, and do not suffer from limit cycle issues such as filter oscillating.

The IIR filter is an alternative filter structure known as *recursive*, meaning the output samples are fed back to make a contribution to the next output. IIR filters tend to have better magnitude responses, require fewer coefficients, require less storage for variables, have lower latency, and are closer to analog models.

Figure 1 shows the frequency response of a FIR filter and an IIR filter. This figure shows that for a sample rate of 100 MHz and a cut-off frequency of 10 MHz with stop frequency of 11 MHz, a 259th order was needed for the FIR filter implementation. However, only a 9th order filter was required for a sample rate of 100 MHz, and a cut-off frequency of 10 MHz with stop frequency of 11 MHz.

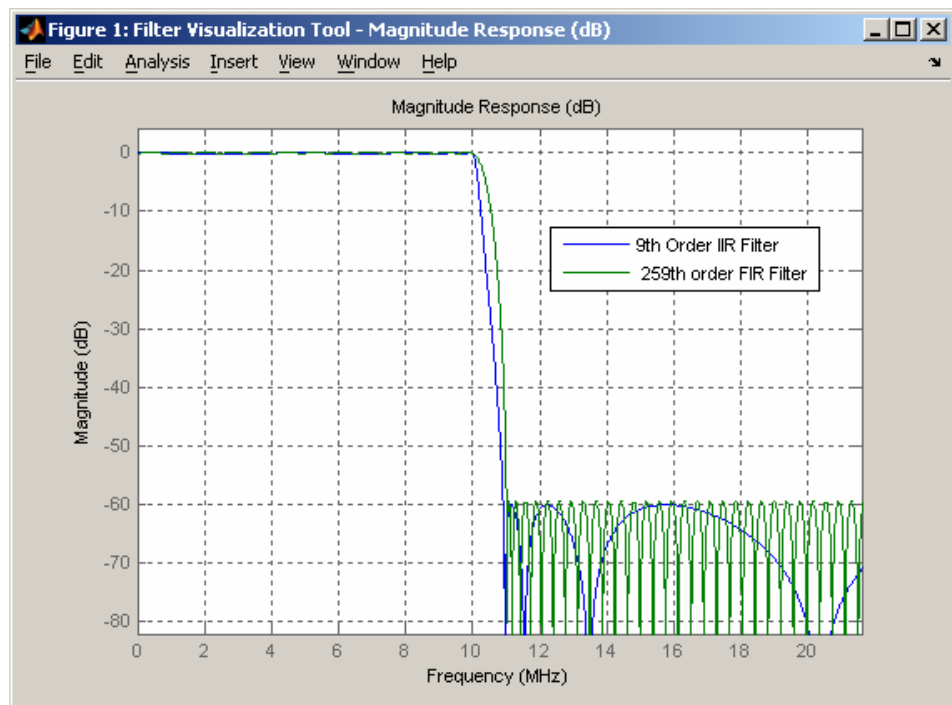


Figure 1: Filter Visualization Tool: Magnitude Response (dB)

Looking at the phase response diagram in Figure 2, the phase for the FIR filter is linear, yet the phase for the IIR filter is not linear. There are IIR filters that provide a better phase response, but these are not considered in this paper.

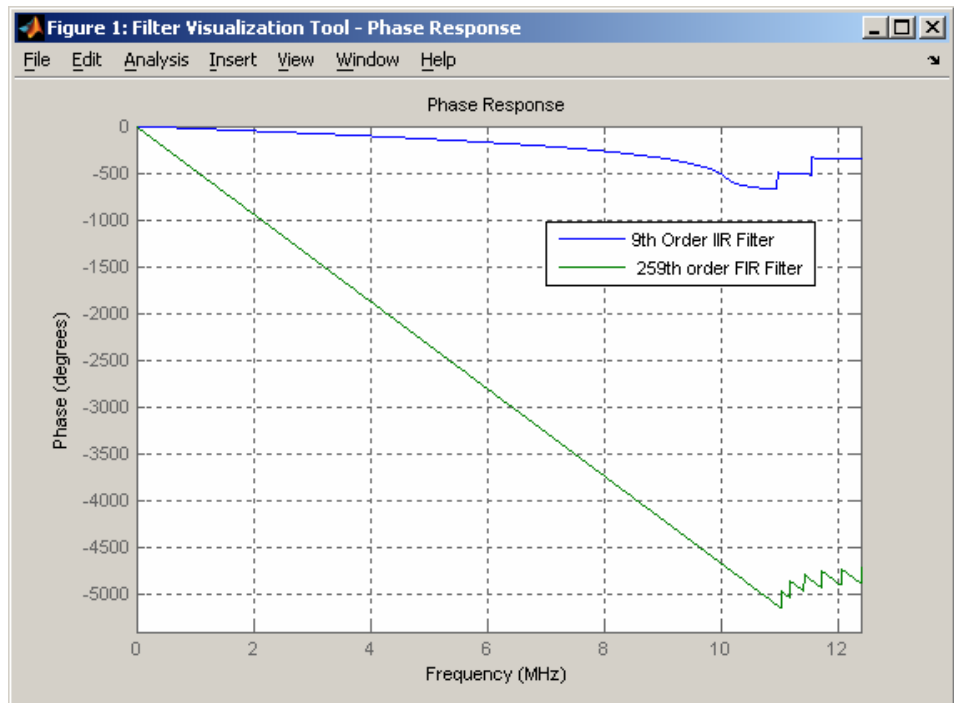


Figure 2: Filter Visualization Tool: Phase Response

For example, a square wave contains odd harmonics of the fundamental frequency and produces a *ringing* output, as shown in Figure 3, because different frequency components have different delays through an IIR filter. However, the latency through the IIR filter is less.

