Error Analysis and Verification of an IEEE 802.11 OFDM Modem using Theorem Proving [1]

Abu Nasser Mohammed Abdullah[a,3], Behzad Akbarpour[b,4] and Sofiène Tahar[c,5]

[a] *Cadence Design Systems, Chelmsford, Massachusetts, USA* [2]

[b] *Computer Laboratory, University of Cambridge, England*

[c] *Electrical and Computer Engineering Department, Concordia University, Montreal, Quebec, Canada*

Abstract

IEEE 802.11 is a widely used technology which powers many of the digital wireless communication revolutions currently taking place. It uses OFDM (Orthogonal Frequency Division Multiplexing) in its physical layer which is an efficient way to deal with multipath, good for relatively slow time-varying channels, and robust against narrowband interference. In this paper, we formally specify and verify an implementation of the IEEE 802.11 standard physical layer based OFDM modem using the HOL (Higher Order Logic) theorem prover. The versatile expressive power of HOL helped us model the original design at all abstraction levels starting from a floating-point model to the fixed-point design and then synthesized and implemented in FPGA technology. We have been able to find a bug in one of the blocks of the design that is responsible for modulation which implementation diverts from the constellation provided in the IEEE standard specification. The paper also derives new expressions for the rounding error accumulated during ideal real to floating-point and fixed-point transitions at the algorithmic level and performs a formal error analysis for the OFDM modem in HOL.

Keywords: Formal Verification, Theorem Proving, Error Analysis, OFDM, Wireless Communication.

1 Introduction

IEEE 802.11 [19] refers to a family of IEEE standards about local area and metropolitan area wireless networks. The services and protocols specified in IEEE 802.11 map to the lower two layers, namely Data Link layer (DLL) and Physical layer (PHY) of the seven-layer OSI (Open Systems Interconnection) networking reference model. DLL consists of two sub-layers named Logical Link Control (LLC) and Media Access Control (MAC). The PHY of IEEE 802.11 is based on orthogonal frequency division multiplexing (OFDM) [28],

[1] A two pages abstract preliminary version of this work has been published as a "short paper" at FMCAD 2006 [A. N. M. Abdullah, B. Akbarpour, and S. Tahar: Formal Analysis and Verification of an OFDM Modem Design using HOL, in: Proceedings IEEE International Conference on Formal Methods in Computer-Aided Design, IEEE Computer Society Press, San Jose, California, USA, November 2006, pp. 189-190].

[2] The work was done when this author was with Concordia University.

[3] Email: nasser@cadence.com

[4] Email: ba265@cl.cam.ac.uk

[5] Email: tahar@ece.concordia.ca

a modulation technique that uses multiple carriers to mitigate the effects of multipath.

Usually, the analysis and functional verification of communications and other electronics designs, such as OFDM modems, are done using simulation. But, simulation is inadequate to check all possible inputs of a design even of moderate size and thus leaves the design partially verified. Formal verification is a technique which has proved itself as a complement to simulation to achieve a rigorous verification. Among established formal verification techniques theorem proving is particularly powerful for verifying complex systems at higher levels of abstraction.

In this paper, we use the general hierarchical methodology proposed by Akbarpour [2] for the formal modeling and verification of DSP (Digital Signal Processing) designs, to verify an implementation of the IEEE 802.11a physical layer OFDM modem [25] using the HOL theorem prover [9]. The verification is performed at all levels of abstraction starting from real, floating-point, and fixed-point number systems down to Register Transfer Level (RTL) hardware implementation. For the purpose of verification, both the design specification and implementation are modeled in formal logic and then mathematical theorems are proved for correctness. We were able to find a bug in the modulation block where the constellation used in the implementation did not follow the IEEE standard specification. Besides, we derive new expressions for the round-off error accumulation while converting from one number domain to the other and carry out a formal error analysis of the OFDM modem in HOL.

The rest of the paper is organized as follows. Section 2 reviews some related work. Section 3 describes details of the OFDM modem implementation to be verified and the methodology used for verification. Section 4 describes the verification of RTL blocks of the OFDM system. Section 5 describes the error analysis of the OFDM modem and its formalization using HOL. The last section concludes the paper and provides hints for future work directions.

2 Related Work

There are numerous research work done on the design and implementation of the IEEE 802.11a physical layer. Although no significant work is done about using theorem proving for the verification of the OFDM or part of the system, we still mention some important implementations of OFDM systems. In [8], a coded OFDM system was developed using the TMS320C6201 processor for telemetry applications in the racing and automotive environment. In [30] the authors developed a wireless LAN (Local Area Network) system using the TI C6x platform. A real time software implementation of OFDM modem optimized for software defined radio is implemented in [6]. Software modules representing discrete system blocks are created and sequentially called upon as needed in this implementation. This software reconfigurable system is developed on a TMS320C6201 evaluation module, which is based on a fixed-point processor. The work also explored different combinations of arithmetic precision and speed for the fixed-point operations. In this paper, we consider the design of [25]. Unlike [8], the design under verification is not optimized for telemetry applications and it does not use the coded OFDM technology. The OFDM design in [30] is targeted for a specific platform and used the high level procedural language subroutine provided by the platform extensively; whereas [25] used Xilinx library to implement some high performance computational blocks. The work described in [6] also designed OFDM

system but it is optimized specially for software defined radio. Both [8] and [6] used the same processor platform, but [25] has a more generic design that can be accommodated in various applications.

There exists a couple of work related to the application of formal methods for the IEEE 802.11. Both use probabilistic model checking but none of them analyzes the design or implementation of the system from the hardware viewpoint. The first one [23] models the two-way handshake mechanism of the IEEE 802.11 standard with a fixed network topology using probabilistic timed automata, a formal description mechanism, in which both nondeterministic and probabilistic choices can be represented. Then from the probabilistic timed automaton model a finite-state Markov decision process is obtained which in turn is verified using PRISM [22], a probabilistic model checking tool. In the second work [29], which identifies ways to increase the scope of application of probabilistic model checking to the 802.11 MAC (Media Access Control), presents a generalized probabilistic timed automata model optimized through an abstraction technique. Here also the results were verified using PRISM. In contrast to these related work, we focus on a completely different direction. While the first work performs model checking on a IEEE 802.11 network setting and concentrates on the protocol issues, it is concerned more about the upper layers of the OSI (Open System Interconnect) model than the physical layer. The second work also uses model checking to verify the MAC protocol which resides just above the physical layer. In this paper, we concentrate only on the physical layer and its hardware implementation. Moreover, instead of model checking we use theorem proving techniques based on HOL. The above two work are totally related with the protocol verification and address the verification issues related with the upper layers of OSI model and hence more related with software verification. Besides, in the work we present here, we propose a formal error analysis of the physical layer implementation, which is to the best of our knowledge the first work of its kind to tackle this issue.

Previous work on the error analysis in formal verification was done by Harrison [11] who verified 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. In another work, Huhn *et al.* [17] 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 in [17] 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.

Arithmetic errors in the implementation of digital filters and the FFT algorithm have long been analysed using traditional mathematics and simulation. For instance, Tran-Thong and Liu [31] presented a detailed analysis of roundoff error in various versions of the FFT algorithm using fixed-point arithmetic. Jackson [20] analysed the roundoff noise for the cascade and parallel realizations of fixed-point digital filters. Liu and Kaneko [24, 21] presented a general approach to the error analysis problem of digital filters and FFT algorithm using floating-point arithmetic and calculated the error at the output due to the roundoff accumulation and input quantization. Error analysis is traditionally validated by comparing such theoretical results with experimental simulations.

In contrast to [17], Akbarpour [2] developed a framework for the error analysis of DSP systems using the HOL theorem prover. He showed how the error analysis above, particularly those of Liu and Kaneko [24, 21], can be verified mechanically. He extended this analysis to cover floating-point and fixed-point digital filters and FFT algorithms. Akbarpour's analysis of DSP algorithms follows Harrison's verification [11] of the floating-point algorithm for the exponential function using the HOL Light theorem prover which is a prior example of formalized error analysis.

In this paper, we intend to investigate error analysis in the same way as proposed by [2] but on a larger case study, here an IEEE 802.11 OFDM modem. Our work proves that the approach in [2] is scalable. On top of that, in contrast to [31, 21], which perform error analysis on single structures of FFT algorithm, we derive new expressions for the accumulation of roundoff error in IFFT-FFT combination which is radix-4 and 64 point in computation as a computation model for the whole OFDM structure. In ideal case, the output signal of the modem should be equal to the input. But, we show that in the real implementation this is never the case because of the finite precision effects.

3 IEEE 802.11 OFDM Modem and Verification Methodology

A standard block diagram implementation of the IEEE 802.11 physical layer OFDM modem is shown in Figure 1. The first block is the random data generator, which is shown here merely for completion purpose. The next block is a quadrature amplitude modulation block (QAM). For our specific implementation, 64-QAM is used. The next block is a serial to parallel (S/P) block that can also be found in the receiver side of the block diagram. The next block is the IFFT block, one of the most important blocks of OFDM. The design uses a 64-point complex IFFT core from Xilinx Coregen Library [34].

The IFFT uses the same IP core as FFT block that comes in the receiver. The parallel to serial (P/S) circuitry makes the next block. The next block in the transmitter line is the guard interval insertion circuitry. In the receiver side, the first block is guard interval removal block. We skip to QAM demapper (DQAM) block since we discussed the other blocks before. From this block the data is serialized again and the output is received sequentially.

The design flow chosen for the OFDM modem implementation under study starts from the floating-point modeling. For this OFDM modem design, the environment used for floating-point modeling is the Signal Processing Worksystem (SPW) from Cadence [5]. The second step in the design flow is fixed-point modeling and simulation. The environment used for this purpose is the Hardware Design System (HDS), which is a set of libraries from SPW. Then VHDL codes are generated automatically for the whole system using HDS

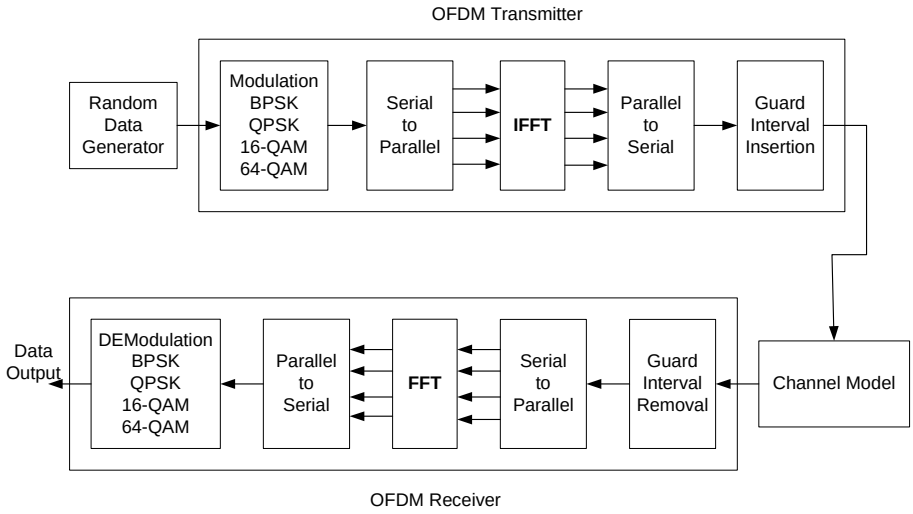


Fig. 1. OFDM Block Diagram [25]

also. But, for some blocks like FFT/IFFT there was no HDS counterpart and those were imported from the Xilinx Coregen Library. Some of the VHDL codes were prepared manually [25]. After VHDL code generation, these blocks are synthesized in Synopsys Design Compiler targeting FPGA as the hardware for implementation. Finally, the synthesized circuitry is mapped into FPGA using "Place and Route" techniques and a bit file is generated.

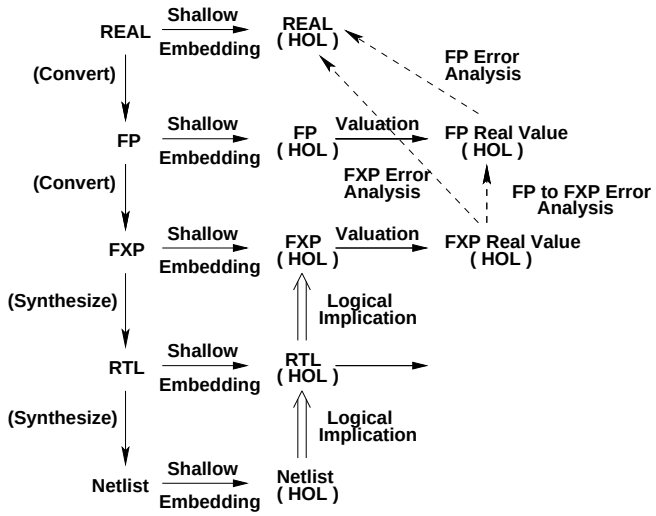


Fig. 2. DSP Specification and Verification Approach [2]

The formal specification, verification and error analysis used in this paper is adopted from DSP verification framework proposed by Akbarpour [2]. The commutating diagram shown in Figure 2 demonstrates the basic idea of the framework. The methodology proposes that the ideal real specification of the DSP algorithms and the corresponding floating-point (FP) and fixed-point (FXP) representations as well as the RTL (Register Transfer Level) and gate level implementations be modeled in higher order logic based on the idea of shallow embedding [4] of languages using the HOL theorem proving environment.

For the transition from real to FP and FXP levels, an error analysis is used in which

the real values of the floating-point and fixed-point outputs are compared with the corresponding output of the ideal real specification. The verification of the RTL is performed using well-known classical hierarchical proof approaches in HOL. 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 the figure. This analysis is not covered in this paper.

4 Formal Functional Verification

In this section we describe the verification of the RTL blocks of OFDM using HOL according to the methodology described in Section 3. The whole design is segmented into different blocks and then modeled using HOL. The resulting model is in turn set against an ideal specification and the HOL tool is used interactively to prove its correctness. In the following sections we will describe in details the verification of QAM, DQAM, serial to parallel (S/P) and parallel to serial (P/S) blocks. For the blocks described below, the corresponding abstract models, HOL models and parts of the proof strategy are provided to explain the verification in its entirety. For more details please refer to [1].

4.1 Verification of QAM and DQAM Block

QAM (Quadrature Amplitude Modulation) is a modulation scheme which conveys data by changing the amplitude of two carrier waves. These two waves, usually sinusoids, are out of phase with each other by 90° and are thus called quadrature carriers—hence the name of the scheme. It is a kind of *M-ary* signaling technique where one of M possible signals, $s_1(t)$, $s_2(t)$, ..., $s_M(t)$ may be sent during each signaling interval of duration *T*. Unlike *M-ary PSK* (Phase Shift Keying), where in-phase and quadrature components of the modulated signals are interrelated in such a way that the envelope is constrained to remain constant, QAM has this constraint removed. The general form of *M-ary* QAM is defined by the following transmitted signal:

$$s_i(t) = \sqrt{\frac{2E_0}{T}}\, a_i\, cos\,(2\pi f_c t)\; +\; \sqrt{\frac{2E_0}{T}}\, b_i\, sin\,(2\pi f_c t) \quad 0 \leq t \leq T \tag{1}$$

where E_0 is the energy of the signal with the lowest amplitude, and a_i and b_i are a pair of independent integers chosen in accordance with the location of the pertinent message point [14]. According to the IEEE 802.11a standard, the OFDM subcarriers shall be modulated by using *BPSK* (Binary Phase Shift Keying), *QPSK* (Quadrature Phase Shift Keying), *16-QAM*, or *64-QAM* modulation depending on the rate requested. The encoded and interleaved binary serial input data shall be divided into bit groups and converted into complex numbers representing BPSK, QPSK, 16-QAM or 64-QAM constellation points. The conversion shall be performed according to Gray-coded constellation mappings, illustrated in Figure 3, with the input bit, b0, being the earliest in the stream. The output values, d, are formed by multiplying the resulting $I + jQ$, where I and Q are the x-axis and y-axis of the constellation respectively, value by a normalization factor K_{MOD}

$$d \;=\; (I + jQ)\, K_{MOD} \tag{2}$$

The normalization factor, K_{MOD}, depends on the base modulation mode, as prescribed in Table 1. The purpose of the normalization factor is to achieve the same average power for all mappings. In practical implementations, an approximate value of the normalization factor can be used, as long as the device conforms with the modulation accuracy as specified in the draft standard of IEEE 802.11a in [19]. A question might arise in terms of what QAM constellation should be used for OFDM? The answer lies in the fact that, although higher constellation gives more bits per symbol, if the mean energy is to remain the same, the points must be closer together and are thus more susceptible to noise and other corruption; this results in a higher bit error rate and so higher-order QAM can deliver more data less reliably than lower-order QAM.

