DESIGNING FIXED-POINT IIR FILTERS USING SIMULINK

M. Christensen and F. Taylor
University of Florida

Abstract

Today, the availability of design of commercially off-the-shelf (COTS) design software has greatly simplified the design and analysis of infinite impulse response (IIR) digital filters. The successful implementation of a fixed-point IIR, however, remains problematic. The fundamental problem is controlling degrading finite wordlength effects, which can take the form of damaging run-time register overflow and arithmetic roundoff errors. The severity of such effects is directly influenced by a number of choices, including filter architecture and arithmetic precision. State variables provide a means of facilitating the analysis of finite wordlength effects in the context of specific architectural choices. In this paper, IIR design methodology, state variables, and finite wordlength analysis techniques are fused together to define a viable fixed-point IIR design outcome.

I. Introduction

An important DSP infrastructure element is the fixed-coefficient infinite impulse response (IIR) filter. Fixed-point IIRs can achieve a high frequency selectivity with a relatively low-complexity solution. As such, IIRs are often the technology of choice in realizing a range of frequency dependent applications including tone detection, spectral filtering, noise rejection, digital control, and so forth. Today’s IIRs can be designed using any number of commercially available software packages (e.g., Mathworks Matlab) which translate a filter’s frequency domain specifications into a transfer function having the following form:

\[ H(z) = k \frac{\sum_{i=0}^{N} b_i z^{-i}}{\sum_{i=0}^{N} a_i z^{-i}} = k \prod_{i=1}^{N} \frac{z - z_i}{z - p_i} \]

To illustrate, Matlab’s cheby2 function creates a Chebyshev-II lowpass IIR filter using the command \( [b,a] = \text{cheby2}(N,Rs,Wn) \). Upon execution, cheby2 creates an \( N \)th order lowpass digital Chebyshev II filter with a normalized cutoff frequency \( Wn \) and a stopband ripple of \( Rs \) dB. The filter program returns a vector \( b \) of length \( N+1 \) that contains the numerator coefficients (i.e., \( b \)), defined by Equation 1. Similarly, the vector \( a \) is of length \( N+1 \) and contains the denominator coefficients \( a \). For those preferring to use GUIs (see Figure 1), the Filter Design and Analysis Tool (FDATool), included in the Mathworks Signal Processing Toolbox, provides access to a collection of filter design options that can be used to design classical IIRs:

- Butterworth digital filter design
- Chebyshev Type I filter design (passband ripple)
- Chebyshev Type II filter design (stopband ripple)
- Elliptic (Cauer) filter design
- Discrete-time filters
- Yule Walker recursive digital filter
The availability of COTS filter design software has greatly simplified the design process, leaving the true design challenge to be one of physical implementation. For financial and performance reasons, fixed-point solutions are generally preferred over floating-point instantiations. Unfortunately, fixed-point designs are highly susceptible to a range of finite wordlength effects that can seriously degrade IIR performance and, in some cases, render a design unusable. As a result, extreme care is needed to insure the final outcome meets the target design requirements.

II. Finite Wordlength Effects

Table 1 summarizes finite wordlength effects in order of their severity. The goal of the fixed-point IIR design engineer is to maximize performance while minimizing finite wordlength effects, beginning with register overflow. If register overflow occurs, the system will behave in an unpredictable non-linear manner, introducing potentially large run-time errors. The effects of register overflow can be mitigated to some degree by using saturating arithmetic. A saturating arithmetic unit "clamps" a register's contents at the register's extreme values if overflow occurs. To illustrate, if an arithmetic unit is designed to operate linearly over $X \in [-M, M]$, then a saturating arithmetic unit will map an overflow condition to the register's maximum or minimum, as value as shown in Figure 2. Even with saturating arithmetic, the effect of register overflow can be devastating. It is therefore incumbent on the IIR designer to protect the filter against this type of error through the proper management of filter data and architectural choices.
Table 1: Digital Filter Finite Wordlength Effects

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register overflow</td>
<td>Occurs when the dynamic range limit of a filter’s register is exceeded. The effect is called register overflow, or saturation, and can introduce potentially large errors.</td>
</tr>
<tr>
<td>Arithmetic roundoff</td>
<td>Arithmetic roundoff errors are the result of imprecise arithmetic which results in and outcome of reduced precision.</td>
</tr>
<tr>
<td>Coefficient roundoff</td>
<td>Coefficient roundoff errors occur when the filter’s real coefficients are converted into digital number through quantization.</td>
</tr>
<tr>
<td>Zero input limit</td>
<td>Zero input limit cycling refers to a condition where the response of a stable filter does not converge to zero when the input is zero.</td>
</tr>
</tbody>
</table>