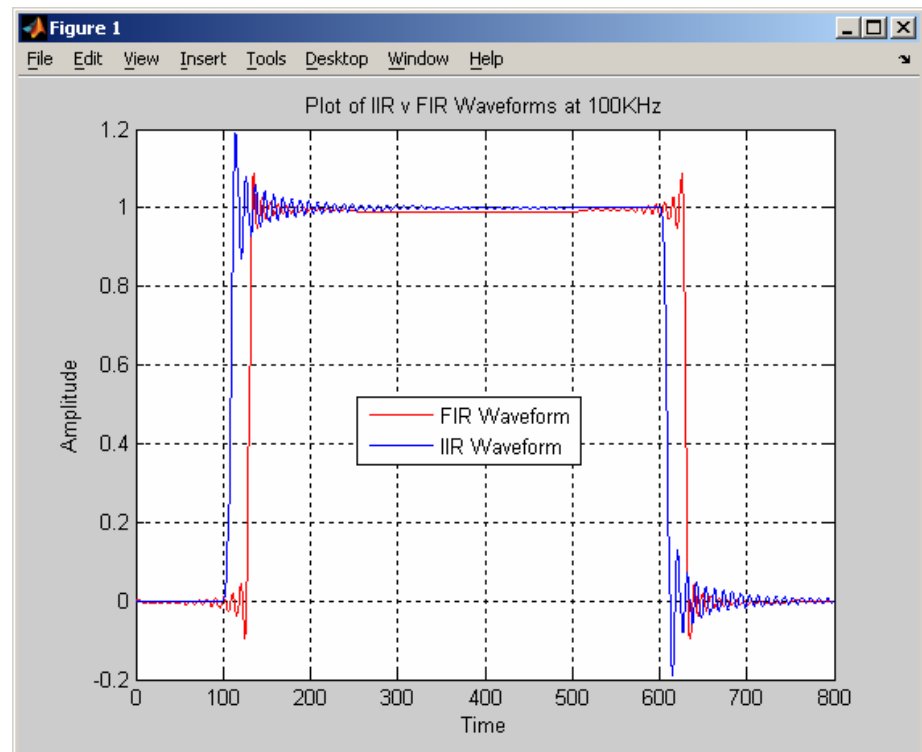


Figure 3: IIR vs. FIR Waveforms at 100 kHz

Classical IIR Filters

A number of IIR filter formats can be used, depending upon the filter characteristics. The plots in [Figure 4](#) show the magnitude response and phase response for the different IIR filters, apart from the Bessel filter.

Elliptic Filter

Also known as Cauer filters, Elliptic filters are equiripple in both the passband and stopband. They generally meet filter requirements with the lowest order of any supported filter type. For a given filter order, passband ripple, and stopband ripple, elliptic filters minimize transition width.

Butterworth

The Butterworth filter provides the best approximation to the ideal lowpass filter response at analog frequencies. Passband and stopband response is maximally flat.

Chebyshev Type I

The Chebyshev Type I filter minimizes the absolute difference between the ideal and actual frequency response over the entire passband by incorporating equal ripple in the passband. Stopband response is maximally flat. The transition from passband to stopband is more rapid than for the Butterworth filter.

Chebyshev Type II

The Chebyshev Type II filter minimizes the absolute difference between the ideal and actual frequency response over the entire stopband by incorporating an equal amount of ripple in the stopband. Passband response is maximally flat. The stopband does not approach zero as quickly as the type I filter (and does not approach zero at all for even-valued filter order n). The absence of ripple in the passband, however, is often an important advantage.

Bessel

Analog Bessel lowpass filters have maximally flat group delay at zero frequency and retain nearly constant group delay across the entire passband. Filtered signals therefore maintain their wave shapes in the passband frequency range. Frequency mapped and digital Bessel filters, however, do not have this maximally flat property. Bessel filters generally require a higher filter order than other filters for satisfactory stopband attenuation. As the MathWorks tools do not support the digital model of the Bessel, this format will not be covered in this document.

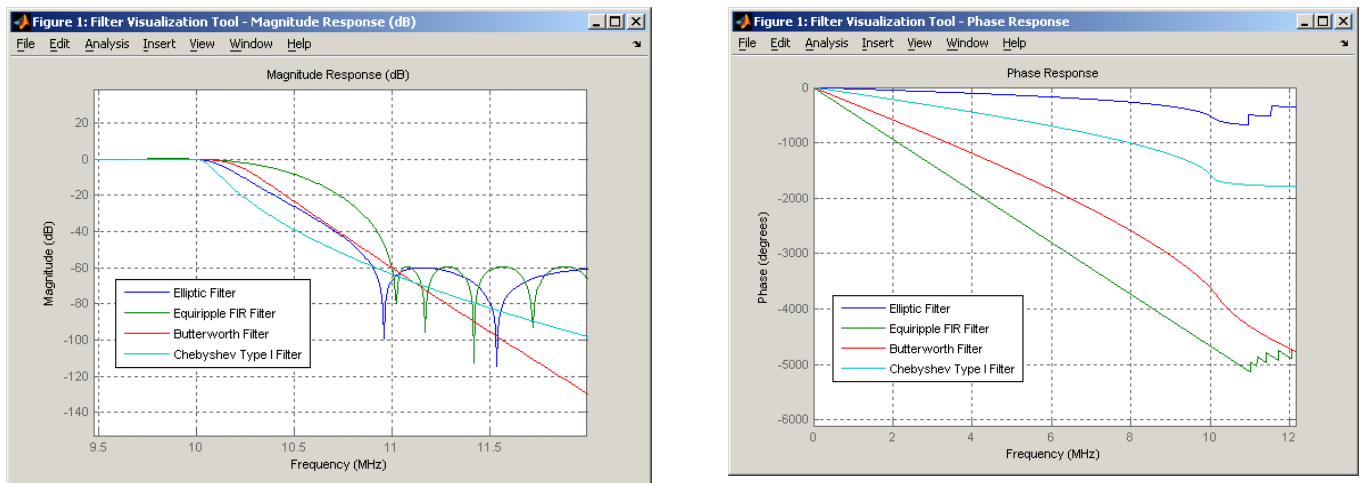


Figure 4: Magnitude and Phase Response for Different IIR Filters

IIR Filter Structure

The IIR filter consists of a forward FIR filter, also known as all-zero filter, comprising of the numerator, or b , coefficients for the zeros, and a feedback FIR for the denominator, or a , coefficients for the poles. A way to express an IIR Filter is as a z -transfer function with numerator coefficients b_i and denominator coefficients a_i .

$$H(z) = \frac{\sum_{i=0}^n b_i z^{-i}}{\sum_{i=0}^m a_i z^{-i}}$$

Equation 1

The time domain expression for the IIR is shown in Equation 2, and it can be seen that some delayed version of the $y(n)$ output is playing a part in the output:

$$y(n) = \sum_{i=0}^n b_{(i)} x(n-i) + \sum_{i=0}^m a_{(i)} y(n-i)$$

Equation 2

$a(i)$ and $b(i)$ are the coefficients of the IIR filter. The expression is now showing summation, multiplication, and subtraction, which are basic DSP building blocks and can be implemented in FPGA architecture using tools like System Generator. The IIR filter can be implemented using different structures. Example structures considered are:

- Direct Form I
- Direct Form II
- Biquad
 - ◆ Direct Form I/II
 - ◆ Cascade Biquad
 - ◆ Parallel Biquad

- Pipelined IIR Biquad

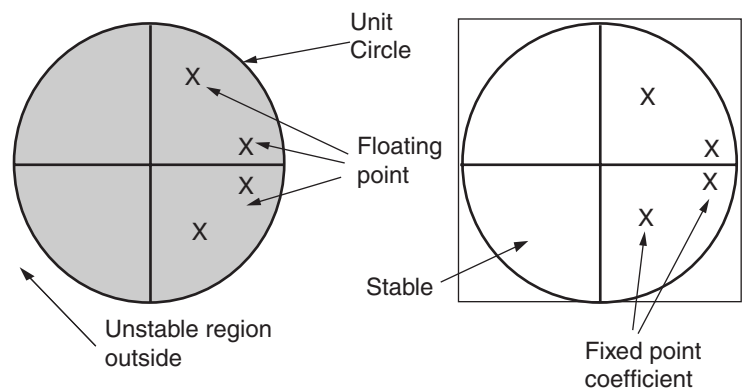
Fixed-Point Implementations

To ensure satisfactory fixed-point operation of the IIR filter, we will examine the following in detail:

- Coefficient Quantization
- Internal Quantization
- Overflow
- Stability

Coefficient Quantization

Coefficient quantization affects the frequency response. To look at the effect of quantization, it is useful to look at the filter in the z-domain and use pole/zero plot. This shows how the zeros (depths in the frequency response plot) and poles (peaks in the frequency response plot) are positioned. In fact, the issue with IIR stability has to do with the denominator coefficients and their positions, as poles, on the pole/zero plot. See [Figure 5](#).



wp330_05_020508

Figure 5: Denominator Coefficients and Pole Positions

The poles for the floating-point version of the plot are shown on the left, and it can be seen that they are within the unit circle, that is, the values of the coefficients are less than 1 and thus in the stable region. After the coefficients are quantized, these poles move, having an effect on the frequency response, and if they move onto the unit circle, that is, the poles equal 1, then potentially it is an oscillator, and if the poles become greater than 1, then the filter definitely becomes unstable. The embedded multipliers in the Spartan®-3 and Virtex®-4 FPGA families allow the coefficients to be up to 18 bits for unsigned and 25 bits on the XtremeDSP™ slice in the Virtex-5 FPGAs. The flexibility of the FPGA fabric allows larger multipliers to be constructed to reduce the effects of coefficient quantization.

Internal Quantization

With a DSP function, there are multiplication and addition/subtraction operations. However, there is bit growth due to these operations, and at some point, the bit widths have to be reduced. Operations like wrapping/saturation for the most significant bits and rounding/truncation for the least significant bits have to be used. More detail can be found in DSP reference books. The rounding process reduces the bit-width, but it is

a source of noise and contributes to output round-off noise and, hence, affects the signal-to-noise ratio. The flexibility of FPGA architecture allows for increasing the word length and reducing the round-off noise. So, utilizing the XtremeDSP slices allows up to 40 bits growth on the Spartan-3A DSP FPGA and 48 bits on Virtex-4 and Virtex-5 FPGAs, with the additional advantage of lower power. Larger bit widths can be handled by cascading the XtremeDSP slice and guidance for this can be found in [Ref 1], [Ref 2], and [Ref 3].

Overflow

For a fixed-point implementation, there is a certain bit width and, hence, a range. As a result of calculations, the filter may exceed its maximum/minimum ranges. For example, a two's complement value of $01111000 (+120) + 00001001 (+9) = 10000001 = (-127)$. So, the large positive number becomes a large negative number. This is known as *wraparound*, which can cause huge errors. Using saturation logic can deal with this situation. In the example, the results would be $01111111 (+127)$.

To minimize the effects of overflows, scaling can be used. Therefore, values can never overflow. There are different kinds of scaling and these tend to be used by DSP processors to fit within their fixed structure. However, this has an effect on the signal to noise ratios.

Stability

Looking at just the feedback path of the reverse filter, it is basically a higher-order polynomial:

$$1 + a_1.Z^{-1} + a_2.Z^{-2} + a_3.Z^{-3} + \dots + a_n.Z^{-n}$$

The MathWorks tools provide mathematical functions like *roots* to help solve these equations and provide the roots of the polynomial. A script can be used to determine the roots of the double-precision polynomial and output the magnitude, and so discover if any of the roots are >1 , which indicates instability. It also indicates the number of poles that are close to the unit circle, and so can highlight which coefficients may need more attention and thus more precision bits.

Higher-Order IIR Filter Coefficient Values

Table 1 shows the coefficients for 9th direct form for the IIR Filter specification used
Table 1: Coefficients for 9th Direct Form

Denominator Coefficient (a_n)		Numerator Coefficient (b_n)	
		b_0	0.00245014558048
a_0	-7.04394978610584	b_1	-0.00966138645865
a_1	22.98981235527600	b_2	0.01939701064155
a_2	-45.39640044672937	b_3	-0.02129070350197
a_3	59.59070437306473	b_4	0.00964412229742
a_4	-53.82417235803681	b_5	0.00964412229742
a_5	33.41364866776762	b_6	-0.02129070350197
a_6	-13.73985515793203	b_7	0.01939701064155
a_7	3.39583699025570	b_8	-0.00966138645865
a_8	-0.38454626044234	b_9	0.00245014558048

Even though this is a 9th order filter, it actually has 19 coefficients; 10 for the numerator and 9 for denominator. This indicates the design challenge, as the numerator coefficients are fractional, whereas the denominator coefficients have an integer range as well as precision. So, both sets of coefficients require fixed-point representation with adequate range and precision. However, the numerator coefficients are symmetrical in this case and thus could be used to advantage in FPGA implementation.

For the numerator coefficients, increasing the bit width does improve the frequency response, but because the filter uses the embedded multipliers within the FPGA fabric, making use of the multiplier width which is 18 bits is recommended. However, to get greater improvement, it may require greater than 18 bits. Examining the binary representation of the coefficients might help. For example, the coefficient value 0.00245014558048 if using 17 bits of precision results in `00_0000_0001_0100_0001` for a two's complement number. In order not to have the embedded multipliers spending time multiplying by all those zeros between the binary point and the first active 1, it is suggested to scale the numerator coefficients by powers of 2 and scale down the resulting products by the same power of 2 value.

