Error Analysis of Digital Filters using Theorem Proving

Behzad Akbarpour and Sofiène Tahar

Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8, Canada
{behzad,tahar}@ece.concordia.ca

Abstract. When a digital filter is realized with floating-point or fixed-point arithmetics, errors and constraints due to finite word length are unavoidable. In this paper, we show how these errors can be mechanically analysed using the HOL theorem prover. We first model the ideal real filter specification and the corresponding floating-point and fixed-point implementations as predicates in higher-order logic. We use valuation functions to find the real values of the floating-point and fixed-point filter outputs and define the error as the difference between these values and the corresponding output of the ideal real specification. Fundamental analysis lemmas have been established to derive expressions for the accumulation of roundoff error in parametric Lth-order digital filters, for each of the three canonical forms of realization: direct, parallel, and cascade. The HOL formalization and proofs are found to be in a good agreement with existing theoretical paper-and-pencil counterparts.

1 Introduction

Signal processing through digital techniques has become increasingly attractive with the rapid technological advancement in digital integrated circuits, devices, and systems. The availability of large scale general purpose computers and special purpose hardware has made real time digital filtering both practical and economical. Digital filters are a particularly important class of DSP (Digital Signal Processing) systems. A digital filter is a discrete time system that transforms a sequence of input numbers into another sequence of output, by means of a computational algorithm [13]. Digital filters are used in a wide variety of signal processing applications, such as spectrum analysis, digital image and speech processing, and pattern recognition. Due to their well-known advantages, digital filters are often replacing classical analog filters. The three distinct and most outstanding advantages of the digital filters are their flexibility, reliability, and modularity. Excellent methods have been developed to design these filters with desired characteristics. The design of a filter is the process of determination of a transfer function from a set of specifications given either in the frequency domain, or in the time domain, or for some applications, in both. The design of a digital filter starts from an ideal real specification. In a theoretical analysis of the digital filters, we generally assume that signal values and system coefficients are represented in the real number system and are expressed to an infinite precision. When implemented as a special-purpose digital hardware or as a computer algorithm, we must represent the signals and coefficients in some digital number system that must always be of a finite precision. Therefore, arithmetic operations must be carried out with an accuracy limited by this finite word length. There is a variety of types of arithmetic used in the implementation of digital systems. Among the most common are the floating-point and fixed-point. Here, all operands are represented by a special format or assigned a fixed word length and a fixed exponent, while the control structure and the operations of the ideal program remain unchanged. The transformation from the real to the floating-point and fixed-point forms is quite tedious and error-prone. On the implementation side, the fixed-point model of the algorithm has to be transformed into the best suited target description, either using a hardware description or a programming language. This design process can be aided by a number of specialized CAD tools such as SPW (Cadence) [3], CoCentric (Synopsys) [20], Matlab-Simulink (Mathworks) [16], and FRIDGE (Aachen UT) [22].

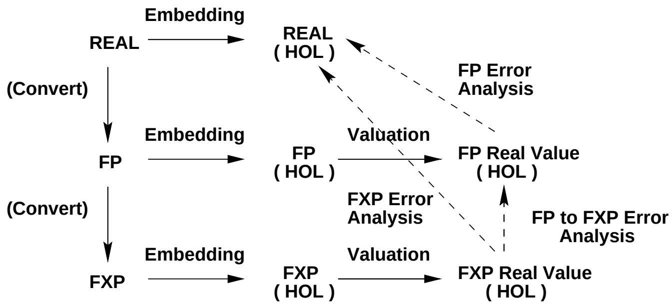


Fig. 1. Error analysis approach

In this paper we describe the error analysis of digital filters using the HOL theorem proving environment [5] based on the commutating diagram shown in Figure 1. Thereafter, we first model the ideal real filter specification and the corresponding floating-point and fixed-point implementations as predicates in higher-order logic. For this, we make use of existing theories in HOL on the construction of real numbers [7], the formalization of IEEE-754 standard based floating-point arithmetic [8, 9], and the formalization of fixed-point arithmetic [1, 2]. We use valuation functions to find the real values of the floating-point and fixed-point filter outputs and define the errors as the differences between these values and the corresponding output of the ideal real specification. Then we establish fundamental lemmas on the error analysis of the floating-point and fixed-point roundings and arithmetic operations against their abstract mathematical counterparts. Finally, we use these lemmas as a model to derive expressions for the accumulation of the roundoff error in parametric Lth-order digital filters, for each of the three canonical forms of realization: direct, parallel, and cascade [18].

Using these forms, our verification methodology can be scaled up to any larger-order filter, either directly or by decomposing the design into a combination of internal sub-blocks. While the theoretical work on computing the errors due to finite precision effects has been extensively studied since the late sixties [15], it is for the first time in this paper, that a formalization and proof of this analysis for digital filters is done using a mechanical theorem prover, here the HOL. Our results are found to be in a good agreement with the theoretical ones.

The rest of this paper is organized as follows: Section 2 gives a review of the related work. Section 3 introduces the fundamental lemmas in HOL for the error analysis of the floating-point and fixed-point rounding and arithmetic operations. Section 4 describes the details of the error analysis in HOL of the class of linear difference equation digital filters implemented in the three canonical forms of realization. Finally, Section 5 concludes the paper.