In the absence of register overflow errors, the system will behave linearly but possibly with reduced precision due to the accumulation of various arithmetic errors. It is expected that the filter design engineer is able to quantify and control the effects of arithmetic errors to ensure that the final outcome meets some minimum precision requirement. The error quantification process begins with the statistical definition of how rounding (quantizing) a signed real number $x$ into an $B$-bit fractional fixed-point fractional representation $X$ introduces rounding errors. It is assumed that the signed $B$-bit fractional fixed-point data format consists of a sign bit, $I$ integer bits, and $F$ fractional bits having a format $X \Rightarrow [\pm : I \Diamond F]$, where $\Diamond$ denotes the binary-point location and $B = I + F + 1$. The dynamic range of $X$ is bounded by $\pm 2^I$ and the precision associated with the fixed-point number is defined in terms of the weight of the least significant (fractional) bit (LSB). The physical value of the least significant bit is the quantization step-size, denoted $Q$, which is generally established by the system’s front-end analog to digital converter (ADC). Consider, for illustrative purposes, a $B$-bit double ended ADC that converts an analog signal $x$, where $|x| \leq V$ (volts), into a signed fixed-point number $X$. The quantization step size is given by $Q = 2V/2^B$ and is measured in volts/bit. A double ended 16-bit ADC can therefore convert a $\pm 16V$ signal into a signed 16-bit fixed-point number having a quantization step size $Q = 32/2^{16} = 2^{-11} \approx 0.488$ mV/bit. The ADC output $X$ would be coded as a signed 16-bit fixed-point number having $I = 4$ integer bits and $F = 11$ fractional bits. The roundoff error is defined to be the difference between a real number and its fixed-point equivalent. The error is statistically modeled to be a uniformly distributed random process defined over $\varepsilon = [-Q/2, Q/2)$. Based on this classic modeling convention, the error variance is given by

$$
\sigma^2 = \text{var}(\varepsilon) = Q^2 / 12 .
$$

2.

The standard error deviation, $\sigma$, is often interpreted in bits as $\log_2(\sigma)$. That is:

$$
\sigma \text{ (bits)} = \log_2(\sigma) = \log_2(Q) - \log_2(\sqrt{12}) = \log_2(Q) - 1.78
$$

3.

For the case where $Q = 2^{-11}$, $\sigma \text{ (bits)} = \log_2(\sigma) = \log_2(Q) - \log_2(\sqrt{12}) = -11 - 1.78 = -12.78$ bits. That is, the fractional statistical precision of the 16-bit word with 11 fractional bits, but 12.78 fractional bits.
To illustrate, consider the **SAXPY** \((S=AX+Y)\) or multiply-accumulate (**MAC**) unit shown in Figure 3. Suppose that \(X\) and \(A\) are \(B = 16\)-bit signed fixed-point numbers where \(I = 4\) integer bits, leaving \(F = 11\) fractional bits. In addition, assume that the data is defined in terms of a quantization step-size \(Q = 0.488\) mV/bit (the weight of the LSB). A signal represented in this system is therefore bounded by \(\pm Q2^{15} = \pm 16\) V. The full precision multiplier outcome can be modeled as a 32-bit number, which may be rounded to \(B_0\)-bits before being sent to the accumulator (e.g., \(B_0=16\)-bits). The full precision \((B_0=2B=32\)-bit\) product can also be sent, without rounding, to an extended precision accumulator would have wordwidth of \(B_1=2B+\Delta B\) word width. The presence of extended precision accumulators is, in fact, a attribute common to many commercial DSP solutions. A conservative design assumes that the accumulator word width is sufficient to encounter a worst-case signal condition without producing register overflow. To illustrate, suppose the worst-case outcome is bounded by \(\pm S\) and \(|S| \leq 2^{B_1}\), where \(|V| \leq 2^{\Delta B}\), the extending the accumulator word width by an addition \(\Delta B\)-bits will insure that the accumulator will not overflow during run-time. The extended precision accumulator can be, if desired, rounded to \(B_2\)-bits of precision upon completion of a filter’s multiply-accumulate cycle. Unfortunately, determining the worst-case gain requirements for an arbitrary IIR is, in general, a challenging problem. This problem is addressed in Section VII.

### III. IIR Architecture

The implementation of a digital filter is defined in terms of its architecture. Architecture refers to how a filter is constructed using the primitive building block elements such as shift registers, memory, multipliers, and adders. Many DSP engineers are aware of only one or a few possible filter architectures. There are, in fact, many architectures found in common use. Some are application specific, whereas others have general implications. Some are recognized to better control finite wordlength effects, whereas others emphasize reduced complexity and speed. Included in the list of common architectures are:

- Direct I
- Direct II
- Cascade
- Parallel
- Normal Cascade
- Normal Parallel
- Lattice/Ladder
- Wave
- Bi-quadratic (Biquad)

The two most popular choices are Direct II and Cascade. However, all are capable of implementing a given transfer function \(H(z)\) and, if implemented in floating point, would have identical input-output behavior. This is not the case when designs are implemented in fixed-point. Finite wordlength effects are strongly influenced by the choice of architecture. Regardless
of the architecture selection, the study of finite wordlength effects requires that an assessment framework be established to quantify the internal behavior of a filter.

IV. State Variable Model

In order to quantify the performance of a specific architecture, a mathematical model is needed that monitors register behavior. This is normally assumed to be a state variable model. A state in a state variable model represents the contents of a filter’s registers. In general, an \( N \)th order at-rest (zero initial condition) digital IIR filter has a state variable representation given by:

State equation: \( x[k+1] = A x[k] + b v[k] \)
Output equation: \( y[k] = c^T x[k] + d u[k] \),

A state variable model provides a wiring diagram for the filter, which is analogous to a net list. Each unique architecture is associated with a state 4-tuple \([A, b, c, d]\), which identifies how each element in the state model communicates with other states. A graphical interpretation of the state variable model is presented in Figure 4 and consists of:

- \( v[k] \) the input,
- \( y[k] \) the output,
- \( x[k] \) is an \( N \)-dimensional state vector,
- \( A \) is an \( N \times N \) state matrix,
- \( b \) an input \( N \) vector input coefficient vector,
- \( c \) an input \( N \) vector output coefficient vector, and
- \( d \) an input-output scalar gain.

It can be noted that the actual input, \( u[k] \), is scaled by \( k \) of Equation 1, to produce an input that is presented to the state variable model. That is, the state variable model is ignorant of any input scale factor.

V. Direct II Architecture

A general monic \( N \)th order IIR has a transfer function \( H(z) \) that can be factored to read:
The input gain constant \( k \) is applied to the input signal and is otherwise ignored by the state model. The transfer function defined by Equation 5 defines a Direct II architecture which has a state variable representation \([A, b, c, d]\) given by:

\[
H(z) = k \frac{N(z)}{D(z)} = k \frac{\sum_{i=0}^{N} b_i z^{-i}}{1 + \sum_{i=0}^{N} a_i z^{-i}} = k \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} = k \left( b_0 + \left( b_1 - b_0 a_1 / a_0 \right) z^{-1} + \cdots + \left( b_N - b_0 a_N / a_0 \right) z^{-N} \right) \frac{1}{1 + \left( a_1 z^{-1} + \cdots + a_N z^{-N} \right)}. \]

The input gain constant \( k \) is applied to the input signal and is otherwise ignored by the state model. The transfer function defined by Equation 5 defines a Direct II architecture which has a state variable representation \([A, b, c, d]\) given by:

\[
A = \begin{bmatrix}
-a_1 & -a_2 & -a_3 & \cdots & -a_{N-1} & -a_N \\
1 & 0 & 0 & \cdots & 0 & 0 \\
0 & 1 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & 0 \\
\end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad c^T = (v_N = b_N - b_0 a_N \quad v_{N-1} = b_{N-1} - b_0 a_{N-1} \quad \cdots \quad v_1 = b_1 - b_0 a_1) \quad d = b_0.
\]

The conversion of a transfer function \( H(z) \) into a Direct II architecture can be performed using the MATLAB function TF2SS (transfer function to state-space conversion) or ZP2SS (zero-pole to state-space conversion). The TF2SS format is given by \([A, B, C, D] = TF2SS(N, D)\). To illustrate, consider a monic 3\(^{rd}\) order IIR digital IIR filter having a transfer function given by \( H(z) \) \((k=1)\), which can be reduced using Equation 5, to read:

\[
H(z) = \frac{N(z)}{D(z)} = \frac{1 - 0.5 z^{-1} - 0.315 z^{-2} - 0.0185 z^{-3}}{1 - 0.5 z^{-1} + 0.5 z^{-2} - 0.25 z^{-3}} = 1 + \frac{0.0185 z^{-3}}{1 - 0.5 z^{-1} + 0.5 z^{-2} - 0.25 z^{-3}} \]

The response of TF2SS is shown below and graphically interpreted in Figure 5.

\[
NUM = [1 \quad -0.5 \quad -0.315 \quad -0.0185]; \quad DEN = [1 \quad -0.5 \quad 0.5 \quad -0.25];
\]

\[
[A, B, C, D] = TF2SS(NUM, DEN)
\]

\[
A = \begin{bmatrix}
-1 & -0.5 & -0.315 & -0.0185 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 1 \\
\end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad C^T = (v_0 = b_0 - b_0 a_0 \quad v_1 = b_1 - b_0 a_1 \quad v_2 = b_2 - b_0 a_2 \quad \cdots \quad v_3 = b_3 - b_0 a_3), \quad D = b_0.
\]

Figure 5: Third-order state-determined digital filter reduced to a Direct-II architecture showing state assignments.
\[
\begin{bmatrix}
0.5000 & -0.5000 & 0.2500 \\
1.0000 & 0 & 0 \\
& 0 & 1.0000
\end{bmatrix}
\}
\{ a_{11}, a_{12}, a_{13} \text{ Figure 5}\}
\begin{bmatrix}
x_2[k+1] = x_1[k] \text{ in Figure 5}\n\end{bmatrix}
\begin{bmatrix}
x_3[k+1] = x_2[k] \text{ in Figure 5}\n\end{bmatrix}
\]
\[B = \]
\[1 \quad \{ b_1 \text{ in Figure 5}\}
0
0
\[
C = 
\begin{bmatrix}
0 & -0.8150 & 0.0650 \quad \{ c_1, c_2, c_3 \text{ in Figure 5}\}
\end{bmatrix}
\[
D = 
1 \quad \{ d \text{ in Figure 5}\}
\]

### VI. Cascade Architecture

A general \(N\)th order fixed-coefficient IIR can also be factored as

\[
H(z) = \frac{N(z)}{D(z)} = k \frac{\prod_{j=1}^{\infty} (z - z_j)}{\prod_{j=1}^{\infty} (z - p_j)} = k \prod_{k=1}^{Q} \frac{b_{1k} + b_{2k} z^{-1} + b_{3k} z^{-2}}{1 + a_{1k} z^{-1} + a_{2k} z^{-2}}, \quad 7.
\]

where \(H(z)\) is the \(i\)th of \(Q\) first or second order subfilters with zeros \(z\) and poles \(p\). The input scale factor \(k\) is again ignored by the state model. First order subfilters are defined in terms of only real poles and second order subfilters are defined by complex conjugate poles pairs which, when combined, create a second order filter having only real coefficients. As a general rule, pole and zeros are combined with nearest poles (proximity rule) to define each of the \(Q\) subfilters. The \(i\)th subfilter has a state variable representation \([A, b, c, d]\) along with an attendant state assignment (i.e., \(x[k]\)). For a cascade architecture, the output of the \((i-1)\)st subsystem is the input to the \(i\)th subsystem. Therefore:

<table>
<thead>
<tr>
<th>Stage</th>
<th>State model</th>
<th>Output model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (x_1[k+1]) = (A_1x_1[k] + b_1u[k])</td>
<td>(y_1[k] = c_1^\top x_1[k] + d_1u[k])</td>
<td></td>
</tr>
<tr>
<td>2 (x_2[k+1]) = (A_2x_1[k] + b_2u[k])</td>
<td>(y_2[k] = c_2^\top x_2[k] + d_2u[k])</td>
<td></td>
</tr>
<tr>
<td>3 (x_3[k+1]) = (A_3x_2[k] + b_3u[k])</td>
<td>(y_3[k] = c_3^\top x_3[k] + d_3u[k])</td>
<td></td>
</tr>
</tbody>
</table>

and so forth. This process can be formalized in the state determined form shown in Equation 9.

\[
A = 
\begin{bmatrix}
A_1 & 0 & 0 & \ldots & 0 \\
b_1c_1^\top & A_2 & 0 & \ldots & 0 \\
b_2d_1c_2^\top & b_2c_2^\top & A_3 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
(b_0d_{o_1}d_{o_2} \ldots d_{o_2}c_1^\top) & b_0d_{o_1}d_{o_2} \ldots d_{o_2}c_2^\top & b_0d_{o_1}d_{o_2} \ldots d_{o_2}c_3^\top & \ldots & A_0
\end{bmatrix}
\]
The Matlab program \texttt{tf2sos} converts a digital filter’s transfer function $H(z)$ into first or second-order subsystems $H_i(z)$ using the format $[\text{sos}, \text{g}] = \text{tf2sos}(N, D)$. If the IIR filter is given in terms of its pole-zero distribution, the MATLAB program \texttt{zp2sos} and its attendant syntax, $[\text{sos}, \text{g}] = \text{zp2sos}(z, p, k)$ can be used to partition the filter into cascade sections. In either case, \text{SOS} is a $Q$-by-6 matrix of subsections transfer functions $H_i(z)$, having poles and zeros matched by their proximity, and is given by:

$$
\begin{bmatrix}
  b_{01} & b_{11} & b_{21} & 1 & a_{11} & a_{21} \\
  b_{02} & b_{12} & b_{22} & 1 & a_{12} & a_{22} \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  b_{0Q} & b_{1Q} & b_{2Q} & 1 & a_{1Q} & a_{2Q}
\end{bmatrix}
$$

To illustrate, consider again the 3rd order IIR digital having the transfer function

$$
H(z) = \frac{N(z)}{D(z)} = \frac{1-0.5z^{-1} - 0.315z^{-2} - 0.185z^{-3}}{1 - 0.5z^{-1} + 0.5z^{-2} - 0.25z^{-3}},
$$

\((k=1)\) which is interpreted by \texttt{tf2sos}(N, D) as follows:

\begin{verbatim}
NUM=[1 - .5 - .315 -.0185];  DEN=[1 -.5 .5 -.25];  [SOS,G]=TF2SOS(NUM,DEN)
SOS =
  1.0000  -0.8813     0      1.0000   -0.5000     0      {H_1(z)}
  1.0000   0.3813   0.0210   1.0000    0.0000   0.5000   {H_2(z)}
G = 1
\end{verbatim}

The outcome defines two sections of the cascade transfer filter, namely:

$$
H(z) = G H_1(z) H_2(z) = \frac{1-0.8813z^{-1} + 0.3813z^{-1} + 0.021z^{-2}}{1 + 0.5z^{-2}}
$$

Once the $Q$ subfilters have been defined, the individual transfer functions can be mapped into a state determined model using the Matlab program \texttt{sos2ss} and the format $[A, B, C, D] = \text{sos2ss}(\text{SOS}, \text{G})$ for each individual subfilter (note $c = c^T$). To illustrate, the production of the state models for $H_1(z)$ and $H_2(z)$ for the 3rd order example are:

\begin{verbatim}
SOS1=SOS(1,1:6);  \{first IIR section $H_1(z)$\}
[A, B, C, D]=SOS2SS(SOS1,1)
A =  0.5000
B = 1
C = -0.3813
D = 1
\end{verbatim}
SOS2=SOS(2,1:6);  \{second IIR section $H_2(z)$\}

$[A,B,C,D]=\text{SOS2SS}(\text{SOS2},1)$

\[
A = \begin{bmatrix}
-0.0000 & -0.5000 \\
1.0000 & 0
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 \\
0
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.3813 \\
-0.4790
\end{bmatrix}
\]

\[
D = 1
\]

The individual filter sections' are of Direct II form as reported below:

\[
A_i = [t]; \quad b_i = [t]; \quad c_i = [-0.3813] \quad d_i = 1
\]

\[
A_2 = \begin{bmatrix}
0 & -0.5 \\
1 & 0
\end{bmatrix}; \quad b_2 = [1] \quad c_2 = [0.3813 \quad -0.479]; \quad d_2 = 1
\]

