Incorporating Formal Methods in the Design Flow of DSP Systems

Behzad Akbarpour and Sofiène Tahar

Dept. of Electrical and Computer Engineering, Concordia University

1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8, Canada

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

Abstract

In this paper we propose a framework for the incorporation of formal methods in the design flow of DSP (Digital Signal Processing) systems in a rigorous way. In the proposed approach we model and verify DSP descriptions at different abstraction levels using higher-order logic based on the HOL (Higher Order Logic) theorem prover. This framework enables the formal verification of DSP designs which in the past could only be done partially using conventional simulation techniques. To this end, we provide a shallow embedding of DSP descriptions in HOL at the floating-point, fixed-point, RTL (Register Transfer Level), and netlist gate levels. We make use of existing formalization of floating-point theory in HOL and introduce a parallel one for fixed-point arithmetic. The high ability of abstraction in HOL allows a seamless hierarchical verification encompassing the whole DSP design path, starting from top level floating- and fixed-point algorithmic descriptions down to RTL, and gate level implementations. We illustrate the new verification framework using different case studies such as digital filters and FFT (Fast Fourier Transform) algorithms.

1 Introduction

Digital system design is characterized by ever increasing system complexity that has to be implemented within reduced time, resulting in minimum costs and short time-to-market. These characteristics call for a seamless design flow that allows to perform the design steps on the highest suitable level of abstraction. For most digital signal processing systems, the design has to result in a fixed-point implementation. This is due to the fact that these systems are sensitive to power consumption, chip size and price per device. Fixed point realizations outperform floating-point realizations by far with regard to these criteria. Figure 1 illustrates a general DSP design flow as used nowadays in leading industry design projects.

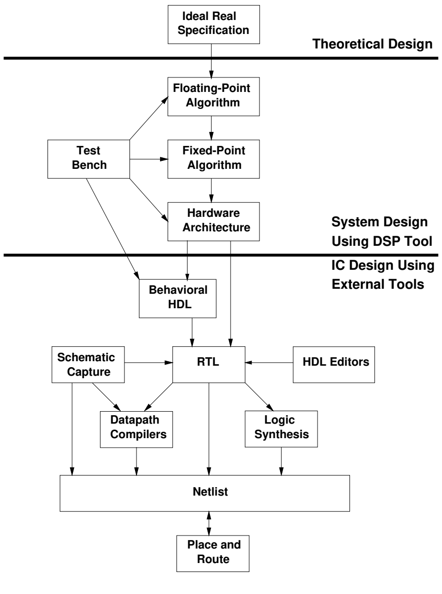

Figure 1: DSP design flow

The design of digital signal processing systems starts from an ideal real number specification. In theoretical analysis of digital systems, we generally assume that signal values and system coefficients are represented in the real number system and expressed to infinite precision. This allows to ignore the effects of finite wordlengths and fixed exponents and to abstract from all implementation details. When implemented in 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. In this case, attention must be paid to the effects of using finite register lengths to represent all relevant design parameters [26]. Despite the advantages offered by digital networks, there is an inherent accuracy problem associated with digital signal processing systems, since the signals are represented by a finite number of bits and the arithmetic operations must be carried out with an accuracy limited by the finite word length of the number representation. Depending on the type of arithmetic used in the system algorithm, the type of quantization used to reduce the word length to a desired size, and the exact system structure used, one can generally estimate how system performance is affected by these finite precision effects. There are several types of arithmetics used in the implementation of digital systems. Among the most common are floating-point and fixed-point. At the floating- and fixed-point levels, 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 (numbers) 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/synthesized into the best suited target description, either using a hardware description language (HDL) or a programming language. Meeting the above (sometimes conflicting) requirements is a great challenge in any DSP design project.

The above design process can be aided by a number of specialized CAD tools such as SPW (Cadence) [7], CoCentric (Synopsys) [8], Matlab-Simulink (Mathworks) [24], and FRIDGE (Aachen UT) [22]. The conformance of the fixed-point implementation with respect to the descriptions in floating-point or real algorithm on one hand, and the RT (Register Transfer) and gate levels on the other hand is verified by simulation techniques. Simulation, however, is known to provide partial verification as it cannot cover all design errors, especially for large systems. On the other hand, adopting formal verification in system design generally means using methods of mathematical proof rather than simulation to ensure the quality of the design, to improve the robustness of a design and to speed up the development.