For example, 18 bits for the denominator coefficients a_3 requires 7 bits to represent the integer leaving just 11 bits to represent precision, which could lead to instability. One possible option is to increase the width but this would result in 25×18 multiplications, which is acceptable for DSP48E but could take two hardware multipliers in the other FPGA families. Another option is recognizing that $59.59070437306473 = 64 - 4.40929562693527$ and 4.40929562693527 can be represented by `FIX_18_14`, that is, 18 bits value with 14 bits to right of the binary point. The multiplication by 64 is nothing but a shift implemented via wire routing in FPGA.

Direct Form I Structures

The structure can be seen as two FIR filters: one forward FIR filter comprising of the b coefficients and a reverse FIR using the a coefficients. The IIR structure can be re-ordered in an alternative form. See [Figure 6](#). The transpose Direct Form I structure is shown on the right. It has a centre spine consisting of delay elements and adders. There are two multipliers *sections* for the numerator and denominator coefficients feeding the spine.

In the transpose IIR structure, the forward filter is followed by the all-pole filter. So, the signal is attenuated before it is applied to the higher gain of the all-pole filter. The forward filter is defined by the data input bit width.

With the forward filter, there is more delay, hence, the use of pipeline registers. The all-pole section, because of the feedback, is the limiting factor and runs slower. Because there can be a higher clock in the FPGA, multi-cycles are available for the implementation of the forward filter. The forward filter can be implemented using the Xilinx CORE Generator™ FIR Compiler allowing some level of abstraction, that is, down to designing to the lowest level, but providing access to the maximum performance available on the FPGA. For the all-pole filter, the data and coefficient width can be larger and would require larger multipliers.

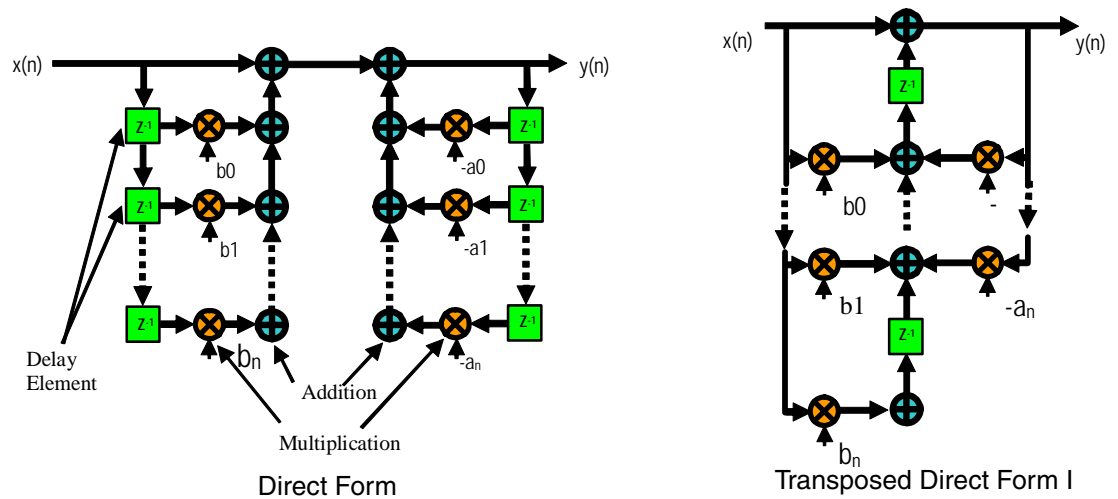


Figure 6: Direct and Transposed Direct Form I Structures

Direct Form II Structures

For the Direct Form I architecture, it was mentioned that the forward and reverse FIR filter stages can be swapped. If this is done, then the center consists of two columns of delay elements. This can be turned into a one column structure, known as *canonical*, meaning it requires the minimum amount of storage. In this case, distributed memory, which is available across the FPGA fabric, can be utilized.

The Direct Form-II structure has the advantage over the Direct Form I as it requires less memory storage for the data samples. These types of filters are an all-pole filter followed by an all-zero (forward) filter. The issue is the high gain of the all-pole section. The adders are larger to handle the potential overflow. The advantage of the reduced storage is offset by the larger adders. Therefore, this structure tends to scale the input to reduce the gain, but this could result in a worse signal-to-noise ratio (SNR) for larger order filters. Therefore, it is best not to use this structure for greater than 2nd order.

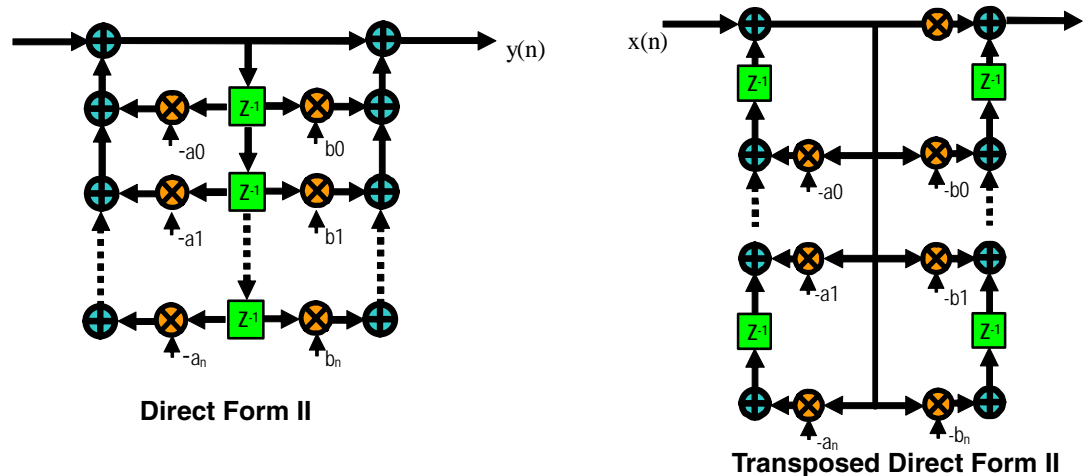


Figure 7: Direct and Transposed Direct Form II Structures

BiQuad

The issue with *N*th Order structures mentioned earlier is that as the order of the filter increases, so does the complexity of the implementation. This leads to worsening range and precision of numbers, and for larger number of taps, it becomes a challenge. The round-off errors that come from all the multiplications and additions contribute to rounding errors. This can be seen when looking at the difference between fixed-point implementation in System Generator and Simulink® models. The structure used to minimize quantization errors is the 2nd order filter, also known as a *biquad*. The transfer function is given in [Equation 3](#).

$$H(z) = \frac{b_0 + b_1Z^{-1} + b_2Z^{-2}}{1 + a_0Z^{-1} + a_1Z^{-2}}$$

Equation 3

The advantage of the 2nd order structure is that it is not so sensitive to coefficient quantization; that is, so many bits are not needed to represent coefficients values. Now the width of the embedded multipliers is sufficient to produce results better than the *N*th order filter. By using the MATLAB FDATool, the 9th order filter can be broken down into a five second-order sections (SOS). The coefficients are shown in [Table 2](#).

