An Approach for the Formal Verification of DSP Designs using Theorem Proving

Behzad Akbarpour, *Member, IEEE,* Sofiène Tahar, *Member, IEEE*

***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 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, behavioral, RTL, and netlist gate levels. We make use of existing formalization of floating-point theory in HOL and a parallel one developed 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 on FFT algorithm as case study.

I. 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. 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 [32]. 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

Manuscript received XXXX XX, 200X; revised XXXX XX, 200X. This work is partially supported by NSERC strategic research grant no. STP234820. The experiments were carried out with CAD tools provided by the Canadian Microelectronics Corporation.

B. Akbarpour is with the Department of Electrical and Computer Engineering, Concordia University, Montreal, P.Q., H3G 1M8 Canada (e-mail: behzad@ece.concordia.ca)

S. Tahar is with the Department of Electrical and Computer Engineering, Concordia University, Montreal, P.Q., H3G 1M8 Canada (e-mail: tahar@ece.concordia.ca)

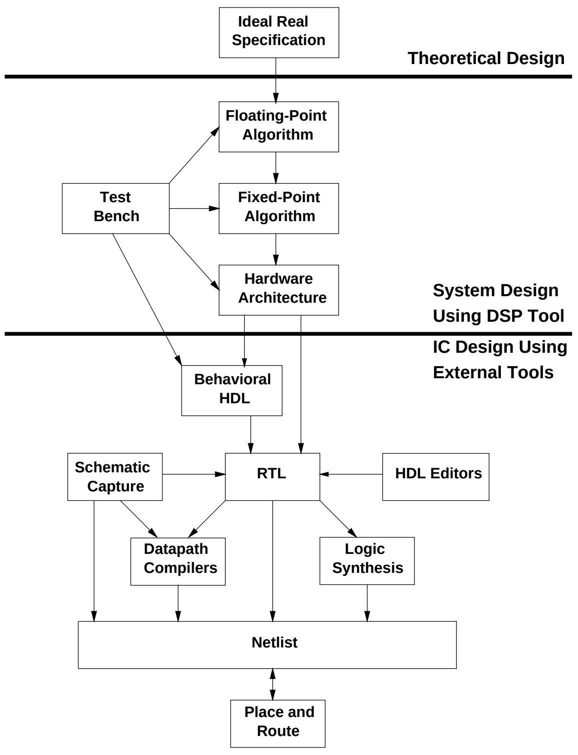

Fig. 1. DSP design flow

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) [10], CoCentric (Synopsys) [11], Matlab-Simulink (Mathworks) [28], and FRIDGE (Aachen UT) [26]. 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. The overall aim of this paper is to propose a general methodology for the formalization and verification of DSP descriptions at different abstraction levels using higher-order logic. To this end, we adopt a shallow embedding for DSP descriptions in which we translate the intended meaning of design blocks into higher-order logic and then complete the formal proof in the HOL [17] theorem proving environment. To our best knowledge, this is the first time formal methods are applied to DSP modeling and verification in such a rigorous way.

The rest of the paper is organized as follows: Section 2 describes the proposed DSP formal verification methodology. Section 3 presents a case study verification of FFT algorithms in HOL from real numbers specification to RTL implementation. Section 4 discusses related work. Finally, Section 5 concludes the paper and outlines future research directions.

II. Proposed DSP Verification Framework

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. 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 [17]. 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 [18] and complex [21] numbers, the formalization of IEEE-754 standard [24] based floating-point arithmetic [19], [20], and the formalization of fixed-point arithmetic [5]. 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 [25], 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 [29]. 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 process of specifying a hardware description language in higher-order logic is commonly known as semantic embedding. There are two main approaches [6]: deep embedding and shallow embedding. In deep embedding, the abstract syntax of a design description is represented by terms, which are then interpreted by semantic functions defined in the logic that assign meaning to the design. With this method, it is possible to reason about classes of designs, since one can quantify over the syntactic structures. However, setting up HOL types of abstract syntax and semantic functions can be very tedious. In a shallow

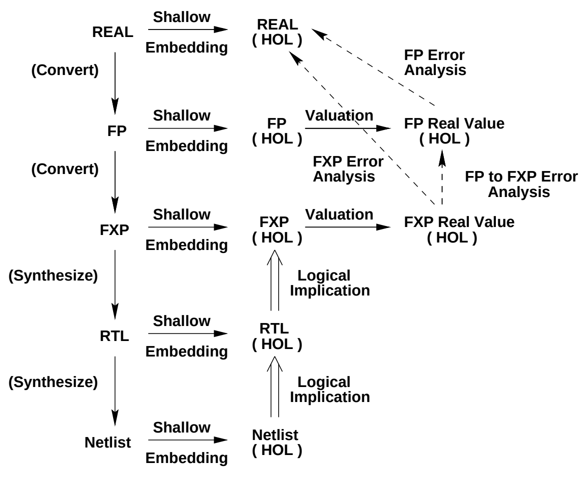

Fig. 2. Proposed DSP specification and verification approach

embedding on the other hand, the design is modeled directly by a formal specification of its functional behavior. This eliminates the effort of defining abstract syntax and semantic functions, but it also limits the proofs to functional properties. In this paper, since our main concern is to check the correctness of the designs based on their functionality, we propose the shallow embedding for DSP descriptions: translate the intended meaning of DSP block designs as described in its documentation into HOL and then complete the formal proof in the HOL theorem prover.

A. Application with SPW

In this section, we demonstrate how the proposed methodology can be used for the verification of an *Integrator* designed in SPW. The Signal Processing WorkSystem (SPW) of Cadence [10] is an integrated framework for developing DSP and communications products. It graphically represents a system as a network of functional blocks and comes with a vast library of DSP blocks and users can also add their own blocks or build IP (Intellectual Property) blocks by composition of primitive blocks. SPW provides all the tools needed to interactively capture, simulate, test, and implement a broad range of DSP designs. Typical design applications include digital communication systems, image processing, multimedia, radar systems, control systems, digital audio, and high-definition television. SPW can be used to evaluate various architectural approaches to a design and to develop, simulate, and fine-tune algorithms. A design project in SPW typically consists of the same steps depicted in Figure 1. More details about SPW design flow and the application of our methodology with it can be found in [4]. To briefly illustrate our approach, we show next the application of our methodology on a simple integrator designed in SPW.

A digital integrator is a discrete time system that transforms a sequence of input numbers into another sequence of output, by means of a specific computational algorithm. To describe the general functionality of a digital integrator, let $\{x_t\}$, $\{w_t\}$, and *a* denote the input sequence, output sequence, and constant coefficient of the integrator, respectively. Then the integrator can be specified by the difference equation:

$$w_t = x_{t-1} + a\ w_{t-1} \tag{1}$$

Thereafter, the output sequence at time *t* is equal to the input sequence at time *t - 1*, added to the output at time *t - 1* multiplied by the integrator coefficient. Figure 3 shows the SPW design of an integrator. The integrator is first designed and simulated using the SPW predefined floating-point blocks and parameters (Figure 3 (a)). The design is composed of an adder (*M1*), a multiplier by constant (*M2*), and a delay (*M3*) block, together with signal source (*M4*) and sink (*M5*) elements. The input signal, the output signal, and the output of the adder and multiplier blocks are labeled by *IN'*, *OUT'*, *S1'*, and *S2'*, respectively. Figure 3 (b) shows the converted fixed-point design in which each block is replaced with the corresponding fixed-point block (*M1'*, *M2'*, *M3'*, *M4'*, *M5'*). Fixed-point blocks are shown by double circles and squares to distinguish them from the floating-point blocks. The attributes of all fixed-point block outputs are set to $(64, 31, t)$ to ensure that overflow and quantization do not affect the system operation. The corresponding fixed-point signals are labeled by *IN"*, *OUT"*, *S1"*, and *S2"*. In HOL, we