The resulting cascade design is interpreted in Figure 6. The state variable model returned by \texttt{TF2SS} is that of a Direct II subfilter rather than an $N^{th}$–order cascade filter. The system cascade state model, according to Equation 8, is not a pre-defined Matlab function and would need to be scripted. If scripted, the $3^{rd}$ order system under study would result is a cascade state model given by:

\[
A = \begin{bmatrix}
A_i & 0 \\
b_2 c_i^T & A_2
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
-0.3813 & 0 & 0.5
\end{bmatrix}; \quad b = \begin{bmatrix}1\end{bmatrix}; \quad c = \begin{bmatrix}
-0.3813 \\
0.3813
\end{bmatrix}; \quad d = 1
\]

\[
0.3813
\]

\[-0.479
\]

\[
-0.5
\]

\[
-0.3813
\]

\[
-0.5
\]

\[
0
\]

\[
0
\]

\[
1
\]

The successful design of an IIR requires that finite wordlength effects be controlled. First and foremost is controlling potential run-time register overflows. Historically, this has required that the contents of filter’s shift registers be bounded in some acceptable manner. Mathematically, the dynamic range of a register can be bounded in either the time or frequency-domain in terms of a specific $l_p$ norm which is formally defined to be:

\[
\text{VII. Quantifying Finite Wordlength Effects}
\]

Figure 6: Cascade IIR Filter.
The $l_2$ norm is the familiar Euclidean norm. The $l_\infty$ norm is equal to the maximum \{\(|x[k]|\)\}. Mathwork’s Matlab $l_p$ norm determination function is called \texttt{norm}. The norms computed in this manuscript are, however, are based on first principles (i.e., direct implementation of Equation 11).

One method for determining the dynamic range requirements of a filter is based on determining the $l_2$ norm of all the state registers. Another requires the measurement of the maximum frequency response ($l_\infty$ norm) at each shift register location. These methods, unfortunately, will not guarantee an overflow-free outcome. The most reliable method of determining a register’s dynamic range requirements is to perform a worst-case analysis or, as it is sometimes referred to, an $l_1$ norm analysis. First, assume that the input $u[k]$ is bounded by some value. For discussion purposes, assume that the scaled input $|w[k]| \leq 1$ on a sample-by-sample basis (i.e.). The worst-case input, denoted $w[k]$, produces the maximum measured response at one or more of the filter’s shift-register locations. The worst-case response is defined in terms of the convolution of the filter’s impulse response with the worst-case input. Specifically, assume that the impulse response, measured at output of the $i$th state location, is $h_i[k] = \{h_i[0], h_i[1], h_i[2], … \}$. If the input is bounded by unity, then the worst-case input from the perspective of $i$th state $x_i[k]$, is given by the binary valued time-series:

$$w_i[k] = \text{sign}(h_i[D-k]) \text{ } \text{ } w_i[k] \in \{-1,1\},$$

where $D$ is an arbitrary integer delay. The worst-case input can be created simply by reversing the impulse response in time and then quantizing each impulse response sample $h[M-k]$ into a sample value of 1 if $h[M-k] \geq 0$, -1 otherwise. The worst case gain, measured at state $x_i[k]$, can then be mathematically defined to be the $l_1$ norm of state impulse response, given by:

$$|x_i|_1 \leq G_i = \sum_{k=0}^\infty |h_i[k]|$$

Since computing an infinitely long IIR’s impulse response is problematic. However, it is reasonable to assume that the IIR being implemented is asymptotically stable, which guarantees that the impulse response $h[k]$ will eventually converge to zero resulting in a finite $G$. The number of samples it would take for $h[k]$ to converge to within an arbitrary small envelope about zero can be theoretically predicted using analysis of the dominant eigenvalues of the filter, or experimentally estimated using numerical studies. From these studies, a sample index $K_0$ can be found such that

$$G_i = \sum_{k=0}^\infty |h_i[k]| \approx \sum_{k=0}^{K_0} |h_i[k]|$$

Repeating this process for all states and the output $y[k]$, the maximum worst-case gain is $G = \max(G_i)$. If, for example, $G \leq 2^{\Delta B}$, then technically the register length should be extended by $\Delta B$-bits in order to avoid run-time register overflow. If the design protocol does not permit the use of extended precision registers, then input $u[k]$ should be scaled by $1/G$ to insure overflow-free performance with an attendant reduction in precision. This action, obviously, can seriously degrade the quality of a filter if $\Delta B$ is large.
If a system is designed to operate without run-time register overflow, then attention turns to quantifying finite wordlength effects. The amount of noise added to an IIR’s output can be determined analytically or experimentally. The analytical approach begins with defining a subfilter’s state variable model:

\[
x_i[k+1] = \sum_{j=1}^{N} a_{ij} x_j[k] + b_j v[k] = a_{1i} x_1[k] + a_{Ni} x_N[k] + b_j v[k]
\]

15.

It can be seen that the production of \( x[k+1] \) includes up to \((N+1)\) rounding operations whose individual variances are \( \sigma^2 = Q^2/12 \). If all the partial products shown in Equation 15 are performed with full precision, then summed using an extended precision accumulator and rounded, only one rounding error is embedded in \( x[k+1] \). In general, each shift register can be assumed to be presented with \( 0 \leq m_i \leq N+1 \) independent roundoff errors of variance \( Q^2/12 \) depending of the choice of architecture and arithmetic system design. What is needed is the determination of the noise amplification existing between a noise injection point (shift register input) and output. This process requires that the impulse response, say \( g[k] \), defined from the \( i^{th} \) shift register’s input to the filter’s output be determined for the purposes motivated in Figure 7. In Figure 7 it is assumed that the power of \( m_i \) roundoff errors are part of the production of \( x[k+1] \). Under the influence of the IIR filter dynamics (impulse response), the roundoff error power is amplified at the filter’s output by an amount called the noise power gain for the \( i^{th} \) state, or \( \text{NPG}_i \). Due to internal feedback, the noise power gain can be significant. What is needed is an efficient means of auditing these power levels.