Table 2: SOS Coefficients

SOS	Numerator Coefficients			Denominator Coefficients			Scale
	b ₀	b ₁	b ₂	a ₀	a ₁	a ₂	
SOS 1	1	1	0	1	-0.7641	0	0.0362
SOS 2	1	-0.5762	1	1	-1.5405	0.6717	0.3880
SOS 3	1	-1.3257	1	1	-1.5624	0.8241	0.69
SOS 4	1	-1.4972	1	1	-1.5799	0.9269	0.8422
SOS 5	1	-1.5441	1	1	-1.5969	0.9809	0.3002

The range of the coefficients are seen to now within a -2 to a +2 range. While previously there was wide range and precision for the denominator coefficients, a more limited range is now observed. Any increase in bit width can be assigned to the precision. The numerator coefficients are still symmetrical so pre-adders can be used to save multipliers. In this 9th order example, a total of 18 multiplications is required. Some of the multiplications are due to the scale factor for each biquad, and these can be on the input or output of the biquad. All the individual biquad gains can be multiplied together to give an overall gain.

Associated with each of the biquad sections are a couple of other parameters which affect SNR; specifically, scale factor and reordering of the biquad. The MathWorks FDATool has the following tool for use with SOS. See [Figure 8](#).

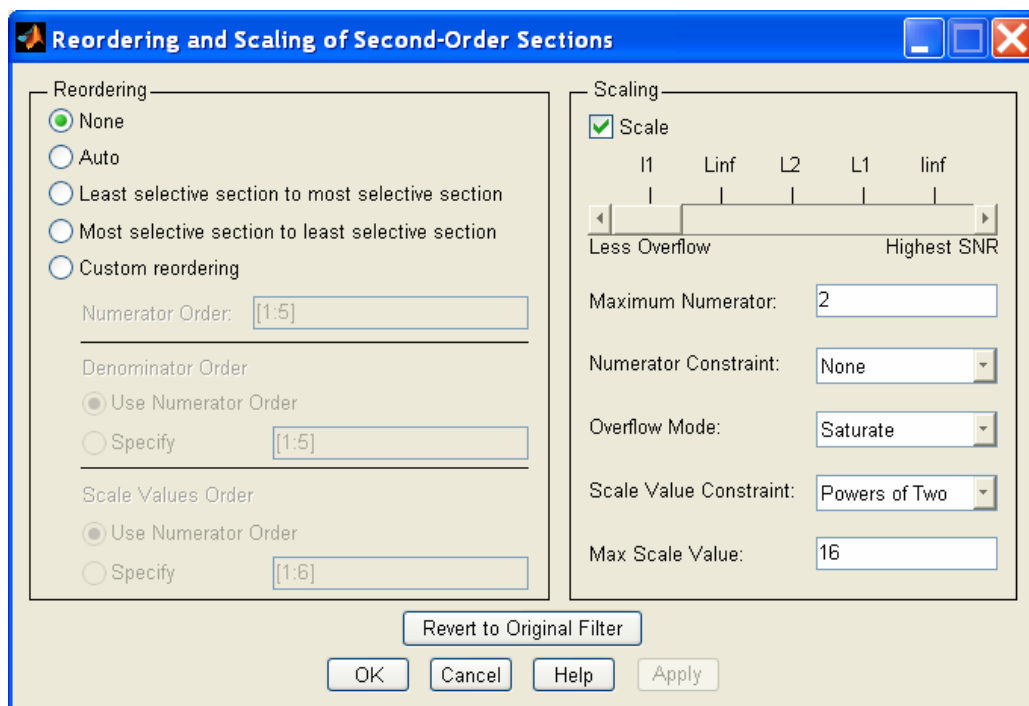


Figure 8: Reordering and Scaling of Second-Order Sections

Scaling

Scaling can be used as a compromise between using the full dynamic range and maximizing SNR and keeping it low enough to minimize overflow. Scaling ensures that values rarely (or never) overflow. Saturation arithmetic can be used to ensure that if overflow occurs, its effects are greatly reduced. Scaling from the frequency response, known as Lp scaling, can be used. In summary, there are three types of scaling: L1, L2, and Linf; each type changes the numerator and scaling values.

- L1 is the most conservative scaling and ensures that there will be no overflow. This is the summation of the absolute values of the impulse response.
- L2 is the absolute sum of the root-mean-square of the impulse response.
- Linf is the maximum amplitude when looking at the frequency response of the transfer response.

For example, for L2 scaling, the SOS coefficients are shown in [Table 3](#).

Table 3: SOS Coefficients for L2 Scaling

SOS	Numerator Coefficients			Denominator Coefficients			Scale
SOS 1	0.76606	-0.4413	0.76606	1	-1.5405	0.6717	0.25
SOS 2	1.5087	-2	1.5087	1	-1.5624	0.8241	0.125
SOS 3	1.3358	-2	1.3358	1	-1.5799	0.9269	0.25
SOS 4	1.2952	-2	1.2952	1	-1.5969	0.9809	0.25
SOS 5	0.50872	0.50872	0	1	-0.7641	0	0.6166

The total number of actual multiplications requiring an embedded multiplier is now 16, because some of the coefficients are powers of 2, which is nothing but a bit shift requiring zero logic. The scaling coefficients could be part of the numerator filter and

thus save another multiplication. In the example shown in Table 3, the scaling has been brought into the numerator coefficients as shown in Table 4

Table 4: Scaling in Numerator Coefficients

SOS	Numerator Coefficients			Denominator Coefficients		
SOS 1	0.0362	0.0362	0	1	-0.7641	0
SOS 2	0.3880	-0.5762*0.3880	0.3880	1	-1.5405	0.6717
SOS 3	0.69	-1.3257*0.69	0.69	1	-1.5624	0.8241
SOS 4	0.8422	-1.4972*0.8422	0.8422	1	-1.5799	0.9269
SOS 5	0.3002	-1.5441*0.3002	0.3002	1	-1.5969	0.9809

This example is for a total of 18 multiplications. Remember the all-pole section of the biquad cannot be run at clock rate. However, there are multiple clock cycles for the forward filter.