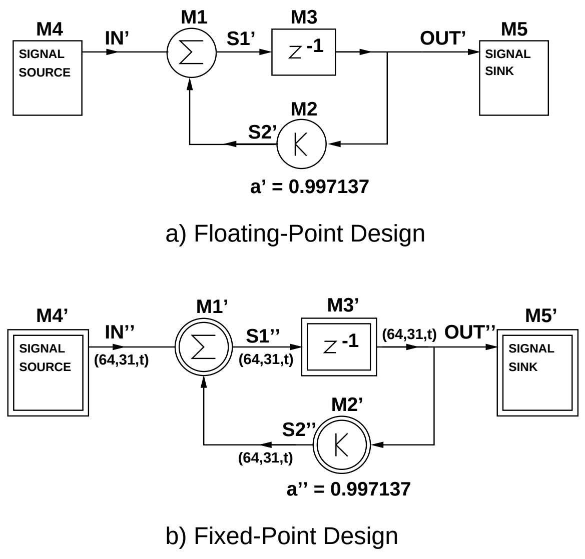

Fig. 3. SPW design of an Integrator

first model the design at each level as predicates in higher-order logic. The predicates corresponding to the floating-point design in IEEE 64 bit double precision format are as follows:

```
⊢def  Float_Gain_Block a′ b′ c′ =
         (∀t. c′ t = a′ t float_mul b′)
⊢def  Float_Delay_Block a′ b′ =
         (∀t. b′ t = a′ (t − 1))
⊢def  Float_Add_Block a′ b′ c′ =
         (∀t. c′ t = a′ t float_add b′ t)

⊢def  Float_Integrator_Imp a′ IN′ OUT′ =
     ∃ S1′ S2′.
     Float_Add_Block IN′ S2′ S1′ ∧
     Float_Delay_Block S1′ OUT′ ∧
     Float_Gain_Block OUT′ a′ S2′
```

The HOL description of the fixed-point implementation is as follows:

```
⊢def  Fxp_Gain_Block a′′ b′′ c′′ =
         (∀t. c′′ t = a′′ t fxp_mul b′′)
⊢def  Fxp_Delay_Block a′′ b′′ =
         (∀t. b′′ t = a′′ (t − 1))
⊢def  Fxp_Add_Block a′′ b′′ c′′ =
         (∀t. c′′ t = a′′ t fxp_add b′′ t)

⊢def  Fxp_Integrator_Imp a′′ IN′′ OUT′′ =
     ∃ S1′′ S2′′.
     Fxp_Add_Block IN′′ S2′′ S1′′ ∧
     Fxp_Delay_Block S1′′ OUT′′ ∧
     Fxp_Gain_Block OUT′′ a′′ S2′′
```

In the next step, we describe each design as a difference equation relating the input and output samples according to Equation (1).

```
⊢def  FLOAT_Integrator_Spec X a′ IN′ OUT′ =
     ∀t. OUT′ t = (IN′ (t − 1) float_add
     (a′ float_mul OUT′ (t − 1)))

⊢def  FXP_Integrator_Spec X′ o_mode q_mode
     n_bits a′′ IN′′ OUT′′ =
     ∀t. OUT′′ t = (IN′′ (t − 1) fxp_add
     (a′′ fxp_mul OUT′′ (t − 1)))
```

The following theorems (Theorems 1 and 2) ensure that the implementation at each level satisfies the corresponding specification.

```
Theorem 1: FLOAT_INTEGRATOR_IMP_TO_SPEC_THM
⊢  Float_Integrator_Imp a′ IN′ OUT′ ⟹
   Float_Integrator_Spec a′ IN′ OUT′

Theorem 2: FXP_INTEGRATOR_IMP_SPEC
⊢  Fxp_Integrator_Imp a′′ IN′′ OUT′′ ⟹
   Fxp_Integrator_Spec a′′ IN′′ OUT′′
```

Now we assume that the floating and fixed-point input sequences are the rounded versions of an infinite precision ideal input *IN*. We also make some other assumptions on finiteness and validity of floating-point and fixed-point inputs, coefficients, and intermediate results, in order to have finite and valid final outputs. Using these assumptions, we proved the following theorem (Theorem 3) concerning the error between the real values of the floating-point and fixed-point precision integrator output samples.

```
Theorem 3: INTEGRATOR_FXP_TO_FLOAT_THM
⊢  Fxp_Integrator_Imp a′′ IN′′ OUT′′
   ⟹
   Float_Integrator_Imp a′ IN′ OUT′ ∧
   Floaterror a′ IN′ OUT′ ∧
   Fxperror a′′ IN′′ OUT′′
```

According to this theorem, for a valid and finite set of input and output sequences at time *(t - 1)* to the integrator design at the floating-point and fixed-point levels, we can have finite and valid outputs at time *t*, and the difference in the real values corresponding to these output samples can be expressed as the difference in input and output values multiplied by the corresponding coefficients, taking into account the effects of finite precision in coefficients and arithmetic operations. The functions `Floaterror` and `Fxperror` represent the errors resulting from rounding the real operation results to a fixed-point and floating-point number, respectively. These errors are already quantified using the theorems mentioned in [5] for fixed-point arithmetic, and the corresponding theorems for error analysis in the floating-point case [20].

Next, we generated with SPW the VHDL code corresponding to the Filter design, and used Synopsys to synthesize the gate level netlist. The resulting codes are then translated into HOL notation and the corresponding correctness theorems established as follows (Theorems 4 and 5):

```
Theorem 4: INTEGRATOR_Netlist_TO_RTL_THM

⊢  Netlist_Integrator_Imp a′′′ IN′′′ OUT′′′
   ⟹  RTL_Integrator_Imp a′′′ IN′′′ OUT′′′

Theorem 5: INTEGRATOR_RTL_TO_FXP_THM

⊢  RTL_Integrator_Imp a′′′ IN′′′ OUT′′′
   ⟹  Fxp_Integrator_Imp Fxp (a′′′)
       Fxp (IN′′′) Fxp (OUT′′′)
```

Here the input and output signals `IN'''` and `OUT'''` are Boolean words. To relate them to the corresponding specifications in fixed- and floating-point, we make use of the bijection functions `Fxp` [5] and `Float` [20], respectively. In the proof of these theorems we used the modular behavior of the circuit, so that we proved separate lemmas for different modules such as adder, multiplier, and delay and then used these lemmas in the proof of the original theorems.

Finally, using the obtained Theorems 1 to 5, we can easily deduce our ultimate theorem (Theorem 6) proving the correctness of the floating-point specification from the gate level implementation, taking into account the error analysis computed beforehand.

Theorem 6: INTEGRATOR_Netlist_TO_FLOAT_THM

```
⊢  Netlist_Integrator_Imp a''' IN''' OUT'''
   ⟹
   Float_Integrator_Spec Float (a''')
   Float (IN''') Float (OUT''') ∧
   Floaterror a''' IN''' OUT''' ∧
   Fxperror a''' IN''' OUT'''
```

More details about this analysis to find the error bounds can be found in [5]. In the rest of the paper, we demonstrate in more detail how the error analysis and verification methodology presented in this section can be used for the verification of the fast Fourier transform (FFT) algorithms implemented in different canonical forms of realization. Similar discussion can be applied to other types of signal analysis algorithms.

III. Case Study: FFT Error Analysis and Verification

The fast Fourier transform (FFT) [8], [12] 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 [31], 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 conventional methods. The first proposed by Cooley and Tukey [13], 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 [16], the sequence of discrete Fourier transform coefficients is decomposed into smaller subsequences, hence its name, decimation-in-frequency (DIF).

As our case study in this paper, we consider the formal verification of the decimation-in-time and decimation-in-frequency 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 [25], we perform a similar analysis using the HOL theorem proving environment. The formal results are found to be in good agreement with the theoretical ones. Finally we prove hierarchically that the FFT RTL implementation implies the high level fixed-point algorithmic specification which has already been related to the floating-point description and ideal real specification through the error analysis.

A. Error Analysis of FFT Algorithms in HOL

In this section, the principal results for accumulation of error in FFT algorithms using HOL theorem proving are derived and summarized. For the most part, the following discussion is phrased in terms of the radix-2 algorithm. However, 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 first analyze the error in Decimation-in-Frequency (DIF) FFT Algorithm. Then, we perform a similar analysis for Decimation-in-Time (DIT) FFT Algorithm. In either cases,we will first describe in detail the theory behind the analysis and then explain how this analysis is performed in HOL.