Figure 7: Roundoff error model.

Consider setting a roundoff error injection point to the input of the \( j^{th} \) shift register. This can be accomplished by redefining the input vector \( b \) to read

\[
b^i=0, \ldots, 1 (i^{th} \text{ element}), \ldots, 0; \quad d=0
\]

16.

Note that the direct input-output path is disabled \((d=0)\) and that the only the \( i^{th} \) shift register is stimulated by an input which is assumed to model roundoff errors. It should be fully appreciated that the measured impulse response, \( w[k] \), includes the effects of all the feedback loops which are connected to the \( i^{th} \) shift register. The predicted noise power gain (NPG), or the power amplification of noise power source injected into the \( i^{th} \) shift register, shall be denoted \( W_i = \text{NPG}_i \), for \( i \in [1,N] \). Mathematically \( W_i \) is the \( l^2 \) norm of the impulse response \( w[k] \) (i.e., \( ||w_i||_2 = W_i \)). Statistically, the error variance measured at the output due to locally injected noise sources and is given by:
\[ \sigma^2 = \frac{Q^2}{12} \sum_{k=1}^{N} m_i W_i = \frac{Q^2}{12} \sum_{k=1}^{N} m_i NPG_i = \frac{Q^2}{12} (NPG) \]

where \( m_i \) is the number of independent roundoff error sources attached to the input of the \( i \)th shift register with each source contributing a noise variance \( Q^2/12 \). The value \( \sigma^2 \) represents the roundoff error variance measured at the output due to all internal roundoff error sources.

Fortunately, an algebraic means is available that can compute the individual power gains in terms of an \( N \times N \) Lyapunov matrix which satisfies:

\[ W = A^T WA + cc^T \]

where \( W_i \) is the \( i \)th on-diagonal term of \( W \), for \( i \in [1,N] \). \( W \) can be computed using the following algorithm:

1. Initialize: \( W_0 = cc^T \)
2. Loop on \( i \)
   \[ W[i+1] = A^T W[i] A + W_0 \]
3. Continue until \( \Delta W[i+1] = W[i+1] - W[i] \sim [0] \)

Given the state 4-tuple \([A, b, c, d]\) and Equation 18, the noise power gain can be computed for the state model. A complete end-to-end analysis needs to also account for the input scale factor \( k \) appearing in Equations 5 and 7, and ignored by the state model. The function of the scale factor is to set the passband gain to unity. A scale factor of \( k=2 \), for example, would reduce the output precision by one-bit. The noise power gain estimate, therefore needs to be corrected for the presence of \( k \) by scaling Equation 17 to read:

\[ \sigma^2 = \frac{\sigma^2 \text{ (Equation 17)}}{k^2} \]

Another problem arises when Matlab is used. Whereas an arbitrary \( N^\text{th} \) order transfer function can be mapped into an \( N^\text{th} \) order Direct II 4-tuple \([A, b, c, d]\), the transfer function for the Cascade architecture reduces to a collection of 1\text{st} or 2\text{nd} order Direct II sections having a state 4-tuple \([A, b, c, d]\). This information can theoretically be used to construct the state 4-tuple \([A, b, c, d]\) for a the entire cascade architecture, as shown in Equation 9. Once this is accomplished, the Lyapunov matrix and attendant noise power gain can be computed. Another approach is to compute the noise power gain stage-by-stage. This requires that the noise power gain \((NPG)\) be computed for each individual first or second order section, and the results chained together. Referring to Equation 17, it can be noted that the weight of the rounding error is \( Q^2/12 \) and would appear at the output of the first cascaded section scaled by \((Q^2/12)\times NPG_1\). The second stage would then accept roundoff errors of weight \((Q^2/12)\times NPG_1\) and export it with weight \(((Q^2/12)\times NPG_1)\times NPG_2\), and so forth. The result is an output round off error power gain estimate taking the form

\[ \sigma^2 = W_{11} \times W_{22} \times \ldots \times W_{LL} \]

where \( W_{ii} \) is the noise power gain of the \( i \)th stage.
V. Design Example

As an illustrative example, consider an 8th order Chebyshev-II IIR lowpass filter is implemented using a Direct II and Cascade architecture. The design specifications are given as follows:

- Sampling Frequency: 100kHz
- Attenuation Frequency: 20kHz
- Stopband Attenuation: 30dB

The following filter was designed using Matlab’s FDATool:

$$H(z) = 0.078 \frac{z^8 + 1.404z^7 + 3.145z^6 + 3.793z^5 + 4.431z^4 + 3.793z^3 + 3.145z^2 + 1.404z + 1}{z^8 - 1.063z^7 + 1.853z^6 - 0.841z^5 + 0.802z^4 - 0.083z^3 + 0.116z^2 + 0.016z + 0.007}$$

The input scale factor $k=0.078$ and is not considered as part of the state model.

Direct II Implementation

Analyzing the transfer function with the state space tool TF2SS, the following $[A,b,c,d]$ 4-tuple results:

$$A = \begin{bmatrix} 1.063 & -1.853 & 0.841 & -0.802 & 0.083 & -0.116 & -0.016 & -0.007 \\ 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 \end{bmatrix}$$

$$b = (1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)^T$$

$$c = (0.193 \ 0.101 \ 0.362 \ 0.284 \ 0.303 \ 0.237 \ 0.109 \ 0.078)$$

$$d = 0.078$$

The predicted worst-case filter dynamic range requirements were estimated using simulation. The process begins with the determination of the impulse response associated with each shift register location. It can be noted that for a Direct II architecture the impulse response measured at the state $x_1[k]$ is identical to the other states except for a simple delay. The impulse response, can be produced using the Matlab function IMPULSE(). The $l_1$ worst-case state norm of the 8th order system, measured at state location $x_1[k]$ is shown in Figure 8. It can be seen that the worst-case gain (Equation 14) is approximately $G=2.25$, which indicates that at the filter’s shift registers need an additional $\log_2(2.25)\sim1.17$-bits. In practice, since 1.17-bits is sufficiently close to 1-bit that a only one-bit of additional head room might be recommended.
Figure 8: Direct II impulse response measured at the $x_1[k]$ shift register location (Left) and $l_1$ norm estimate (Right).

Cascade Implementation

For the cascade architecture, the 8\textsuperscript{th}-order filter $H(z)$ is decomposed into four individual subfilters:

$$
H_1(z) = \frac{1+1.731z^{-1} + z^{-2}}{1+0.240z^{-1} + 0.053z^{-2}} \quad H_2(z) = \frac{1+0.524z^{-1} + z^{-2}}{1-0.083z^{-1} + 0.266z^{-2}}
$$

$$
H_3(z) = \frac{1-0.268z^{-1} + z^{-2}}{1-0.469z^{-1} + 0.551z^{-2}} \quad H_4(z) = \frac{1-0.583z^{-1} + z^{-2}}{1-0.751z^{-1} + 0.841z^{-2}}
$$

such that $H(z) = 0.078\ H_1(z)\ H_2(z)\ H_3(z)\ H_4(z)$ where each of which is modeled as a 2\textsuperscript{nd}-order Direct II section using Matlab’s \texttt{SOS2SS} function. The individual 2\textsuperscript{nd}-order Direct II subfilters are shown below:

$$
A_1 = \begin{bmatrix} -0.2404 & -0.0535 \\ 1 & 0 \end{bmatrix} \quad c_1 = [1.4905 \quad 0.946]
$$

$$
A_2 = \begin{bmatrix} 0.0834 & -0.2664 \\ 1 & 0 \end{bmatrix} \quad c_2 = [0.6075 \quad 0.7336]
$$

$$
A_3 = \begin{bmatrix} 0.4688 & -0.5514 \\ 1 & 0 \end{bmatrix} \quad c_3 = [0.2006 \quad 0.4486]
$$

$$
A_4 = \begin{bmatrix} 0.7514 & -0.8414 \\ 1 & 0 \end{bmatrix} \quad c_4 = [0.1686 \quad 0.1586]
$$

The product of the DC gains for each 2\textsuperscript{nd}-order section (i.e., $H(z=1)$) has an approximate value of $1/k$. Form this, the overall state model follows and is defined by Equation 9 to be:
The impulse responses and worst-case state norms can be deduced from a set of simulations illustrated in Figure 9. It can be noted that the largest $l_1$ norm is found in the 4th subfilter and is about 1.8 ($\log_2(1.8) \approx 0.85$-bits), which is less than the maximal $l_1$ norm of the Direct II filter model.

The statistical noise power gain of each design could have been predicted by calculating the elements of the Lyapunov matrix $W$. For the Direct II design, a value of $W_{11} = \ldots = W_{68} = 0.35$ was computed. The design assumes that extended precision accumulators are used, resulting in $m_1 = 1$ (Equation 17), $m_2 = m_3 = \cdots = m_8 = 0$. Correcting for the scale factor $k = 0.078$, the noise power gain is predicted to be:

$$\sigma^2 = W_{11}/k^2 = 58.5 = NPG$$

or $\sigma = 7.63$ which corresponds to approximately a 3-bit of lost precision.

The noise power gain for the 4-stage cascaded subfilters is based on the following computed results:

<table>
<thead>
<tr>
<th>Stage $i$</th>
<th>$NPG$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.848</td>
</tr>
<tr>
<td>2</td>
<td>1.04</td>
</tr>
<tr>
<td>3</td>
<td>0.47</td>
</tr>
<tr>
<td>4</td>
<td>0.307</td>
</tr>
</tbody>
</table>

Computing $\sigma^2$ using Equation 21, one obtains $\sigma^2 = 94.91$, or $\sigma = 9.74$ which corresponds to about a 3.3-bit loss in output precision (approximately 0.5-bit more than the Direct II model). The consequence of this analysis is that for a 16-bit design, the optimal choice of fractional precision (i.e., $F$) for the Cascade filter is about $F = 14$-bits based on the overflow analysis, whereas $F = 13$-bits is required by the Direct II. Furthermore, it is predicted that the output Direct II filter is
slightly more precise than the Direct II when operating in the linear (on-saturated) regime. As a result, the non-saturated performance of Direct II and Cascade filters performance should be very comparable.

Figure 9: Cascade impulse responses section by section (Left) and $l_1$ norm estimates (Right).
The analytical methods previously introduced admittedly require some mathematical maturity and are based on a set of assumptions that do not necessarily exactly represent the true behavior of a system. The alternative is to use simulation. The use of Simulink for this task offers several distinct advantages, namely in the ease of access to pertinent system information. While effectively any operation can be formed in either an imperative or graphical paradigm, the latter is of greater utility to a designer who maintains a chief interest in the end result. In this way, the relationship between the traditional workspace and the Simulink interface is seamless. To facilitate this transition, one can exploit the existence of predefined blocks by invoking the Simulink Library Browser from the Launch Pad in Matlab.