BiQuad Structure

The Direct Form I structure is shown in Figure 9 and its time domain difference equation is shown in Equation 4.

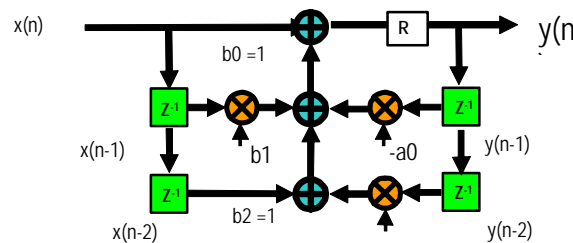


Figure 9: Direct Form I Structure

$$y(n) = b_0.x(n) + b_1.x(n-1) + b_2.x(n-2) + a_1.y(n-1) + a_2.y(n-2)$$

Equation 4

Two data storage areas for the time delayed input data $x(n)$ and output data $y(n)$ are required. The first stage is the all-zero filter which has a smaller gain than the all-pole filter. This means that the overflow logic and the rounding logic is on the output. The suggestion is that the full precision from the multipliers is carried through. The number of multiplications is three, and the number of additions is four. The key factor is that the size of the denominator coefficients can be 18 bits (within the width of the embedded multipliers).

The Direct Form II structure is shown in Figure 10 and time domain difference equation is shown in Equation .

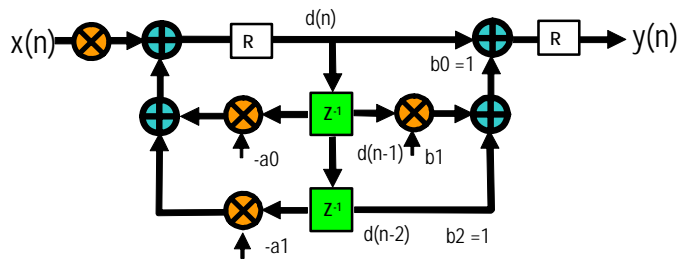


Figure 10: Direct Form II Structure

$$d(n) = x(n) + d(n-1).a0 + d(n-2).a1 \tag{Equation 5}$$

$$y(n) = d(n).b0 + d(n-1).b1 + d(n-2).b2 \tag{Equation 6}$$

This structure is favored by DSP processors because it requires less storage and takes fewer cycles. To reduce the number of numerator coefficients they are scaled, which can result in the following form requiring four multiplications and four additions. There tends to be scaling on the input to reduce the chances of overflow in the all-pole section of the filter for the $d(n)$ signal. The multiplications are 18×18 and the full 36-bit values for the feedback filter can be added together. The denominator coefficients are expressed as $-ve$ values in the diagram, but these can be expressed as positive values. The outputs from the reverse filter can be added together before being subtracted from the $x(n)$ input. At the output of the subtractor, rounding can take place so as to bring the result back within the 18 bits required for the delay stages. With this structure, there are two rounding points **R**: one at the $d(n)$ stage and the other at the $y(n)$ output.

By using Direct Form II structure, the numerator coefficients can be multiplied by the scaling factor. If there is a number of zeros between the binary point and the first 1, then power of two multiplications calculate the result, and division by the same power of two can be used.

Cascaded BiQuad Structure

The basic biquad can be extended so as to provide better attenuation. It involves having more than one biquad cascaded. For higher order filters, several biquads are cascaded. Equation 7 is shown where * represents product, and N is the number of biquads. In this case, it's five.

$$H(z) = \prod_{i=1}^N \left(\frac{b_{i0} + b_{i1}Z^{-1*} + b}{1 + a_{i0}Z^{-1*} + a_{i1}Z^{-2}} \right)$$

Equation 7

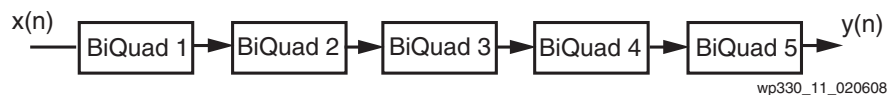


Figure 11: Cascaded BiQuad Structure

The cascaded biquads can be reordered to reduce the round-off noise. However, for N biquads, that becomes N possible options! To address this, we will consider the following Parallel structure option.

Parallel BiQuad Structure

Another structure using biquads is the parallel form as shown in Figure 12. Again this structure has been utilized by processors.

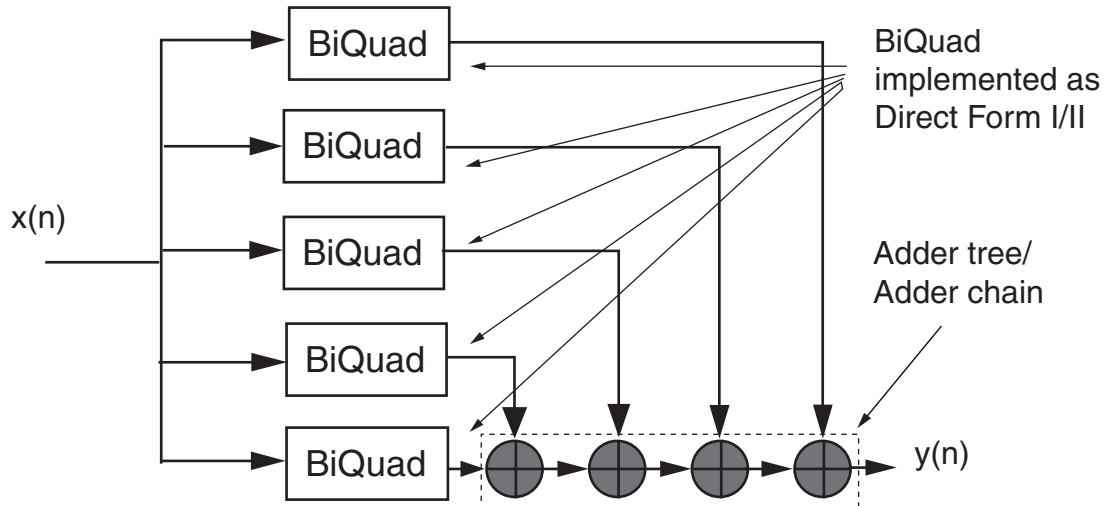


Figure 12: Parallel BiQuad Structure

There is an alternative structure which uses biquads. The equation for parallel form is shown in Equation 8.

$$H(z) = K + \sum_{i=1}^N \left(\frac{c_{i0}Z^{-1} + c_{i1}Z^{-2}}{1 + a_{i0}Z^{-1} + a_{i1}Z^{-2}} \right)$$

Equation 8

To get from cascade structure to a parallel structure, use the MATLAB *residuez* function. This function takes the polynomial coefficients and converts it to partial fractions. Second order sections and their coefficients are shown in Table 5. The denominator values are the same but the numerator coefficients are different.