2 Related Work

Work on the analysis of the errors due to the finite precision effects in the realization of the digital filters has always existed since their early days, however, using theoretical paper-and-pencil proofs and simulation techniques. For digital filters realized with the fixed-point arithmetic, error problems have been studied extensively. For instance, Knowles and Edwards [14] proposed a method for analysis of the finite word length effects in fixed-point digital filters. Gold and Radar [6] carried out a detailed analysis of the roundoff error for the first-order and second-order fixed-point filters. Jackson [12] analyzed the roundoff noise for the cascade and parallel realizations of the fixed-point digital filters. While the roundoff noise for the fixed-point arithmetic enters into the system additively, it is a multiplicative component in the case of the floating-point arithmetic. This problem is analyzed first by Sandberg [19], who discussed the roundoff error accumulation and input quantization effects in the direct realization of the filter excited by a deterministic input. He also derived a bound on the time average of the squared error at the output. Liu and Kaneko [15] presented a general approach to the error analysis problem of digital filters using the floating-point arithmetic and calculated the error at the output due to the roundoff accumulation and input quantization. Expressions are derived for the mean square error for each of the three canonical forms of realization: direct, cascade, and parallel. Upper bounds that are useful for a special class of the filters are given. Oppenheim and Weinstein [17] discussed in some details the effects of the finite register length on implementations of the linear recursive difference equation digital filters, and the fast Fourier transform (FFT) algorithm. Comparisons of the roundoff noise in the digital filters using the different types of arithmetics have also been reported in [21].

In order to validate the error analysis, most of the above work compare the theoretical results with corresponding experimental simulations. In this paper, we show how the above error analysis can be mechanically performed using the HOL theorem prover, providing a superior approach to validation by simulation.

Our focus will be on the process of translating the hand proofs into equivalent proofs in HOL. The analysis we propose is mostly inspired by the work done by Liu and Kaneko [15], who defined a general approach to the error analysis problem of digital filters using the floating-point arithmetic. Following a similar approach, we have extended this theoretical analysis for fixed-point digital filters. In both cases, a good agreement between the HOL formalized and the theoretical results are obtained.

Through our work, we confirmed and strengthened the main results of the previously published theoretical error analysis, though we uncovered some minor errors in the hand proofs and located a few subtle corners that are overlooked informally. For example, in the theoretical fixed-point error analysis it is always assumed that the fixed-point addition causes no error and only the roundoff error in the fixed-point multiplication is analyzed [17]. This is under the assumption that there is no overflow in the result and also the input operands have the same attributes as the output. Using a mechanical theorem prover, we provide a more general error analysis in which we cover the roundoff errors in both the fixed-point addition and multiplication operations. On top of that, for the floating-point error analysis, we have used the formalization in HOL of the IEEE-754 [8], a standard which has not yet been established at the time of the above mentioned theoretical error analysis. This enabled us to cover a more complete set of rounding and overflow modes and degenerate cases which are not discussed in earlier theoretical work.

Previous work on the error analysis in formal verification was done by Harrison [9] who verified the floating-point algorithms such as the exponential function against their abstract mathematical counterparts using the HOL Light theorem prover. As the main theorem, he proved that the floating-point exponential function has a correct overflow behavior, and in the absence of overflow the error in the result is bounded to a certain amount. He also reported on an error in the hand proof mostly related to forgetting some special cases in the analysis. This error analysis is very similar to the type of analysis performed for DSP algorithms. The major difference, however, is the use of statistical methods and mean square error analysis for DSP algorithms which is not covered in the error analysis of the mathematical functions used by Harrison. In this method, the error quantities are treated as independent random variables uniformly distributed over a specific interval depending on the type of arithmetic and the rounding mode. Then the error analysis is performed to derive expressions for the variance and mean square error. To perform such an analysis in HOL, we need to develop a mechanized theory on the properties of random variables and random processes. This type of analysis is not addressed in this paper and is a part of our work in progress. Huhn *et al.* [11] proposed a hybrid formal verification method combining different state-of-the-art techniques to guide the complete design flow of imprecisely working arithmetic circuits starting at the algorithmic down to the register transfer level. The usefulness of the method is illustrated with the example of the discrete cosine transform algorithms. In particular, the authors have shown the use of computer algebra systems like Mathematica or Maple at the algorithmic level to reason about real numbers and to determine certain error bounds for the results of numerical operations. In contrast to [11], we propose an error analysis for digital filters using the HOL theorem prover. Although the computer algebraic systems such as Maple or Mathematica are much more popular and have many powerful decision procedures and heuristics, theorem provers are more expressive, more precise, and more reliable [10]. One option is to combine the rigour of the theorem provers with the power of computer algebraic systems as proposed in [10].

3 Error Analysis Models

In this section we introduce the fundamental error analysis theorems [23, 4], and the corresponding lemmas in HOL for the floating-point [8, 9] and fixed-point [1, 2] arithmetics. These theorems are then used in the next sections as a model for the analysis of the roundoff error in digital filters.

3.1 Floating-Point Error Model

In analyzing the effects of floating-point roundoff, the effects of rounding will be represented multiplicatively. The following theorem is the most fundamental in the floating-point rounding-error theory [23, 4].

Theorem 1: If the real number x located within the floating-point range, is rounded to the closest floating-point number x_R, then

$$x_R = x(1+\delta), \text{ where } |\delta| \leq 2^{-p} \tag{1}$$

and p is the precision of the floating-point format.

In HOL, we proved this theorem in the IEEE single precision floating-point format for the case of rounding to nearest as follows:

```
Lemma 1: FLOAT_ROUND_RELATIVE_ERROR
⊢  normalizes x ⟹  ∃ e. abs (e) < (1 / 2 pow ((fracwidth X) + 1)) ∧
   (Val (float (round X To_nearest x)) = x * (1 + e))
```

where the function *normalizes* defines the criteria for an arbitrary real number to be in the normalized range of floating-point numbers [8], *fracwidth* extracts the fraction width parameter from the floating-point format X, *Val* is the floating-point valuation function, *float* is the bijection function that converts a triple of natural numbers into the floating-point type, and *round* is the floating-point rounding function [9].

To prove this theorem [4], we first proved the following lemma which locates a real number in a binade (the floating-point numbers between two adjacent powers of 2):

```
Lemma 2: REAL_IN_BINADE
⊢  normalizes x ⟹  ∃ j. j ≤ ((emax X) − 2) ∧
   (2 pow (j + 1) / 2 pow (bias X)) ≤ abs x ∧
   abs x < (2 pow (j + 2) / 2 pow (bias X))
```

where the function *emax* defines the maximum exponent in a given floating-point format, and *bias* defines the exponent bias in the floating-point format which is a constant used to make the exponent's range nonnegative. Using this lemma we can rewrite the general floating-point absolute error bound theorem (ERROR_BOUND_NORM_STRONG) developed in [9] as follows:

```
Lemma 3: ERROR_BOUND_NORM_STRONG_NORMALIZE
⊢  normalizes x ⟹
   ∃ j. abs (error x) ≤ (2 pow j / 2 pow (bias X + fracwidth X))
```

which states that if the absolute value of a real number is in the representable range of the normalized floating-point numbers, then the absolute value of the error is less than or equal to $2^j/2^{(bias\ X\ +\ fracwidth\ X)}$. The function *error*, defines the error resulting from rounding a real number to a floating-point value which is defined as follows [9]:

```
⊢def  error x = (Val (float (round X To_nearest x)) − x)
```

Since $(2^{(j+1)}\ /\ 2^{(bias\ X)}) \ \leq\ |x|$ for the real numbers in the normalized region as proved in Lemma 2, we have $(|error\ x|\ /\ |x|)\ \leq\ (2^j\ /\ 2^{(bias\ X\ +\ fracwidth\ X)})\ /\ (2^{(j+1)}\ /\ 2^{(bias\ X)})$ or $(|error\ x|\ /\ |x|)\ \leq\ (1\ /\ 2^{((fracwidth\ X)\ +\ 1)})$. Finally, defining $e = (error\ x\ /\ x)$ will complete the proof of the floating-point relative error bound theorem as described in Lemma 1.

Next, we apply the floating-point relative rounding error analysis theorem (Theorem 1) to the verification of the arithmetic operations. The goal is to prove the following theorem in which floating-point arithmetic operations such as addition, subtraction, multiplication, and division are related to their abstract mathematical counterparts according to the corresponding errors.

Theorem 2: Let $*$ denote any of the floating-point operations +, -, $\times$, /. Then

$$fl\ (x\ *\ y)\ =\ (x\ *\ y)(1\ +\ \delta),\quad \text{where}\ \ |\delta|\ \leq\ 2^{-p} \tag{2}$$

and p is the precision of the floating-point format. The notation *fl (.)* is used to denote that the operation is performed using the floating-point arithmetic.

To prove this theorem in HOL, we start from the already proved lemmas on the absolute analysis of rounding error in the floating-point arithmetic operations (FLOAT_ADD) developed in [9]. We have converted these lemmas to the following relative error analysis version, using the relative error bound analysis of floating-point rounding (Lemma 1):

```
Lemma 4: FLOAT_ADD_RELATIVE
⊢  Finite a ∧ Finite b ∧ normalizes (Val a + Val b)
   ⟹  Finite (a + b) ∧ ∃ e. abs e ≤ (1 / 2 pow ((fracwidth X) + 1))
        ∧ (Val (a + b) = (Val a + Val b) * (1 + e))
```

where the function *Finite* defines the finiteness criteria for the floating-point numbers. Note that we use the conventional symbols for arithmetic operations on floating-point numbers using the operator overloading in HOL.

3.2 Fixed-Point Error Model

While the rounding error for the floating-point arithmetic enters into the system multiplicatively, it is an additive component for the fixed-point arithmetic. In this case the fundamental error analysis theorem can be stated as follows [23].

Theorem 3: If the real number x located in the range of the fixed-point numbers with format *X'*, is rounded to the closest fixed-point number x'_R, then

$$x'_R \;=\; x \;+\; \epsilon, \quad \text{where} \;\; |\epsilon| \;\leq\; 2^{-fracbits\ (X')} \tag{3}$$

and *fracbits* is a function that extracts the number of bits that are to the right of the binary point in the given fixed-point format.

This theorem is proved in HOL as follows [1]:

```
Lemma 5: FXP_ROUND_ABSOLUTE_ERROR_BOUND
⊢  (validAttr X′) ∧ (representable X′ x) ⟹
   abs (Fxp_error X′ x) ≤ (1 / 2 pow (fracbits X′))
```

where the function *validAttr* defines the validity of the fixed-point format, *representable* defines the criteria for a real number to be in the representable range of the fixed-point format, and *Fxp_error* defines the fixed-point rounding error.

The verification of the fixed-point arithmetic operations using the *absolute* error analysis of the fixed-point rounding (Theorem 3) can be stated as in the following theorem in which the fixed-point arithmetic operations are related to their abstract mathematical counterparts according to the corresponding errors.

Theorem 4: Let $*$ denote any of the fixed-point operations +, -, $\times$, /, with a given format *X'*. Then

$$fxp\,(x \;*\; y) \;=\; (x \;*\; y) \;+\; \epsilon, \quad \text{where} \;\; |\epsilon| \;\leq\; 2^{-fracbits\ (X')} \tag{4}$$

and the notation *fxp (.)* is used to denote that the operation is performed using the fixed-point arithmetic. This theorem is proved in HOL using the following lemma [1]:

```
Lemma 6: FXP_ADD_ABSOLUTE
⊢  (Isvalid a) ∧ (Isvalid b) ∧ validAttr (X′) ∧
   representable X′ (value a + value b) ⟹  (Isvalid (FxpAdd X′ a b)) ∧
   ∃ e. abs e ≤ (1 / 2 pow (fracbits X′)) ∧
   value (FxpAdd X′ a b) = (value a + value b) + e
```

where *Isvalid* defines the validity of a fixed-point number, *value* is the fixed-point valuation, and *FxpAdd* is the fixed-point addition.

4 Error Analysis of Digital Filters in HOL

In this section, the principal results for the roundoff accumulation in digital filters using the mechanized theorem proving are derived and summarized. We shall employ the models for the floating-point and fixed-point roundoff errors in HOL presented in the previous section. In the following, we will first describe in details the theory behind the analysis and then explain how this analysis is performed in HOL.

The class of digital filters considered in this paper is that of linear constant coefficient filters specified by the difference equation:

$$w_n \;=\; \sum_{i=0}^{M} b_i \; x_{n-i} \;-\; \sum_{i=1}^{L} a_i \; w_{n-i} \tag{5}$$

where $\{x_n\}$ is the input sequence and $\{w_n\}$ is the output sequence. L is the order of the filter, and M can be any positive number less than L. There are three canonical forms of realizing a digital filter, namely the direct, parallel, and cascade forms (Figure 2) [18].

If the output sequence is calculated by using the equation (5), the digital filter is said to be realized in the direct form. Figure 2 (a) illustrates the direct form realization of the filter using the corresponding blocks for the addition, multiplication by a constant operations, and the delay element.

The implementation of a digital filter in the parallel form is shown in Figure 2 (b) in which the entire filter is visualized as the parallel connection of the simpler filters H_i of a lower order. In this case, K intermediate outputs $\{w_n^i\}$, $i = 1,2,\ldots,K$ are first calculated and then summed to form the total output $\{w_n\}$. Therefore, for the input sequence $\{x_n\}$ we have:

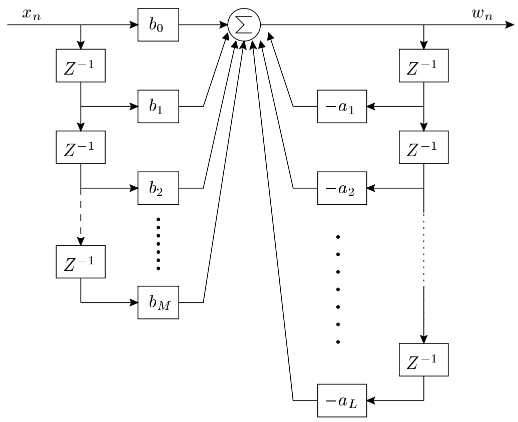
$$w_n^i = f_i x_n + g_i x_{n-1} - c_i w_{n-1}^i - d_i w_{n-2}^i \tag{6}$$

where the parameters f_i, g_i, c_i, and d_i are obtained from the parameters a_i and b_i in equation (5) using the parallel expansion. The output of the entire filter w_n, is then related to w_n^i by:

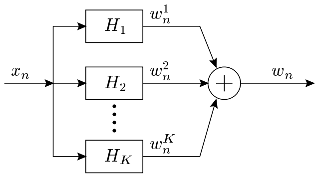
$$w_n = w_n^1 + w_n^2 + \cdots + w_n^K \tag{7}$$

The implementation of a digital filter in the cascade form is shown in Figure 2 (c) in which the filter is visualized as a cascade of lower filters. From the input $\{x_n\}$, the intermediate output $\{w_n^1\}$ is first calculated, and then this is the input to the second filter. Continuing in this manner, the final output $w_n^K = w_n$ is calculated. Since the output of the ith section (w_n^i) is the input of the *(i+1)*th section, the following equation holds:

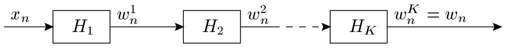
$$w_n^{i+1} = w_n^i + k_i w_{n-1}^i + l_i w_{n-2}^i - c_i w_{n-1}^{i+1} - d_i w_{n-2}^{i+1} \tag{8}$$



a) Direct form



b) Parallel form



c) Cascade form

Fig. 2. Canonical forms of digital filter realizations

where the parameters k_i, l_i, c_i, and d_i are obtained from the parameters a_i and b_i in equation (5) using the serial expansion.

There are three common sources of errors associated with the filter of the equation (5), namely [15]:

1. **input quantization:** caused by the quantization of the input signal $\{x_n\}$ into a set of discrete levels.
2. **coefficient inaccuracy:** caused by the representation of the filter coefficients $\{a_k\}$ and $\{b_k\}$ by a finite word length.
3. **round-off accumulation:** caused by the accumulation of roundoff errors at arithmetic operations.

Therefore, for the digital filter of the equation (5) the actual computed output reference is in general different from $\{w_n\}$. We denote the actual floating-point and fixed-point outputs by $\{y_n\}$ and $\{v_n\}$, respectively. Then, we define the corresponding errors at the nth output sample as:

$$e_n = y_n - w_n \tag{9}$$

$$e'_n = v_n - w_n \tag{10}$$
$$e''_n = v_n - y_n \tag{11}$$

where e_n and e'_n are defined as the errors between the actual floating-point and fixed-point implementations and the ideal real specification, respectively. e''_n is the error in the transition from the floating-point to fixed-point levels.

It is clear from the above discussion that for the digital filter of the equation (5) realized in the direct form, we have:

$$y_n \;=\; fl\;(\sum_{k=0}^{M} b_k\; x_{n-k} \;-\; \sum_{k=1}^{L} a_k\; y_{n-k}) \tag{12}$$

and

$$v_n \;=\; fxp\;(\sum_{k=0}^{M} b_k\; x_{n-k} \;-\; \sum_{k=1}^{L} a_k\; v_{n-k}) \tag{13}$$

The notations *fl (.)* and *fxp (.)* are used to denote that the operations are performed using the floating-point and fixed-point arithmetics, respectively. The calculation is to be performed in the following manner. First, the output products $a_k\; y_{n-k},\;\; k = 1,2,\ldots,L$ are calculated separately and then summed. Next, the same is done for the input products $b_k\; x_{n-k},\;\; k = 0,1,\ldots,M$. Finally, the output summation is subtracted from the input one to obtain the main floating-point output y_n. Similar discussion can be applied for the calculation of the fixed-point output v_n. The corresponding flowgraph showing the effect of roundoff error using the fundamental error analysis theorems (Theorems 2 and 4) according to the equations (2) and (4), is given by Figure 3, which also indicates the order of the calculation.

Formally, a flowgraph is a network of directed branches that connect at nodes. Associated with each node is a variable or node value. Each branch has an input signal and an output signal with a direction indicated by an arrowhead on it. In a linear flowgraph, the output of a branch is a linear transformation of the input to the branch. The simplest examples are constant multipliers and adders, i.e., when the output of the branch is simply a multiplication or an addition of the input to the branch with a constant value, which are the only classes we consider in this paper. The linear operation represented by the branch is typically indicated next to the arrowhead showing the direction of the branch. For the case of a constant multiplier and adder, the constant is simply shown next to the arrowhead. When an explicit indication of the branch operation is omitted, this indicates a branch transmittance of unity, or identity transformation. By definition, the value at each node in a flowgraph is the sum of the outputs of all the branches entering the node. To complete the definition of the flowgraph notation, we define two special types of nodes. *(1) Source nodes* that have no entering branches. They are used to represent the injection of the external inputs or signal sources into a flowgraph. *(2) Sink nodes* that have only entering branches. They are used to extract the outputs from a flowgraph [18].

The quantities $\delta_{n,k}, k = 0,1,\ldots,M,\;\; \epsilon_{n,k}, k = 1,2,\ldots,L,\;\; \zeta_{n,k}, k = 1,2,\ldots,M,$

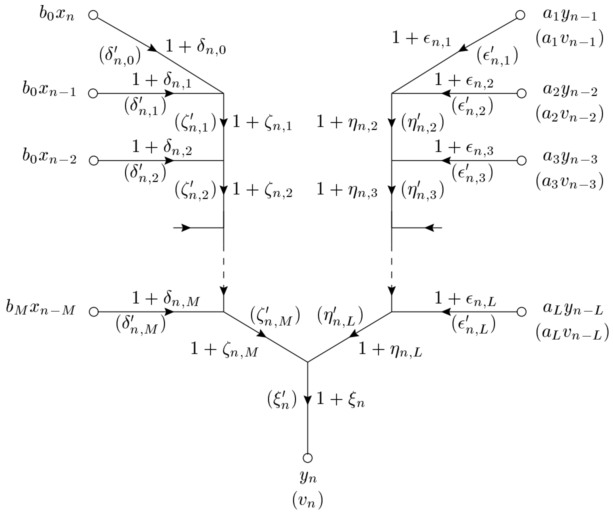


Fig. 3. Error flowgraph for Lth-order filter (Direct form)

$\eta_{n,k}, k = 2,3,\ldots,L$, and ξ_n in Figure 3 are errors caused by the floating-point roundoff at each arithmetic step. The corresponding error quantities for the fixed-point roundoff (shown in parentheses) are $\delta'_{n,k}, k = 0,1,\ldots,M$, $\epsilon'_{n,k}, k = 1,2,\ldots,L$, $\zeta'_{n,k}, k = 1,2,\ldots,M$, $\eta'_{n,k}, k = 2,3,\ldots,L$, and ξ'_n. Note that we have used one flowgraph to represent both the floating-point and fixed-point cases, simultaneously. For floating-point errors, the branch operations are interpreted as constant multiplications, while for fixed-point errors the branch operations are interpreted as constant additions. We have surrounded the fixed-point error quantities and output samples by parentheses to distinguish them from their floating-point counterparts. Therefore, the actual outputs y_n and v_n are seen to be given explicitly by:

$$y_n = \sum_{k=0}^{M} b_k \, \theta_{n,k} \, x_{n-k} \; - \; \sum_{k=1}^{L} a_k \, \phi_{n,k} \, y_{n-k} \tag{14}$$

where

$$\theta_{n,0} = (1+\xi_n)(1+\delta_{n,0}) \prod_{i=1}^{M} (1+\zeta_{n,i})$$

$$\theta_{n,j} = (1+\xi_n)(1+\delta_{n,j}) \prod_{i=j}^{M} (1+\zeta_{n,i}), \quad \text{where } j = 1,2,\ldots,M$$

$$\phi_{n,1} = (1+\xi_n)(1+\epsilon_{n,1}) \prod_{i=2}^{L} (1+\eta_{n,i})$$

$$\phi_{n,j} = (1+\xi_n)(1+\epsilon_{n,j}) \prod_{i=j}^{L} (1+\eta_{n,i}), \quad \text{where } j = 2,3,\ldots,L$$

and

$$v_n = \sum_{k=0}^{M} b_k\ x_{n-k} - \sum_{k=1}^{L} a_k\ v_{n-k} + \sum_{k=0}^{M} \delta'_{n,k} + \sum_{k=1}^{M} \zeta'_{n,k} + \sum_{k=1}^{L} \epsilon'_{n,k} + \sum_{k=2}^{L} \eta'_{n,k} + \xi'_n \quad (15)$$

For the error analysis, we need to calculate the y_n and v_n sequences from the equations (14) and (15), and compare them with the ideal output sequence w_n specified by the equation (5) to obtain the corresponding errors e_n, e'_n, and e''_n, according to the equations (9), (10), and (11), respectively. Therefore, the difference equations for the errors between the different levels showing the accumulation of the roundoff error are derived as the following error analysis cases:

1. *Real to Floating-Point Error Analysis:*

$$e_n\ +\ \sum_{k=1}^{L} a_k\ e_{n-k} = \sum_{k=0}^{M} b_k\ (\theta_{n,k} - 1)\ x_{n-k}\ -\ \sum_{k=1}^{L} a_k\ (\phi_{n,k} - 1)\ y_{n-k} \quad (16)$$

2. *Real to Fixed-Point Error Analysis:*

$$e'_n\ +\ \sum_{k=1}^{L} a_k\ e'_{n-k} = \sum_{k=0}^{M} \delta'_{n,k} + \sum_{k=1}^{M} \zeta'_{n,k} + \sum_{k=1}^{L} \epsilon'_{n,k} + \sum_{k=2}^{L} \eta'_{n,k} + \xi'_n \quad (17)$$

3. *Floating-Point to Fixed-Point Error Analysis:*

$$e''_n\ +\ \sum_{k=1}^{L} a_k\ e''_{n-k} = \sum_{k=0}^{M} \delta'_{n,k} + \sum_{k=1}^{M} \zeta'_{n,k} + \sum_{k=1}^{L} \epsilon'_{n,k} + \sum_{k=2}^{L} \eta'_{n,k} + \xi'_n - \quad (18)$$

$$\sum_{k=0}^{M} b_k\ (\theta_{n,k} - 1)\ x_{n-k}\ +\ \sum_{k=1}^{L} a_k\ (\phi_{n,k} - 1)\ y_{n-k}$$

Similar analysis is performed for the parallel and cascade forms of realization based on the error flowgraphs as shown in Figures 4 and 5, respectively.

In HOL, we first specified a parametric Lth-order digital filters at the real, floating-point, and fixed-point abstraction levels, as predicates in higher-order logic. The direct form is defined in HOL using the equation (5). For the real specification, we used the expression *sum (m,n) f* denoting $\sum_{i\,=\,m}^{m+n-1} f(i)$, which is a function available in the HOL real library [7] and defines the finite summation on the real numbers. For the floating-point and fixed-point specifications, we defined similar functions for the finite summations on the floating-point (*float_sum*) and fixed-point (*fxp_sum*) numbers, using the recursive definition in HOL. For the parallel form, we first specified the ith parallel path using the equation (6). Then, we specified the entire output as defined in equation (7), using the finite summation functions. Finally, we specified the cascade form of realization as defined in equation (8), using recursive definitions in HOL. For the error analysis

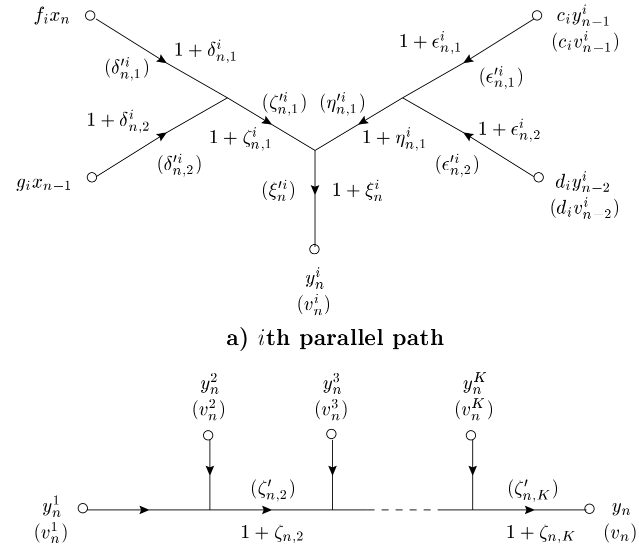


Fig. 4. Error flowgraph for Lth-order filter (Parallel form)

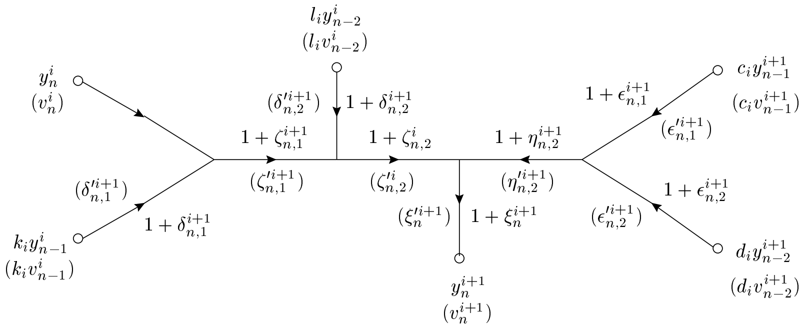


Fig. 5. Error flowgraph for Lth-order filter (Cascade form)

of the digital filters in HOL, we first established lemmas to compute the output real values of the floating-point and fixed-point filters according to the equations (14) and (15), for the direct form of realization. For this, we need to define the finite product on the real numbers. We defined this function in HOL recursively as the expression *mul (m,n) f* denoting $\prod_{i\,=\,m}^{m+n-1} f(i)$. Finally, we defined the errors as the differences between the output of the real filter specification, and the corresponding real values of the floating-point and fixed-point filter implementations (*Real_To_Float_Error*,*Real_To_Fxp_Error*), and the error in transition from the floating-point to fixed-point levels (*Float_To_Fxp_Error*), according to the equations (9), (10), and (11), respectively. Then, we established lemmas for the accumulation of the round-off error between the different levels, according to the equations (16), (17), and (18). Finally, we proved these lemmas using the fundamental floating-point and fixed-point error analysis lemmas, based on the error models presented in Section 3. The lemmas are proved by induction on the parameters L and M for the direct form of realization. Similar analysis is performed in HOL for the parallel and cascade realization forms. For these cases, we proved the corresponding lemmas by induction on the parameter K which is defined as the number of the internal sub-filters connected in parallel or cascade forms to generate the final output. The corresponding error analysis lemmas in HOL for the direct form of realization are listed in Appendix A.