In this paper we propose a methodology for applying formal methods to the design flow of DSP systems in a rigorous way. The corresponding commutating diagram is shown in Figure 2.

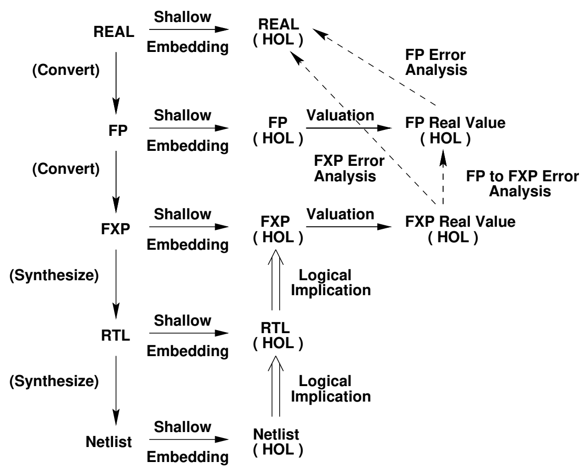

Figure 2: Proposed DSP specification and verification approach

Thereafter, we model the ideal real specification of the DSP algorithms and the corresponding floating-point (FP) and fixed-point (FXP) representations as well as the RT and gate level implementations as predicates in higher-order logic. The overall methodology for the formal specification and verification of DSP algorithms will be based on the idea of shallow embedding of languages [6] using the HOL theorem proving environment[13]. In the proposed approach, we first focus on the transition from real to floating- and fixed-point levels. For this, we make use of existing theories in HOL on the construction of real [14] and complex [17] numbers, the formalization of IEEE-754 standard [18] based floating-point arithmetic [15, 16], and the formalization of fixed-point arithmetic [4]. We use valuation functions to find the real values of the floating- and fixed-point DSP 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 roundings and arithmetic operations against their abstract mathematical counterparts. Finally, based on these lemmas, we derive expressions for the accumulation of roundoff error in floating- and fixed-point DSP algorithms using recursive definitions and initial conditions. While theoretical work on computing the errors due to finite precision effects in the realization of DSP algorithms with floating- and fixed-point arithmetics has been extensively studied since the late sixties [21], this paper contains the first formalization and proof of this analysis using a mechanical theorem prover, here HOL. The formal results are found to be in good agreement with the theoretical ones.

After handling the transition from real to floating- and fixed-point levels, we turn to the HDL representation. At this point, we use well known techniques to model the DSP design at the RTL level within the HOL environment. The last step is to verify this level using a classical hierarchical proof approach in HOL [25]. In this way, we hierarchically prove that the DSP RTL implementation implies the high level fixed-point algorithmic specification, which has already been related to the floating-point description and the ideal real specification through the error analysis. The verification can be extended, following similar manner, down to gate level netlist either in HOL or using other commercial verification tools as depicted in Figure 2.

The rest of the paper is organized as follows: Section 2 describes the fixed-point arithmetic and the details of its formalization in higher-order-logic. Section 3 describes the error analysis of digital filters using HOL theorem proving. Section 4 presents the verification of FFT algorithms in HOL from real specification to gate level implementation. Finally, Section 5 concludes the paper and outlines the future research directions.

2 Formalization of Fixed-Point Arithmetic in HOL

In this section, the objective is to formalize the fixed-point arithmetic in higher-order logic as a basis for checking the correctness of the implementation of DSP designs against higher level algorithmic descriptions in floating-point and fixed-point representations.

2.1 Fixed-Point Numbers

We first describe the fixed-point arithmetic definitions on which we base our formalization. Unlike floating-point arithmetic which is standardized in IEEE-754 [18] and IEEE-854 [19], current fixed-point arithmetic does not follow any particular standard and depends on the tool and the language used to design the DSP chip. While we tried to keep these definitions as general as possible, the fixed-point numbers format, arithmetic operations, overflow and quantization modes, and exception handling adopted are to some extent influenced by the fixed-point arithmetic defined by Cadence SPW [7] and Synopsys SystemC [27].

A fixed-point number has a fixed number of binary digits and a fixed position for the decimal point with respect to that sequence of digits. Fixed-point numbers can be either unsigned (always positive) or signed (in two's complement representation). Fixed-point numbers are expressed as a pair consisting of a binary string and a set of attributes. The attributes specify how the binary string is interpreted. Generally, the attributes consists of three parameters to represent the total word length, the integer word length, and the sign format. Operations performed on fixed-point data types are done using arbitrary and full precision. After the operation is complete, the resulting operand is cast to fit the fixed-point data type object. The casting operation applies the quantization behavior of the target object to the new value and assigns the new value to the target object. Then, the appropriate overflow behavior is applied to the result of the process which gives the final value. In addition to the parameters corresponding to the input operands and output result, the arithmetic operations take specific parameters defining the overflow and quantization modes. These parameters are the Quantization mode, Overflow mode, and Number of saturated bits. Quantization effects are used to determine what happens to the LSBs of a fixed-point type when more bits of precision are required than are available. The quantization modes are: Quantization to Plus Infinity, Quantization to Zero, Quantization to Minus Infinity, Quantization to Infinity, Convergent Quantization, Truncation, and Truncation to Zero. In addition to quantization modes, we can use overflow modes to approximate a higher range for fixed-point operations. Usually, overflow occurs when the result of an operation is too large or too small for the available bit range. Specific overflow modes can then be implemented to reduce the loss of data. Overflow modes are: Saturation, Saturation to Zero, Symmetrical Saturation, Wrap-Around, and Sign Magnitude Wrap-Around.

2.2 Fixed-point Formalization in HOL

In this section, we present formalization of the fixed-point arithmetic in higher-order logic, based on the general purpose HOL theorem prover. HOL's basic types include the natural numbers and booleans. It also includes other specific extensions like John Harrison's reals library [14] which proved to be essential for our fixed-point arithmetic formalization.

Fixed point numbers are modeled in HOL as a pair of elements composed of a bit string and a set of attributes. The bit string is represented by a boolean word and the set of attributes is itself a combination of six elements representing the word length, integer word length, sign type, rounding mode, overflow mode, and the number of saturation bits. The fixed-point numbers are then partitioned using special predicates into signed and un-signed numbers. The validity of a fixed-point number and a set of attributes is defined using special predicates. Then we defined the actual HOL type for the fixed-point numbers. The valuation function is then defined to specify a real value to fixed-point numbers using separate formulas for signed and unsigned numbers. The constants for the smallest and largest fixed-point numbers for a given format together with their corresponding real values are also defined using specific functions. Then, we defined enumerated data types for seven rounding modes and five overflow modes in fixed-point arithmetic. The rounding function is then defined case by case on the rounding and overflow modes. Then, we defined the operations on fixed-point numbers which are performed using the arbitrary precision in real domain and then the result is casted to the output format.

2.3 Verification of Fixed-Point Operations

According to the discussion in the previous section, each fixed-point number has a corresponding real number value. The correctness of a fixed-point operation can be specified by comparing its output with the true mathematical result, using the valuation function *value* that converts a fixed-point to an infinitely precise number. For example, the correctness of a fixed-point adder *fxpAdd* is specified by comparing it with its ideal counterpart $+$. That is, for each pair of fixed-point numbers *(a,b)*, we compare *value (a)* $+$ *value (b)* and *value (fxpAdd (a,b))*. In other words, we check if the diagram in Figure 3 commutes.

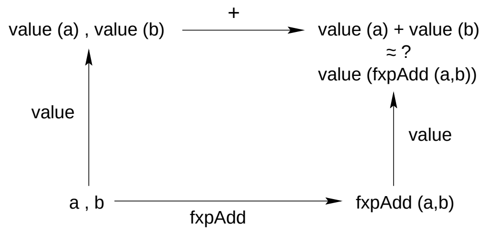

Figure 3: Correctness criteria for fixed-point addition

Therefore, the correctness of fixed-point operations can be specified by comparing the operation's output with the true mathematical result. Since the operations are defined as if they first performed using infinite precision and then the result is rounded to fit in the destination format, the verification of operations is closely related to bounding the error in rounding function. The steps in analysis of fixed-point rounding error in HOL are as follows:

- **Lemmas for Analyzing the Fixed-Point Rounding Operation:** We first proved lemmas concerning with the approximation of a real number with a fixed-point number. We proved that in a finite nonempty set of fixed-point numbers we can find the best approximation to a real number based on a given valuation function. Then, we proved that the chosen best approximation to a real number satisfying a property p from a finite and non empty set of fixed-point numbers is unique and is itself a member of the set. Finally, we proved that the chosen best approximation to a real number satisfying a property p from the set of all valid fixed-point numbers with a given attributes is itself a valid fixed-point number.

- **Rounding Error in Fixed-Point Arithmetic Operations:** Then, we defined the error resulting from rounding a real number to a fixed-point value. Then, we established the first main theorems on the correctness of fixed-point arithmetic operations. According to these theorems, if the input fixed-point operands and the output attributes are valid then the result of fixed-point operations is valid. Also the result of the operations is related to the real result considering the error.

- **General Fixed-Point Error Bound Theorem:** In the next step, we established the second main theorem on fixed-point rounding error analysis which concerns bounding the error. According to this theorem, the error in rounding a real number which is in the range representable by a given set of attributes X is less than the quantity $1/2^{fracbits(X)}$, where *fracbits* is the number of bits that are to the right of the binary point in the given fixed-point format X. To explain the theorem, we consider the distribution of fixed-point number on the real axis as shown by Figure 4. Thereafter, the representable range of fixed-point numbers is divided into 2^N equispaced quantization steps with the distance between two successive steps equal to $1/2^b$. Suppose that $x \in \mathbb{R}$ is approximated by a fixed-point number a. The position of these values are labeled in the figure. The error $\mid x - a \mid$ is hence less than the length of one interval, or $1/2^b$, as mentioned in the second theorem. The details can be found in [4].

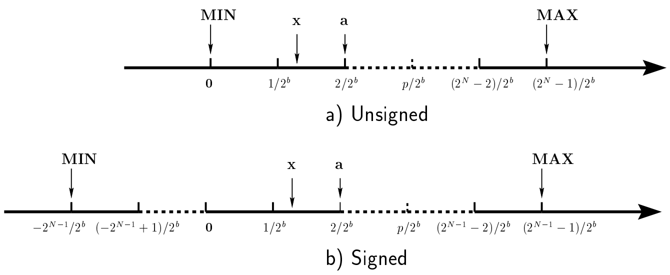

Figure 4: Fixed-Point Values on the Real Axis

3 Error Analysis of Digital Filters

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 [20]. 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. 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 section, as the first case study for our verification methodology, we show how these errors can be mechanically analyzed 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 basic 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.

3.1 Error Analysis Models

In this section we introduce the fundamental error analysis theorems [28, 11], and the corresponding lemmas in HOL for the floating-point [15, 16] and fixed-point arithmetics [4]. These theorems are then used in the next sections as a model for the analysis of the roundoff error in digital filters.

In analyzing the effect of floating-point roundoff, the effect of rounding will be represented multiplicatively. Letting $*$ denote any of the arithmetic operations +, -, $\times$, /, it is known [11, 28] that, if p represents the precision of the floating-point format, then

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

The notation fl (.) is used to denote that the operation is performed using floating-point arithmetic. The theorem relates the floating-point arithmetic operations such as addition, subtraction, multiplication, and division to their abstract mathematical counterparts according to the corresponding errors.

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 for fixed-point arithmetic operations against their abstract mathematical counterparts can be stated as

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

The notation fxp (.) is used to denote that the operation is performed using fixed-point arithmetic. We have proved equations (1) and (2) as theorems in higher-order logic within HOL. The theorems are proved under the assumption that there is no overflow or underflow in the operation result. This means that the input values are scaled so that the real value of the result is located in the ranges defined by the maximum and minimum representable values of the given floating-point and fixed-point formats.

3.2 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.

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

$$w_n\ =\ \sum_{i=0}^{M} b_i\ x_{n-i}\ -\ \sum_{i=1}^{L} a_i\ w_{n-i} \tag{3}$$

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 basic forms of realizing a digital filter, namely the direct, parallel, and cascade forms. If the output sequence is calculated by using the equation (3), the digital filter is said to be realized in the direct form. In parallel form, the entire filter is visualized as the parallel connection of the simpler filters of a lower order. In cascade form, the filter is visualized as a cascade of lower filters.

There are three common sources of errors associated with the filter of the equation (3), namely [23]: input quantization, coefficient inaccuracy, and round-off accumulation. Therefore, for the digital filter of the equation (3) 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, e'_n and e''_n, 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.

The corresponding flowgraph showing the effect of roundoff error using the fundamental error analysis theorems according to the equations (1) and (2), is given by Figure 5, which also indicates the order of the calculation. The quantities $\delta, \epsilon_{n,k}, \zeta_{n,k}, \eta_{n,k}$, and ξ_n in Figure 5 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 $\delta'_{n,k}, \epsilon'_{n,k}, \zeta'_{n,k}, \eta'_{n,k}$, and ξ'_n.

For the error analysis, we need to calculate the y_n and v_n sequences, and compare them with the ideal output sequence w_n specified by the equation (3) to obtain the corresponding errors e_n, e'_n, and e''_n. Therefore, we derived the difference equations for the errors between the different levels showing the accumulation of the roundoff error in HOL. Similar analysis is performed for the parallel and cascade forms of realization based on the corresponding error flowgraphs. The details can be found in [3].

4 Verification of FFT algorithm

The fast Fourier transform (FFT) [5, 9] is an algorithm to compute the discrete Fourier transform with a substantial time saving over conventional methods. FFT algorithms are based on the fundamental principle

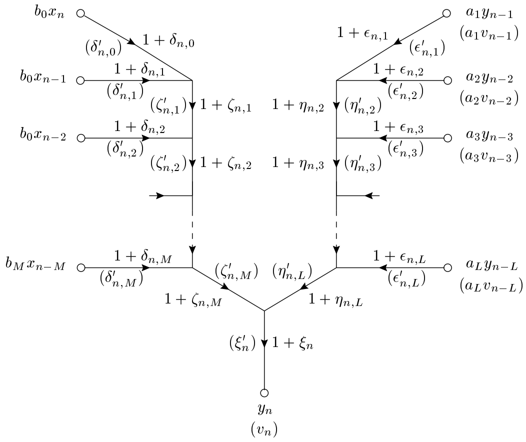

Figure 5: Error flowgraph for Lth-order filter (Direct form)

of decomposing the computation of the discrete Fourier transform of a sequence of length N into successively smaller discrete Fourier transforms. The manner in which this principle is implemented leads to a variety of different algorithms, all with comparable improvements in computational speed. Two basic classes of FFT algorithms are the decimation-in-time (DIT) [10] and decimation-in-frequency (DIF) [12]. As the second case study in this project, we considered the formal verification of FFT algorithms. We used our methodology to derive expressions for the accumulation of roundoff error in floating- and fixed-point FFT algorithms by recursive definitions and initial conditions, considering the effects of input quantization and inaccuracy in the coefficients. Based on the extensively studied theoretical work on computing the errors due to finite precision effects in the realization of FFT algorithms with floating- and fixed-point arithmetics, we performed a similar analysis using the HOL theorem proving environment. The formal results are found to be in good agreement with the theoretical ones.

4.1 Error Analysis of FFT Algorithms in HOL

In this section, the principal results for roundoff accumulation in FFT algorithms using HOL theorem proving are derived and summarized. For the most part, the following discussion is phrased in terms of the decimation-in-frequency form of radix-2 algorithm. The results, however, are applicable with only minor modification to the decimation-in-time form. Furthermore, most of the ideas employed in the error analysis of the radix-2 algorithms can be utilized in the analysis of other algorithms.

Let $\{A_k(p)\}_{p=0}^{N-1}$ denote the N complex numbers calculated at the kth step. Then the decimation in frequency (DIF) FFT algorithm can be expressed as [21]

$$A_{k+1}(p) = \begin{cases} A_k(p) + A_k(p + 2^{m-1-k}) & \text{if } p_k = 0 \\ [A_k(p - 2^{m-1-k}) - A_k(p)]\; w_k(p) & \text{if } p_k = 1 \end{cases} \tag{4}$$

where $w_k(p)$ is a power of the principle roots of unity, and $N = 2^m$, where m is an integer value. Equation (4) is carried out for $k = 0, 1, 2, \ldots, m-1$, with $A_0(p) = x(p)$, where $\{x(n)\}_{n=0}^{N-1}$ is the set of input sequence. It can be shown [12] that at the last step $\{A_m(p)\}_{p=0}^{N-1}$ are the discrete Fourier coefficients in rearranged order.

There are three common sources of errors associated with the FFT algorithms, namely: input quantization, coefficient inaccuracy, and round-off accumulation. Therefore, the actual array computed by using equation (4) is in general different from $\{A_k(p)\}_{p=0}^{N-1}$. We denote the actual floating- and fixed-point computed arrays by $\{A'_k(p)\}_{p=0}^{N-1}$ and $\{A''_k(p)\}_{p=0}^{N-1}$, respectively. Then, we define the corresponding errors of the pth element at step k as $e_k(p), e'_k(p)$, and $e''_k(p)$, where $e_k(p)$ and $e'_k(p)$ are defined as the error between the actual floating- and fixed-point implementations and the ideal real specification, respectively. $e''_k(p)$ is the error in transition from floating- to fixed-point levels.

In equation (4) the $\{A_k(p)\}$ are complex numbers, so their real and imaginary parts are calculated separately. We define the real and imaginary of $A_k(p)$ and $w_k(p)$ as $B_k(p), C_k(p)$ and $U_k(p), V_k(p)$, respectively. Similarly, we express the real and imaginary parts of $A'_k(p)$, $B'(p)$ and $C'_k(p)$, and $A''_k(p)$, $B''_k(p)$ and $C''_k(p)$,

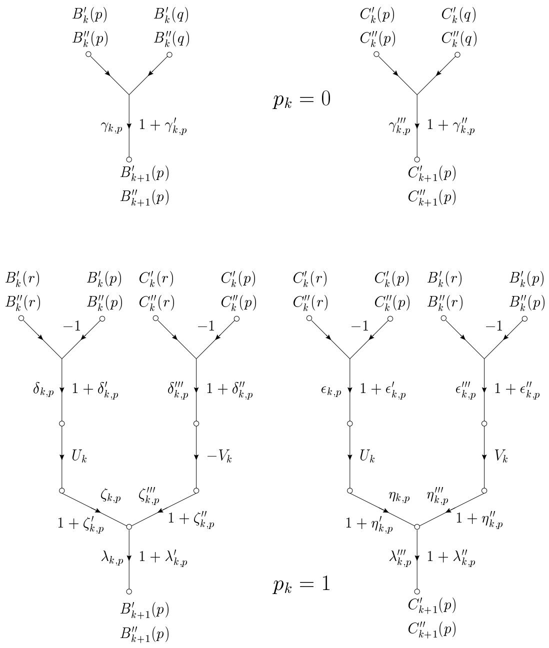

Figure 6: Error flowgraph for decimation-in-frequency FFT

using the floating- and fixed-point operations, respectively. The corresponding error flowgraph showing the effect of roundoff error using the fundamental floating- and fixed-point error analysis theorems according to the equations (1) and (2), respectively, is given in Figure 6, which also indicates the order of the calculation.

The quantities γ', γ'', δ', δ'', ϵ', ϵ'', ζ', ζ'', η', η'', λ', and λ'' in Figure 6 are errors caused by floating-point roundoff at each arithmetic step. The corresponding error quantities for fixed-point roundoff are γ, γ''', δ, δ''', ϵ, ϵ''', ζ, ζ''', η, η''', λ, and λ'''.

Therefore, we derived in HOL, expressions for the accumulation of roundoff error for FFT algorithm by recursive equations and initial conditions. The details can be found in [2].

4.2 FFT Design Implementation Verification

In this section, we describe the application of the proposed approach for the verification in HOL of the transition from real, floating- and fixed-point specifications to RTL implementation of an FFT algorithm. We have chosen the case study of a radix-4 pipelined 64-point complex FFT core available as VHDL RTL model in the Xilinx Coregen library [29]. All proofs have been conducted in HOL, hence establishing a correctness of the FFT design implementation with respect to its high level algorithmic specifications. Figure 7 shows the overall block diagram of the Radix-4 64-point pipelined FFT design.

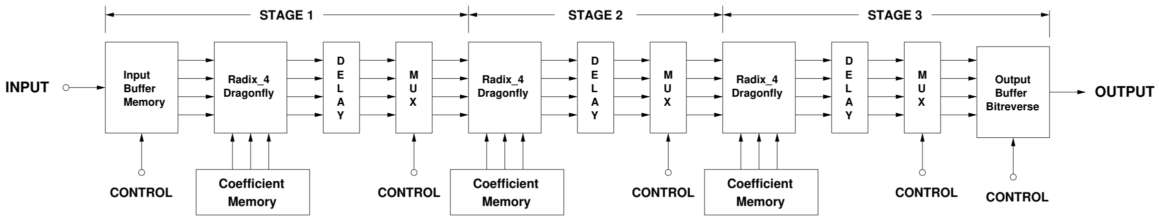

Figure 7: Radix-4 64-point pipelined FFT implementation

The basic elements are memories, delays, multiplexers, and dragonflies. In general, the 64-point pipelined FFT requires the calculation of three radix-4 dragonfly ranks. Each radix-4 dragonfly is a successive combination of a radix-4 butterfly with four twiddle factor multipliers. The FFT core accepts naturally ordered data on the input buses in a continuous stream, performs a complex FFT, and streams out the DFT samples on the output buses in a natural order. These buses are respectively the real and imaginary components of the input and output sequences. An internal input data memory controller orders the data into blocks to be presented to the FFT processor. The twiddle factors are stored in coefficient memories. The real and imaginary components of complex input and output samples and the phase factors are represented as 16-bit 2's complement numbers. The unscrambling operation is performed using the output bit-reversing buffer.

In HOL, we first modeled the RTL description of a radix-4 butterfly as a predicate in higher-order logic. The block takes a vector of four complex input data and performs the operations, to generate a vector of four complex output signals. The real and imaginary parts of the input and output signals are represented as 16-bit Boolean words. We defined separate functions in HOL for arithmetic operations such as addition, subtraction, and multiplication on complex two's complement 16-bit Boolean words. Then, we built the complete butterfly structure using a proper combination of these primitive operations.

Thereafter, we described a radix-4 dragonfly block as a conjunction of a radix-4 butterfly and four 16-bit twiddle factor complex multipliers. Finally, we modeled the complete RTL description of the radix-4 64-point structure in HOL. The FFT block is defined as a conjunction of 48 instantiations of radix-4 dragonfly blocks. Proper time instances of the input and output signals are applied to each block.

Following similar steps, we described the radix-4 64-point FFT structures as fixed-point, floating-point, and real domains in HOL using the corresponding complex data types and arithmetic operations.

The formal verification of the radix-4 decimation in frequency FFT algorithm case study was performed based on the commutating diagram in Figure 2, in that we proved hierarchically that the FFT Netlist implies the FFT RTL; and then proved that the FFT RTL description implies the corresponding fixed-point model. The proof of the FFT block is then broken down into the corresponding proof of the dragonfly block, which itself is broken down into the proofs of butterfly and primitive arithmetic operations. We used the data abstraction functions to convert a complex vector of 16-bit two's complement Boolean words into the corresponding fixed-point vector.

Then, we proved three theorems encompassing the error analysis of the radix-4 decimation in frequency FFT algorithm. The first lemma represents the error between the real number specification and the floating-point specification. The second lemma represents the error between the real number and the fixed-point specifications. The third lemma represents the error between floating-point and fixed-point specifications. According to these lemmas, the floating-point and fixed-point implementations and the real specification of a radix-4 decimation in frequency FFT algorithm are related to each other based on the corresponding data abstraction, and error analysis functions.

Finally, using the obtained theorems, we easily deduced our ultimate theorem proving the correctness of the real specification from the RTL implementation, taking into account the error analysis computed beforehand. A complete list of the derived HOL definitions and theorems can be found in [1].

5 Conclusions

In this paper, we described a methodology for the formal specification and verification of DSP systems designs at different abstraction levels. We proposed a shallow embedding of DSP descriptions at different levels in HOL. For the verification of the transition from floating- to fixed-point levels, we proposed an error analysis approach in which we consider the effects of finite precision in the implementation of DSP systems. These include errors due to the quantization of input samples and system coefficients, and also roundoff accumulation in arithmetic operations. The verification from fixed-point to RTL and netlist levels is performed using traditional hierarchical verification in HOL. In this paper we demonstrated our methodology using the case studies of digital filters and the fast Fourier transform algorithms. The approach covers three basic forms (direct, parallel, and cascade) of realization of the digital filters, and the two canonical forms (decimation-in-time, and decimation-in-frequency) of realization of the FFT algorithm using real, floating-, and fixed-point arithmetic as well as their RT implementations, each entirely specified in HOL. We proved lemmas to derive expressions for the accumulation of roundoff error in floating- and fixed-point designs compared to the ideal real specification. Then we proved that the FFT RTL implementation implies the corresponding specification at the fixed-point level using classical hierarchical verification in HOL, hence bridging the gap between hardware implementation and high levels of mathematical specification. In this work we also have contributed to the upgrade and application of established real, complex real, floating- and fixed-point theories in HOL to the analysis of errors due to finite precision effects, and applied them on the realization of the FFT algorithms. Error analyses using theoretical paper-and-pencil proofs did exist since the late sixties while design verification is exclusively done by simulation techniques. We believe this is the first time a complete formal framework has been proposed for the specification and verification of the DSP algorithms at different levels of abstraction. The methodology presented in this paper opens new avenues in using formal methods for the verification of digital signal processing (DSP) systems as complement to traditional theoretical (analytical) and simulation techniques. We are currently investigating the verification of complex wired and wireless communication systems, whose building blocks, heavily make use of several instances of the FFT algorithms. As a future work, we also plan to extend the error analyses to cover worst-case, average, and variance errors. Finally, we plan to link HOL with computer algebra systems to create a sound, reliable, and powerful system for the verification of DSP systems.