Table 5:

SOS	Numerator Coefficients			Denominator Coefficients		
SOS 1	0	0.3738	0	1	-0.7641	0
SOS 2	-0.5048	0.4457	0	1	-1.5405	0.6717
SOS 3	0.1439	-0.2083	0	1	-1.5624	0.8241
SOS 4	0.0130	0.0383	0	1	-1.5799	0.9269
SOS 5	-0.0170	0.0058	0	1	-1.5969	0.9809

The structure is shown in Figure 12, with the biquads in parallel. If they are Direct Form I structure, even though more data storage is needed for the forward filter, the input storage for the forward filter is common to all phases of the parallel biquad.

Thus, optimization can be implemented here. However, the forward filter numerator coefficients are not symmetrical, but there are fewer of them.

The denominator section stays the same and the coefficients are the same value. So, rounding can be done prior to storing the internal value in the data storage, and then being applied to the adder stage. The adder stage can be implemented as an adder tree, or adder chain. Virtex-4, Virtex-5, and Spartan-3A DSP FPGAs have XtremeDSP slices in the fabric, which are well suited to adder chains as well as providing high performance. See [Ref 1], [Ref 2], and [Ref 3].

Higher Performance IIR Filters

With an IIR filter, the key requirement is that the calculations are performed within a sample period. Unlike the forward filter, the feedback filter cannot have any sample delays in it. So the question is: how can the feedback filter speed be increased. There are a couple of options:

1. **Using multiple cycles:** This option uses the fact that there is a higher clock in the system, or there could be a higher clock, introduced via the digital clock managers in Xilinx FPGA architecture. Then more pipelining stages can be added and still meet the requirement for getting the sample calculated for the next sample period. However, there is a limit on how many stages can be pipelined. For example, for a Virtex-4 FPGA, a clock rate of 400 MHz and a clock enable every four clock periods can result in a performance of 100 MHz.
2. **Pipelining the feedback filter:** This option adds extra canceling poles and zeros in the Z-plane to the original transfer function. There are a couple of methods:
 - a. **Cluster Lookahead Method:** See [Ref 4] for more details, but essentially for this method, the active poles are clustered together and the lower coefficients are set to zero, leaving just the sample registers. However, this method can result in the final transfer function being unstable, even though the original filter was stable for a low number of pipeline stages, and is not investigated further.
 - b. **Scattered Lookahead:** See [Ref 5]. This method holds that for each pole $M-1$ additional poles/zeros are introduced at equal angles around the Z circle. This method ensures that the new transfer function is still stable. As an example for $M = 3$ stages of pipelining, adding poles and zeros at $ae^{\pm j}2\pi/M = ae^{\pm j}$ are shown in Figure 13.

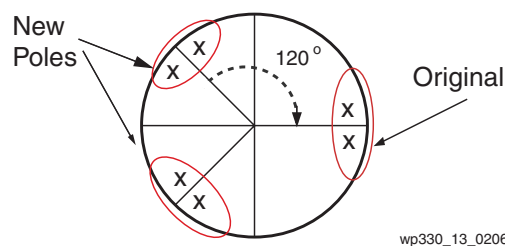


Figure 13: Scattered Lookahead using Extra Poles

So, considering just the all-pole section of the biquad:

$$H(z) = \frac{1}{1 + a_1 Z^{-1} + a_2 Z^{-2}}$$

Adding the extra poles/zeros gives Equation 9.

$$H(z) = \frac{1 - a_0 Z^{-1} + (a_0^2 - a_1)^{-2} - (a_0 \cdot a_1) Z^{-3} + a_1^2 Z^{-4}}{1 + (a_0^3 + 3a_0 \cdot a_1) Z^{-3} + a_1^3 Z^{-6}}$$

Equation 9

As shown, the poles are scattered, that is, the zero coefficients between Z^{-3} and Z^{-6} . The intermediate Z still exist as registers, but there is no requirement for multiplication by a coefficient. These are the registers that are used as pipeline registers to speed up the recursive path. Looking at just the recursive section, the implementation is shown in Figure 14. The original recursive is shown Figure 14 (a), and the result of adding the extra poles giving the extra pipeline registers is shown in Figure 14 (b). The structure is changed to a transpose structure and the multiplier and adder are now registered, shown in Figure 14 (c).

However, there is an increase in computation for the numerator as it is now a 4th order numerator, shown in Equation 9.

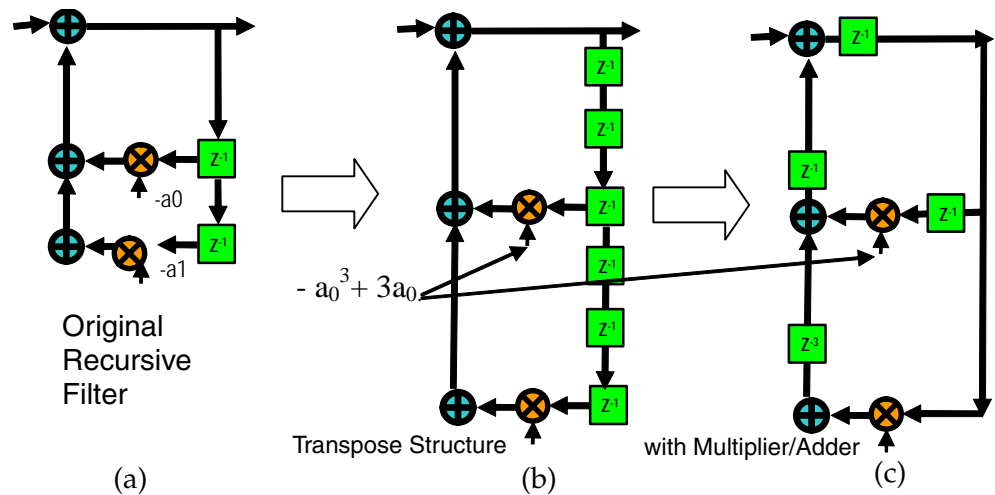


Figure 14: Pipelined IIR Filter

Multi-Cycle BiQuad

If the requirement is for less number of multipliers using the Direct Form II as an example, it is possible to create a multi-cycle of the equation which uses 5 cycles where one multiplier per biquad is used.