5 Conclusions

In this paper, we describe a comprehensive methodology for the error analysis of generic digital filters using the HOL theorem prover. The proposed approach covers the three canonical forms (direct, parallel and cascade) of realization entirely specified in HOL. We make use of existing theories in HOL on real, IEEE standard based floating-point, and fixed-point arithmetic to model the ideal filter specification and the corresponding implementations in higher-order logic. We used valuation functions to define the errors as the differences between the real values of the floating-point and fixed-point filter implementation outputs and the corresponding output of the ideal real filter specification. Finally, we established fundamental analysis lemmas as our model to derive expressions for the accumulation of the roundoff error in digital filters. Related work did exist since the late sixties using theoretical paper-and-pencil proofs and simulation techniques. We believe this is the first time a complete formal framework is considered using mechanical proofs in HOL for the error analysis of digital filters. As a future work, we plan to extend these lemmas to analyse the worst-case, average, and variance errors. We also plan to extend the verification to the lower levels of abstraction, and prove that the implementation of a digital filter at the register transfer and netlist gate levels implies the corresponding fixed-point specification using classical hierarchical verification in HOL, hence bridging the gap between the hardware implementation and high levels of the mathematical specification. Finally, we plan to link HOL with computer algebra systems to create a sound, reliable, and powerful system for the verification of DSP systems. This opens new avenues in using formal methods for the verification of DSP systems as a complement to the traditional theoretical (analytical) and simulation techniques. We are currently investigating the verification of other DSP algorithms such as the fast Fourier transform (FFT) which is widely used as a building block in the design of complex wired and wireless communication systems.

References

1. B. Akbarpour, S. Tahar, and A. Dekdouk, "Formalization of Cadence SPW Fixed-Point Arithmetic in HOL," In Integrated Formal Methods, LNCS 2335, Springer-Verlag, pp. 185-204, 2002.

2. B. Akbarpour, and S. Tahar, “Modeling SystemC Fixed-Point Arithmetic in HOL,” In Formal Methods and Software Engineering, LNCS 2885, Springer-Verlag, pp. 206-225, 2003.
3. Cadence Design Systems, Inc., “Signal Processing WorkSystem (SPW) User's Guide,” USA, July 1999.
4. G. Forsythe, and C. B. Moler, “Computer Solution of Linear Algebraic Systems,” Prentice-Hall, 1967.
5. M. J. C. Gordon, and T. F. Melham, “Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic,” Cambridge University Press, 1993.
6. B. Gold, and C. M. Radar, “Effects of Quantization Noise in Digital Filters,” In Proceedings AFIPS Spring Joint Computer Conference, vol. 28, pp. 213-219, 1966.
7. J. R. Harrison, “Constructing the Real Numbers in HOL,” Formal Methods in System Design, 5 (1/2): 35-59, 1994.
8. J. R. Harrison, “A Machine-Checked Theory of Floating-Point Arithmetic,” In Theorem Proving in Higher Order Logics, LNCS 1690, Springer-Verlag, pp. 113-130, 1999.
9. J. R. Harrison, “Floating-Point Verification in HOL Light: The Exponential Function,” Formal Methods in System Design, 16 (3): 271-305, 2000.
10. J. R. Harrison, and L. Théry, “A Skeptic's Approach to Combining Hol and Maple,” Journal of Automated Reasoning, 21: 279-294, 1998.
11. M. Huhn, K. Schneider, T. Kropf, and G. Logothetis, “Verifying Imprecisely Working Arithmetic Circuits,” In Proceedings Design Automation and Test in Europe Conference, pp. 65-69, March 1999.
12. L. B. Jackson, “Roundoff-Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form,” IEEE Transactions on Audio and Electroacoustics, AU-18: 107-122, June 1970.
13. J. F. Kaiser, “Digital Filters,” In System Analysis by Digital Computer, F. F. Kuo and J. F. Kaiser, Eds., pp. 218-285, Wiley, 1966.
14. J. B. Knowles, and R. Edwards, “Effects of a Finite-Word-Length Computer in a Sampled-Data Feedback System,” IEE Proceedings, 112: 1197-1207, June 1965.
15. B. Liu, and T. Kaneko, “Error Analysis of Digital Filters Realized with Floating-Point Arithmetic,” Proceedings of the IEEE, 57: 1735-1747, October 1969.
16. Mathworks, Inc., “Simulink Reference Manual,” USA, 1996.
17. A. V. Oppenheim, and C. J. Weinstein, “Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform,” Proceedings of the IEEE, 60 (8): 957-976, August 1972.
18. A. V. Oppenheim, and R. W. Schafer, “Discrete-Time Signal Processing,” Prentice-Hall, 1989.
19. I. W. Sandberg, “Floating-Point-Roundoff Accumulation in Digital Filter Realization,” The Bell System Technical Journal, 46: 1775-1791, October 1967.
20. Synopsys, Inc., “CoCentric^{TM} System Studio User's Guide,” USA, August 2001.
21. C. Weinstein, and A. V. Oppenheim, “A Comparison of Roundoff Noise in Floating Point and Fixed Point Digital Filter Realizations,” Proceedings of the IEEE (Proceedings Letters), 57: 1181-1183, June 1969.
22. H. Keding, M. Willems, M. Coors, and H. Meyr, “FRIDGE: A Fixed-Point Design and Simulation Environment,” In Proceedings Design Automation and Test in Europe Conference, pp. 429-435, February 1998.
23. J. H. Wilkinson, “Rounding Errors in Algebraic Processes,” Prentice-Hall, 1963.

A Digital Filter Error Analysis Lemmas in HOL

```
Lemma 7: L_ORDER_FILTER_DIRECT_FORM_REAL_TO_FLOAT_THM
⊢  L_Order_Filter_Direct_Form_Ideal_Spec a b x w M L ∧
   L_Order_Filter_Direct_Form_Float_Imp X a′ b′ x′ y M L ⟹
   ∃ t f.
   if (L = 0) then
        (Real_To_Float_Error n = sum (0,SUC M) (λ i. Val (b′ i) *
        (t i − 1) * Val (x′ (n − i))))
   else
        ((Real_To_Float_Error n + sum (1,L) (λ i. a i *
        Real_To_Float_Error (n − i)) = sum (0,SUC M) (λ i. Val (b′ i) *
        (t i − 1) * Val (x′ (n − i))) − sum (1,L) (λ i. Val (a′ i) *
        (f i − 1) * Val (y (n − i))))) ∧
   ∃ k d p e z.
   (abs k ≤ (1 / 2 pow ((fracwidth X) + 1))) ∧
   (∀ i. (i ≤ M) ⟹  (abs (d i) ≤ (1 / 2 pow ((fracwidth X) + 1)))) ∧
   (∀ i. (i ≤ M) ⟹  (abs (p i) ≤ (1 / 2 pow ((fracwidth X) + 1)))) ∧
   (∀ i. (i ≤ L) ⟹  (abs (e i) ≤ (1 / 2 pow ((fracwidth X) + 1)))) ∧
   (∀ i. (i ≤ L) ⟹  (abs (z i) ≤ (1 / 2 pow ((fracwidth X) + 1)))) ∧
   (t 0 = (1 + k) * (1 + d 0) * (mul (1,M) (λ i. (1 + p i)))) ∧
   (∀ j. (1 ≤ j ∧ j ≤ M) ⟹  (t j = (1 + k) * (1 + d j) *
   (mul (j,(M − (j − 1))) (λ j. (1 + p j))))) ∧
   (f 1 = (1 + k) * (1 + e 1) * (mul (2,(L − 1)) (λ i. (1 + z i)))) ∧
   (∀ j. (2 ≤ j ∧ j ≤ L) ⟹
   (f j = (1 + k) * (1 + e j) * (mul (j,(L − j + 1)) (λ j. (1 + z j)))))
```

```
Lemma 8: L_ORDER_FILTER_DIRECT_FORM_REAL_TO_FXP_THM
⊢  L_Order_Filter_Direct_Form_Ideal_Spec a b x w M L ∧
   L_Order_Filter_Direct_Form_Fxp_Imp X′ a′′ b′′ x′′ v M L ⟹
   ∃ k′ d′ p′ e′ z′.
   abs k′ ≤ (1 / 2 pow (fracbits X′)) ∧
   (∀ i. (i ≤ M) ⟹  abs (d′ i) ≤ (1 / 2 pow (fracbits X′))) ∧
   (∀ i. (i ≤ M) ⟹  abs (p′ i) ≤ (1 / 2 pow (fracbits X′))) ∧
   (∀ i. (i ≤ L) ⟹  abs (e′ i) ≤ (1 / 2 pow (fracbits X′))) ∧
   (∀ i. (i ≤ L) ⟹  abs (z′ i) ≤ (1 / 2 pow (fracbits X′))) ∧
   if (L = 0) then
        (Real_To_Fxp_Error n = sum (0,SUC M) (λ i. d′ i) +
        sum (1,M) (λ j. p′ j) + k′)
   else
        (Real_To_Fxp_Error n + sum (1,L) (λ i. a i * Real_To_Fxp_Error
        (n − i)) = sum (0,SUC M) (λ i. d′ i) + sum (1,M) (λ j. p′ j) +
        sum (1,L) (λ i. e′ i) + sum (2,(L − 1)) (λ j. z′ j) + k′))
```

```
Lemma 9: L_ORDER_FILTER_DIRECT_FORM_FLOAT_TO_FXP_THM
⊢  L_Order_Filter_Direct_Form_Ideal_Spec a b x w M L ∧
   L_Order_Filter_Direct_Form_Float_Imp X a′ b′ x′ y M L ∧
   L_Order_Filter_Direct_Form_Fxp_Imp X′ a′′ b′′ x′′ v M L ⟹
   ∃ t f k′ d′ p′ e′ z′.
   if (L = 0) then
        (Float_To_Fxp_Error n = sum (0,SUC M) (λ i. d′ i) +
        sum (1,M) (λ j. p′ j) + k′ − (sum (0,SUC M)
        (λ i. Val (b′ i) * (t i − 1) * Val (x′ (n − i)))))
   else
        (Float_To_Fxp_Error n + sum (1,L) (λ i. a i * Float_To_Fxp_Error
        (n − i)) = sum (0,SUC M) (λ i. d′ i) + sum (1,M) (λ j. p′ j) +
        sum (1,L) (λ i. e′ i) + sum (2,(L − 1)) (λ j. z′ j) + k′ −
        sum (0, (SUC M)) (λ i. Val (b′ i) * (t i − 1) * Val (x′ (n − i)))
        + sum (1,L) (λ i. Val (a′ i) * (f i − 1) * Val (y (n − i)))))
```