References

[1] B. Akbarpour, "Modeling and Verification of DSP Designs in HOL," Ph.D. Thesis, Concordia University, Department of Electrical and Computer Engineering, Montreal, Canada, March 2005.

[2] B. Akbarpour and S. Tahar, "A Methodology for the Formal Verification of FFT Algorithms in HOL," In Formal Methods in Computer-Aided Design, LNCS 3312, pp. 37-51, Springer-Verlag, 2004.

[3] B. Akbarpour and S. Tahar, "Error Analysis of Digital Filters using Theorem Proving," In Theorem Proving in Higher Order Logics, LNCS 3223, pp. 1-16, Springer-Verlag, 2004.

[4] B. Akbarpour, S. Tahar, and A. Dekdouk, "Formalization of Fixed-Point Arithmetic in HOL," Formal Methods in Systems Design, 27: 173-200, 2005.

[5] E. O. Brigham, "The Fast Fourier Transform," Prentice Hall, 1974.

[6] 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.

[7] Cadence Design Systems, Inc., "Signal Processing WorkSystem (SPW) User's Guide," USA, July 1999.

[8] Synopsys, Inc., "CoCentric^{TM} System Studio User's Guide," USA, Aug. 2001.

[9] W. T. Cochran et. al., "What is the Fast Fourier Transform," IEEE Transactions on Audio and Electroacoustics, AU-15: 45-55, Jun. 1967.

[10] J. W. Cooley, J. W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," Mathematics of Computation, 19: 297-301, Apr. 1965.

[11] G. Forsythe and C. B. Moler, "Computer Solution of Linear Algebraic Systems," Prentice-Hall, 1967.

[12] W. M. Gentleman and G. Sande, "Fast Fourier Transforms - For Fun and Profit," In AFIPS Fall Joint Computer Conference, Vol. 29, pp. 563-578, Spartan Books, Washington, DC, 1966.

[13] M. J. C. Gordon and T. F. Melham, "Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic," Cambridge University Press, 1993.

[14] J. R. Harrison, "Constructing the Real Numbers in HOL," Formal Methods in System Design, 5 (1/2): 35-59, 1994.

[15] J. R. Harrison, "A Machine-Checked Theory of Floating-Point Arithmetic," In Theorem Proving in Higher Order Logics, LNCS 1690, pp. 113-130, Springer-Verlag, 1999.

[16] J. R. Harrison, "Floating-Point Verification in HOL Light: The Exponential Function," Formal Methods in System Design, 16 (3): 271-305, 2000.

[17] J. R. Harrison, "Complex Quantifier Elimination in HOL," In Supplemental Proceedings of the International Conference on Theorem Proving in Higher Order Logics, pp. 159-174, Edinburgh, Scotland, UK, Sep. 2001.

[18] 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.

[19] The Institute of Electrical and Electronic Engineers, Inc., "IEEE, Standard for Radix-Independent Floating-Point Arithmetic," ANSI/IEEE Std 854, USA, 1987.

[20] J. F. Kaiser, "Digital Filters," In System Analysis by Digital Computer, F. F. Kuo and J. F. Kaiser, Eds., pp. 218-285, Wiley, 1966.

[21] T. Kaneko and B. Liu, "Accumulation of Round-Off Error in Fast Fourier Transforms," Journal of Association for Computing Machinery, 17 (4): 637-654, Oct. 1970.

[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, Paris, France, February 1998.

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

[24] Mathworks, Inc., "Simulink Reference Manual," USA, 1996.

[25] T. Melham, "Higher Order Logic and Hardware Verification," Cambridge Tracts in Theoretical Computer Science 31, Cambridge University Press, 1993.

[26] 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.

[27] Open SystemC Initiative, "SystemC Language Reference Manual," USA, 2004.

[28] J. H. Wilkinson, "Rounding Errors in Algebraic Processes," Prentice-Hall, 1963.

[29] Xilinx, Inc., "High-Performance 64-Point Complex FFT/IFFT V2.0, Product Specification," USA, Aug. 2000, http://xilinx.com/ipcenter.