1) Decimation-in-Frequency (DIF) FFT Algorithm: The discrete Fourier transform of a sequence $\{x(n)\}_{n=0}^{N-1}$ is defined as in [31]

$$\begin{aligned} A(p) &= \textstyle\sum_{n=0}^{N-1} x(n)\ (W_N)^{np}, \\ p &= 0,1,2,\ldots,N-1 \end{aligned} \tag{2}$$

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

$$\begin{aligned} p &= 2^{m-1}p_0 + 2^{m-2}p_1 + \cdots + 2p_{m-2} + p_{m-1}, \\ p_k &= 0 \ \text{ or } \ 1 \end{aligned} \tag{3}$$

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

$$p^* = 2^{m-1}p_{m-1} + 2^{m-2}p_{m-2} + \cdots + 2p_1 + p_0 \tag{4}$$

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 [25] can be expressed as

$$A_{k+1}(p) = \begin{cases} A_k(p) + A_k(p + 2^{m-1-k}) \\ \text{if } \ p_k = 0 \\ [A_k(p - 2^{m-1-k}) - A_k(p)] \ w_k(p) \\ \text{if } \ p_k = 1 \end{cases} \tag{5}$$

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

$$z_k(p) = 2^k \ (2^{m-1-k} p_k + 2^{m-2-k} p_{k+1} + \cdots + 2p_{m-2} + p_{m-1}) - 2^{m-1} p_k \tag{6}$$

Equation (5) is carried out for $k = 0, 1, 2, \ldots, m-1$, with $A_0(p) = x(p)$. It can be shown [16] 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 (3) and (4), respectively. Figure 4 shows the signal flowgraph of the actual computation for the case $N = 2^4$.

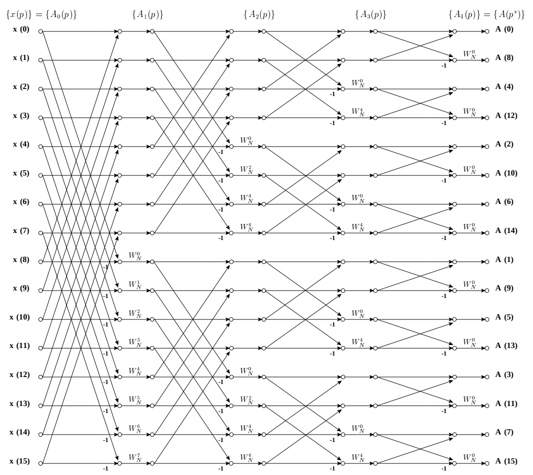

Fig. 4. Signal flowgraph of decimation-in-frequency FFT, $N = 2^4$

There are three common sources of errors associated with the FFT algorithms [25], namely:

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 (5) 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

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

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

$$e''_k(p) = A''_k(p) - A'_k(p) \tag{9}$$

where $e_k(p)$ and $e'_k(p)$ are the errors between the actual floating- and fixed-point implementations and the ideal real specification, respectively. $e''_k(p)$ is the error in the transition from floating- to fixed-point.

In analyzing the effect of floating-point roundoff, the effect of rounding will be represented multiplicatively. Let $*$ denote any of the arithmetic operations +, -, $\times$, /, it is known [14], [36] that, if *p* represents the precision of the floating-point format, then

$$fl \ (x \ * \ y) \ = \ (x \ * \ y)(1 \ + \ \delta), \quad \text{where} \ \ |\delta| \ \le \ 2^{-p} \tag{10}$$

The notation fl (.) is used to denote that the operation is performed using floating-point arithmetic. The above 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

$$fxp \ (x \ * \ y) \ = \ (x \ * \ y) \ + \ \epsilon, \quad \text{where} \ \ |\epsilon| \ \le \ 2^{-fracbits \ (X)} \tag{11}$$

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 (10) and (11) as theorems in higher-order logic within HOL. These 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 actual 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 (5), the $\{A_k(p)\}$ are complex numbers, so their real and imaginary parts are calculated separately. Let

$$\begin{array}{ll} B_k(p) = Re \ [A_k(p)] & C_k(p) = Im \ [A_k(p)] \\ U_k(p) = Re \ [w_k(p)] & V_k(p) = Im \ [w_k(p)] \end{array} \tag{12}$$

where the notations Re [.] and Im [.] denote, respectively, the real and imaginary parts of the quantity inside the bracket [.]. Equation (5) can be rewritten as

$$\left.\begin{aligned} B_{k+1}(p) &= B_k(p) + B_k(q) \\ C_{k+1}(p) &= C_k(p) + C_k(q) \end{aligned}\right\} \tag{13}$$
$$\text{if} \quad p_k = 0$$
$$\left.\begin{aligned} &B_{k+1}(p) = [B_k(r) - B_k(p)]\ U_k(p) - \\ &[C_k(r) - C_k(p)]\ V_k(p) \\ &C_{k+1}(p) = [C_k(r) - C_k(p)]\ U_k(p) + \\ &[B_k(r) - B_k(p)]\ V_k(p) \end{aligned}\right\}$$
$$\text{if} \quad p_k = 1$$

where $q = p + 2^{m-1-k}$ and $r = p - 2^{m-1-k}$. On using the prime, and double prime to denote the calculated floating-point and fixed-point results, the real and imaginary parts of $A'_{k+1}(p)$ and $A''_{k+1}(p)$ are given respectively by

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

$$\left.\begin{aligned} B''_{k+1}(p) &= fxp\ \{B''_k(p) + B''_k(q)\} \\ C''_{k+1}(p) &= fxp\ \{C''_k(p) + C''_k(q)\} \end{aligned}\right\} \tag{15}$$
$$\text{if} \quad p_k = 0$$
$$\left.\begin{aligned} &B''_{k+1}(p) = fxp\ \{[B''_k(r) - B''_k(p)]\ U_k(p) - \\ &[C''_k(r) - C''_k(p)]\ V_k(p)\} \\ &C''_{k+1}(p) = fxp\ \{[C''_k(r) - C''_k(p)]\ U_k(p) + \\ &[B''_k(r) - B''_k(p)]\ V_k(p)\} \end{aligned}\right\}$$
$$\text{if} \quad p_k = 1$$

The corresponding error flowgraph showing the effect of roundoff error using the fundamental floating- and fixed-point error analysis theorems according to Equations (10) and (11), respectively, is given in Figure 5, which also indicates the order of the calculation.

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 5 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, can be given explicitly by

$$\left.\begin{aligned} B'_{k+1}(p) &= [B'_k(p) + B'_k(q)](1 + \gamma'_{k,p}) \\ C'_{k+1}(p) &= [C'_k(p) + C'_k(q)](1 + \gamma''_{k,p}) \end{aligned}\right\}$$
$$\text{if} \quad p_k = 0 \tag{16}$$

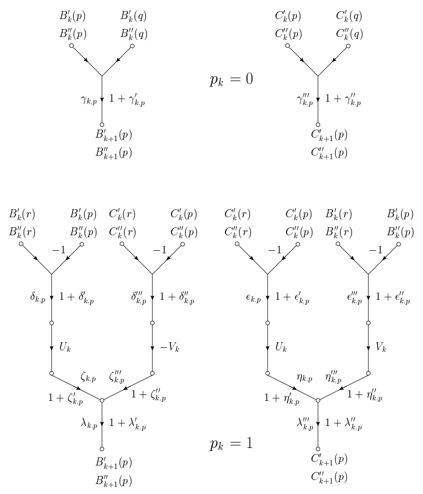

Fig. 5. Error flowgraph for decimation-in-frequency FFT

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

and

$$\left.\begin{aligned} B''_{k+1}(p) &= [B''_k(p) + B''_k(q)] + \gamma_{k,p} \\ C''_{k+1}(p) &= [C''_k(p) + C''_k(q)] + \gamma'''_{k,p} \end{aligned}\right\}$$
$$\text{if} \quad p_k = 0 \tag{17}$$

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

The errors $e_k(p)$, $e'_k(p)$, and $e''_k(p)$ defined in Equations (7), (8), and (9) are complex and can be rewritten as

$$e_k(p) = B'_k(p) - B_k(p) + j[C'_k(p) - C_k(p)] \tag{18}$$
$$e'_k(p) = B''_k(p) - B_k(p) + j[C''_k(p) - C_k(p)] \tag{19}$$
$$e''_k(p) = B''_k(p) - B'_k(p) + j[C''_k(p) - C'_k(p)] \tag{20}$$
$$k = 1, 2, \ldots, m,\ p = 0, 1, \ldots, N-1$$

with

$$e_0(p) = e'_0(p) = e''_0(p) = 0, \qquad p = 0, 1, \ldots, N-1 \tag{21}$$

From Equations (13), (16), (17), (18), (19), and (20), we derive the following error analysis cases:

1) **DIF FFT Real to Floating-Point:**

$$e_{k+1}(p) = \begin{cases} e_k(p) + e_k(q) + f_k(p) \\ \text{ if } \; p_k = 0 \\ [e_k(r) - e_k(p)]\; w_k(p) + f_k(p) \\ \text{if } \; p_k = 1 \end{cases} \tag{22}$$

where $f_k(p)$ is given by

$$f_k(p) = \begin{cases} \gamma'_{k,p}[B'_k(p) + B'_k(q)] + j\gamma''_{k,p}[C'_k(p) + \\ C'_k(q)] \quad \text{if} \quad p_k = 0 \\ [(1+\delta'_{k,p})(1+\zeta'_{k,p})(1+\lambda'_{k,p}) - 1] \\ [B'_k(r) - B'_k(p)]U_k(p) - [(1+\delta''_{k,p}) \\ (1+\zeta''_{k,p})(1+\lambda'_{k,p}) - 1][C'_k(r) - \\ C'_k(p)]V_k(p) + j[(1+\epsilon'_{k,p})(1+\eta'_{k,p}) \\ (1+\lambda''_{k,p}) - 1][C'_k(r) - \; C'_k(p)]U_k(p) \\ +j[(1+\epsilon''_{k,p})(1+\eta''_{k,p})(1+\lambda''_{k,p}) - \\ 1][B'_k(r) - B'_k(p)]V_k(p) \\ \text{if} \quad p_k = 1 \end{cases} \tag{23}$$

2) **DIF FFT Real to Fixed-Point:**

$$e'_{k+1}(p) = \begin{cases} e'_k(p) + e'_k(q) + f'_k(p) \\ \text{if } \; p_k = 0 \\ [e'_k(r) - e'_k(p)]\; w_k(p) + f'_k(p) \\ \text{if } \; p_k = 1 \end{cases} \tag{24}$$

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

$$f'_k(p) = \begin{cases} \gamma_{k,p} + j\gamma'''_{k,p} \quad \text{if } \; p_k = 0 \\ \delta_{k,p}U_k(p) + \zeta_{k,p} - \delta'''_{k,p}V_k(p) - \\ \zeta'''_{k,p} + \lambda_{k,p} + j(\epsilon_{k,p}U_k(p) + \eta_{k,p} + \\ \epsilon'''_{k,p}V_k(p) + \eta'''_{k,p} + \lambda'''_{k,p}) \\ \text{if } \; p_k = 1 \end{cases} \tag{25}$$

3) **DIF FFT Floating- to Fixed-Point:**

$$e''_{k+1}(p) = \begin{cases} e''_k(p) + e''_k(q) + f'_k(p) - f_k(p) \\ \text{if } \; p_k = 0 \\ [e''_k(r) - e''_k(p)]\; w_k(p) + f'_k(p) - \\ f_k(p) \\ \text{if } \; p_k = 1 \end{cases} \tag{26}$$

where $f_k(p)$ and $f'_k(p)$ are given by Equations (23) and (25).

The accumulation of roundoff error is determined by the recursive Equations (22), (23), (24), (25), and (26), with initial conditions given by Equation (21).

2) Decimation-in-Time (DIT) FFT Algorithm: Let $\{A_k(p)\}_{p=0}^{N-1}$ denote the N complex numbers calculated at the kth step. Then the decimation in time (DIT) FFT algorithm [27] can be expressed as

$$A_{k+1}(p) = \begin{cases} A_k(p) + w_k(p)\; A_k(p+2^k) \\ \text{if } \; p_{m-1-k} = 0 \\ A_k(p-2^k) - w_k(p)\; A_k(p) \\ \text{if } \; p_{m-1-k} = 1 \end{cases} \tag{27}$$

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

$$z_k(p) = 2^{m-1-k}\; (2^k p_{m-1-k} + 2^{k-1}p_{m-k} + \cdots + \\ 2p_{m-2} + p_{m-1}) - 2^{m-1}p_{m-1-k} \tag{28}$$

Equation (27) is carried out for $k = 0, 1, 2, \ldots, m-1$, with $A_0(p) = x(p^*)$, where p and p^* are expanded and defined as in Equations (7) and (8), respectively. It can be shown [13] that at the last step, $\{A_m(p)\}_{p=0}^{N-1}$ are the discrete Fourier coefficients in the normal order. Specifically, $A_m(p) = A(p)$. Figure 6 shows the signal flowgraph of the actual computation for the case $N = 2^4$.

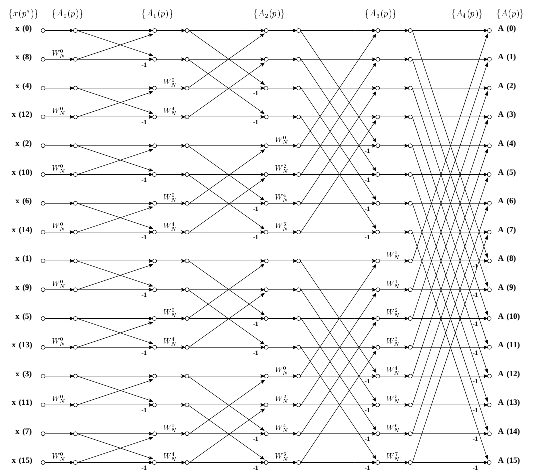

Fig. 6. Signal flowgraph of decimation-in-time FFT, $N = 2^4$

Similar to the discussion of error analysis of decimation-in-frequency (DIF) FFT, we first rewrite Equation (27) using the real and imaginary parts as

$$\left.\begin{array}{l} B_{k+1}(p) = B_k(p) + U_k(p)\; B_k(q) - \\ V_k(p)\; C_k(q) \\ C_{k+1}(p) = C_k(p) + U_k(p)\; C_k(q) + \\ V_k(p)\; B_k(q) \end{array}\right\} \tag{29}$$
$$\text{if } \; p_{m-1-k} = 0$$
$$\left.\begin{array}{l} B_{k+1}(p) = B_k(r) - U_k(p)\; B_k(p) + \\ V_k(p)\; B_k(q) \\ C_{k+1}(p) = C_k(r) - U_k(p)\; C_k(p) - \\ V_k(p)\; B_k(p) \end{array}\right\}$$
$$\text{if } \; p_{m-1-k} = 1$$

where $q = p + 2^k$ and $r = p - 2^k$. We also use the prime, and double prime to denote the calculated floating-point and fixed-point results as $A'_{k+1}(p)$ and

$A''_{k+1}(p)$. 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.

$$\left.\begin{array}{l} B'_{k+1}(p) = fl\ \{B'_k(p) + U_k(p)\ B'_k(q) - \\ V_k(p)\ C'_k(q)\} \\ C'_{k+1}(p) = fl\ \{C'_k(p) + U_k(p)\ C'_k(q) + \\ V_k(p)\ B'_k(q)\} \end{array}\right\} \quad \text{if } p_{m-1-k} = 0$$
$$\left.\begin{array}{l} B'_{k+1}(p) = fl\ \{B'_k(r) - U_k(p)\ B'_k(p) + \\ V_k(p)\ B'_k(q)\} \\ C'_{k+1}(p) = fl\ \{C'_k(r) - U_k(p)\ C'_k(p) - \\ V_k(p)\ B'_k(p)\} \end{array}\right\} \quad \text{if } p_{m-1-k} = 1 \qquad (30)$$

$$\left.\begin{array}{l} B''_{k+1}(p) = fxp\ \{B''_k(p) + U_k(p)\ B''_k(q) - \\ V_k(p)\ C''_k(q)\} \\ C''_{k+1}(p) = fxp\ \{C''_k(p) + U_k(p)\ C''_k(q) + \\ V_k(p)\ B''_k(q)\} \end{array}\right\} \quad \text{if } p_{m-1-k} = 0$$
$$\left.\begin{array}{l} B''_{k+1}(p) = fxp\ \{B''_k(r) - U_k(p)\ B''_k(p) + \\ V_k(p)\ B''_k(q)\} \\ C''_{k+1}(p) = fxp\ \{C''_k(r) - U_k(p)\ C''_k(p) - \\ V_k(p)\ B''_k(p)\} \end{array}\right\} \quad \text{if } p_{m-1-k} = 1 \qquad (31)$$

The corresponding error flowgraph showing the effect of roundoff error using the fundamental floating- and fixed-point error analysis theorems according to the Equations (10) and (11), respectively, is given in Figure 7, which also indicates the order of the calculation.

The quantities $\gamma'_{k,p}$, $\gamma''_{k,p}$, $\delta'_{k,p}$, $\delta''_{k,p}$, $\epsilon'_{k,p}$, $\epsilon''_{k,p}$, $\zeta'_{k,p}$, $\zeta''_{k,p}$, $\eta'_{k,p}$, $\eta''_{k,p}$, $\lambda'_{k,p}$, $\lambda''_{k,p}$, $\alpha'_{k,p}$, $\alpha''_{k,p}$, $\beta'_{k,p}$, and $\beta''_{k,p}$ in Figure 7 are errors caused by floating-point roundoff at each arithmetic step. The corresponding error quantities for fixed-point roundoff are $\gamma_{k,p}$, $\gamma'''_{k,p}$, $\delta_{k,p}$, $\delta'''_{k,p}$, $\epsilon_{k,p}$, $\epsilon'''_{k,p}$, $\zeta_{k,p}$, $\zeta'''_{k,p}$, $\eta_{k,p}$, $\eta'''_{k,p}$, $\lambda_{k,p}$, $\lambda'''_{k,p}$, $\alpha_{k,p}$, $\alpha'''_{k,p}$, $\beta_{k,p}$, and $\beta'''_{k,p}$. Thereafter, the actual real and imaginary parts of the floating- and fixed-point outputs $A'_{k+1}(p)$ and $A''_{k+1}(p)$, respectively can be given explicitly by

$$\left.\begin{array}{l} B'_{k+1}(p) = [[B'_k(q)\ U_k(p)\ (i + \zeta'_{k,p}) - C'_k(q) \\ V_k(p)\ (1 + \zeta''_{k,p})]\ (1 + \delta'_{k,p}) + B'_k(p)]\ (1 + \lambda'_{k,p}) \\ C'_{k+1}(p) = [[C'_k(p)\ U_k(p)\ (1 + \eta'_{k,p}) + B'_k(q) \\ V_k(p)\ (1 + \eta''_{k,p})](1 + \delta''_{k,p}) + C'_k(p)](1 + \lambda''_{k,p}) \end{array}\right\} \quad \text{if } p_{m-1-k} = 0$$
$$\left.\begin{array}{l} B'_{k+1}(p) = [[B'_k(p)\ (-U_k(p))\ (1 + \alpha'_{k,p}) + C'_k(p) \\ V_k(p)\ (1 + \alpha''_{k,p})]\ (1 + \epsilon'_{k,p}) + B'_k(r)]\ (1 + \gamma'_{k,p}) \\ C'_{k+1}(p) = [[C'_k(p)\ (-U_k(p))\ (1 + \beta'_{k,p}) + B'_k(p) \\ (-V_k(p))\ (1 + \beta'_{k,p})](1 + \epsilon''_{k,p}) + C'_k(r)]\ (1 + \gamma''_{k,p}) \end{array}\right\} \quad \text{if } p_{m-1-k} = 1 \qquad (32)$$

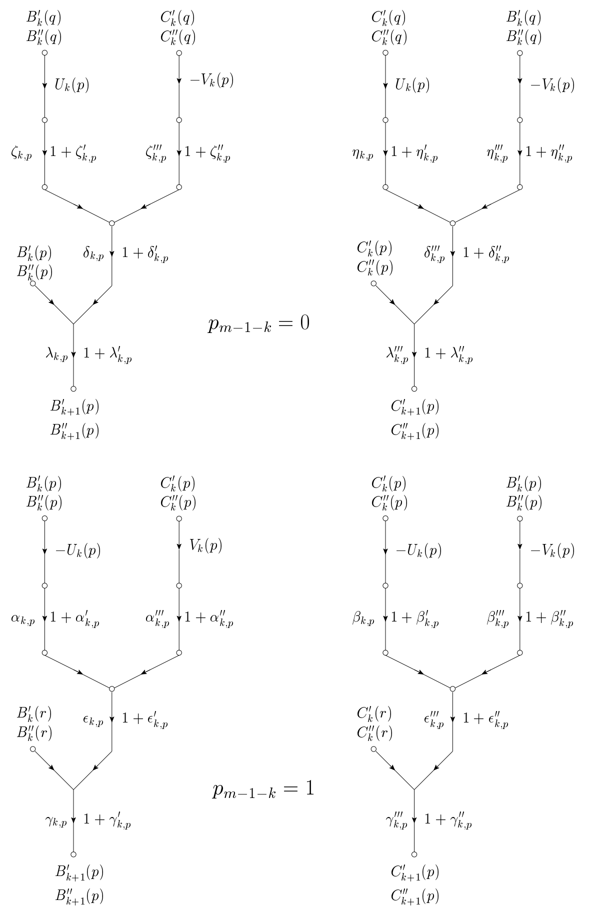
$B'_k(q)$ $B''_k(q)$
$C'_k(q)$ $C''_k(q)$
$U_k(p)$
$-V_k(p)$
$\zeta_{k,p}$ $1+\zeta'_{k,p}$
$\zeta'''_{k,p}$ $1+\zeta''_{k,p}$
$B'_k(p)$ $B''_k(p)$
$\delta_{k,p}$ $1+\delta'_{k,p}$
$\lambda_{k,p}$ $1+\lambda'_{k,p}$
$B'_{k+1}(p)$ $B''_{k+1}(p)$
$p_{m-1-k}=0$
$C'_k(q)$ $C''_k(q)$
$B'_k(q)$ $B''_k(q)$
$\eta_{k,p}$ $1+\eta'_{k,p}$
$\eta'''_{k,p}$ $1+\eta''_{k,p}$
$C'_k(p)$ $C''_k(p)$
$\delta'''_{k,p}$ $1+\delta''_{k,p}$
$\lambda'''_{k,p}$ $1+\lambda''_{k,p}$
$C'_{k+1}(p)$ $C''_{k+1}(p)$
$B'_k(p)$ $B''_k(p)$
$C'_k(p)$ $C''_k(p)$
$-U_k(p)$
$V_k(p)$
$\alpha_{k,p}$ $1+\alpha'_{k,p}$
$\alpha'''_{k,p}$ $1+\alpha''_{k,p}$
$B'_k(r)$ $B''_k(r)$
$\epsilon_{k,p}$ $1+\epsilon'_{k,p}$
$\gamma_{k,p}$ $1+\gamma'_{k,p}$
$B'_{k+1}(p)$ $B''_{k+1}(p)$
$p_{m-1-k}=1$
$C'_k(p)$ $C''_k(p)$
$B'_k(p)$ $B''_k(p)$
$-U_k(p)$
$-V_k(p)$
$\beta_{k,p}$ $1+\beta'_{k,p}$
$\beta'''_{k,p}$ $1+\beta''_{k,p}$
$C'_k(r)$ $C''_k(r)$
$\epsilon'''_{k,p}$ $1+\epsilon''_{k,p}$
$\gamma'''_{k,p}$ $1+\gamma''_{k,p}$
$C'_{k+1}(p)$ $C''_{k+1}(p)$

Fig. 7. Error flowgraph for decimation-in-time FFT

and

$$\left.\begin{array}{l} B''_{k+1}(p) = B''_k(q)\ U_k(p) - C''_k(q)\ V_k(p) + \\ B''_k(p) + \zeta_{k,p} + \zeta'''_{k,p} + \delta_{k,p} + \lambda_{k,p} \\ C''_{k+1}(p) = C''_k(q)\ U_k(p) + B''_k(q)\ V_k(p) + \\ C''_k(p) + \eta_{k,p} + \eta'''_{k,p} + \delta'''_{k,p} + \lambda'''_{k,p} \end{array}\right\} \quad \text{if } p_{m-1-k} = 0$$
$$\left.\begin{array}{l} B''_{k+1}(p) = B''_k(p)\ (-U_k(p)) + C''(p)\ V_k(p) + \\ B''_k(r) + \alpha_{k,p} + \alpha'''_{k,p} + \epsilon_{k,p} + \delta_{k,p} \\ C''_{k+1}(p) = C''_k(p)\ (-U_k(p)) + B''_k(p) \\ (-V_k(p)) + C''_k(r) + \beta_{k,p} + \beta'''_{k,p} + \epsilon'''_{k,p} + \gamma'''_{k,p} \end{array}\right\} \quad \text{if } p_{m-1-k} = 1 \qquad (33)$$

From Equations (29), (32), and (33), we derive the following error analysis cases:

1) **DIT FFT Real to Floating-Point:**

$$e_{k+1}(p) = \left\{ \begin{array}{l} e_k(p) + e_k(q)\ w_k(p) + f_k(p) \\ \ \text{if } p_{m-1-k} = 0 \\ e_k(r) - e_k(p)\ w_k(p) + f_k(p) \\ \text{if } p_{m-1-k} = 1 \end{array}\right. \qquad (34)$$

where $f_k(p)$ is given by

$$f_k(p) = \begin{cases} B'_k(p)\ \lambda'_{k,p} + B'_k(q)\ U_k(p)\ [(1+\zeta'_{k,p})\ (1+ \\ \delta'_{k,p})\ (1+\lambda'_{k,p}) - 1] - C'_k(q)\ V_k(p)\ [(1+ \\ \zeta''_{k,p})\ (1+\delta_{k,p}\ (1+\lambda'_{k,p}) - 1] + j[C'_k(p) \\ \lambda'_{k,p} + C'_k(q)\ U_k(p)\ [(1+\zeta'_{k,p})\ (1+\delta''_{k,p}) \\ (1+\lambda''_{k,p}) - 1] + B'_k(q)\ V_k(p)\ [(1+\zeta''_{k,p}) \\ (1+\delta''_{k,p})(1+\lambda''_{k,p}) - 1] \\ \text{if} \quad p_{m-1-k} = 0 \\ B'_k(r)\ \gamma'_{k,p} - B'_k(p)\ U_k(p)\ [(1+\alpha'_{k,p}\ (1+ \\ \epsilon'_{k,p})\ (1+\gamma'_{k,p}) - 1] - C'_k(p)\ V_k(p)\ [(1+ \\ \alpha''_{k,p})\ (1+\epsilon'_{k,p}\ (1+\gamma'_{k,p}) - 1] + j\ [C'_k(r) \\ \gamma''_{k,p} - C'_k(p)\ U_k(p)\ [(1+\beta'_{k,p})\ (1+\epsilon''_{k,p}) \\ (1+\gamma''_{k,p}) - 1] +\ B'_k(q)\ V_k(p)\ [(1+\zeta''_{k,p}) \\ (1+\gamma''_{k,p})\ (1+\lambda''_{k,p}) - 1] \\ \text{if} \quad p_{m-1-k} = 1 \end{cases} \tag{35}$$

2) **DIT FFT Real to Fixed-Point:**

$$e'_{k+1}(p) = \begin{cases} e'_k(p) + e'_k(q)\ w_k(p) + f'_k(p) \\ \text{if} \quad p_{m-1-k} = 0 \\ e'_k(r) - e'_k(p)\ w_k(p) + f'_k(p) \\ \text{if} \quad p_{m-1-k} = 1 \end{cases} \tag{36}$$

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

$$f'_k(p) = \begin{cases} \zeta_{k,p} + \zeta'''_{k,p} + \delta_{k,p} + \lambda_{k,p} + \\ j\ (\eta_{k,p} + \eta'''_{k,p} + \delta'''_{k,p} + \lambda'''_{k,p}) \\ \text{if} \quad p_{m-1-k} = 0 \\ \alpha_{k,p} + \alpha'''_{k,p} + \epsilon_{k,p} + \gamma_{k,p} + \\ j\ (\beta_{k,p} + \beta'''_{k,p} + \epsilon'''_{k,p} + \gamma'''_{k,p}) \\ \text{if} \quad p_{m-1-k} = 1 \end{cases} \tag{37}$$

3) **DIT FFT Floating- to Fixed-Point:**

$$e''_{k+1}(p) = \begin{cases} e''_k(p) + e''_k(q)\ w_k(p) + f'_k(p) - \\ f_k(p) \qquad \text{if} \quad p_{m-1-k} = 0 \\ e''_k(r) - e''_k(p)\ w_k(p) + f'_k(p) - \\ f_k(p)\ \text{if} \quad p_{m-1-k} = 1 \end{cases} \tag{38}$$

where $f_k(p)$ and $f'_k(p)$ are given by Equations (35) and (37), respectively.

The accumulation of roundoff error is determined by the recursive Equations (34), (35), (36), (37), and (38), with initial conditions given by Equation (21).

3) Effects of Input Quantization and Coefficient Inaccuracy: The discussion presented in previous sections concerns only the round-off accumulation effect. As mentioned before, there are two other common causes of error due to the finite word length in computing the Fourier coefficients. They are the quantization of the input data $x(n)$ and the inaccuracy of the coefficients $w_k(p)$. The effect of the quantization of $x(n)$ can be treated as follows. Let $x'(n)$ and $x''(n)$ be the floating- and fixed-point quantized versions of $x(n)$, respectively. Then from the discussion in Section III-A.1 we can write

$$Re[x'(n)] = (1+\theta_n)\ Re[x(n)], \\ Im[x'(n)] = (1+\xi_n)\ Im[x(n)] \tag{39}$$

$$Re[x''(n)] = Re[x(n)] + \theta'_n, \\ Im[x''(n)] = Im[x(n)] + \xi'_n \tag{40}$$

where θ_n and ξ_n are the errors caused by floating-point quantization, and θ'_n and ξ'_n are the errors caused by fixed-point quantization in the input signal. The effect of Equations (39) and (40) modifies the initial conditions as described in Equation (21) to

$$e_0(n) = \theta_n\ Re[x(n)] + j\ \xi_n\ Im[x(n)] \tag{41}$$

$$e'_0(n) = \theta'_n + j\ \xi'_n \tag{42}$$

$$e''_0(n) = e'_0(n) - e_0(n) \tag{43}$$

It can be shown that with these modifications, the final results of the mean square errors remain the same except for an addition term which is independent of *p* [25].

Another cause for error that has been neglected in the treatment of the previous sections is the fact that the coefficients $w_k(p)$ can only be represented in finite accuracy. It is possible to analyze the effect of the inaccuracy of $w_k(p)$ as follows. Let $U_k(p)$ and $V_k(p)$ be the real and imaginary parts of $w_k(p)$ as defined in Equation (13), respectively. Also, let $U'_k(p)$ and $U''_k(p)$ be the floating- and fixed-point quantized version of $U_k(p)$, and $V'_k(p)$ and $V''_k(p)$ be the floating- and fixed-point quantized version of $V_k(p)$, respectively. Then from the discussion in Section III-A.1 we can write

$$U'_k(p) = (1+\varphi_{k,p})\ U_k(p), \qquad V'_k(p) = (1+\psi_{k,p})\ V_k(p) \tag{44}$$

$$U''_k(p) = U_k(p) + \varphi'_{k,p}, \qquad V''_k(p) = V_k(p) + \psi'_{k,p} \tag{45}$$

where $\varphi_{k,p}$ and $\psi_{k,p}$ are the errors caused by floating-point quantizations, and $\varphi'_{k,p}$ and $\psi'_{k,p}$ are the errors caused by fixed-point quantizations in the coefficients. One may now proceed with the analysis of Sections III-A.1 and III-A.2 by adding the factors $(1+\varphi_{k,p})$, $(1+\psi_{k,p})$, $\varphi'_{k,p}$, and $\psi'_{k,p}$ in appropriate places in Equations (23), (25), (35), and (37).

4) Error Analysis in HOL: In HOL, we first constructed complex numbers on reals similar to [21]. 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. We also defined arithmetic operations such as addition, subtraction, and multiplication on complex numbers. We overloaded the usual symbols $(+, -, \times)$ for $\mathbb{C}$ and $\mathbb{R}$. Furthermore, we defined, using recursive definition in HOL, expressions for the finite summation on complex numbers. Similarly, we constructed complex numbers on floating-point and fixed-point variables. We also defined rounding and valuation functions for floating-point and fixed-point complex numbers. Then we defined the principal N-roots on unity ($e^{-j2\pi n/N} = cos\ (2\pi n/N) - j\ sin\ (2\pi n/N)$), and its powers as a complex number using the sine and cosine functions available in the transcendental theory of the HOL reals library [18]. We specified expressions in HOL for expansion of a natural number into a binary form in normal and rearranged order according to Equations (3), (4), (6), and (28). The above enables us to specify the FFT algorithms in real, floating-, and fixed-point abstraction levels using recursive definitions in HOL as described in Equations (5) and (27). Then we define the real and imaginary parts of the FFT algorithm, and powers of the principal N-roots on unity according to the Equation (12). Later, we proved in separate lemmas that the real and imaginary parts of the FFT algorithm in real, floating-point, and fixed-point levels can be expanded as in Equations (13) and (29). Then we proved lemmas to introduce an error in each of the arithmetic steps in real and imaginary parts of the floating-point and fixed-point FFT algorithms according to the Equations (16), (17), (32), and (33). We proved these lemmas using the fundamental error analysis lemmas for basic arithmetic operations [3] according to the Equations (10) and (11). Then we defined in HOL the error of the *p*th element of the floating- and fixed-point FFT algorithms at step *k*, and the corresponding error in transition from floating- to fixed-point, according to the Equations (7), (8), and (9). Thereafter, we proved lemmas to rewrite the errors as complex numbers using the real and imaginary parts according to Equations (18), (19), and (20). Finally, we proved a set of lemmas to determine the accumulation of roundoff error in floating- and fixed-point FFT algorithms by recursive equations and initial conditions according to Equations (21), (22), (23), (24), (25), and (26) for DIF, and (34), (35), (36), (37), and (38) for DIT FFT. A complete list of the derived HOL definitions and theorems can be found in [1].

B. Radix-4 64-Point FFT Design 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 [37]. 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 8 shows the overall block diagram of the Radix-4 64-point pipelined FFT design. 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.

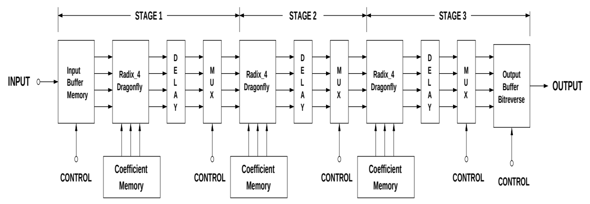

Fig. 8. Radix-4 64-point pipelined FFT implementation

To define the radix-4 64-point FFT algorithm [8], [31], we represent the indices *p* and *n* in Equation (2) in a base 4 (quaternary number system) as

$$p = 16p_2 + 4p_1 + p_0, \qquad p_2, p_1, p_0 = 0, 1, 2, 3 \quad (46)$$

$$n = 16n_2 + 4n_1 + n_0, \qquad n_2, n_1, n_0 = 0, 1, 2, 3 \quad (47)$$

It is easy to verify that as n_0, n_1, and n_2 take on all possible values in the range indicated, n goes through all possible values from 0 to 63 with no values repeated. This is also true for the frequency index p. Using these index mappings, we can express the radix-4 64-point FFT algorithm recursively as

$$A_1(p_0, n_1, n_0) = \sum_{n_2=0}^{3} x(n_2, n_1, n_0)\ (W_{64})^{16p_0 n_2} \quad (48)$$

$$A_2(p_0, p_1, n_0) = \sum_{n_1=0}^{3} A_1(p_0, n_1, n_0)\ (W_{64})^{(4p_1+p_0)4n_1} \quad (49)$$

$$A_3(p_0, p_1, p_2) = \sum_{n_0=0}^{3} A_2(p_0, p_1, n_0)\ (W_{64})^{(16p_2+4p_1+p_0)n_0} \quad (50)$$

The final result can be written as

$$A(p_2, p_1, p_0) = A_3(p_0, p_1, p_2) \quad (51)$$

Thus, as in the radix-2 algorithm, the results are in reversed order. Based on Equations (48), (49), (50), and (51), we can develop a signal flowgraph for the radix-4 64-point FFT algorithm as shown in Figure 9, which is an expanded version of the pipelined implementation of Figure 8. The graph is composed of three successive radix-4 dragonfly stages, with each stage comprising 16 dragonflies.

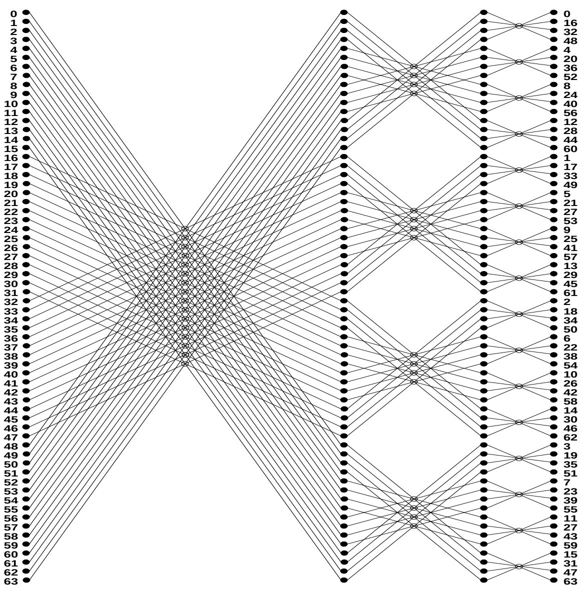

Fig. 9. Signal flowgraph of radix-4 64-point FFT

From Equations (48), (49), (50), and (51) we can express the input-output relationship of radix-4 64-point FFT as follows

$$A(p_2,p_1,p_0) = \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} x(n_2,n_1,n_0) (W_{64})^{16p_0n_2+4(4p_1+p_0)n_1+(16p_2+4p_1+p_0)n_0} \tag{52}$$

Equation (52) can be rewritten using the real and imaginary parts as follows

$$\begin{aligned}
B(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} \\
&B_0(n_2,n_1,n_0)\ U_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0) - C_0(n_2,n_1,n_0)\ V_{64} \\
&(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0) \\
C(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} \\
&C_0(n_2,n_1,n_0)\ U_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0) + B_0(n_2,n_1,n_0)\ V_{64} \\
&(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0)
\end{aligned} \tag{53}$$

The corresponding error flowgraph is given in Figure 10. Therefore, the actual real and imaginary parts of the

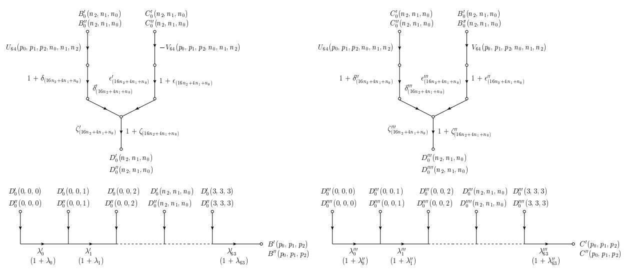

Fig. 10. Error flowgraph for radix-4 64-point FFT

floating- and fixed-point outputs can be given as follows

$$\begin{aligned}
B'(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} \\
&(B'_0(n_2,n_1,n_0)\ U_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0)\ (1+\delta_{(16n_2+4n_1+n_0)}) - \\
&C'_0(n_2,n_1,n_0)\ V_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0)\ (1+\epsilon_{(16n_2+4n_1+n_0)})) \\
&(1+\zeta_{(16n_2+4n_1+n_0)}) \prod_{i=16n_2+4n_1+n_0}^{63} (1+\lambda_i)
\end{aligned} \tag{54}$$

$$\begin{aligned}
C'(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} \\
&(C'_0(n_2,n_1,n_0)\ U_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0)\ (1+\delta''_{(16n_2+4n_1+n_0)}) + \\
&B'_0(n_2,n_1,n_0)\ V_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0)\ (1+\epsilon''_{(16n_2+4n_1+n_0)})) \\
&(1+\zeta''_{(16n_2+4n_1+n_0)}) \prod_{i=16n_2+4n_1+n_0}^{63} (1+\lambda''_i)
\end{aligned} \tag{55}$$

$$\begin{aligned}
B''(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} \\
&B''_0(n_2,n_1,n_0)\ U_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0) - C''_0(n_2,n_1,n_0) \\
&V_{64}(16p_0n_2+4(4p_1+p_0)n_1+(16p_2+4p_1+ \\
&p_0)n_0)\ + \delta'_{(16n_2+4n_1+n_0)} + \epsilon'_{(16n_2+4n_1+n_0)} + \\
&\zeta'_{(16n_2+4n_1+n_0)} + \sum_{i=16n_2+4n_1+n_0}^{63} \lambda'_i
\end{aligned} \tag{56}$$

$$\begin{aligned}
C''(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3} \\
&C''_0(n_2,n_1,n_0)\ U_{64}(16p_0n_2+4(4p_1+p_0)n_1+ \\
&(16p_2+4p_1+p_0)n_0) + B''_0(n_2,n_1,n_0)\ V_{64}(16p_0n_2 \\
&+4(4p_1+p_0)n_1+(16p_2+4p_1+p_0)n_0)\ + \\
&\delta'''_{(16n_2+4n_1+n_0)} + \epsilon'''_{(16n_2+4n_1+n_0)} + \zeta'''_{(16n_2+4n_1+n_0)} + \\
&\sum_{i=16n_2+4n_1+n_0}^{63} \lambda'''_i
\end{aligned} \tag{57}$$