Cycle 1: Accum = x(n) * scaling

Cycle 2: Accum = Accum + d(n-1) * a0

Cycle 3: Accum = Accum + d(n-2) * a1

Cycle 4: Accum = Accum + d(n-1) * b1

Cycle 5: Accum = Accum + d(n-2) * b2. And output y(n).

By using an XtremeDSP slice in the FPGA and the appropriate opcode, the multi-cycle biquad can be implemented. With the high performance of the XtremeDSP slices, higher sample rates can be supported.

IIR Filter Structure Observations

IIR filter structure observations are as follows:

- FIR Filter
 - ◆ Normal and easy to implement. The obvious choice implemented by many people.
 - ◆ Use cores to implement filter.
- Higher Order IIR Filter
 - ◆ Nth order filter implemented with MAC for forward filter and parallel transpose for all-pole section.
 - ◆ The forward filter can be implemented using FIR compiler IP which supports up to 32 bits input data.
 - ◆ Frequency response, peak error, and error variance not as good as FIR filter.
 - ◆ Complications due to bit growth and overflow. Would consider this for low order filters.
- Cascade Biquad
 - ◆ Filter implemented as a series of 2nd order filters. Gives better results than Nth order for similar number of multipliers.
 - ◆ Four multipliers per biquad. Limit on performance is the feedback path.
 - ◆ Can be viewed as a basic IP block and cascaded as many times as required.
 - ◆ Filter also can be implemented as a series of 2nd order filters using multi-cycles to do the computation with one multiplier per biquad. Limit on performance is the feedback path. But, as FPGA clock rates increase can support increasing sample rates.
 - ◆ Can be implemented in either Direct Form I or Direct Form II.
- Parallel Biquad
 - ◆ Filter implemented as 2nd order filters in parallel.
 - ◆ The number of numerators is reduced and the data samples shared. The adder stage can be implemented using adder chain or accumulator. Advantage is that adder chain can deal with odd number of biquads. Time-division-multiplexed techniques can be used for the filter to reduce area with adder stage implemented using accumulator.
 - ◆ Use the XtremeDSP slice to save on area and increase the performance.
- Pipelined Filter
 - ◆ Feedback section pipelined to allow increased performance.
 - ◆ Dependent upon coefficient and data width, but as these increase more pipeline stages are needed, and the complexity of the forward filter increases.
 - ◆ The numerator order increases requiring more multipliers. However, due to higher clocks, this can be implemented using MAC techniques. Design is simplified by the use of FIR compiler IP.

Implementation of IIR Filters in Xilinx FPGAs

Although there is not a specific IIR Filter IP core, the basic DSP functions, that is, addition, accumulation, multiplication, and the more complex functions like filtering are available in the Xilinx CORE Generator software, System Generator software, and AccelDSP tools.

The CORE Generator software delivers a library of parameterizable and fixed netlist LogiCORE IP cores with corresponding data sheets, designed and supported by Xilinx. The CORE Generator software can be accessed as a standalone tool or from the Xilinx ISE® design environment. Generating a core is straightforward. The output is an optimized core for the targeted FPGA device family that includes the following files:

- A tailored Xilinx implementation netlist
- VHDL or Verilog instantiation code
- VHDL or Verilog wrapper for simulation support
- A symbol for schematic capture tools

System Generator for DSP is a highly productive design environment for the development and prototyping of DSP systems using FPGAs. System Generator software enables the use of The MathWorks Simulink®/MATLAB modeling environments for FPGA design by providing a Xilinx-specific block set for use within the Simulink modeling environment. DSP algorithm developers who do not know HDL design techniques can quickly capture their designs and accelerate their simulations using push-button hardware co-simulation flows.

The AccelDSP Synthesis Tool is a high-level MATLAB language-based tool for designing DSP blocks for Xilinx FPGAs. This tool also has the ability to be imported into a System Generator design. The tool automates floating-to-fixed-point conversion, generates synthesizable VHDL or Verilog, and creates a test bench for verification. By offering such alternatives, the user can design in the environment that is most comfortable and can build systems without necessarily having to learn completely new FPGA design methodologies and hardware design languages, but still have access to the high performance architecture.

Conclusions

This white paper has presented different structures of IIR filters and the issues that must be considered when implementing on the latest FPGA architectures. Using FPGAs allows flexibility in implementation depending upon sample rates, and freedom of choice between the hard IP, like the XtremeDSP slices, and the general FPGA fabric if so required. Using The MathWorks with System Generator and AccelDSP tools adds an extra advantage, because it allows quick exploration and investigation of different techniques and implementations.

In building an IIR filter, even though the higher order structures can be built and the transpose structure can offer advantages because of the adder chain, the risks due to coefficient quantization might be too much. A starting point could be the Biquad, either in a cascaded or a parallel architecture. If higher performance is required and the extra area can be tolerated, then using higher clock rates and multiple cycles, or pipelining IIR filters, is suggested.

References

1. *Xilinx Virtex-5 FPGA XtremeDSP Design Considerations User Guide* (UG193)
2. *Xilinx Virtex-4 FPGA XtremeDSP Design Considerations User Guide* (UG 073)
3. *Xilinx XtremeDSP DSP48A for Spartan-3A DSP FPGAs User Guide* (UG 431)
4. H.H.Loomis and B. Sinha, *High Speed Recursive Digital Filter Realization Circuits, Systems and Signal Processing*, Vol. 3, pp. 267-294, 1984.
5. K.K. Parhi and D.G. Messerschmitt, *Pipeline Interleaving and Parallelism in Recursive Digital Filters – Parts I and II*, IEEE Trans Acoustic Speech, Signal processing

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
03/05/08	1.0	Initial Xilinx release.
04/07/08	1.1	Changed “ ≤ 1 ” to “ > 1 ” in “Stability.”
08/10/09	1.2	Frequency changed to 10 and 11 MHz. Updated Equation 5 and added Equation 6.

Notice of Disclaimer

The information disclosed to you hereunder (the “Information”) is provided “AS-IS” with no warranty of any kind, express or implied. Xilinx does not assume any liability arising from your use of the Information. You are responsible for obtaining any rights you may require for your use of this Information. Xilinx reserves the right to make changes, at any time, to the Information without notice and at its sole discretion. Xilinx assumes no obligation to correct any errors contained in the Information or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE INFORMATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS.