Error Analysis of Digital Filters using HOL Theorem Proving

Behzad Akbarpour and Sofiène Tahar

Electrical and Computer Engineering Department

Concordia University, Montreal, Canada

Email: {behzad, tahar}@ece.concordia.ca

Technical Report

February 2004

Abstract

When a digital filter is realized with floating- and fixed-point arithmetics, errors and constraints due to finite word length are unavoidable. These error problems have already been extensively studied using theoretical paper and pencil proofs and simulation techniques since the late sixties. In this report, we show how this error analysis can be mechanically performed using the HOL theorem prover. We first model the ideal real filter specification and the corresponding floating- and fixed-point implementations as predicates in higher-order logic. For this, we make use of existing theories in HOL on real, IEEE standard based floating-point, and fixed-point arithmetics. We use valuation functions to find the real values of the floating- and fixed-point filter outputs and define the error as the difference between these values and the corresponding output of the ideal real specification. Then, we established fundamental analysis lemmas as our model to derive expressions for the accumulation of roundoff error in a parametric Lth-order digital filter, for each of the three canonical forms of realization: direct, parallel, and cascade. The HOL formalization and proofs are found to be in good agreement with the theoretical counterparts.

Keywords: Error Analysis, Digital Filters, Theorem Proving, Fixed-point, Floating-point, HOL

Contents

1 Introduction

Signal processing by 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 digital filters are their flexibility, reliability, and modularity. Excellent methods have been developed to design these filters with desired characteristics. The design of the filter is the process of the determination of a transfer function from a set of specifications given either in frequency domain, or in time domain, or for some applications, in both. The design of digital filters starts from an ideal real specification. In a theoretical analysis of digital filters, we generally assume that signal values and system coefficients are represented in the real number system and expressed to infinite precision. When implemented as a special-purpose digital hardware or as a computer algorithm, we must represent signals and coefficients in some digital number system that must always be of finite precision. Despite their advantages, there is an inherent accuracy problem associated with digital filters, since the signals are represented by a finite number of bits and the arithmetic operations must be carried out with an accuracy limited by this finite word length. There are a variety of types of arithmetic used in the implementation of digital systems. Among the most common are floating- 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 real to floating- and fixed-point 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 (Computer Aided Design) tools. In the design flow of digital filters we will focus on the transition from real to floating- and fixed-point levels.

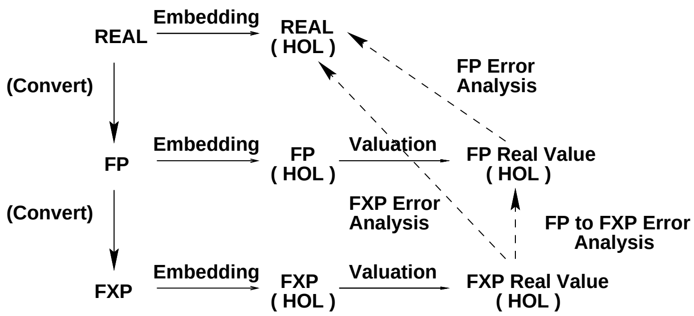


Figure 1: Error Analysis Approach

In this report we describe the error analysis of digital filters using 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- and fixed-point implementations as predicates in higher-order logic. For this, we make use of existing theories in HOL on construction of real numbers [7], formalization of IEEE-754 standard [10] based floating-point arithmetic [8, 9], and formalization of fixed-point arithmetic [1, 2]. We use valuation functions to find the real values of the floating- and fixed-point filter outputs and define the error as the difference between these values and the corresponding output of the ideal real specification. Then we establish fundamental lemmas on the error analysis of floating- and fixed-point rounding and arithmetic operations against their abstract mathematical counterparts. Finally, we use these lemmas as a model to derive expressions for the accumulation of roundoff error in a parametric Lth-order digital filter, for each of the three canonical forms of realization: direct, parallel, and cascade [17]. 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 theoretical work on computing the errors due to finite precision effects in the realization of digital filters with floating- and fixed-point arithmetics has been extensively studied since the late sixties [15], it is for the first time in this report, that a formalization and proof of this analysis is done using a mechanical theorem prover, here HOL. The formal results are found to be in good agreement with the theoretical ones.

The rest of this report 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 floating- 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 report.

2 Related Work

Work on the analysis of errors due to finite precision effects in the realization of digital filters has always existed since their early days, however, using theoretical paper and pencil proofs and simulation techniques. For digital filters realized with fixed-point arithmetic, the error problems have been studied extensively. For instance, Knowles and Edwards [14] proposed a method for analysis of finite word length effects in a feedback system that can be applied directly to the calculation of roundoff errors in fixed-point digital filters. Their paper also demonstrates that a higher-order filter should be realized as a cascade or parallel connection of first- and second-order filter sections from the error point of view. Gold and Radar [6] carried out a detailed analysis of the roundoff error for first- and second-order fixed-point filters. Jackson [11] has analyzed the roundoff noise for cascade and parallel realizations of fixed-point digital filters under certain dynamic range constraints. While the roundoff noise for fixed-point arithmetic enters into the system additively, it is a multiplicative component in the case of floating-point arithmetic. This problem is analyzed first by Sandberg [18], who discussed roundoff error accumulation and quantization effects of the input in the direct realization of the filter excited by a deterministic input, and 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 floating-point arithmetic and calculated the error at the output due to 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 filters are given. Kan and Aggarwal [12] studied the error accumulation in a digital filter employing floating-point arithmetic. Oppenheim and Weinstein [16] discussed in some detail the effects of finite register length on implementations of linear recursive difference equation digital filters, and the fast Fourier transform (FFT) algorithm. For these algorithms, the differing quantization effects of floating- and fixed-point arithmetic are examined and compared. Comparisons of the roundoff noise in digital filters using different types of arithmetics have also been reported in [19]. Most work compare the theoretical results with the experimental simulation results to validate the proposed analysis.

In this report, we show how the above error analysis for digital filters can be mechanically performed using the HOL theorem prover. Our focus will be on the process of translating the hand proofs into equivalent proofs in HOL. We propose a formal verification methodology based on the shallow embedding [3] of the intended meaning of digital filter block designs at the real, floating- and fixed-point levels in higher-order logic. Thereafter, we complete the formal proof in the HOL theorem proving environment hence increasing the confidence of reusability of the blocks, and providing a superior approach to validation by simulation. The analysis we propose is mostly inspired by the work done by Liu and Kaneko [15] who proposed a general approach to the error analysis problem of digital filters using floating-point arithmetic. Following similar approach, we have extended this analysis for the fixed-point digital filters. The methodology presented in this report is to our best knowledge, the first project of its kind in which we cover the formal specification and verification of integrated digital filters at different abstraction levels starting from real specification to floating- and fixed-point algorithmic descriptions. Good agreement between formal and theoretical results are obtained.

To formally specify and verify digital filters in higher-order logic, the need for a formalization of real, floating-, and fixed-point arithmetics is unavoidable. For real and floating-point arithmetics, we have used the work done by Harrison who constructed the real numbers library [7] in HOL, and developed a generic floating-point theory [8] to define and prove the most fundamental terms and lemmas about the IEEE-754 standard [10], and to formalize and verify floating-point algorithms such as exponential function [9] against their behavioral specification[1]. For fixed-point arithmetic, we developed in [1] and [2] a theory in HOL on similar properties of fixed-point arithmetic as described in the Cadence SPW tool and the SystemC language, respectively. Using these formalizations, we were able to describe the behavior of digital filters at different algorithmic levels in HOL and analyze the error in transition from one level to the other one.

3 Error Analysis Models

In this section we introduce the fundamental error analysis theorems [20, 4], and the corresponding lemmas in HOL for floating- [8, 9] and fixed-point [1, 2] arithmetics. These theorems are then used in the next sections as a model for the analysis of roundoff error in digital filters.

[1] These theories were developed using HOL Light. As part of this project, we have ported most of them to HOL4 Kananaskis 2.

3.1 Floating-Point Error Model

In analyzing the effect of floating-point roundoff, the effect of rounding will be represented multiplicatively. The following theorem is the most fundamental in floating-point rounding-error theory which gives a convenient expression for the relative error committed if a given real number x is rounded to the closest floating-point number x_R [20, 4].

Theorem 1: If x is a real number within the floating-point range, then

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

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

In HOL, we proved this theorem in IEEE single precision floating-point format ($p = 24$) for the case of rounding to nearest as follows:

```
Lemma 1: FLOAT_ROUND_RELATIVE_ERROR
⊢  normalizes x ⟹  ∃ e. abs (e) < (1 / 2 pow 24) ∧
   (Val (float (round float_format 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], *Val* is the floating-point valuation function, *float* is the bijection function that converts a triple of natural numbers to floating-point type, *round* is the floating-point rounding function, and *float_format* is the floating-point format [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 float_format − 2) ∧
   (2 pow j / 2 pow 126) ≤ abs x ∧ abs x < (2 pow SUC j / 2 pow 126)
```

where the function *emax* defines the maximum exponent in a given floating-point format. 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 150)
```

which states that if the absolute value of a real number is in the representable range of normalized floating-point numbers and located in the jth binade, i.e., its absolute value is less than $2^{j+1}/2^{126}$ and greater than or equal to $2^j/2^{126}$, then the absolute value of error is less than or equal to $2^j/2^{150}$. 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 float_format To_nearest x)) − x)
```

Since $2^j \ / \ 2^{126} \ \leq \ |x|$ as proved in Lemma 2 for the real numbers in the normalized region, we have $(\ |error\ x| \ / \ |x| \ ) \ \leq \ (2^j \ / \ 2^{150}) \ / \ (2^j \ / \ 2^{126}) = (1 \ / \ 2^{24})$. 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

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

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 absolute analysis of rounding error in floating-point arithmetic operations (`FLOAT_ADD,FLOAT_SUB,FLOAT_MUL,FLOAT_DIV`) 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 24) ∧
       (Val (a + b) = (Val a + Val b) * (1 + e))

Lemma 5: FLOAT_SUB_RELATIVE
⊢  Finite a ∧ Finite b ∧ normalizes (Val a − Val b)
   ⟹  Finite (a − b) ∧ ∃ e. abs e ≤ (1 / 2 pow 24) ∧
       (Val (a − b) = (Val a − Val b) * (1 + e))

Lemma 6: FLOAT_MUL_RELATIVE
⊢  Finite a ∧ Finite b ∧ normalizes (Val a * Val b)
   ⟹  Finite (a * b) ∧ ∃ e. abs e ≤ (1 / 2 pow 24) ∧
       (Val (a * b) = (Val a * Val b) * (1 + e))

Lemma 7: FLOAT_DIV_RELATIVE
⊢  Finite a ∧ Finite b ∧ ¬ Iszero b ∧ normalizes (Val a / Val b)
   ⟹  Finite (a / b) ∧ ∃ e. abs e ≤ (1 / 2 pow 24) ∧
       (Val (a / b) = (Val a / Val b) * (1 + e))
```

where the function *Finite* defines the finiteness criteria for the floating-point numbers, and the function *Iszero* checks if a given floating-point number is equal to zero. Note that we use the conventional symbols for arithmetic operations on floating-point numbers using the operator overloading feature of HOL. The lemmas are composed of two parts. The first part is about the finiteness of the floating-point operation output. It states that for each pair of finite floating-point numbers, if the real result is in the representable range of normalized floating-point numbers, then the output result is also finite. For floating-point division, the second operand should be nonzero to avoid the division by zero. The second part of the lemmas states that the result of a floating-point operation is the exact result, perturbed by a relative error of bounded magnitude.

3.2 Fixed-Point Error Model

While the rounding error for floating-point arithmetic enters into the system multiplicatively, it is an additive component for fixed-point arithmetic. In this case the fundamental error analysis theorem can be stated as follows [20].

Theorem 3: If x is a real number within the range of fixed-point numbers with format X, then

$$x_R \;=\; x \;+\; \epsilon, \;\; \mathit{where} \;\; |\epsilon| \;\leq\; 2^{-\mathit{fracbits}\ (X)} \tag{3}$$

and *fracbits* is 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 8: FXP_ROUND_ABSOLUTE_ERROR_BOUND
⊢  (validAttr X) ∧ (representable X x) ⟹
   abs (Fxp_error X x) ≤ inv (2 pow fracbits (X))
```

where the function *validAttr* defines the validity for a set of fixed-point attributes to represent the fixed-point format, *representable* defines the criteria for an arbitrary real number to be in the representable range of a given fixed-point format defined by its maximum and minimum values, and *Fxp_error* is the fixed-point rounding error function which defines the error resulting from rounding a real number to a fixed-point value.

The verification of fixed-point arithmetic operations using the *absolute* error analysis of fixed-point rounding (Theorem 3) can be stated as in the following theorem in which 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

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

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 lemmas [1]:

```
Lemma 9: FXP_ADD_ABSOLUTE
⊢  (Isvalid a) ∧ (Isvalid b) ∧ validAttr (X) ∧
   representable X (value a + value b) ⟹  (Isvalid (FxpAdd X a b)) ∧
   ∃ e. abs e ≤ inv (2 pow (fracbits X)) ∧
   value (FxpAdd X a b) = (value a + value b) + e

Lemma 10: FXP_SUB_ABSOLUTE
⊢  (Isvalid a) ∧ (Isvalid b) ∧ validAttr (X) ∧
   representable X (value a − value b) ⟹  (Isvalid (FxpSub X a b)) ∧
   ∃ e. abs e ≤ inv (2 pow (fracbits X)) ∧
   value (FxpSub X a b) = (value a − value b) + e
```

```
Lemma 11: FXP_MUL_ABSOLUTE
⊢  (Isvalid a) ∧ (Isvalid b) ∧ validAttr (X) ∧
   representable X (value a * value b) ⟹  (Isvalid (FxpMul X a b)) ∧
   ∃ e. abs e ≤ inv (2 pow (fracbits X)) ∧
   value (FxpMul X a b) = (value a * value b) + e

Lemma 12: FXP_DIV_ABSOLUTE
⊢  (Isvalid a) ∧ (Isvalid b) ∧ ¬ (value b = 0) ∧ validAttr (X) ∧
   representable X (value a / value b) ⟹  (Isvalid (FxpDiv X a b)) ∧
   ∃ e. abs e ≤ inv (2 pow (fracbits X)) ∧
   value (FxpDiv X a b) = (value a / value b) + e
```

where the function *Isvalid* defines the validity of a fixed-point number, *value* is the fixed-point valuation function, and *FxpAdd, FxpSub, FxpMul,* and *FxpDiv* are the corresponding functions for fixed-point addition, subtraction, multiplication, and division operations, respectively. According to these lemmas, if the input fixed-point numbers and the output attributes are valid, then the result of fixed-point operations is valid. For fixed-point division, the second operand should be nonzero to avoid the division by zero. The result of the fixed-point operations is the exact result, perturbed by an absolute error of bounded magnitude.

4 Error Analysis of Digital Filters using HOL

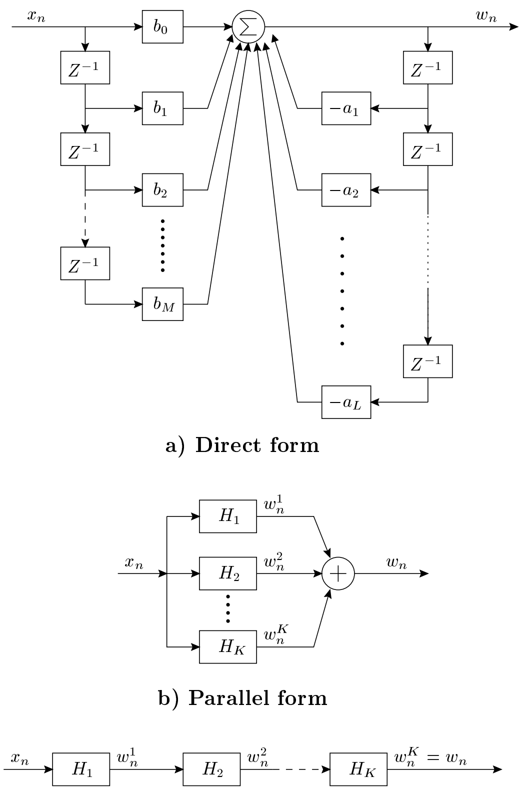
In this section, the principal results for roundoff accumulation in digital filters using theorem proving are derived and summarized. We shall employ the models for floating- and fixed-point roundoff errors in HOL presented in the previous section. To illustrate our approach, we first considered the case of first- and second-order digital filters. Then, we extended this analysis to the general case of the direct form realization of a parametric Lth-order filter of which the first- and second-order filters are special cases. Finally, we applied our approach to the parallel and cascade forms. Using these forms, larger-order filters can be treated as a combination of first- and second-order filters. Then, the total error is computed by accumulating the error in all internal sub-filters. In the following, we will first describe in details the theory behind the analysis and then explain how each step of this analysis is performed in HOL.

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

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

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 (Figure 2) [17].

If the output sequence is calculated by using Equation (5), the digital filter is said to be realized in direct form. Figure 2 (a) illustrates the direct form realization of the filter using corresponding blocks for addition, and multiplication by a constant operations, and the delay element. The implementation of a digital filter in parallel form is shown in Figure 2 (b) in which the entire filter is visualized as the parallel connection of simpler filters H_i of



a) Direct form

b) Parallel form

c) Cascade form

Figure 2: Canonical forms of digital filter realizations

lower order. In this case, K intermediate outputs $\{w_n^i\}$, $i = 1, 2, ..., K$ are first calculated and then summed to form the total output $\{w_n\}$. The implementation of a digital filter in 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 this is the input to the second filter. Continuing in this manner, the final output $w_n^K = w_n$ is calculated.

In terms of Z transform, Equation (5) can be written as:

$$W(z) = H(z)X(z) \tag{6}$$

where $X(z)$ and $W(z)$ are the Z transforms of the input and output sequences respectively, and $H(z)$ is the so-called pulse transfer function defined as:

$$H(z) = \frac{\sum_{i=0}^{M} b_i \ z^{-i}}{1 + \sum_{i=1}^{L} a_i \ z^{-i}} \tag{7}$$

If the order of the filter L is higher than 2, $H(z)$ may be written as a sum of partial fractions as:

$$H(z) = \sum_{i=1}^{K} H_i(z) \tag{8}$$

where

$$H_i(z) = \frac{N_i(z)}{D_i(z)} = \frac{f_i + g_i z^{-1}}{1 + c_i z^{-1} + d_i z^{-2}} \tag{9}$$

In this case, the filter is said to be realized in the parallel form. The parameters f_i, g_i, c_i, and d_i are obtained from a_i and b_i by partial fraction expansion of Equation (7). The above expression includes the case of a real pole or a constant by setting $g_i = d_i = 0$ or $g_i = c_i = d_i = 0$, respectively. For parallel form of realization as described by Equations (8) and (9), let $\{w_n^i\}$, $i = 1, 2, ..., K$, be the output sequence of the filter $H_i(z)$. Then, for the input sequence $\{x_n\}$ we have:

$$w_n^i = f_i x_n + g_i x_{n-1} - c_i w_{n-1}^i - d_i w_{n-2}^i \tag{10}$$

The output of the entire filter $H(z)$, w_n, is then related to w_n^i by:

$$w_n = w_n^1 + w_n^2 + ... + w_n^K \tag{11}$$

Similarly, if $H(z)$ is written as a product of quadratic factors:

$$H(z) = c \prod_{i=1}^{K} H_i(z) \tag{12}$$

where c is a constant and

$$H_i(z) = \frac{N_i(z)}{D_i(z)} = \frac{1 + k_i z^{-1} + l_i z^{-2}}{1 + c_i z^{-1} + d_i z^{-2}} \tag{13}$$

The original filter is said to be realized in the cascade form. The parameters k_i, l_i, c_i, and d_i are obtained from a_i and b_i by expansion of Equation (7). Again the expression for $H_i(z)$ includes the possibility of a real pole or a real zero by simply setting $d_i = 0$ or $l_i = 0$, respectively. For cascade form of realization as described by Equations (12) and (13), let w_n^i be the output of the ith section. Since the output of the ith transfer function $H_i(z)$ is the input of the $(i+1)$th transfer function $H_{i+1}(z)$, the following equation holds:

$$w_n^{i+1} = w_n^i + k_i w_{n-1}^i + l_i w_{n-2}^i - c_i w_{n-1}^{i+1} - d_i w_{n-2}^{i+1} \tag{14}$$

There are three common sources of errors associated with the filter of 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 accuracy:** 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 Equation (5) the actual computed output reference is in general different from $\{w_n\}$. We denote the actual floating- and fixed-point outputs by $\{y_n\}$ and $\{v_n\}$, respectively. Then, we define the corresponding errors at the nth output sample as:

$$e_n = y_n - w_n \tag{15}$$

$$e'_n = v_n - w_n \tag{16}$$

$$e''_n = v_n - y_n \tag{17}$$

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

4.1 First-Order Filter

To illustrate our approach for the analysis of roundoff errors with floating- and fixed-point arithmetic, let us consider a first-order filter. Let x_n, w_n, and a denote the ideal real input signal, output response, and the coefficient of the filter, that is, the filter parameters with no roundoff noise and x'_n, y_n, a' and x''_n, v_n, a'' denote the corresponding actual floating-point and fixed-point filter parameters in the presence of roundoff noise, respectively. Then we can write:

$$w_n = aw_{n-1} + x_n \tag{18}$$

The corresponding computed floating- and fixed-point outputs are:

$$y_n = fl\ [a'y_{n-1} + x'_n] \tag{19}$$

$$v_n = fxp\ [a''v_{n-1} + x''_n] \tag{20}$$

The notations $fl\ (.)$ and $fxp\ (.)$ are used to denote that the operations are performed using the floating- and fixed-point arithmetics, respectively. In HOL, we specified the first-order digital filter in real, floating-, and fixed-point abstraction levels, as predicates in higher-order logic. The corresponding codes are as follows.

```
⊢def  First_Order_Filter_Ideal_Spec a x w =
       ∀n. w n = a * w (n - 1) + x n

⊢def  First_Order_Filter_Float_Imp a′ x′ y =
       ∀n. y n = a′ * y (n - 1) + x′ n

⊢def  First_Order_Filter_Fxp_Imp X a″ x″ v =
       ∀n. v n = (FxpAdd X (FxpMul X a″ (v (n - 1))) (x″ n))
```

The calculation of Equation (19) is to be performed in the following manner. First the product $a'y_{n-1}$ is calculated separately. Then it is added to x'_n to obtain y_n. Similar discussion can be applied for the calculation of the fixed-point output v_n according to the Equation (20). Following Sandberg [18], a flowgraph as given in Figure 3 may be drawn by using the fundamental error analysis theorems on floating-point and fixed-point arithmetic operations introduced in Section 3 as given in Equations (2) and (4). The quantities ϵ_n and ξ_n are errors caused by roundoff at each floating-point arithmetic step. The corresponding error quantities for fixed-point roundoff are ϵ'_n and ξ'_n.

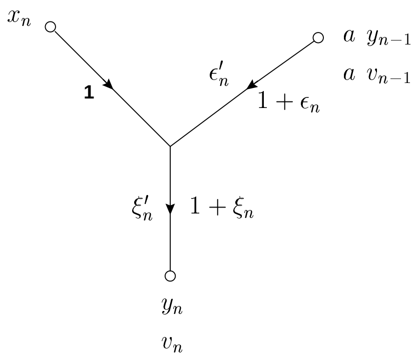


Figure 3: Error flowgraph for the first-order filter

Therefore the actual y_n is seen to be given explicitly by

$$y_n = [ay_{n-1}(1+\epsilon_n) + x_n](1+\xi_n) \tag{21}$$

In HOL, we established the following lemma to compute the real value of the floating-point output for the first-order filter according to the Equation (21).

```
Lemma 13: FIRST_ORDER_FILTER_FLOAT_OUTPUT_VALUE
⊢  First_Order_Filter_Float_Imp a′ x′ y ⟹
   ∃ e1 e2. abs e1 ≤ (1 / 2 pow 24) ∧ abs e2 ≤ (1 / 2 pow 24) ∧
   (Val (y n) = (Val (a′) * Val (y (n − 1)) * (1 + e1) +
   Val (x′ n)) * (1 + e2))
```

Similarly, the actual fixed-point output v_n is given explicitly by

$$v_n = [av_{n-1} + \epsilon'_n + x_n] + \xi'_n \tag{22}$$

and the corresponding lemma is established in HOL as follows:

```
Lemma 14: FIRST_ORDER_FILTER_FXP_OUTPUT_VALUE
⊢  First_Order_Filter_Fxp_Imp X a″ x″ v ⟹
   ∃ e1 e2. abs e1 ≤ inv (2 pow (fracbits X)) ∧
   abs e2 ≤ inv (2 pow (fracbits X)) ∧
   value (FxpAdd X (FxpMul X a″ (v (n − 1))) (x″ n)) =
   ((value (a″) * value (v (n − 1)) + e1) + value (x″ n)) + e2
```

For error analysis, we need to calculate the y_n and v_n sequences from Equations (21) and (22), and compare them with the ideal output sequence w_n specified by the Equation (18) to obtain the corresponding errors e_n, e'_n, and e''_n, according to the Equations (15), (16), and

(17). Therefore, the difference equations for the errors between different levels showing the accumulation of roundoff error are derived as follows:

1) Floating-Point Error Analysis:

$$e_n - a\ e_{n-1} = a\ y_{n-1}(\epsilon_n + \xi_n + \epsilon_n \xi_n) + x_n \xi_n \tag{23}$$

To prove this theorem in HOL, we first defined the error as the difference between the output of the real filter specification, and the corresponding real value of the floating-point filter implementation (*Float_Error*). Then, we established the following lemma for the accumulation of round-off error in floating-point realization of the first-order filter, according to the Equation (23).

```
Lemma 15: FIRST_ORDER_FILTER_FLOAT_TO_REAL_THM
⊢ First_Order_Filter_Ideal_Spec a x w ∧
  First_Order_Filter_Float_Imp a′ x′ y ⟹
  ∃ e1 e2. abs e1 ≤ (1 / 2 pow 24) ∧ abs e2 ≤ (1 / 2 pow 24) ∧
  (Float_Error n − a ∗ Float_Error (n − 1) =
  a ∗ Val (y (n − 1)) ∗ (e1 + e2 + e1 ∗ e2) + (x n) ∗ e2)
```

2) Fixed-Point Error Analysis:

$$e'_n - a\ e'_{n-1} = \epsilon'_n + \xi'_n \tag{24}$$

To prove this theorem in HOL, we first defined the error as the difference between the output of the real filter specification, and the corresponding real value of the fixed-point filter implementation (*Fxp_Error*). Then, we established the following lemma for the accumulation of round-off error in fixed-point realization of the first-order filter, according to the Equation (24).

```
Lemma 16: FIRST_ORDER_FILTER_FXP_TO_REAL_THM
⊢ First_Order_Filter_Ideal_Spec a x w ∧
  First_Order_Filter_Fxp_Imp X a″ x″ v ⟹
  ∃ e1 e2. abs e1 ≤ inv (2 pow (fracbits X)) ∧
  abs e2 ≤ inv (2 pow (fracbits X)) ∧
  (Fxp_Error n − a ∗ Fxp_Error (n − 1) = e1 + e2)
```

3) Floating- to Fixed-Point Error Analysis:

$$e''_n - a\ e''_{n-1} = \epsilon'_n + \xi'_n - a\ y_{n-1}(\epsilon_n + \xi_n + \epsilon_n \xi_n) - x_n \xi_n \tag{25}$$

To prove this theorem in HOL, we first defined the error as the difference between the real value of the output of the fixed-point filter implementation, and the corresponding real value of the floating-point filter implementation (*Float_Fxp_Error*). Then, we established the following lemma for the accumulation of round-off error in transition from floating-point to fixed-point levels of the first-order filter, according to the Equation (25).

```
Lemma 17: FIRST_ORDER_FILTER_FXP_TO_FLOAT_THM
⊢ First_Order_Filter_Ideal_Spec a x w ∧
  First_Order_Filter_Float_Imp a′ x′ y ∧
  First_Order_Filter_Fxp_Imp X a″ x″ v ⟹
  ∃ e1 e2 e3 e4. abs e1 ≤ inv (2 pow (fracbits X)) ∧
  abs e2 ≤ inv (2 pow (fracbits X)) ∧
  abs e3 ≤ (1 / 2 pow 24) ∧ abs e4 ≤ (1 / 2 pow 24) ∧
  (Float_Fxp_Error n − a ∗ Float_Fxp_Error (n − 1) =
  e1 + e2 − a ∗ Val (y (n − 1)) ∗ (e3 + e4 + e3 ∗ e4) − (x n) ∗ e4)
```

We proved these lemmas using the fundamental error analysis lemmas (Lemmas 4,5, and 6 for floating-point, and Lemmas 9,10, and 11 for fixed-point), based on the error models presented in Section 3.

4.2 Second-Order Filter

A second-order filter is specified by

$$w_n = b_0 x_n - (a_1 w_{n-1} + a_2 w_{n-2}) \tag{26}$$

The corresponding computed floating- and fixed-point outputs are

$$y_n = fl\ [b_0 x_n - (a_1 y_{n-1} + a_2 y_{n-2})] \tag{27}$$

$$v_n = fxp\ [b_0 x_n - (a_1 v_{n-1} + a_2 v_{n-2})] \tag{28}$$

In HOL, we specified the second-order digital filter in real, floating-, and fixed-point abstraction levels, as predicates in higher-order logic. The corresponding codes are as follows.

```
⊢def  Second_Order_Filter_Ideal_Spec a b x w =
      ∀n. w n = (b 0 * x n - (a 1 * w (n - 1) + a 2 * w (n - 2)))

⊢def  Second_Order_Filter_Float_Imp a′ b′ x′ y =
      ∀n. y n = (b′ 0 * x′ n - (a′ 1 * y (n - 1) + a′ 2 * y (n - 2)))

⊢def  Second_Order_Filter_Fxp_Imp a″ b″ x″ v =
      ∀n. v n = (FxpSub X (FxpMul X (b″ 0) (x″ n))
                (FxpAdd X (FxpMul X (a″ 1) (v (n - 1)))
                (FxpMul X (a″ 2) (v (n - 2)))))
```

The calculation of Equation (27) is performed in the following manner. First, the products $a_1 y_{n-1}$, $a_2 y_{n-2}$, and $b_0 x_n$ are calculated separately. Then $a_1 y_{n-1}$ and $a_2 y_{n-2}$ are added. Finally, this sum is subtracted from $b_0 x_n$ to obtain y_n. Similar discussion can be applied for the calculation of the fixed-point output v_n according to the Equation (28). A flowgraph for the error of this case is drawn in Figure 4. The quantities $\delta_{n,0}$, $\epsilon_{n,1}$, $\epsilon_{n,2}$, η_n, ξ_n are errors caused by floating-point roundoff at each arithmetic step. The corresponding error quantities for fixed-point roundoff are $\delta'_{n,0}$, $\epsilon'_{n,1}$, $\epsilon'_{n,2}$, η'_n, ξ'_n.

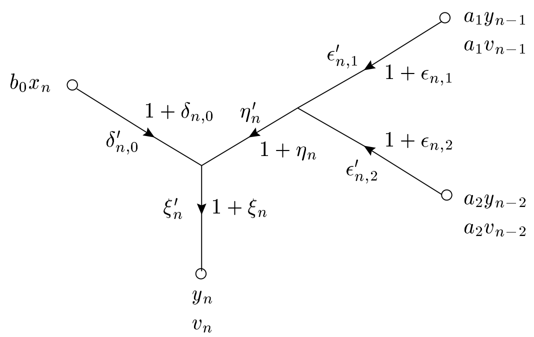


Figure 4: Error flowgraph for the second-order filter

Therefore the actual y_n is seen to be given explicitly by

$$y_n = b_0 \ \theta_{n,0} \ x_n \ - \ \sum_{k=1}^{2} a_k \ \phi_{n,k} \ y_{n-k} \tag{29}$$

where

$$\begin{aligned} \theta_{n,0} &= (1+\delta_{n,0})(1+\xi_n) \\ \phi_{n,1} &= (1+\epsilon_{n,1})(1+\eta_n)(1+\xi_n) \\ \phi_{n,2} &= (1+\epsilon_{n,2})(1+\eta_n)(1+\xi_n) \end{aligned}$$

In HOL, we established the following lemma to compute the real value of the floating-point output for the second-order filter according to the Equation (29).

```
Lemma 18: SECOND_ORDER_FILTER_FLOAT_OUTPUT_VALUE
⊢  Second_Order_Filter_Float_Imp a′ b′ x′ y ⟹
   ∃ t f. Val (y n) = Val (b′ 0) * (t 0) * Val (x′ n) −
   sum (1,2) (λ i. Val (a′ i) * (f i) * Val (y (n − 1))) ∧
   ∃ e1 e2 e3 e4 e5. abs e1 ≤ (1 / 2 pow 24) ∧
   abs e2 ≤ (1 / 2 pow 24) ∧ abs e3 ≤ (1 / 2 pow 24) ∧
   abs e4 ≤ (1 / 2 pow 24) ∧ abs e5 ≤ (1 / 2 pow 24) ∧
   t 0 = (1 + e1) * (1 + e5) ∧
   f 1 = (1 + e2) * (1 + e4) * (1 + e5) ∧
   f 2 = (1 + e3) * (1 + e4) * (1 + e5)
```

Similarly, the actual fixed-point output v_n is given explicitly by

$$v_n = [b_0 x_n - (a_1 v_{n-1} + a_2 v_{n-2})] + \delta'_{n,0} + \epsilon'_{n,1} + \epsilon'_{n,2} + \eta'_n + \xi'_n \tag{30}$$

and the corresponding lemma is established in HOL as follows:

```
Lemma 19: SECOND_ORDER_FILTER_FXP_OUTPUT_VALUE
⊢  Second_Order_Filter_Fxp_Imp X a″ b″ x″ v ⟹
   ∃ e1 e2 e3 e4 e5. abs e1 ≤ inv (2 pow (fracbits X)) ∧
   abs e2 ≤ inv (2 pow (fracbits X)) ∧
   abs e3 ≤ inv (2 pow (fracbits X)) ∧
   abs e4 ≤ inv (2 pow (fracbits X)) ∧
   abs e5 ≤ inv (2 pow (fracbits X)) ∧
   value (v n) = value (b″ 0) * value (x″ n) −
   (value (a″ 1) * value (v (n − 1)) +
   value (a″ 2) * value (v (n − 2))) +
   e1 + e2 + e3 + e4 + e5
```

For error analysis, we need to calculate the y_n and v_n sequences from Equations (29) and (30), and compare them with the ideal output sequence w_n specified by the Equation (26) to obtain the corresponding errors e_n, e'_n, and e''_n, according to the Equations (15), (16), and (17). Therefore, the difference equations for the errors between different levels showing the accumulation of roundoff error are derived as follows:

1) Floating-Point Error Analysis:

$$e_n \ + \ a_1 \ e_{n-1} \ + \ a_2 \ e_{n-2} \ = b_0 x_n (\theta_{n,0} - 1) \ - [a_1 y_{n-1} \ (\phi_{n,1} - 1) + \tag{31}$$

$a_2 y_{n-2}\ (\phi_{n,2} - 1)]$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in floating-point realization of the second-order filter, according to the Equation (31).

```
Lemma 20: SECOND_ORDER_FILTER_FLOAT_TO_REAL_THM
⊢  Second_Order_Filter_Ideal_Spec a b x w ∧
   Second_Order_Filter_Float_Imp a′ b′ x′ y ⟹
   ∃ t f. Float_Error n + a 1 * Float_Error (n − 1) +
   a 2 * Float_Error (n − 2) = b 0 * x n * (t 0 − 1) −
   (a 1 * Val (y (n − 1)) * (f 1 − 1) +
   a 2 * Val (y (n − 2)) * (f 2 − 1)) ∧
   ∃ e1 e2 e3 e4 e5. abs e1 ≤ (1 / 2 pow 24) ∧
   abs e2 ≤ (1 / 2 pow 24) ∧ abs e3 ≤ (1 / 2 pow 24) ∧
   abs e4 ≤ (1 / 2 pow 24) ∧ abs e5 ≤ (1 / 2 pow 24) ∧
   t 0 = (1 + e1) * (1 + e5) ∧
   f 1 = (1 + e2) * (1 + e4) * (1 + e5) ∧
   f 2 = (1 + e3) * (1 + e4) * (1 + e5)
```

2) Fixed-Point Error Analysis:

$$e'_n \ + \ a_1\ e'_{n-1} \ + \ a_2\ e_{n-2} \ = \ \delta'_{n,0} \ + \ \epsilon'_{n,1} \ + \ \epsilon'_{n,2} \ + \ \eta'_n \ + \ \xi'_n \tag{32}$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in fixed-point realization of the second-order filter, according to the Equation (32).

```
Lemma 21: SECOND_ORDER_FILTER_FXP_TO_REAL_THM
⊢  Second_Order_Filter_Ideal_Spec a b x w ∧
   Second_Order_Filter_Fxp_Imp X a″ b″ x″ v ⟹
   ∃ e1 e2 e3 e4 e5. abs e1 ≤ (inv (&2 pow (fracbits X))) ∧
   abs e2 ≤ inv (2 pow (fracbits X)) ∧
   abs e3 ≤ inv (2 pow (fracbits X)) ∧
   abs e4 ≤ inv (2 pow (fracbits X)) ∧
   abs e5 ≤ inv (2 pow (fracbits X)) ∧
   Fxp_Error n + a 1 * Fxp_Error (n − 1) + a 2 * Fxp_Error (n − 2) =
   e1 + e2 + e3 + e4 + e5
```

3) Floating- to Fixed-Point Error Analysis:

$$\begin{aligned} e''_n \ + \ a_1\ e''_{n-1} \ + \ a_2\ e''_{n-2} \ &= \ \delta'_{n,0} \ + \ \epsilon'_{n,1} \ + \ \epsilon'_{n,2} \ + \ \eta'_n \ + \ \xi'_n \ - \\ & b_0 x_n(\theta_{n,0} - 1) \ + \ a_1 y_{n-1}\ (\phi_{n,1} - 1) \ + \ a_2 y_{n-2}\ (\phi_{n,2} - 1) \end{aligned} \tag{33}$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in transition from floating-point to fixed-point levels of the second-order filter, according to the Equation (33).

```
Lemma 22: SECOND_ORDER_FILTER_FXP_TO_FLOAT_THM
⊢  Second_Order_Filter_Ideal_Spec a b x w ∧
   Second_Order_Filter_Float_Imp a′ b′ x′ y ∧
   Second_Order_Filter_Fxp_Imp X a″ b″ x″ v ⟹
   ∃ t f e1′ e2′ e3′ e4′ e5′.
   abs e1 ≤ inv (2 pow (fracbits X)) ∧
   abs e2 ≤ inv (2 pow (fracbits X)) ∧
   abs e3 ≤ inv (2 pow (fracbits X)) ∧
   abs e4 ≤ inv (2 pow (fracbits X)) ∧
   abs e5 ≤ inv (2 pow (fracbits X)) ∧
   (Float_Fxp_Error n + a 1 * Float_Fxp_Error (n − 1) +
   a 2 * Float_Fxp_Error (n − 2) =
   e1′ + e2′ + e3′ + e4′ + e5′ − b 0 * x n * (t 0 − 1) +
   (a 1 * Val (y (n − 1)) * (f 1 − 1) +
   a 2 * Val (y (n − 2)) * (f 2 − 1))) ∧
   ∃ e1 e2 e3 e4 e5. abs e1 ≤ (1 / 2 pow 24) ∧
   abs e2 ≤ (1 / 2 pow 24) ∧ abs e3 ≤ (1 / 2 pow 24) ∧
   abs e4 ≤ (1 / 2 pow 24) ∧ abs e5 ≤ (1 / 2 pow 24) ∧
   t 0 = (1 + e1) * (1 + e5) ∧
   f 1 = (1 + e2) * (1 + e4) * (1 + e5) ∧
   f 2 = (1 + e3) * (1 + e4) * (1 + e5)
```

We proved these lemmas using the fundamental error analysis lemmas (Lemmas 4,5, and 6 for floating-point, and Lemmas 9,10, and 11 for fixed-point), based on the error models presented in Section 3.

4.3 Lth-Order Filter (Direct Form)

The direct form realization of a parametric Lth-order filter is specified by Equation (5). The corresponding computed floating- and fixed-point outputs are

$$y_n \;=\; fl\;(\sum_{k=0}^{M} b_k\; x_{n-k} \;-\; \sum_{k=1}^{L} a_k\; y_{n-k}) \tag{34}$$

and

$$v_n \;=\; fxp\;(\sum_{k=0}^{M} b_k\; x_{n-k} \;-\; \sum_{k=1}^{L} a_k\; v_{n-k}) \tag{35}$$

In HOL, we specified the direct form realization of a parametric Lth-order digital filter in real, floating-, and fixed-point abstraction levels, as predicates in higher-order logic. The real specification is defined in HOL using Equation (5). For this 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. The floating- and fixed-point specifications are defined in HOL according to the Equations (34) and (35). For these cases, we defined similar functions for finite summation on the floating-point (*float_sum*) and fixed-point (*fxp_sum*) numbers, using recursive definition in HOL. The corresponding codes in HOL are as follows.

```
⊢def  L_Order_Filter_Direct_Form_Ideal_Spec a b x w M L =
      ∀n. w n = sum (0,SUC M) (λ i. b i * x (n − i)) −
                sum (1,L) (λ i. a i * w (n − i))

⊢def  ∀ f n m. (float_sum (n,0) f = float (0,0,0)) ∧
               (float_sum (n,SUC m) f = float_sum (n,m) f + f (n + m))

⊢def  L_Order_Filter_Direct_Form_Float_Imp a′ b′ x′ y M L =
      ∀ n. y n = float_sum (0,SUC M) (λ i. b′ i * x′ (n − i)) −
                 float_sum (1,L) (λ i. a′ i * y (n − i))

⊢def  ∀ X f n m. (fxp_sum (n,0) X f =
                 (fxp (WORD (REPLICATE (streamlength X) F),X))) ∧
                 (fxp_sum (n,SUC m) X f =
                 FxpAdd X (fxp_sum (n,m) X f) f (n+m))

⊢def  L_Order_Filter_Direct_Form_Fxp_Imp X a′′ b′′ v M L =
      ∀ n. v n = FxpSub X
                 (fxp_sum (0,SUC M) X (λ i. FxpMul X b′′ i x′′ (n − i)))
                 (fxp_sum (1,L) X (λ i. FxpMul X a i y (n − i)))
```

The calculation of Equation (34) is to be performed in the following manner. First, the output products $a_k\ y_{n-k}$, $k = 1, 2, ..., L$ are calculated separately and then summed. Next, the same is done for the input products $b_k\ x_{n-k}$, $k = 0, 1, ..., M$. 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 according to the Equation (35). 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 5 which also indicates the order of the calculation. The quantities $\delta_{n,k}$, $k = 0, 1, ..., M$, $\epsilon_{n,k}$, $k = 1, 2, ..., L$, $\zeta_{n,k}$, $k = 1, 2, ..., M$, $\eta_{n,k}$, $k = 2, 3, ..., L$, and ξ_n are errors caused by floating-point roundoff at each arithmetic step. The corresponding error quantities for fixed-point roundoff are $\delta'_{n,k}$, $k = 0, 1, ..., M$, $\epsilon'_{n,k}$, $k = 1, 2, ..., L$, $\zeta'_{n,k}$, $k = 1, 2, ..., M$, $\eta'_{n,k}$, $k = 2, 3, ..., L$, and ξ'_n.

Therefore, the actual floating-point output y_n is seen to be given explicitly by:

$$y_n = \sum_{k=0}^{M} b_k\ \theta_{n,k}\ x_{n-k}\ -\ \sum_{k=1}^{L} a_k\ \phi_{n,k}\ y_{n-k} \tag{36}$$

where

$$\begin{aligned}
\theta_{n,0} &= (1+\xi_n)(1+\delta_{n,0}) \prod_{i=1}^{M}(1+\zeta_{n,i}) \\
\theta_{n,j} &= (1+\xi_n)(1+\delta_{n,j}) \prod_{i=j}^{M}(1+\zeta_{n,i}) \quad j \ = \ 1, 2, ..., M \\
\phi_{n,1} &= (1+\xi_n)(1+\epsilon_{n,1}) \prod_{i=2}^{L}(1+\eta_{n,i}) \\
\phi_{n,j} &= (1+\xi_n)(1+\epsilon_{n,j}) \prod_{i=j}^{L}(1+\eta_{n,i}) \quad j \ = \ 2, 3, ..., L
\end{aligned}$$

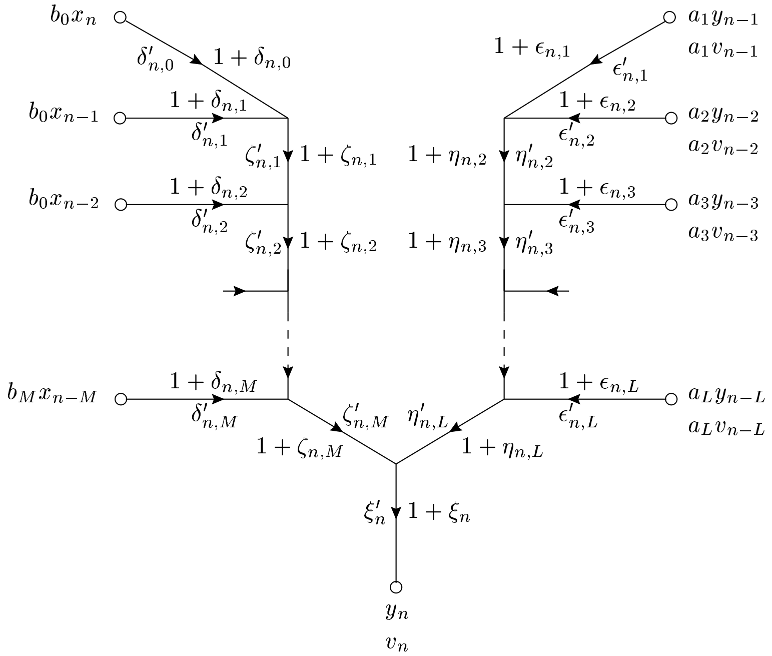


Figure 5: Error flowgraph for Lth-order filter (Direct form)

In HOL, we first defined finite multiplication on the real numbers recursively as the expression $mul\ (m,n)\ f$ denoting $\prod_{i\,=\,m}^{m+n-1} f(i)$ as follows:

```
⊢def  ∀ f n m. (mul (n,0) f = 1) ∧
               (mul (n,SUC m) f = mul (n,m) f * f (n + m))
```

Then, we established the following lemma to compute the real value of the floating-point filter output for the direct form of realization according to the Equations (36).

```
Lemma 23: L_ORDER_FILTER_DIRECT_FORM_FLOAT_OUTPUT_VALUE
⊢  L_Order_Filter_Direct_Form_Float_Imp a′ b′ x′ y M L ⟹
   ∃ t f. (Val (y n) = (if L = 0 then
   sum (0,SUC M) (λ i. (Val (b′ i) * t i * Val (x′ (n − i))))
   else sum (0,SUC M) (λ i. (Val (b′ i) * t i * Val (x′ (n − i)))) −
   sum (1,L) (λ i. (Val (a′ i) * f i * Val (y (n − i)))))) ∧
   ∃ k d p e z. abs k ≤ (1 / 2 pow 24) ∧
   (∀ i. (i ≤ M) ⟹  (abs (d i) ≤ (1 / 2 pow 24))) ∧
   (∀ i. (i ≤ M) ⟹  (abs (p i) ≤ (1 / 2 pow 24))) ∧
   (∀ i. (i ≤ L) ⟹  (abs (e i) ≤ (1 / 2 pow 24))) ∧
   (∀ i. (i ≤ L) ⟹  (abs (z i) ≤ (1 / 2 pow 24))) ∧
   (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)))))
```

Similarly, the actual fixed-point output v_n is given explicitly by

$$v_n = \sum_{k=0}^{M} b_k \ x_{n-k} - \sum_{k=1}^{L} a_k \ v_{n-k} + \sum_{k=0}^{M} \delta'_{n,k} + \sum_{k=1}^{M} \zeta'_{n,k} + \sum_{k=1}^{L} \epsilon'_{n,k} + \sum_{k=2}^{L} \eta'_{n,k} + \xi'_n \tag{37}$$

and the corresponding lemma is established in HOL as follows:

```
Lemma 24: L_ORDER_FILTER_DIRECT_FORM_FXP_OUTPUT_VALUE_EXPAND
⊢  L_Order_Filter_Direct_Form_Fxp_Imp X a'' b'' x'' v M L ⟹
   ∃ k d p e z. abs k ≤ (inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ M) ⟹  abs (d i) ≤ inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ M) ⟹  abs (p i) ≤ inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ L) ⟹  abs (e i) ≤ inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ L) ⟹  abs (z i) ≤ inv (2 pow (fracbits X))) ∧
   (value (v n) = if (L = 0) then
   sum (0,SUC M) (λ i. value (b'' i) * value (x'' (n − i))) +
   sum (0,SUC M) (λ i. d i) + sum (1,M) (λ j. p j) + k else
   sum (0,SUC M) (λ i. value (b'' i) * value (x'' (n − i))) −
   sum (1,L) (λ i. value (a'' i) * value (v (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)
```

For error analysis, we need to calculate the y_n and v_n sequences from Equations (36) and (37), 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 (15), (16), and (17). Therefore, the difference equations for the errors between different levels showing the accumulation of roundoff error are derived as follows:

1) Floating-Point Error Analysis:

$$e_n \ + \ \sum_{k=1}^{L} a_k \ e_{n-k} = \sum_{k=0}^{M} b_k \ (\theta_{n,k} - 1) \ x_{n-k} \ - \ \sum_{k=1}^{L} a_k \ (\phi_{n,k} - 1) \ y_{n-k} \tag{38}$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in floating-point realization of the direct form filter, according to the Equation (38).

```
Lemma 25: L_ORDER_FILTER_DIRECT_FORM_FLOAT_TO_REAL_THM
⊢  L_Order_Filter_Direct_Form_Ideal_Spec a b x w M L ∧
   L_Order_Filter_Direct_Form_Float_Imp a′ b′ x′ y M L ⟹
   ∃ t f. (if L = 0 then (Float_Error n =
   sum (0,SUC M) (λ i. Val (b′ i) * (t i − 1) * Val (x′ (n − i)))) else
   (((Float_Error n) + sum (1,L) (λ i. a i * (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 24)) ∧
   (∀ i. (i ≤ M) ⟹  (abs (d i) ≤ (1 / 2 pow 24))) ∧
   (∀ i. (i ≤ M) ⟹  (abs (p i) ≤ (1 / 2 pow 24))) ∧
   (∀ i. (i ≤ L) ⟹  (abs (e i) ≤ (1 / 2 pow 24))) ∧
   (∀ i. (i ≤ L) ⟹  (abs (z i) ≤ (1 / 2 pow 24))) ∧
   (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)))))
```

2) Fixed-Point Error Analysis:

$$e'_n \ + \ \sum_{k=1}^{L} a_k \ e'_{n-k} = \sum_{k=0}^{M} \delta'_{n,k} + \sum_{k=1}^{M} \zeta'_{n,k} + \sum_{k=1}^{L} \epsilon'_{n,k} + \sum_{k=2}^{L} \eta'_{n,k} + \xi'_n \tag{39}$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in fixed-point realization of the direct form filter, according to the Equation (39).

```
Lemma 26: L_ORDER_FILTER_DIRECT_FORM_FXP_TO_REAL_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 ≤ inv (2 pow (fracbits X)) ∧
   (∀ i. (i ≤ M) ⟹  abs (d i) ≤ inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ M) ⟹  abs (p i) ≤ inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ L) ⟹  abs (e i) ≤ inv (2 pow (fracbits X))) ∧
   (∀ i. (i ≤ L) ⟹  abs (z i) ≤ inv (2 pow (fracbits X))) ∧
   (if (L = 0) then (Fxp_Error n =
   sum (0,SUC M) (λ i. d i) + sum (1,M) (λ j. p j) + k) else
   (Fxp_Error n + sum (1,L) (λ i. a i * 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))
```

3) Floating- to Fixed-Point Error Analysis:

$$e''_n \ + \ \sum_{k=1}^{L} a_k \ e''_{n-k} = \sum_{k=0}^{M} \delta'_{n,k} + \sum_{k=1}^{M} \zeta'_{n,k} + \sum_{k=1}^{L} \epsilon'_{n,k} + \sum_{k=2}^{L} \eta'_{n,k} + \xi'_n \ - \tag{40}$$

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

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in transition from floating-point to fixed-point levels of the direct form filter, according to the Equation (40).

```
Lemma 27: L_ORDER_FILTER_DIRECT_FORM_FXP_TO_FLOAT_THM
⊢ L_Order_Filter_Direct_Form_Ideal_Spec a b x w M L ∧
  L_Order_Filter_Direct_Form_Float_Imp a′ b′ x′ y M L ⟹
  ∃ t f k1 d1 p1 e1 z1.
  (abs k1 ≤ inv (2 pow (fracbits X))) ∧
  (∀ i. (i ≤ M) ⟹  abs (d1 i) ≤ inv (2 pow (fracbits X))) ∧
  (∀ i. (i ≤ M) ⟹  abs (p1 i) ≤ inv (2 pow (fracbits X))) ∧
  (∀ i. (i ≤ L) ⟹  abs (e1 i) ≤ inv (2 pow (fracbits X))) ∧
  (∀ i. (i ≤ L) ⟹  abs (z1 i) ≤ inv (2 pow (fracbits X))) ∧
  (if (L = 0) then
  (Float_Fxp_Error n = sum (0,SUC M) (λ i. d1 i) +
  sum (1,M) (λ j. p1 j) + k1 − (sum (0,SUC M)
  (λ i. Val (b′ i) ∗ (t i − 1) ∗ Val (x′ (n − i))))) else
  (Float_Fxp_Error n + sum (1,L) (λ i. a i ∗ Float_Fxp_Error (n − i)) =
  sum (0,SUC M) (λ i. d1 i) + sum (1,M) (λ j. p1 j) +
  sum (1,L) (λ i. e1 i) + sum (2,(L − 1)) (λ j. z1 j) + k1 −
  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))))) ∧
  ∃ k2 d2 p2 e2 z2. abs k2 ≤ (1 / 2 pow 24) ∧
  (∀ i. (i ≤ M) ⟹  abs (d2 i) ≤ (1 / 2 pow 24)) ∧
  (∀ i. (i ≤ M) ⟹  abs (p2 i) ≤ (1 / 2 pow 24)) ∧
  (∀ i. (i ≤ L) ⟹  abs (e2 i) ≤ (1 / 2 pow 24)) ∧
  (∀ i. (i ≤ L) ⟹  abs (z2 i) ≤ (1 / 2 pow 24)) ∧
  (t 0 = (1 + k2) ∗ (1 + d2 0) ∗ (mul (1,M) (λ i. (1 + p2 i)))) ∧
  (∀ j. (1 ≤ j ∧ j ≤ M) ⟹  (t j = (1 + k2) ∗ (1 + d2 j) ∗
  (mul (j,(M − (j − 1))) (λ j. (1 + p2 j))))) ∧
  (f 1 = (1 + k2) ∗ (1 + e2 1) ∗ (mul (2,(L − 1)) (λ i. (1 + z2 i)))) ∧
  (∀ j. (2 ≤ j ∧ j ≤ L) ⟹  (f j = (1 + k2) ∗ (1 + e2 j) ∗
  (mul (j,(L − j + 1)) (λ j. (1 + z2 j)))))
```

We proved these lemmas using the fundamental error analysis lemmas (Lemmas 4,5, and 6 for floating-point, and Lemmas 9,10, and 11 for fixed-point), based on the error models presented in Section 3. The lemmas are proved by induction on parameters L and M for the direct form of realization.

4.4 Lth-Order Filter (Parallel Form)

For parallel form of realization, the ith parallel path is described by Equation (10). The corresponding computed floating- and fixed-point outputs are

$$y_n^i = fl\ [f_i x_n + g_i x_{n-1} - c_i y_{n-1}^i - d_i y_{n-2}^i] \tag{41}$$

$$v_n^i = fxp\ [f_i x_n + g_i x_{n-1} - c_i v_{n-1}^i - d_i v_{n-2}^i] \tag{42}$$

The output of the entire parallel form filter is described by Equation (11). The corresponding computed floating- and fixed-point outputs are

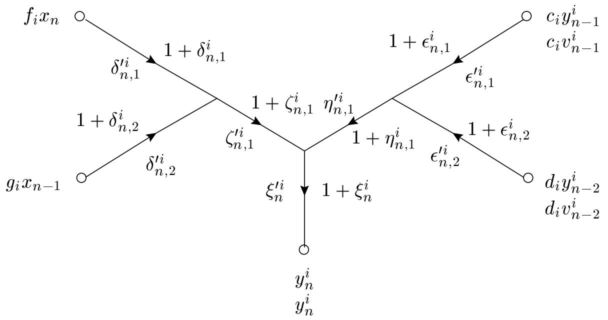
$$y_n = fl\ [y_n^1 + y_n^2 + ... + y_n^K] \tag{43}$$

$$v_n = fxp\ [v_n^1 + v_n^2 + ... + v_n^K] \tag{44}$$

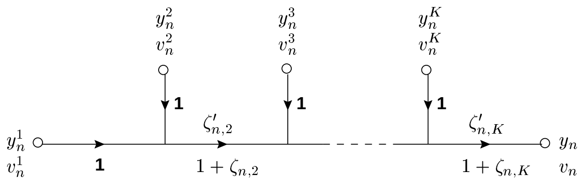
In HOL, we first specified the ith parallel path in real, floating-, and fixed-point abstraction levels, using Equations (10), (41), and (42). Then, we specified the entire output as defined in Equations (11), (43), and (44) using the finite summation functions. The corresponding codes are as follows.

```
⊢def  Parallel_Form_Ideal_Spec c d f g x ww w K =
      ∀ n. w n = sum (1, K) (λ i. ww i n) ∧
      ∀ i. ww i n = f i * x n + g i * x (n − 1) −
                  c i * ww i (n − 1) − d i * ww i (n − 2)

⊢def  Parallel_Form_Float_Imp c′ d′ f′ g′ x′ yy y K =
      ∀ n. y n = float_sum (1,K) (λ i. yy i n) ∧
      ∀ i. (i ≥ 1 ∧ i ≤ K) ⟹ yy i n =
      f′ i * x′ n + g′ i * x′ (n − 1) −
      (c′ i * yy i (n − 1) + d′ i * yy i (n − 2))

⊢def  Parallel_Form_Fxp_Imp X c″ d″ f″ g″ x″ vv v K =
      ∀ n. v n = fxp_sum (1,K) X (λ i. vv i n) ∧
      ∀ i. (i ≥ 1 ∧ i ≤ K) ⟹ vv i n = FxpSub X
      (FxpAdd X (FxpMul X (f″ i) (x″ n))
      (FxpMul X (g″ i) (x″ (n − 1))))
      (FxpAdd X (FxpMul X (c″ i) (vv i (n − 1)))
      (FxpMul X (d″ i) (vv i (n − 2))))
```

Figure 6 shows the error flowgraph for the parallel form realization of a parametric Lth-order filter.



a) i**th parallel path**



b) Final parallel output

Figure 6: Error flowgraph for Lth-order filter (Parallel form)

The corresponding error flowgraph for the ith parallel path is shown in Figure 6 (a). The actual floating-point output sequence for the ith parallel path is therefore given by

$$y_n^i = f_i x_n \theta_{n,1}^i + g_i x_{n-1} \theta_{n,2}^i - c_i y_{n-1}^i \phi_{n,1}^i - d_i y_{n-2}^i \phi_{n,2}^i \tag{45}$$

where

$$\theta^i_{n,1} = (1 + \delta^i_{n,1})(1 + \zeta^i_{n,1})(1 + \xi^i_n)$$

$$\theta^i_{n,2} = (1 + \delta^i_{n,2})(1 + \zeta^i_{n,1})(1 + \xi^i_n)$$

$$\phi^i_{n,1} = (1 + \epsilon^i_{n,1})(1 + \eta^i_{n,1})(1 + \xi^i_n)$$

$$\phi^i_{n,2} = (1 + \epsilon^i_{n,2})(1 + \eta^i_{n,1})(1 + \xi^i_n)$$

If the summation of (43) is carried out from the left to the right, a corresponding flow-graph can be drawn as given in Figure 6 (b). Thus

$$y_n = \sum_{i=1}^{K} \psi_{n,i} y^i_n \tag{46}$$

where

$$\psi_{n,i} = \begin{cases} \prod_{j=2}^{K}(1 + \zeta_{n,j}), & \text{i} = 1 \\ \prod_{j=i}^{K}(1 + \zeta_{n,j}), & i \geq 2 \end{cases}$$

In HOL, we established the following lemma to compute the real value of the floating-point output of the parallel form of realization according to the Equations (45) and (46).

```
Lemma 28: PARALLEL_FORM_FLOAT_OUTPUT_VALUE
⊢  Parallel_Form_Float_Imp c′ d′ f′ g′ x′ yy y K ⟹
   (∃ t f. Val (yy i n) = Val (f′ i) ∗ Val (x′ n) ∗ (t 1) +
   Val (g′ i) ∗ Val (x′ (n − 1)) ∗ (t 2) −
   Val (c′ i) ∗ Val (yy i (n − 1)) ∗ (f 1) −
   Val (d′ i) ∗ Val (yy i (n − 2)) ∗ (f 2)) ∧
   ∃ k d1 d2 p e1 e2 z. abs (k i) ≤ (1 / 2 pow 24) ∧
   abs (d1 i) ≤ (1 / 2 pow 24) ∧
   abs (d2 i) ≤ (1 / 2 pow 24) ∧
   abs (p i) ≤ (1 / 2 pow 24) ∧
   abs (e1 i) ≤ (1 / 2 pow 24) ∧
   abs (e2 i) ≤ (1 / 2 pow 24) ∧
   abs (z i) ≤ (1 / 2 pow 24) ∧
   t 1 = (1 + d1 i) ∗ (1 + p i) ∗ (1 + k) ∧
   t 2 = (1 + d2 i) ∗ (1 + p i) ∗ (1 + k) ∧
   f 1 = (1 + e1 i) ∗ (1 + z i) ∗ (1 + k) ∧
   f 2 = (1 + e2 i) ∗ (1 + z i) ∗ (1 + k) ∧
   ∃ s. (Val (y n) = sum (1,K) (λ i. s i ∗ Val (yy i n)) ∧
   ∃ k. s i = (if (i = 1) then (mul (2,(K − 1)) (λ i. (1 + k i))) else
   (mul (i,(K − i + 1)) (λ i. (1 + k i)))))
```

Similarly, the actual fixed-point outputs of the parallel form of realization v_n^i, and v_n are given explicitly by

$$v_n^i = f_i x_n + g_i x_{n-1} - c_i v_{n-1}^i - d_i v_{n-2}^i + \delta_{n,1}^{\prime i} + \delta_{n,2}^{\prime i} + \zeta_{n,1}^{\prime i} + \epsilon_{n,1}^{\prime i} + \epsilon_{n,2}^{\prime i} + \eta_{n,1}^{\prime i} + \xi_n^{\prime i} \tag{47}$$

and

$$v_n = \sum_{i=1}^{K} v_n^i + \sum_{i=2}^{K} \zeta_{n,i}^{\prime} \tag{48}$$

and the corresponding lemma is established in HOL as follows:

```
Lemma 29: PARALLEL_FORM_FXP_OUTPUT_VALUE
⊢  Parallel_Form_Fxp_Imp X c'' d'' f'' g'' x'' vv v K ⟹
   ∃ k' d1' d2' p' e1' e2' z'.
   abs (k' i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d1' i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d2' i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p' i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e1' i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e2' i) ≤ inv (2 pow (fracbits X)) ∧
   abs (z' i) ≤ inv (2 pow (fracbits X)) ∧
   value (vv i n) = value (f'' i) * value (x'' n) +
   value (g'' i) * value (x'' (n - 1)) -
   value (c'' i) * value (vv i (n - 1)) -
   value (d'' i) * value (vv (n - 2)) +
   d1' i + d2' i + p' i + e1' i + e2' i + z' i + k' i ∧
   ∃ s'. value (v n) = sum (1,K) (λ i. value (vv i n)) +
   sum (2,(K - 1)) (λ j. s' j)
```

For error analysis of the parallel form, we first define the corresponding errors at the ith parallel path output sample as:

$$e_n^i = y_n^i - w_n^i \tag{49}$$

$$e_n^{\prime i} = v_n^i - w_n^i \tag{50}$$

$$e_n^{\prime\prime i} = v_n^i - w_n^i \tag{51}$$

Then, we calculate the y_n^i, y_n, v_n^i, and v_n sequences from Equations (45), (46), (47), and (48), respectively and compare them with the ideal output sequences w_n^i, and w_n specified by the Equations (10), and (11) to obtain the corresponding errors e_n^i, $e_n^{\prime i}$, $e_n^{\prime\prime i}$, e_n, $e_n^{\prime}$, and $e_n^{\prime\prime}$ according to the Equations (49), (50), (51), (15), (16), and (17), respectively. Therefore, the difference equations for the errors between different levels showing the accumulation of roundoff error are derived as follows:

1) Floating-Point Error Analysis:

$$e^i_n + c_i e^i_{n-1} + d_i e^i_{n-2} = f_i x_n(\theta^i_{n,1} - 1) + g_i x_{n-1}(\theta^i_{n,2} - 1) - c_i y^i_{n-1}(\phi^i_{n,1} - 1) - d_i y^i_{n-2}(\phi^i_{n,2} - 1) \tag{52}$$

and

$$e_n - \sum_{i=1}^{K} e^i_n = \sum_{i=1}^{K} (\psi_{n,i} - 1)\ y^i_n \tag{53}$$

To prove these theorems in HOL, we established the following lemma for the accumulation of round-off error in floating-point realization of the parallel form filter, according to the Equations (52), and (53).

```
Lemma 30: PARALLEL_FORM_FLOAT_TO_REAL_THM
⊢  Parallel_Form_Ideal_Spec c d f g x ww w K ∧
   Parallel_Form_Float_Imp c′ d′ f′ g′ x′ yy y K  ⟹
   ∃ t f. Float_Error i n + c i * Float_Error i (n − 1) +
   d i * Float_Error i (n − 2) =
   f i * x n * (t 1 − 1) + g i * x (n − 1) * (t 2 − 1) −
   c i * Val (y (n − 1)) * (f 1 − 1) −
   d i * Val (y (n − 2)) * (f 2 − 1) ∧
   ∃ k d1 d2 p e1 e2 z. abs (k i) ≤ (1 / 2 pow 24) ∧
   abs (d1 i) ≤ (1 / 2 pow 24) ∧
   abs (d2 i) ≤ (1 / 2 pow 24) ∧
   abs (p i) ≤ (1 / 2 pow 24) ∧
   abs (e1 i) ≤ (1 / 2 pow 24) ∧
   abs (e2 i) ≤ (1 / 2 pow 24) ∧
   abs (z i) ≤ (1 / 2 pow 24) ∧
   t 1 = (1 + d1 i) * (1 + p i) * (1 + k) ∧
   t 2 = (1 + d2 i) * (1 + p i) * (1 + k) ∧
   f 1 = (1 + e1 i) * (1 + z i) * (1 + k) ∧
   f 2 = (1 + e2 i) * (1 + z i) * (1 + k) ∧
   ∃ s. (Float_Error n = sum (1,K) (λ i. Float_Fxp_Error i n) +
   sum (1, K) (λ i. (((s i) − 1) * Val (yy i n))) ∧
   ∃ k. s i = if (i = 1) then mul (2, (K − 1)) (λ i. (1 + (k i))) else
   mul (i,(K − i + 1)) (λ i. (1 + (k i))))
```

2) Fixed-Point Error Analysis:

$$e'^i_n + c_i e'^i_{n-1} + d_i e'^i_{n-2} = \delta'^i_{n,1} + \delta'^i_{n,2} + \zeta'^i_{n,1} + \epsilon'^i_{n,1} + \epsilon'^i_{n,2} + \eta'^i_{n,1} + \xi'^i_n \tag{54}$$

and

$$e'_n - \sum_{i=1}^{K} e'^i_n = \sum_{i=1}^{K} \zeta'_{n,i} \tag{55}$$

To prove these theorems in HOL, we established the following lemma for the accumulation of round-off error in fixed-point realization of the parallel form filter, according to the Equations (54), and (55).

```
Lemma 31: PARALLEL_FORM_FXP_TO_REAL_THM
⊢  Parallel_Form_Ideal_Spec c d f g x ww w K ∧
   Parallel_Form_Fxp_Imp X c′′ d′′ f′′ g′′ x′′ vv v K ⟹
   ∃ k′ d1′ d2′ p′ e1′ e2′ z′.
   abs (k′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (z′ i) ≤ inv (2 pow (fracbits X)) ∧
   Fxp_Error i n + c i * Fxp_Error i (n − 1) +
   d i * Fxp_Error i (n − 2) =
   d1′ i + d2′ i + p′ i + e1′ i + e2′ i + z′ i + k′ i ∧
   ∃ s′. Fxp_Error n = sum (1,K) (λ i. Fxp_Error i n) +
   sum (2,(K − 1)) (λ j. s′ j)
```

3) Floating- to Fixed-Point Error Analysis:

$$e''^{i}_{n} + c_i e''^{i}_{n-1} + d_i e''^{i}_{n-2} = \delta'^{i}_{n,1} + \delta'^{i}_{n,2} + \zeta'^{i}_{n,1} + \epsilon'^{i}_{n,1} + \epsilon'^{i}_{n,2} + \eta'^{i}_{n,1} + \xi'^{i}_{n} - \tag{56}$$

$$f_i x_n(\theta^{i}_{n,1} - 1) - g_i x_{n-1}(\theta^{i}_{n,2} - 1) + c_i y^{i}_{n-1}(\phi^{i}_{n,1} - 1) \ + d_i y^{i}_{n-2}(\phi^{i}_{n,2} - 1)$$

and

$$e''_{n} - \sum_{i=1}^{K} e''^{i}_{n} = \sum_{i=1}^{K} \zeta'_{n,i} - \sum_{i=1}^{K} (\psi_{n,i} - 1)\ y^{i}_{n} \tag{57}$$

To prove these theorems in HOL, we established the following lemma for the accumulation of round-off error in transition from floating-point to fixed-point realizations of the parallel form filter, according to the Equations (56), and (57).

```
Lemma 32: PARALLEL_FORM_FXP_TO_FLOAT_THM
⊢ Parallel_Form_Ideal_Spec c d f g x ww w K ∧
  Parallel_Form_Float_Imp c′ d′ f′ g′ x′ yy y K ∧
  Parallel_Form_Fxp_Imp X c′′ d′′ f′′ g′′ x′′ vv v K ⟹
  ∃ t f k′ d1′ d2′ p′ e1′ e2′ z′.
  abs (k′ i) ≤ inv (2 pow (fracbits X)) ∧
  abs (d1′ i) ≤ inv (2 pow (fracbits X)) ∧
  abs (d2′ i) ≤ inv (2 pow (fracbits X)) ∧
  abs (p′ i) ≤ inv (2 pow (fracbits X)) ∧
  abs (e1′ i) ≤ inv (2 pow (fracbits X)) ∧
  abs (e2′ i) ≤ inv (2 pow (fracbits X)) ∧
  abs (z′ i) ≤ inv (2 pow (fracbits X)) ∧
  Float_Fxp_Error i n + c i * Float_Fxp_Error i (n − 1) +
  d i * Float_Fxp_Error i (n − 2) =
  d1′ i + d2′ i + p′ i + e1′ i + e2′ i + z′ i + k′ i −
  (f i * x n * (t 1 − 1) +
  g i * (x (n − 1)) * (t 2 − 1) −
  (c i * Val (y (n − 1)) * (f 1 − 1) +
  d i * Val (y (n − 2)) * (f 2 − 1))) ∧
  ∃ k d1 d2 p e1 e2 z. abs (k i) ≤ (1 / 2 pow 24) ∧
  abs (d1 i) ≤ (1 / 2 pow 24) ∧
  abs (d2 i) ≤ (1 / 2 pow 24) ∧
  abs (p i) ≤ (1 / 2 pow 24) ∧
  abs (e1 i) ≤ (1 / 2 pow 24) ∧
  abs (e2 i) ≤ (1 / 2 pow 24) ∧
  abs (z i) ≤ (1 / 2 pow 24) ∧
  t 1 = (1 + d1 i) * (1 + p i) * (1 + k) ∧
  t 2 = (1 + d2 i) * (1 + p i) * (1 + k) ∧
  f 1 = (1 + e1 i) * (1 + z i) * (1 + k) ∧
  f 2 = (1 + e2 i) * (1 + z i) * (1 + k) ∧
  ∃ s s′. Float_Fxp_Error n = sum (1,K) (λ i. Float_Fxp_Error i n) +
  sum (2,(K − 1)) (λ j. s′ j) −
  sum (1, K) (λ i. ((s i − 1) * Val (yy i n))) ∧
  ∃ k. s i = if (i = 1) then mul (2, (K − 1)) (λ i. (1 + k i)) else
  mul (i,(K − i + 1)) (λ i. (1 + k i))
```

We proved these lemmas using the fundamental error analysis lemmas (Lemmas 4,5, and 6 for floating-point, and Lemmas 9,10, and 11 for fixed-point), based on the error models presented in Section 3. The lemmas are proved by induction on the parameter K which is defined as the number of internal sub-filters connected in parallel form to generate the final output, according to the Equation (8).

4.5 Lth-Order Filter (Cascade Form)

The cascade form realization of a parametric Lth-order filter is specified by Equation (14). The corresponding computed floating- and fixed-point outputs are

$$y_n^{i+1} = fl[y_n^i + k_i y_{n-1}^i + l_i y_{n-2}^i - c_i y_{n-1}^{i+1} - d_i y_{n-2}^{i+1}] \tag{58}$$

$$v_n^{i+1} = fxp[v_n^i + k_i v_{n-1}^i + l_i v_{n-2}^i - c_i v_{n-1}^{i+1} - d_i v_{n-2}^{i+1}] \tag{59}$$

In HOL, we specified the second-order digital filter in real, floating-, and fixed-point abstraction levels, using recursive definitions as predicates in higher-order logic. The corresponding codes are as follows.

```
⊢def  Cascade_Form_Ideal_Spec c d k l x w =
      ∀ n. (w 0 n = x n ∧ ∀ i.
      w (SUC i) n = ((((w i n + k i * w i (n − 1)) +
      l i * w i (n − 2)) − c i * w (SUC i) (n − 1)) −
      d i * w (SUC i) (n − 2)))

⊢def  Cascade_Form_Float_Imp c′ d′ k′ l′ x′ y =
      ∀ n. (y 0 n = x′ n ∧ ∀ i.
      y (SUC i) n = ((((y i n + (k′ i * y i (n − 1))) +
      (l′ i * y i (n − 2))) − (c′ i * y (SUC i) (n − 1))) −
      (d′ i * y (SUC i) (n − 2))))

⊢def  Cascade_Form_Fxp_Imp X c′′ d′′ k′′ l′′ x′′ v =
      ∀ n. (v 0 n = x′′ n ∧ ∀ i. v (SUC i) n =
      FxpSub X (FxpSub X (FxpAdd X (FxpAdd X v i n
      (FxpMul X (k′′ i) v i (n − 1)))
      (FxpMul X (l′′ i) v i (n − 2)))
      (FxpMul X (c′′ i) v (i + 1) (n − 1)))
      (FxpMul X (d′′ i) v (i + 1) (n − 2)))
```

Figure 6 shows the error flowgraph for the cascade form realization of a parametric Lth-order filter.

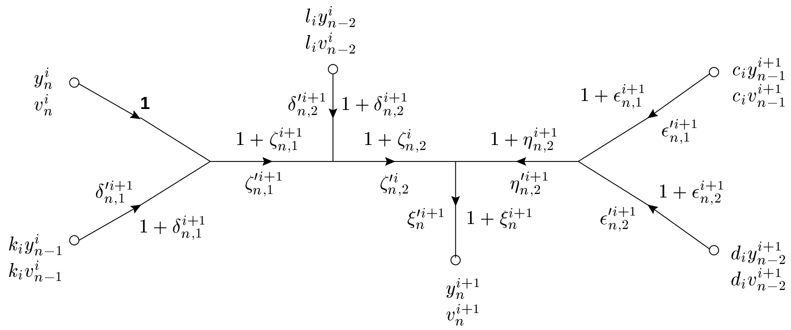


Figure 7: Error flowgraph for Lth-order filter (Cascade form)

The actual floating-point output sequence for the cascade form is therefore given by

$$y_n^{i+1} = y_n^i \theta_{n,0}^{i+1} + f_i y_{n-1}^i \theta_{n,1}^{i+1} + g_i y_{n-2}^i \theta_{n,2}^{i+1} - c_i y_{n-1}^i \phi_{n,1}^{i+1} - d_i y_{n-2}^i \phi_{n,2}^{i+1} \tag{60}$$

where

$$\theta_{n,0}^{i+1} = (1 + \zeta_{n,1}^{i+1})(1 + \zeta_{n,2}^{i+1})(1 + \xi_n^{i+1})$$

$$\theta_{n,1}^{i+1} = (1 + \delta_{n,1}^{i+1})(1 + \zeta_{n,1}^{i+1})(1 + \zeta_{n,2}^{i+1})(1 + \xi_n^{i+1})$$

$$\theta_{n,2}^{i+1} = (1 + \delta_{n,2}^{i+1})(1 + \eta_{n,1}^{i+1})(1 + \xi_n^{i+1})$$

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

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

In HOL, we established the following lemma to compute the real value of the floating-point output of the parallel form of realization according to the Equation (60).

```
Lemma 33: CASCADE_FORM_FLOAT_OUTPUT_VALUE
⊢  Cascade_Form_Float_Imp c′ d′ k′ l′ x′ y ⟹
   ∃ t f. Val (y (SUC i) n) = Val (y i n) * (t 0) +
   Val (k′ i) * Val (y i (n − 1)) * (t 1) +
   Val (l′ i) * Val (y i (n − 2)) * (t 2) −
   Val (c′ i) * Val (y i (n − 1)) * (f 1) −
   Val (d′ i) * Val (y i (n − 2)) * (f 2) ∧
   ∃ k d1 d2 p1 p2 e1 e2 z. abs (k i) ≤ (1 / 2 pow 24) ∧
   abs (d1 i) ≤ (1 / 2 pow 24) ∧
   abs (d2 i) ≤ (1 / 2 pow 24) ∧
   abs (p1 i) ≤ (1 / 2 pow 24) ∧
   abs (p2 i) ≤ (1 / 2 pow 24) ∧
   abs (e1 i) ≤ (1 / 2 pow 24) ∧
   abs (e2 i) ≤ (1 / 2 pow 24) ∧
   abs (z i) ≤ (1 / 2 pow 24) ∧
   t 0 = (1 + p1 i) * (1 + p2 i) * (1 + k) ∧
   t 1 = (1 + d1 i) * (1 + p1 i) * (1 + p2 i) * (1 + k) ∧
   t 2 = (1 + d2 i) * (1 + p2 i) * (1 + k) ∧
   f 1 = (1 + e1 i) * (1 + z i) * (1 + k) ∧
   f 2 = (1 + e2 i) * (1 + z i) * (1 + k)
```

Similarly, the actual fixed-point outputs of the cascade form of realization is given explicitly by

$$v_n^{i+1} = v_n^i + f_i v_{n-1}^i + g_i v_{n-2}^i - c_i v_{n-1}^{i+1} - d_i v_{n-2}^{i+1} + \delta_{n,1}'^{i+1} + \zeta_{n,1}'^{i+1} + \delta_{n,2}'^{i+1} + \zeta_{n,2}'^{i} + \epsilon_{n,1}'^{i+1} + \epsilon_{n,2}'^{i+1} + \eta_{n,2}'^{i+1} + \xi_n'^{i+1} \tag{61}$$

and the corresponding lemma is established in HOL as follows:

```
Lemma 34: CASCADE_FORM_FXP_OUTPUT_VALUE
⊢  Cascade_Form_Fxp_Imp X c′′ d′′ k′′ l′′ x′′ v ⟹
   ∃ k′ d1′ d2′ p1′ p2′ e1′ e2′ z′.
   abs (k′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (z′ i) ≤ inv (2 pow (fracbits X)) ∧
   value (v (SUC i) n) = value (v i n) +
   value (k′′ i) * value (v i (n - 1)) +
   value (l′′ i) * value (v i (n - 2)) +
   value (l′′ i) * value (v i (n - 2)) +
   value (c′′ i) * value (v (SUC i) (n - 1)) +
   value (d′′ i) * value (v (SUC i) (n - 2)) +
   d1′ i + p1′ i + d2′ i + p2′ i + e1′ i + e2′ i + z′ i + k′ i
```