A Simulink simulator was designed to test a 16-bit filter for fractional precisions ranging from $F \in [0, 15]$. Figure 10 displays a Simulink model for a specific instance corresponding to a Direct II architecture with $F=0$ fractional bits. A unit-bound, uniformly distributed random number generator is used as the input to each subsystem to model any potential input signal. The simulation is run for two design types, one accounting for input scaling and the other with a unity gain. The filter is modeled in its state-space form for the appropriate number of fractional bits and is then compared to the floating-point design. The base-two logarithm of the standard deviation is calculated for both the predicted and simulated precision. The simulation is run sufficiently long to acquire 2048 samples for a given sampling period.

Figure 10: Simulink simulation block diagram for $F=0$.  

State-space filters for fixed-point designs are available in the Fixed Point Blockset, and floating-point filters can be found in the Discrete subsection of the main Simulink library, both of which can be accessed by the aforementioned browser. Block connection is akin to many schematically oriented tools: one need only click the first block, hold the SHIFT key, and click the second block. This permits a quick design of the entire system once the individual blocks are selected. Should such a connection be invalid due to different input and output data types, the Simulink compiler will recognize this and will demonstrate where the conflict exists. Type conversion is frequently controlled within a given block, and fixed-point to floating-point conversion can be done in a single block. For composite systems that maintain a specific input-output relationship of interest, subsystems can be created that condense numerous blocks into one, facilitating understanding by minimizing potentially pertinent detail. For example, the
model shown in Figure 10 is in fact a subsystem of a larger model that accounts for different architectural models and fractional precision.

While this design requires minimal overhead to appreciate, certain aspects merit explanation. The fixed-point filters, as an example, offer a measure for the data type used for internal calculations. Specifically, the command \texttt{sfrac(N,I)} creates an N-bit structure having a signed fractional form with I integer bits. Namely, a 16-bit system with no fractional precision can be modeled by the structure \texttt{sfrac(16,15)}. When the simulation is complete, the Fixed-Point GUI can be utilized to examine run-time saturation, as shown in Figure 11. Bookkeeping is maintained within individual fixed-point blocks both for underflow and overflow.

Conclusions

Using Simulink simulation, an end-to-end fixed-point study of the 8th-order design example implemented as a Direct II and Cascade filter architectures, was performed. The key architectural choices were:

- **Data wordlength:** \( N=16 \)-bits
- **Fractional Precision:** \( F \in [0:15] \)-bits.
- **Input data format:** \( x[k] \in [N:F] \)-bits
- **Output data format:** \( y[k] \in [N:F] \)-bits
- **Coefficient data format:** \( c_k \in [N:F] \)-bits
- **Multiplier datapaths:** 16x16 \( \rightarrow \) 32 –bits
- **Multiplier datapaths:** 16x16 \( \rightarrow \) 32 –bits
- **Direct II accumulator datapaths:** \( 32+(32+N_{\text{DII}}) \rightarrow 32+N_{\text{DII}} \) –bits \( (N_{\text{DII}} \geq \log_2(2.25) \sim 1.17 \)-bits\)
- **Cascade accumulator datapaths:** \( 32+(32+N_{\text{C}}) \rightarrow 32+N_{\text{C}} \) –bits \( (N_{\text{C}} \geq \log_2(1.8) \sim 0.8 \)-bits\)

The numerical results are presented in Figure 12 for both architectures. Note that the graph is portioned into three regimes, namely:
• Too little precision – caused by too few fractional bits of accuracy.
• Linear regime – sufficient dynamic range to inhibit run-time overflow and sufficient fractional precision to eliminate traumatic roundoff errors.
• Too little dynamic range – caused by too few integer bits resulting in a too small dynamic range that results in run-time overflow errors.

It can be seen that the Direct II architecture exhibits overflow contamination about 1 bit before overflow become an issue with the Cascade filter. This was predicted by the analytical study. Moreover, the Direct II has slightly better statistical precision over the linear operating range which was again predicted by the analytical study. The simulation would suggest that the Cascade filter carry a [16:14] format, resulting in a solution having (statistically) about (statistically) 11.25 fractional bits of precision. The simulation also suggests that the Direct II filter carry a [16:13] format, resulting in a solution having (statistically) about 11 fractional bits of precision.

![Figure 12: Statistical filter performance versus the number of fractional bits.](Image)

The conclusion is that a robust simulation environment can be used to obtain the same fixed-point IIR design decision as those obtained using classical analytical practices. The advantage of the simulation approach is manifold. First, it is easier and faster to construct than the alternative analytical analysis framework. Second, it is more robust, capable of handling any arbitrary architecture. Third, it is extensible and can be easily modified (e.g., change of design specification). Fourth, it provides open access to the entire signal processing elements used to construct the simulation.
Acronyms

ADC Analog to digital converter
DSP Digital signal processing
FDA Filter Design and Analysis toolkit (Matlab)
FIR Finite impulse response filter
IIR Infinite impulse response filter
LSB Least significant bit
MAC Multiply-accumulate (SAXPY)
NPG Noise power gain
SAXPY S=AX+Y (multiply-accumulate)

VI. References


