A Methodology for the Formal Verification of FFT Algorithms in HOL

Behzad Akbarpour and Sofiène Tahar

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

Abstract. This paper addresses the formal specification and verification of fast Fourier transform (FFT) algorithms at different abstraction levels based on the HOL theorem prover. We make use of existing theories in HOL on real and complex numbers, IEEE standard floating-point, and fixed-point arithmetics to model the FFT algorithms. Then, we derive, by proving theorems in HOL, expressions for the accumulation of roundoff error in floating- and fixed-point FFT designs with respect to the corresponding ideal real and complex numbers specification. The HOL formalization and proofs are found to be in good agreement with the theoretical paper-and-pencil counterparts. Finally, we use a classical hierarchical proof approach in HOL to prove that the FFT implementations at the register transfer level (RTL) implies the corresponding high level fixed-point algorithmic specification.

1 Introduction

The fast Fourier transform (FFT) [6, 9] is a highly efficient method for computing the discrete Fourier transform (DFT) coefficients of a finite sequence of complex data. Because of the substantial time saving over conventional methods, the fast Fourier transform has found important applications in a number of diverse fields such as spectrum analysis, speech and optical signal processing, and digital filter design. FFT algorithms are based on the fundamental principle of decomposing the computation of the discrete Fourier transform of a finite-length 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. There are two basic classes of FFT algorithms for which the number of arithmetic multiplications and additions as a measure of computational complexity is proportional to $N \; log \; N$ rather than N^2 as in the conventional methods. The first proposed by Cooley and Tukey [10], called decimation-in-time (DIT), derives its name from the fact that in the process of arranging the computation into smaller transformations, the input sequence (generally thought of as a time sequence) is decomposed into successively smaller subsequences. In the second general class of algorithms proposed by Gentleman and Sande [13], the sequence of discrete Fourier transform coefficients is decomposed into smaller subsequences, hence its name, decimation-in-frequency (DIF). In a theoretical analysis of the fast Fourier transform, we generally assume that signal values and system coefficients are represented with real numbers 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. There is an inherent accuracy problem in calculating the Fourier coefficients, 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. Among the most common types of arithmetic used in the implementation of FFT systems 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 tools such as SPW (Cadence) [27], CoCentric (Synopsys) [8], Matlab-Simulink (Mathworks) [22], and FRIDGE (Aachen UT) [20].

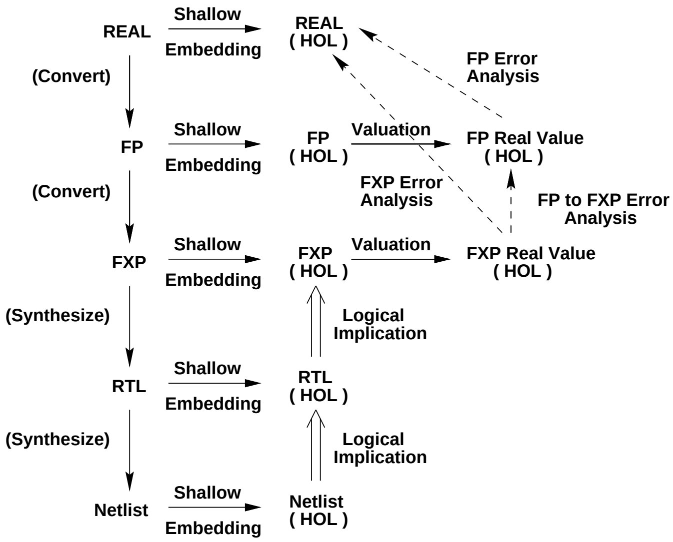

Fig. 1. FFT specification and verification methodology

In this paper, we describe a methodology for the formal specification and verification of FFT algorithms based on shallow embedding technique [5] using the HOL theorem proving environment [14]. The overall methodology is depicted in the commutating diagram shown in Figure 1. We first focus on the transition from real to floating- and fixed-point levels. Here, we model the ideal real specification of the FFT algorithms 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 the construction of real [15] and complex [18] numbers, the formalization of IEEE-754 standard based floating-point arithmetic [16, 17], and the formalization of fixed-point arithmetic [1, 2]. We use valuation functions to find the real values of the floating- and fixed-point FFT 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, for each of the two canonical forms of realization, expressions for the accumulation of roundoff error in floating- and fixed-point FFT algorithms using recursive definitions and initial conditions. While theoretical work on computing the errors due to finite precision effects in the realization of FFT algorithms with floating- and fixed-point arithmetics has been extensively studied since the late sixties [19], 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 FFT 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 [23]. In this way, we hierarchically prove that the FFT 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 1, which is not covered in this paper.

The rest of this paper is organized as follows: Section 2 reviews some related work. Section 3 describes the details of the error analysis in HOL of the FFT algorithms at the real, floating-, and fixed-point levels. Section 4 describes the verification of the FFT algorithms in the transition from fixed-point to register transfer levels. Finally, Section 5 concludes the paper.

2 Related Work

Analysis of errors in FFT realizations due to finite precision effects has traditionally relied on paper-and-pencil proofs and simulation techniques. The roundoff error in using the FFT algorithms depends on the algorithm, the type of arithmetic, the word length, and the radix. For FFT algorithms realized with fixed-point arithmetic, the error problems have been studied extensively. For instance, Welch [30] presented an analysis of the fixed-point accuracy of the radix-2 decimation-in-time FFT algorithm. Tran-Thong and Liu [28] presented a general approach to the error analysis of the various versions of the FFT algorithm when fixed-point arithmetic is used. 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 Gentleman and Sande [13], who presented an upper bound on the mean-squared error for floating-point decimation-in-frequency FFT algorithm. Weinstein [29] presented a statistical model for roundoff errors of the floating-point FFT. Kaneko and Liu [19] presented a detailed analysis of roundoff error in the FFT decimation-in-frequency algorithm using floating-point arithmetic. This analysis is later extended by the same authors to the FFT decimation-in-time algorithm [21]. Oppenheim and Weinstein [26] discussed in some detail the effects of finite register length on implementations of digital filters, and FFT algorithms.

In order to validate the error analysis, most of the above work compare the theoretical results with experimental simulation. In this paper, we show how the above error analyses for the FFT algorithms can be mechanically performed using the HOL theorem prover, providing a superior approach to validation by simulation. Our focus will be on the process of translating the hand proofs into equivalent proofs in HOL. The analysis we develop is mainly inspired by the work done by Kaneko and Liu [19], who proposed a general approach to the error analysis problem of the decimation-in-frequency FFT algorithm using floating-point arithmetic. Following a similar idea, we have extended this theoretical analysis for the decimation-in-time and fixed-point FFT algorithms. In all cases, good agreements between formal and theoretical results were obtained.

Prior work on error analysis and theorem proving was done by Harrison [17], who verified floating-point algorithms against their abstract mathematical counterparts using the HOL Light theorem prover. His error analysis is very similar to the type of analysis performed for DSP algorithms. The major difference, however, is the use of statistical methods and mean square error analysis for DSP algorithms which is not covered in the error analysis of the mathematical functions used by Harrison. To perform such an analysis in HOL, we need to develop a mechanized theory on the properties of random processes. This type of analysis is not addressed in this paper and is a part of our work in progress.

Related work on the formalization and mechanical verification of the FFT algorithm was done by Gamboa [12] using the ACL2 theorem prover. The author formalized the FFT as a recursive data-parallel algorithm, using the powerlist data structure. He also presented an ACL2 proof of the correctness of the FFT algorithm, by translating the hand proof taken from Misra's seminal paper on powerlists [24] into a mechanical proof in ACL2. In the same line, Capretta [7] presented the formalization of the FFT using the type theory proof tool Coq. To facilitate the definition of the transform by structural recursion, Capretta used the structure of polynomial trees which is similar to the data structure of powerlists introduced by Misra. Finally, he proved its correctness and the correctness of the inverse Fourier transform (IFT).

Bjesse [4] described the verification of FFT hardware at the netlist level with an automatic combination of symbolic simulation and theorem proving using the Lava hardware development platform. He proved that the sequential pipelined implementation of the radix-4 decimation-in-time FFT is equivalent to the corresponding combinational circuit. He also proved that the abstract implementation of the radix-2 and the radix-4 FFT are equivalent for sizes that are an exponent of four. While [12] and [7] prove the correctness of the high level FFT algorithm against the DFT, the verification of [4] is performed at the netlist level. In contrast, our work tries to close this gap by formally specifying and verifying the FFT algorithm realizations at different levels of abstraction based on different data types. Besides, the definition used for the FFT in [12, 7] is based on the radix-2 decimation-in-time algorithm. We cover both decimation-in-time and decimation-in-frequency algorithms, and radices other than 2. The methodology we propose in this paper is, to the best of our knowledge, the first project of its kind that covers the formal specification and verification of integrated FFT algorithms at different abstraction levels starting from real specification to floating- and fixed-point algorithmic descriptions, down to RT and netlist gate levels.

3 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. In the following, we will first describe in detail the theory behind the analysis and then explain how this analysis is performed in HOL.

The discrete Fourier transform of a sequence $\{x(n)\}_{n=0}^{N-1}$ is defined as [25]

$$A(p) \;=\; \sum_{n=0}^{N-1} x(n)\ (W_N)^{np}, \qquad p \;=\; 0,1,2,\ldots,N-1 \tag{1}$$

where $W_N \;=\; e^{-j2\pi/N}$ and $j \;=\; \sqrt{-1}$. The multiplicative factors $(W_N)^{np}$ are called twiddle factors. For simplicity, our discussion is restricted to the radix-2 FFT algorithm, in which the number of points N to be Fourier transformed satisfy the relationship $N \;=\; 2^m$, where m is an integer value. The results can be extended to radices other than 2. By using the FFT method, the Fourier coefficients $\{A(p)\}_{p=0}^{N-1}$ can be calculated in $m \;=\; \log_2 N$ iterative steps. At each step, an array of N complex numbers is generated by using only the numbers in the previous array. To explain the FFT algorithm, let each integer p, $p = 0,1,2,\ldots,N-1$, be expanded into a binary form as

$$p \;=\; 2^{m-1}p_0 + 2^{m-2}p_1 + \cdots + 2p_{m-2} + p_{m-1}, \qquad p_k = 0 \ \text{ or } \ 1 \tag{2}$$

and let p^* denote the number corresponding to the reverse bit sequences of p, i.e.,

$$p^* = 2^{m-1}p_{m-1} + 2^{m-2}p_{m-2} + \cdots + 2p_1 + p_0 \tag{3}$$

Let $\{A_k(p)\}_{p=0}^{N-1}$ denote the N complex numbers calculated at the kth step. The decimation-in-frequency FFT algorithm can then be expressed as [19]

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

where $w_k(p)$ is a power of W_N given by $w_k(p) = (W_N)^{z_k(p)}$, where

$$z_k(p) = 2^k\ (2^{m-1-k}p_k + 2^{m-2-k}p_{k+1} + \cdots + 2p_{m-2} + p_{m-1}) - 2^{m-1}p_k \tag{5}$$

Equation (4) is carried out for $k = 0, 1, 2, \ldots, m-1$, with $A_0(p) = x(p)$. It can be shown [13] that at the last step $\{A_m(p)\}_{p=0}^{N-1}$ are the discrete Fourier coefficients in rearranged order. Specifically, $A_m(p) = A(p^*)$ with p and p^* expanded and defined as in equations (2) and (3), respectively.

There are three common sources of errors associated with the FFT algorithms, namely [19]:

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 coefficients $\{w_k(p)\}$ by a finite word length.
3. **round-off accumulation:** caused by the accumulation of roundoff errors at arithmetic operations.

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

$$e_k(p) = A'_k(p) - A_k(p) \tag{6}$$
$$e'_k(p) = A''_k(p) - A_k(p) \tag{7}$$
$$e''_k(p) = A''_k(p) - A'_k(p) \tag{8}$$

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 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, 31] that, if p represents the precision of the floating-point format, then

$$fl\ (x\ *\ y) = (x\ *\ y)(1 + \delta), \quad \text{where } |\delta| \leq 2^{-p} \tag{9}$$

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

$$fxp\ (x\ *\ y)\ =\ (x\ *\ y)\ +\ \epsilon,\quad \text{where}\ \ |\epsilon|\ \leq\ 2^{-fracbits\ (X)} \tag{10}$$

and *fracbits* is the number of bits that are to the right of the binary point in the given fixed-point format *X*. The notation $fxp\ (.)$ is used to denote that the operation is performed using fixed-point arithmetic. We have proved equations (9) and (10) 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. The details can be found in [3].

In equation (4) the $\{A_k(p)\}$ are complex numbers, so their real and imaginary parts are calculated separately. Let

$$\begin{array}{ll} B_k(p) = Re\ [A_k(p)] & C_k(p) = Im\ [A_k(p)] \\ U_k(p) = Re\ [w_k(p)] & V_k(p) = Im\ [w_k(p)] \end{array} \tag{11}$$

where the notations $Re\ [.]$ and $Im\ [.]$ denote, respectively, the real and imaginary parts of the quantity inside the bracket [.]. Equation (4) can be rewritten as

$$\left.\begin{array}{l} B_{k+1}(p) = B_k(p) + B_k(q) \\ C_{k+1}(p) = C_k(p) + C_k(q) \end{array}\right\} \quad \text{if}\ \ p_k = 0 \tag{12}$$
$$\left.\begin{array}{l} B_{k+1}(p) = [B_k(r) - B_k(p)]\ U_k(p) - [C_k(r) - C_k(p)]\ V_k(p) \\ C_{k+1}(p) = [C_k(r) - C_k(p)]\ U_k(p) + [B_k(r) - B_k(p)]\ V_k(p) \end{array}\right\} \quad \text{if}\ \ p_k = 1$$

where $q = p + 2^{m-1-k}$ and $r = p - 2^{m-1-k}$. Similarly, we can express the real and imaginary parts of $A'_{k+1}(p)$, $B'_{k+1}(p)$ and $C'_{k+1}(p)$, and $A''_{k+1}(p)$, $B''_{k+1}(p)$ and $C''_{k+1}(p)$, 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 (9) and (10), respectively, is given in Figure 2, which also indicates the order of the calculation.

Formally, a flowgraph consists of nodes and directed branches. Each branch has an input signal and an output signal with a direction indicated by an arrowhead on it. Each node represents a variable which is the weighted sum of the variables at the originating nodes of the branches that terminate on that node. The weights, if other than unity, are shown for each branch. Source nodes have no entering branches. They are used to represent the injection of the external inputs or signal sources into the flowgraph. Sink nodes have only entering branches. They are used to extract the outputs from the flowgraph [25, 3].

The quantities $\gamma'_{k,p}$, $\gamma''_{k,p}$, $\delta'_{k,p}$, $\delta''_{k,p}$, $\epsilon'_{k,p}$, $\epsilon''_{k,p}$, $\zeta'_{k,p}$, $\zeta''_{k,p}$, $\eta'_{k,p}$, $\eta''_{k,p}$, $\lambda'_{k,p}$, and $\lambda''_{k,p}$ in Figure 2 are errors caused by floating-point roundoff at each arithmetic step. The corresponding error quantities for fixed-point roundoff are $\gamma_{k,p}$, $\gamma'''_{k,p}$, $\delta_{k,p}$, $\delta'''_{k,p}$, $\epsilon_{k,p}$, $\epsilon'''_{k,p}$, $\zeta_{k,p}$, $\zeta'''_{k,p}$, $\eta_{k,p}$, $\eta'''_{k,p}$, $\lambda_{k,p}$, and $\lambda'''_{k,p}$. Thereafter, the actual real and imaginary parts of the floating- and fixed-point outputs $A'_{k+1}(p)$ and $A''_{k+1}(p)$, respectively are seen to be given explicitly by

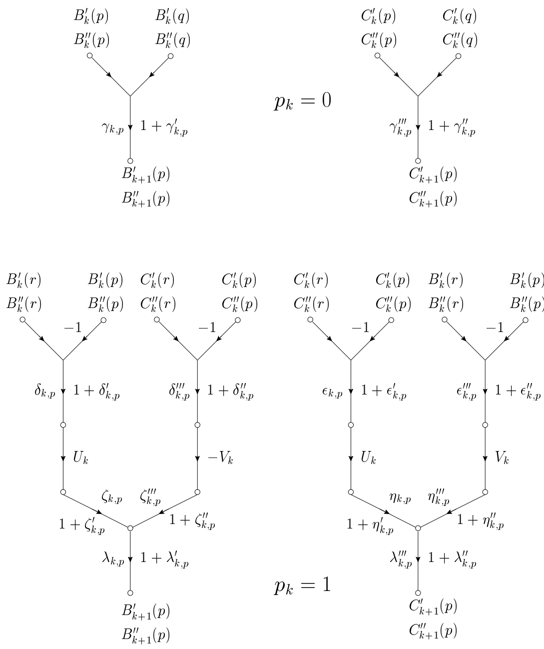

Fig. 2. Error flowgraph for decimation-in-frequency FFT

$$\left.\begin{aligned} B'_{k+1}(p) &= [B'_k(p) + B'_k(q)](1+\gamma'_{k,p}) \\ C'_{k+1}(p) &= [C'_k(p) + C'_k(q)](1+\gamma''_{k,p}) \end{aligned}\right\} \quad \text{if } \ p_k = 0 \tag{13}$$

$$\left.\begin{aligned} & B'_{k+1}(p) = [B'_k(r) - B'_k(p)]\ U_k(p)(1+\delta'_{k,p})(1+\zeta'_{k,p})(1+\lambda'_{k,p}) \\ & - [C'_k(r) - C'_k(p)]\ V_k(p)(1+\delta''_{k,p})(1+\zeta''_{k,p})(1+\lambda'_{k,p}) \\ & C'_{k+1}(p) = [C'_k(r) - C'_k(p)]\ U_k(p)(1+\epsilon'_{k,p})(1+\eta'_{k,p})(1+\lambda''_{k,p}) \\ & + [B'_k(r) - B'_k(p)]\ V_k(p)(1+\epsilon''_{k,p})(1+\eta''_{k,p})(1+\lambda''_{k,p}) \end{aligned}\right\} \quad \text{if } \ p_k = 1$$

and

$$\left.\begin{aligned} B''_{k+1}(p) &= [B''_k(p) + B''_k(q)] + \gamma_{k,p} \\ C''_{k+1}(p) &= [C''_k(p) + C''_k(q)] + \gamma'''_{k,p} \end{aligned}\right\} \quad \text{if } \ p_k = 0 \tag{14}$$

$$\left.\begin{array}{l} B''_{k+1}(p) = [B''_k(r) - B''_k(p) + \delta_{k,p}]\ U_k(p) + \zeta_{k,p} - \\ \quad ([C''_k(r) - C''_k(p) + \delta'''_{k,p}]\ V_k(p) + \zeta'''_{k,p}) + \lambda_{k,p} \\ C''_{k+1}(p) = [C''_k(r) - C''_k(p) + \epsilon_{k,p}]\ U_k(p) + \eta_{k,p} + \\ \quad ([B''_k(r) - B''_k(p) + \epsilon'''_{k,p}]\ V_k(p) + \eta'''_{k,p}) + \lambda'''_{k,p} \end{array}\right\} \quad \text{if } \ p_k = 1$$

The errors $e_k(p)$, $e'_k(p)$, and $e''_k(p)$ defined in equations (6), (7), and (8) are complex and can be rewritten as

$$e_k(p) = B'_k(p) - B_k(p) + j[C'_k(p) - C_k(p)] \tag{15}$$

$$e'_k(p) = B''_k(p) - B_k(p) + j[C''_k(p) - C_k(p)] \tag{16}$$

$$e''_k(p) = B''_k(p) - B'_k(p) + j[C''_k(p) - C'_k(p)] \tag{17}$$

$$k = 1, 2, \ldots, m,\ p = 0, 1, \ldots, N-1$$

with

$$e_0(p) = e'_0(p) = e''_0(p) = 0, \qquad p = 0, 1, \ldots, N-1 \tag{18}$$

From equations (12), (13), (14), (15), (16), and (17), we derive the following error analysis cases:

1. *FFT Real to Floating-Point:*

$$e_{k+1}(p) = \begin{cases} e_k(p) + e_k(q) + f_k(p) & \text{if } \ p_k = 0 \\ [e_k(r) - e_k(p)]\ w_k(p) + f_k(p) & \text{if } \ p_k = 1 \end{cases} \tag{19}$$

where $f_k(p)$ is given by

$$f_k(p) = \begin{cases} \gamma'_{k,p}[B'_k(p) + B'_k(q)] + j\gamma''_{k,p}[C'_k(p) + C'_k(q)] \quad \text{if} \quad p_k = 0 \\ [(1+\delta'_{k,p})(1+\zeta'_{k,p})(1+\lambda'_{k,p}) - 1][B'_k(r) - B'_k(p)]U_k(p) \\ -[(1+\delta''_{k,p})(1+\zeta''_{k,p})(1+\lambda'_{k,p}) - 1][C'_k(r) - C'_k(p)]V_k(p) \\ +j[(1+\epsilon'_{k,p})(1+\eta'_{k,p})(1+\lambda''_{k,p}) - 1][C'_k(r) - C'_k(p)]U_k(p) \\ +j[(1+\epsilon''_{k,p})(1+\eta''_{k,p})(1+\lambda''_{k,p}) - 1][B'_k(r) - B'_k(p)]V_k(p) \\ \hfill \text{if} \quad p_k = 1 \end{cases} \tag{20}$$

2. *FFT Real to Fixed-Point:*

$$e'_{k+1}(p) = \begin{cases} e'_k(p) + e'_k(q) + f'_k(p) & \text{if } \ p_k = 0 \\ [e'_k(r) - e'_k(p)]\ w_k(p) + f'_k(p) & \text{if } \ p_k = 1 \end{cases} \tag{21}$$

where $f'_k(p)$ is given by

$$f'_k(p) = \begin{cases} \gamma_{k,p} + j\gamma'''_{k,p} \quad \text{if } \ p_k = 0 \\ \delta_{k,p}U_k(p) + \zeta_{k,p} - \delta'''_{k,p}V_k(p) - \zeta'''_{k,p} + \lambda_{k,p} + \\ j(\epsilon_{k,p}U_k(p) + \eta_{k,p} + \epsilon'''_{k,p}V_k(p) + \eta'''_{k,p} + \lambda'''_{k,p}) \quad \text{if } \ p_k = 1 \end{cases} \tag{22}$$

3. *FFT Floating- to Fixed-Point:*

$$e''_{k+1}(p) = \begin{cases} e''_k(p) + e''_k(q) + f'_k(p) - f_k(p) & \text{if } \ p_k = 0 \\ [e''_k(r) - e''_k(p)]\ w_k(p) + f'_k(p) - f_k(p) & \text{if } \ p_k = 1 \end{cases} \tag{23}$$

where $f_k(p)$ and $f'_k(p)$ are given by equations (20) and (22).

The accumulation of roundoff error is determined by the recursive equations (19), (20), (21), (22), and (23), with initial conditions given by equation (18). In HOL, we first constructed complex numbers on reals similar to [18]. We defined in HOL a new type for complex numbers, to be in bijection with $\mathbb{R} \times \mathbb{R}$. The bijections are written in HOL as $complex : \mathbb{R}^2 \rightarrow \mathbb{C}$ and $coords : \mathbb{C} \rightarrow \mathbb{R}^2$. We used convenient abbreviations for the real (Re) and imaginary (Im) parts of a complex number, and also defined arithmetic operations such as addition, subtraction, and multiplication on complex numbers. We overloaded the usual symbols $(+, -, \times)$ for $\mathbb{C}$ and $\mathbb{R}$. Similarly, we constructed complex numbers on floating- and fixed-point numbers. Then we defined the principal N-roots on unity ($e^{-j2\pi/N} = cos\ (2\pi n/N) - j\ sin\ (2\pi n/N)$), and its powers ($OMEGA$) as a complex number using the sine and cosine functions available in the transcendental theory of the HOL reals library [15]. We specified expressions in HOL for expansion of a natural number into a binary form in normal and rearranged order according to the equations (2) and (3). The above enables us to specify the FFT algorithms in real ($REAL\_FFT$), floating- ($FLOAT\_FFT$), and fixed-point (FXP_FFT) abstraction levels using recursive definitions in HOL as described in equation (4). Then we defined the real and imaginary parts of the FFT algorithm (FFT_REAL, FFT_IMAGE) and powers of the principal N-roots on unity ($OMEGA\_REAL$,$OMEGA\_IMAGE$) according to the equation (11). Later, we proved in separate lemmas that the real and imaginary parts of the FFT algorithm in real, floating-, and fixed-point levels can be expanded as in equation (12). Then we proved lemmas to introduce an error in each of the arithmetic steps in real and imaginary parts of the floating- and fixed-point FFT algorithms according to the equations (13), and (14). We proved these lemmas using the fundamental error analysis lemmas for basic arithmetic operations [3] according to the equations (9) and (10). Then we defined in HOL the error of the pth element of the floating- ($REAL\_TO\_FLOAT\_FFT\_ERROR$) and fixed-point ($REAL\_TO\_FXP\_FFT\_ERROR$) FFT algorithms at step k, and the corresponding error in transition from floating- to fixed-point ($FLOAT\_TO\_FXP\_FFT\_ERR$ OR), according to the equations (6), (7), and (8). Thereafter, we proved lemmas to rewrite the errors as complex numbers using the real and imaginary parts according to the equations (15), (16), and (17). Finally, we proved lemmas to determine the accumulation of roundoff error in floating- and fixed-point FFT algorithms by recursive equations and initial conditions according to the equations (18), (19), (20), (21), (22), and (23).

4 FFT Design Implementation Verification

In this section, we describe the verification of the transition from fixed-point specification to RTL implementation for FFT algorithms. We have chosen the case study of a radix-4 pipelined 16-point complex FFT core available as a VHDL RTL model in the Xilinx Coregen library [32]. Figure 3 shows the overall block diagram of the design. The basic elements are memories, delays, multiplexers, and dragonflies. In general, the 16-point pipelined FFT requires the calculation of two 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.

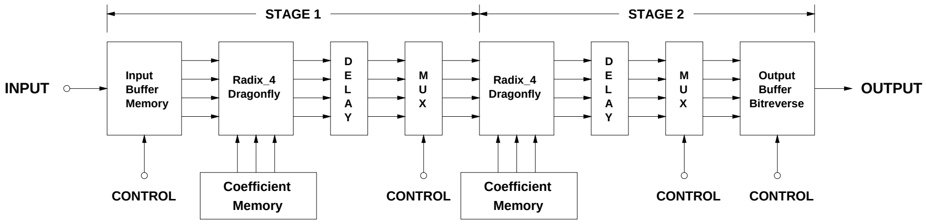

Fig. 3. Radix-4 16-point pipelined FFT implementation

To define the radix-4 FFT algorithm [6, 25], we represent the indices p and n in equation (1) in a base 4 (quaternary number system) as

$$p = 4p_1 + p_0, \qquad p_1, p_0 \;=\; 0, 1, 2, 3 \tag{24}$$

$$n = 4n_1 + n_0, \qquad n_1, n_0 \;=\; 0, 1, 2, 3 \tag{25}$$

It is easy to verify that as n_0 and n_1 take on all possible values in the range indicated, n goes through all possible values from 0 to 15 with no values repeated. This is also true for the frequency index p. Using these index mappings, we can express the radix-4 16-point FFT algorithm recursively as

$$A_1(p_0, n_0) \;=\; \sum_{n_1=0}^{3} x(n_1, n_0)\ (W_{16})^{4p_0 n_1} \tag{26}$$

$$A_2(p_0, p_1) \;=\; \sum_{n_0=0}^{3} A_1(p_0, n_0)\ (W_{16})^{(4p_1+p_0)n_0} \tag{27}$$

The final result can be written as

$$A(p_1, p_0) = A_2(p_0, p_1) \tag{28}$$

Thus, as in the radix-2 algorithm, the results are in reversed order. Based on equations (26), (27), and (28) we can develop a signal flowgraph for the radix-4 16-point FFT algorithm as shown in Figure 4, which is an expanded version of the pipelined implementation of Figure 3. The graph is composed of two successive radix-4 dragonfly stages. To alleviate confusion in this graph we have shown only one of the radix-4 butterflies in the first stage. Also, we have not shown the multipliers for the radix-4 butterflies in the second stage since they are similar to the representative butterfly of the first stage. Figure 4 also illustrates the unscrambling procedure for the radix-4 algorithm.

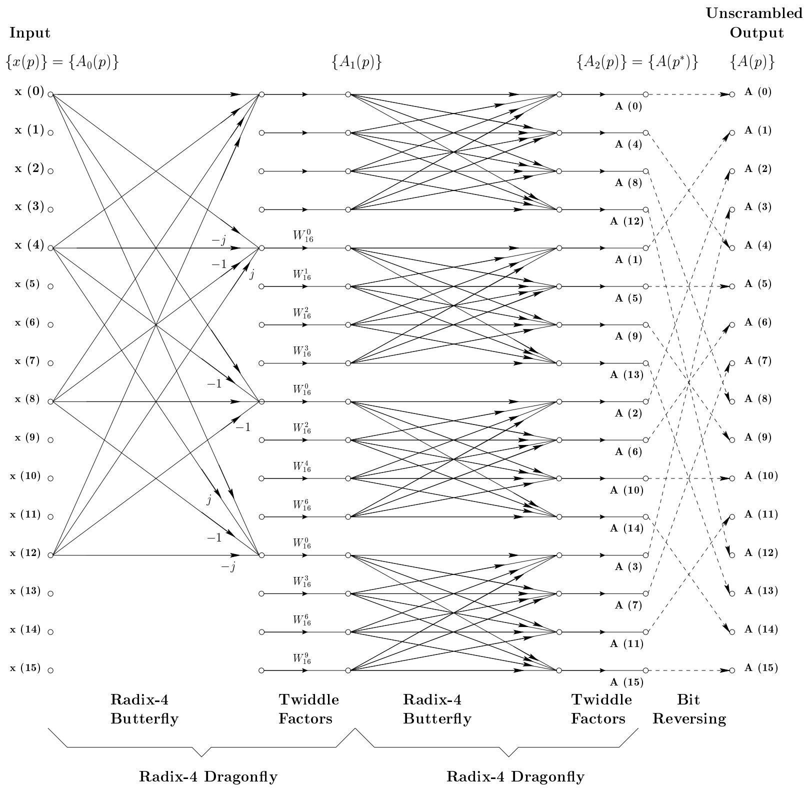

Fig. 4. Signal flowgraph of radix-4 16-point FFT

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 as depicted in the flowgraph of Figure 4, 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 (*DRAGONFLY_RTL*) as a conjunction of a radix-4 butterfly and four 16-bit twiddle factor complex multipliers as shown in Figure 4. Finally, we modeled the complete RTL description of the radix-4 16-point FFT structure (*DIF_FFT_RTL*) in HOL. The FFT block is defined as a conjunction of 8 instantiations of radix-4 dragonfly blocks according to Figure 4, by applying the proper time instances of the input and output signals to each block. Following similar steps, we described a fixed-point radix-4 16-point FFT structure (*DIF_FFT_FXP*) in HOL using complex fixed-point data types and arithmetic operations.

For the formal verification, we first proved that the FFT RTL description implies the corresponding fixed-point model.

$$\begin{array}{ll} \vdash_{thm} & \forall\ N\ X_r\ X_i\ W_r\ W_i\ A_r\ A_i. \\ & DIF\_FFT\_RTL\ N\ X_r\ X_i\ W_r\ W_i\ A_r\ A_i \implies \\ & DIF\_FFT\_FXP\ N\ FXP\_VECT\_COMPLEX\ (X_r, X_i) \\ & \qquad\qquad FXP\_VECT\_COMPLEX\ (W_r, W_i) \\ & \qquad\qquad FXP\_VECT\_COMPLEX\ (A_r, A_i) \end{array} \tag{29}$$

The proof of the FFT block is then broken down into the corresponding proof of the dragonfly block, which itself is broken down to the proof of butterfly and primitive arithmetic operations.

$$\begin{array}{ll} \vdash_{thm} & \forall\ N\ A_r\ A_i\ W_r\ W_i\ Q_r\ Q_i. \\ & DRAGONFLY\_RTL\ N\ A_r\ A_i\ W_r\ W_i\ Q_r\ Q_i \implies \\ & \qquad DRAGONFLY\_FXP\ N\ FXP\ (A_r)\ FXP\ (A_i) \\ & \qquad FXP\ (W_r)\ FXP\ (W_i)\ FXP\ (Q_r)\ FXP\ (Q_i) \end{array} \tag{30}$$

We used the data abstraction functions *FXP* and *FXP_VECT_COMPLEX* to convert a complex vector of 16-bit two's complement Boolean words into the corresponding fixed-point vector. We have also described the radix-4 16-point fixed-point FFT algorithm (*FXP_FFT_ALGORITHM*) using the defining equations (26), (27), and (28). Then we proved that the expanded fixed-point description based on the flowgraph of Figure 4 implies the corresponding closed form fixed-point algorithmic description.

$$\begin{array}{ll} \vdash_{thm} & \forall\ N\ X\ W\ A. \\ & DIF\_FFT\_FXP\ N\ X\ W\ A \implies \\ & \forall\, p.\ A\ (p)\ =\ FXP\_FFT\_ALGORITHM\ N\ X\ W\ p \end{array} \tag{31}$$

In this way we completed the verification of the RTL implementation to fixed-point algorithmic specification of a radix-4 16-point FFT algorithm.

5 Conclusions

In this paper, we described a comprehensive methodology for the verification of generic fast Fourier transform algorithms using the HOL theorem prover. The approach covers 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 implementations compared to the ideal real specification. As a future work, we plan to extend these lemmas to analyse the worst-case, average, and variance errors. Then we proved that the FFT RTL implementation implies the corresponding specification at 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 since the late sixties used theoretical paper-and-pencil proofs and simulation techniques. We believe this is the first time a complete formal framework has been constructed in HOL for the verification of the fast Fourier transform 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.

References

1. B. Akbarpour, S. Tahar, and A. Dekdouk, "Formalization of Cadence SPW Fixed-Point Arithmetic in HOL," In Integrated Formal Methods, LNCS 2335, pp. 185-204, Springer-Verlag, 2002.
2. B. Akbarpour, S. Tahar, "Modeling SystemC Fixed-Point Arithmetic in HOL," In Formal Methods and Software Engineering, LNCS 2885, pp. 206-225, Springer-Verlag, 2003.
3. B. Akbarpour, 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. P. Bjesse, "Automatic Verification of Combinational and Pipelined FFT Circuits," In Computer Aided Verification, LNCS 1633, pp. 380-393, Springer-Verlag, 1999.
5. 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.
6. E. O. Brigham, "The Fast Fourier Transform," Prentice Hall, 1974.
7. V. Capretta, "Certifying the Fast Fourier Transform with Coq," In Theorem Proving in Higher Order Logics, LNCS 2152, pp. 154-168, Springer-Verlag, 2001.
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, C. B. Moler, "Computer Solution of Linear Algebraic Systems," Prentice-Hall, 1967.
12. R. A. Gamboa, "The Correctness of the Fast Fourier Transform: A Structural Proof in ACL2," Formal Methods in System Design, Special Issue on UNITY, Jan. 2002.
13. W. M. Gentleman, 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.
14. M. J. C. Gordon, T. F. Melham, "Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic," Cambridge University Press, 1993.
15. J. R. Harrison, "Constructing the Real Numbers in HOL," Formal Methods in System Design 5 (1/2): 35-59, Kluwer, 1994.
16. 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.
17. J. R. Harrison, "Floating-Point Verification in HOL Light: The Exponential Function," Formal Methods in System Design 16 (3): 271-305, Kluwer, 2000.
18. 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.
19. 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.
20. 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, Feb. 1998.
21. B. Liu, T. Kaneko, "Roundoff Error in Fast Fourier Transforms (Decimation in Time)," Proceedings of the IEEE (Proceedings Letters), 991-992, Jun. 1975.
22. Mathworks, Inc., "Fixed-Point Blockset User's Guide (ver. 2.0)," 1999.
23. T. Melham, "Higher Order Logic and Hardware Verification," Cambridge Tracts in Theoretical Computer Science 31, Cambridge University Press, 1993.
24. J. Misra, "Powerlists: A Structure for Parallel Recursion," In ACM Transactions on Programming Languages and Systems, 16 (6): 1737-1767, Nov. 1994.
25. A. V. Oppenheim, R. W. Schafer, "Discrete-Time Signal Processing," Prentice-Hall, 1989.
26. A. V. Oppenheim, C. J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proceedings of the IEEE, 60 (8): 957-976, Aug. 1972.
27. Cadence Design Systems, Inc., "Signal Processing WorkSystem (SPW) User's Guide," USA, Jul. 1999.
28. T. Thong, B. Liu, "Fixed-Point Fast Fourier Transform Error Analysis," IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP 24 (6): 563-573, Dec. 1976.
29. C. J. Weinstein, "Roundoff Noise in Floating Point Fast Fourier Transform Computation," IEEE Transactions on Audio and Electroacoustics, AU-17 (3): 209-215, Sep. 1969.
30. P. D. Welch, "A Fixed-Point Fast Fourier Transform Error Analysis," IEEE Transactions on Audio and Electroacoustics, AU-17 (2): 151-157, Jun. 1969.
31. J. H. Wilkinson, "Rounding Errors in Algebraic Processes," Prentice-Hall, 1963.
32. Xilinx, Inc., "High-Performance 16-Point Complex FFT/IFFT V1.0.5, Product Specification," USA, Jul. 2000, http://xilinx.com/ipcenter.