For error analysis of the cascade form, we first define the corresponding errors as:

$$e_n^{i+1} = y_n^{i+1} - w_n^{i+1} \tag{62}$$

$$e_n'^{i+1} = v_n^{i+1} - w_n^{i+1} \tag{63}$$

$$e_n''^{i+1} = v_n^{i+1} - w_n^{i+1} \tag{64}$$

Then, we calculate the y_n^{i+1}, and v_n^{i+1} sequences from Equations (60), (61), respectively and compare them with the ideal output sequences w_n^{i+1} specified by the Equations (14) to obtain the corresponding errors e_n^{i+1}, $e_n'^{i+1}$, and $e_n''^{i+1}$ according to the Equations (62), (63), (64), respectively. Therefore, the difference equations for the errors between different levels showing the accumulation of roundoff error are derived as follows:

1) Floating-Point Error Analysis:

$$e_n^{i+1} = e_n^i + f_i e_{n-1}^i + g_i e_{n-2}^i - c_i e_{n-1}^{i+1} - d_i e_{n-2}^{i+1} + y_n^i(\theta_{n,0}^{i+1} - 1) + f_i y_{n-1}^i(\theta_{n,1}^{i+1} - 1) + \tag{65}$$

$$g_i y_{n-2}^i(\theta_{n,2}^{i+1} - 1) - c_i y_{n-1}^i(\phi_{n,1}^{i+1} - 1) - d_i y_{n-2}^i(\phi_{n,2}^{i+1} - 1)$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in floating-point realization of the cascade form filter, according to the Equation (65).

```
Lemma 35: CASCADE_FORM_FLOAT_TO_REAL_THM
⊢  Cascade_Form_Ideal_Spec c d k l x w ∧
   Cascade_Form_Float_Imp c′ d′ k′ l′ x′ y ⟹
   ∃ t f. Float_Error (SUC i) n = Float_Error i n +
   k i * Float_Error i (n − 1) + l i * Float_Error i (n − 2) −
   c i * Float_Error (SUC i) (n − 1) −
   d i * Float_Error (SUC i) (n − 2) +
   Val (y i n) * (t 0 − 1) +
   (k i) * Val (y i (n − 1) (t 1 − 1) +
   (l i) * Val (y i (n − 2)) (t 2 − 1) −
   (c i) * Val (y i (n − 1)) (f 1 − 1) −
   (d i) * Val (y i (n − 2)) (f 2 − 1) ∧
   ∃ k d1 d2 p1 p2 e1 e2 z. abs (k i) ≤ (1 / 2 pow 24) ∧
   abs (d1 i) ≤ (1 / 2 pow 24) ∧
   abs (d2 i) ≤ (1 / 2 pow 24) ∧
   abs (p1 i) ≤ (1 / 2 pow 24) ∧
   abs (p2 i) ≤ (1 / 2 pow 24) ∧
   abs (e1 i) ≤ (1 / 2 pow 24) ∧
   abs (e2 i) ≤ (1 / 2 pow 24) ∧
   abs (z i) ≤ (1 / 2 pow 24) ∧
   t 0 = (1 + p1 i) * (1 + p2 i) * (1 + k) ∧
   t 1 = (1 + d1 i) * (1 + p1 i) * (1 + p2 i) * (1 + k) ∧
   t 2 = (1 + d2 i) * (1 + p2 i) * (1 + k) ∧
   f 1 = (1 + e1 i) * (1 + z i) * (1 + k) ∧
   f 2 = (1 + e2 i) * (1 + z i) * (1 + k)
```

2) Fixed-Point Error Analysis:

$$e_n'^{i+1} = e_n'^{i} + f_i e_{n-1}'^{i} + g_i e_{n-2}'^{i} - c_i e_{n-1}'^{i+1} - d_i e_{n-2}'^{i+1} + \\ \delta_{n,1}'^{i+1} + \zeta_{n,1}'^{i+1} + \delta_{n,2}'^{i+1} + \zeta_{n,2}'^{i+1} + \epsilon_{n,1}'^{i+1} + \epsilon_{n,2}'^{i+1} + \eta_{n,2}'^{i+1} + \xi_n'^{i+1} \tag{66}$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in floating-point realization of the cascade form filter, according to the Equation (66).

```
Lemma 36: CASCADE_FORM_FXP_TO_REAL_THM
⊢  Cascade_Form_Ideal_Spec c d k l x w ∧
   Cascade_Form_Fxp_Imp X c″ d″ k″ l″ x″ v ⟹
   ∃ k′ d1′ d2′ p1′ p2′ e1′ e2′ z′.
   abs (k′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (z′ i) ≤ inv (2 pow (fracbits X)) ∧
   Fxp_Error (SUC i) n = Fxp_Error i n + k i * Fxp_Error i (n − 1) +
   l i * Fxp_Error i (n − 2) − c i * Fxp_Error (SUC i) (n − 1) −
   d i * Fxp_Error (SUC i) (n − 2) +
   d1′ i + p1′ i + d2′ i + p2′ i + e1′ i + e2′ i + z′ i + k′ i
```

3) Floating- to Fixed-Point Error Analysis:

$$e''^{i+1}_n = e''^i_n + f_i e''^i_{n-1} + g_i e''^i_{n-2} - c_i e''^{i+1}_{n-1} - d_i e''^{i+1}_{n-2} + \delta'^{i+1}_{n,1} + \zeta'^{i+1}_{n,1} + \delta'^{i+1}_{n,2} + \zeta'^{i+1}_{n,2} + \tag{67}$$

$$\epsilon'^{i+1}_{n,1} + \epsilon'^{i+1}_{n,2} + \eta'^{i+1}_{n,2} + \xi'^{i+1}_n - y^i_n(\theta^{i+1}_{n,0} - 1) - f_i y^i_{n-1}(\theta^{i+1}_{n,1} - 1) - g_i y^i_{n-2}(\theta^{i+1}_{n,2} - 1) +$$
$$c_i y^i_{n-1}(\phi^{i+1}_{n,1} - 1) + d_i y^i_{n-2}(\phi^{i+1}_{n,2} - 1)$$

To prove this theorem in HOL, we established the following lemma for the accumulation of round-off error in floating-point realization of the cascade form filter, according to the Equation (67).

```
Lemma 37: CASCADE_FORM_FXP_TO_FLOAT_THM
⊢  Cascade_Form_Ideal_Spec c d k l x w ∧
   Cascade_Form_Float_Imp c′ d′ k′ l′ x′ y ∧
   Cascade_Form_Fxp_Imp X c′′ d′′ k′′ l′′ x′′ v ⟹
   ∃ t f k′ d1′ d2′ p1′ p2′ e1′ e2′ z′.
   abs (k′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (d2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (p2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e1′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (e2′ i) ≤ inv (2 pow (fracbits X)) ∧
   abs (z′ i) ≤ inv (2 pow (fracbits X)) ∧
   Float_Fxp_Error (SUC i) n = Float_Fxp_Error i n +
   k i * Float_Fxp_Error i (n − 1) +
   l i * Float_Fxp_Error i (n − 2) −
   c i * Float_Fxp_Error (SUC i) (n − 1) −
   d i * Fxp_Error (SUC i) (n − 2) +
   d1′ i + p1′ i + d2′ i + p2′ i + e1′ i + e2′ i + z′ i + k′ i −
   Val (y i n) * (t 0 − 1) −
   (k i) * Val (y i (n − 1) (t 1 − 1) −
   (l i) * Val (y i (n − 2)) (t 2 − 1) +
   (c i) * Val (y i (n − 1)) (f 1 − 1) +
   (d i) * Val (y i (n − 2)) (f 2 − 1) ∧
   ∃ k d1 d2 p1 p2 e1 e2 z. abs (k i) ≤ (1 / 2 pow 24) ∧
   abs (d1 i) ≤ (1 / 2 pow 24) ∧
   abs (d2 i) ≤ (1 / 2 pow 24) ∧
   abs (p1 i) ≤ (1 / 2 pow 24) ∧
   abs (p2 i) ≤ (1 / 2 pow 24) ∧
   abs (e1 i) ≤ (1 / 2 pow 24) ∧
   abs (e2 i) ≤ (1 / 2 pow 24) ∧
   abs (z i) ≤ (1 / 2 pow 24) ∧
   t 0 = (1 + p1 i) * (1 + p2 i) * (1 + k) ∧
   t 1 = (1 + d1 i) * (1 + p1 i) * (1 + p2 i) * (1 + k) ∧
   t 2 = (1 + d2 i) * (1 + p2 i) * (1 + k) ∧
   f 1 = (1 + e1 i) * (1 + z i) * (1 + k) ∧
   f 2 = (1 + e2 i) * (1 + z i) * (1 + k)
```

We proved these lemmas using the fundamental error analysis lemmas (Lemmas 4,5, and 6 for floating-point, and Lemmas 9,10, and 11 for fixed-point), based on the error models presented in Section 3.

5 Conclusions

In this report, we described 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 of parametric digital filters at the real, floating- and fixed-point levels of abstraction, entirely specified in HOL. We proved lemmas to derive expressions for the accumulation of roundoff error in floating- and fixed-point implementations both compared to the ideal real specification. The main contribution of this work consists in the upgrade and application of established floating- and fixed-point theories in HOL on the analysis of error due to finite precision effects in the realization of 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 mechanized proofs in HOL for the error analysis of digital filters. This opens new avenues in using formal methods for the verification of DSP systems as complement to traditional theoretical (analytical) and simulation techniques.

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, S. Tahar, “Modeling SystemC Fixed-Point Arithmetic in HOL,” In Formal Methods and Software Engineering, LNCS 2885, Springer-Verlag, pp. 206-225, 2003.

[3] R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, and J. Van-Tassel, “Experience with Embedding Hardware Description Languages in HOL,” In Theorem Provers in Circuit Design, pp. 129-156. North-Holland, 1992.

[4] G. Forsythe, C. B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall, Englewood Cliffs, N.J., 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, Washington, D. C.: Spartan, 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] IEEE, Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, The Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA, 1985.

[11] L. B. Jackson, "Roundoff-Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," IEEE Transactions on Audio and Electroacoustics, vol. AU-18, pp. 107-122, June 1970.

[12] E. P. F. Kan, and J. K. Aggarwal, "Error Analysis of Digital Filter Employing Floating-Point Arithmetic," IEEE Transactions on Circuit Theory, vol. CT-18, pp. 678-686, November 1971.

[13] J.F. Kaiser, "Digital Filters," In system Analysis by Digital Computer, F. F. Kuo and J. F. Kaiser, Eds. New York: Wiley, pp. 218-285, 1966.

[14] J. B. Knowles and R. Edwards, "Effects of a Finite-Word-Length Computer in a Sampled-Data Feedback System," IEE Proceedings (London), vol. 112, pp. 1197-1207, June 1965.

[15] B. Liu, and T. Kaneko, "Error Analysis of Digital Filters Realized with Floating-Point Arithmetic," Proceedings of the IEEE, vol. 57, pp. 1735-1747, October 1969.

[16] A. V. Oppenheim, C. J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proceedings of the IEEE, vol. 60, No. 8, August 1972.

[17] A. V. Oppenheim, R. W. Schafer, Discrete-Time Signal Processing, Prentice-Hall, Englewood Cliffs, 1989.

[18] I. W. Sandberg, "Floating-Point-Roundoff Accumulation in Digital Filter Realization," The Bell System Technical Journal, vol. 46, pp. 1775-1791, October 1967.

[19] C. Weinstein, A. V. Oppenheim, "A Comparison of Roundoff Noise in Floating Point and Fixed Point Digital Filter Realizations," Proceedings of the IEEE (Proceedings Letters), vol. 57, pp. 1181-1183, June 1969.

[20] J. H. Wilkinson, Rounding Errors in Algebraic Processes, Prentice-Hall, Englewood Cliffs, New Jersey 1963.