From Equations (53), (54), (55), (56), and (57), we derive the following error analysis cases:

1) **Radix-4 64-Point FFT Real to Floating-Point:**

$$\begin{aligned}
e(p_2,p_1,p_0) &= \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3}(e_0(n_2,n_1,n_0) \\
&(W_{64})^{(16p_0n_2+4(4p_1+p_0)n_1+(16p_2+4p_1+p_0)n_0)}) + \\
&f(p_0,p_1,p_2)
\end{aligned} \tag{58}$$

2) **Radix-4 64-Point FFT Real to Fixed-Point:**

$$e'(p_2, p_1, p_0) \;=\; \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3}(e_0'(n_2, n_1, n_0) \\ (W_{64})^{(16p_0n_2+4(4p_1+p_0)n_1+(16p_2+4p_1+p_0)n_0)})+ \\ f'(p_0, p_1, p_2) \tag{59}$$

3) **Radix-4 64-Point FFT Floating- to Fixed-Point:**

$$e''(p_2, p_1, p_0) \;=\; \sum_{n_2=0}^{3}\sum_{n_1=0}^{3}\sum_{n_0=0}^{3}(e_0''(n_2, n_1, n_0) \\ (W_{64})^{(16p_0n_2+4(4p_1+p_0)n_1+(16p_2+4p_1+p_0)n_0)})+ \\ f'(p_0, p_1, p_2) - f(p_0, p_1, p_2) \tag{60}$$

where *f* and *f'* are the error functions that can be derived based on Figure 10.

1) Verification in HOL: 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, according to Figure 9.

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 described in Section II-A 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, as discussed in Section 3. 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].

IV. Related Work

Work related to our project can be classified in three groups. Namely, using formal methods for error analysis, paper-and-pencil error analysis of FFT algorithms and formal verification of FFT designs.

A. Error Analysis in Formal Verification

Previous work on the error analysis in formal verification was done by Harrison [20] who verified the floating-point algorithms such as the exponential function against their abstract mathematical counterparts using the HOL Light theorem prover. As the main theorem, he proved that the floating-point exponential function has a correct overflow behavior, and in the absence of overflow the error in the result is bounded to a certain amount. He also reported on an error in the hand proof mostly related to forgetting some special cases in the analysis. This error analysis is very similar to the type of analysis performed for DSP algorithms. The major difference, however, is the use of statistical methods and mean square error analysis for DSP algorithms which is not covered in the error analysis of the mathematical functions used by Harrison. In this method, the error quantities are treated as independent random variables uniformly distributed over a specific interval depending on the type of arithmetic and the rounding mode. Then the error analysis is performed to derive expressions for the variance and mean square error. To perform such an analysis in HOL, we need to develop a mechanized theory on the properties of random variables and random processes. This type of analysis is not addressed in this paper and is a part of our future work.

Huhn *et al.* [23] proposed a hybrid formal verification method combining different state-of-the-art techniques to guide the complete design flow of imprecisely working arithmetic circuits starting at the algorithmic down to the register transfer level. The usefulness of the method is illustrated with the example of the discrete cosine transform algorithms. In particular, the authors have shown the use of computer algebra systems like Mathematica or Maple at the algorithmic level to reason about real numbers and to determine certain error bounds for the results of numerical operations. In contrast to [23], we proposed an error analysis for DSP designs using the HOL theorem prover. Although computer algebraic systems such as Maple or Mathematica are much more popular and have many powerful decision procedures and heuristics, theorem provers are more expressive, more precise, and more reliable [22]. One option is to combine the rigour of the theorem provers with the power of computer algebraic systems as proposed in [22].

B. *Error Analysis of FFT Algorithms*

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 [35] presented an analysis of the fixed-point accuracy of the radix-2 decimation-in-time FFT algorithm. Tran-Thong and Liu [33] 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 [16], who presented an upper bound on the mean-squared error for floating-point decimation-in-frequency FFT algorithm. Weinstein [34] presented a statistical model for roundoff errors of the floating-point FFT. Kaneko and Liu [25] 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 [27]. Oppenheim and Weinstein [32] 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 showed 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 was on the process of translating the hand proofs done in the sixties and seventies into equivalent proofs in HOL. The analysis we developed is mainly inspired by the work done by Kaneko and Liu [25], 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.

C. *Formalization and Verification of FFT Algorithms*

Related work on the formalization and mechanical verification of the FFT algorithm was done by Gamboa [15] 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 [30] into a mechanical proof in ACL2. In the same line, Capretta [9] 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). In another related work, Bjesse [7] 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 [15] and [9] prove the correctness of the high level FFT algorithm against the DFT, the verification of [7] is performed at the netlist level. In contrast, our work tried 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 [15], [9] 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 proposed 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.

V. 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 study of the fast Fourier transform algorithms. 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 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 and S. Tahar, “The Application of Formal Verification to SPW Designs,” In Proceedings Euromicro Symposium on Digital System Design, IEEE Computer Society Press, pp. 325 -332, Belek, Turkey, September 2003.

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

[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] P. Bjesse, “Automatic Verification of Combinational and Pipelined FFT Circuits,” In Computer Aided Verification, LNCS 1633, pp. 380-393, Springer-Verlag, 1999.

[8] E. O. Brigham, “The Fast Fourier Transform,” Prentice Hall, 1974.

[9] V. Capretta, “Certifying the Fast Fourier Transform with Coq,” In Theorem Proving in Higher Order Logics, LNCS 2152, pp. 154-168, Springer-Verlag, 2001.

[10] Cadence Design Systems, Inc., “Signal Processing WorkSystem (SPW) User’s Guide,” USA, July 1999.

[11] Synopsys, Inc., “CoCentric^{TM} System Studio User’s Guide,” USA, Aug. 2001.

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

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

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

[15] 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.

[16] 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.

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

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

[19] 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.

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

[21] 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.

[22] J. R. Harrison and L. Théry, “A Skeptic’s Approach to Combining Hol and Maple,” Journal of Automated Reasoning, 21: 279-294, 1998.

[23] M. Huhn, K. Schneider, T. Kropf, and G. Logothetis, “Verifying Imprecisely Working Arithmetic Circuits,” In Proceedings Design Automation and Test in Europe Conference, pp. 65-69, Munich, Germany, March 1999.

[24] 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.

[25] 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.

[26] 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.

[27] B. Liu and T. Kaneko, “Roundoff Error in Fast Fourier Transforms (Decimation in Time),” Proceedings of the IEEE (Proceedings Letters), 991-992, Jun. 1975.

[28] Mathworks, Inc., “Simulink Reference Manual,” USA, 1996.

[29] T. Melham, “Higher Order Logic and Hardware Verification,” Cambridge Tracts in Theoretical Computer Science 31, Cambridge University Press, 1993.
[30] J. Misra, “Powerlists: A Structure for Parallel Recursion,” In ACM Transactions on Programming Languages and Systems, 16 (6): 1737-1767, Nov. 1994.
[31] A. V. Oppenheim and R. W. Schafer, “Discrete-Time Signal Processing,” Prentice-Hall, 1989.
[32] 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.
[33] T. Thong and B. Liu, “Fixed-Point Fast Fourier Transform Error Analysis,” IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP 24 (6): 563-573, Dec. 1976.
[34] 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.
[35] P. D. Welch, “A Fixed-Point Fast Fourier Transform Error Analysis,” IEEE Transactions on Audio and Electroacoustics, AU-17 (2): 151-157, Jun. 1969.
[36] J. H. Wilkinson, “Rounding Errors in Algebraic Processes,” Prentice-Hall, 1963.
[37] Xilinx, Inc., “High-Performance 64-Point Complex FFT/IFFT V2.0, Product Specification,” USA, Aug. 2000, http://xilinx.com/ipcenter.