Modulation	K_{MOD}
BPSK	1
QPSK	$\frac{1}{\sqrt{2}}$
16-QAM	$\frac{1}{\sqrt{10}}$
16-QAM	$\frac{1}{\sqrt{42}}$

Table 1
K_{MOD} Normalization

For the OFDM design verified, 64-QAM constellation was chosen after simulating the floating-point and fixed-point point models in Cadence SPW. The circuitry used for QAM mapping is implemented using combinational logic. It maps the input integer data into a constellation point as shown in Figure 3.

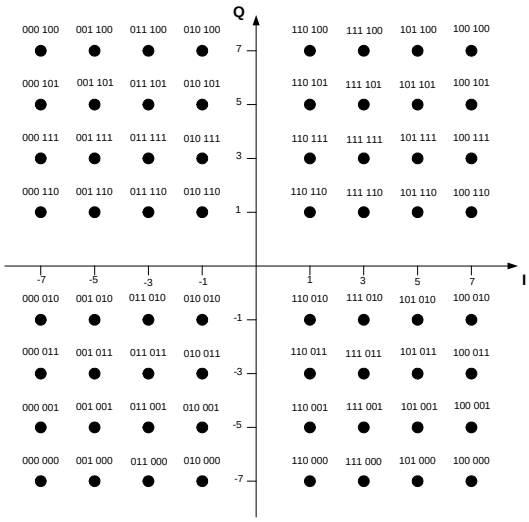
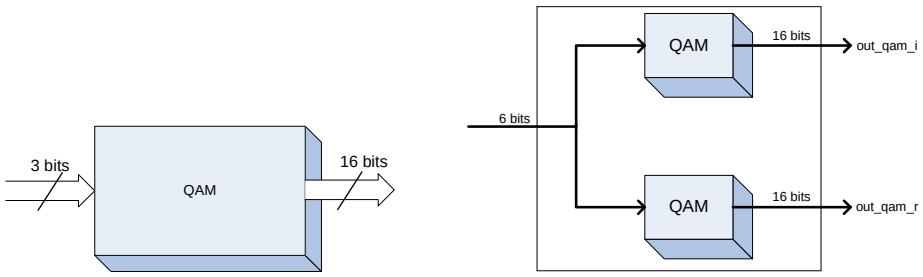


Fig. 3. 64-QAM Constellation Bit Encoding

The VHDL modeling is done using a look-up table approach [25]. The QAM block takes only 3 bits as inputs and maps to an output of 16 bits as shown in Figure 4a. The

QAM block is instantiated two times and designed to generate the real and imaginary components as two separated outputs. Each of them is formatted in 16-bit 2's complement against a 3-bit input chosen from an input of six for each block. These outputs are shown by *out_qam_r* and *out_qam_i* in Figure 4b. The circuitry is fed by the input continuously, therefore *out_qam_r* and *out_qam_i* are generated as continuous streams. The outputs are processed in groups of 48 symbols which are stored in two separated dual port RAMs called *"Dual Port RAM image"* and *"Dual Port RAM real"*, respectively. Since, this type of RAM is generated automatically using the Xilinx Coregen Library [35] it is not discussed further.



(a) QAM Block (b) Instantiation of QAM Block

Fig. 4. QAM Block and its Instantiation

The modeling of QAM is done in HOL using different existing theories. An IF-THEN-ELSE construct is used to embed the VHDL code as shown in the following code:

```
⊢def  ∀input qam_out.
qam_imp (input qam_out) =
 (WORDLEN input = 3) ∧
 (if input = WORD [ F; F; F ] then
   qam_out = WORD [ T; F; F; T; F; F; F; F; F; F; F; F; F; F; F ]
  else
 (if input = WORD [ F; F; T ] then
   qam_out = WORD [ T; F; T; T; F; F; F; F; F; F; F; F; F; F; F ]
  else
 (if input = WORD [ F; T; F ] then
   qam_out = WORD [ T; T; T; T; F; F; F; F; F; F; F; F; F; F; F ]
  else
 (if input = WORD [ F; T; T ] then
   qam_out = WORD [ T; T; F; T; F; F; F; F; F; F; F; F; F; F; F ]
  else
 (if input = WORD [ T; F; F ] then
   qam_out = WORD [ F; T; T; T; F; F; F; F; F; F; F; F; F; F; F]
  else
 (if input = WORD [ T; F; T ] then
   qam_out = WORD [ F; T; F; T; F; F; F; F; F; F; F; F; F; F; F ]
  else
 (if input = WORD [ T; T; F ] then
   qam_out = WORD [ F; F; F; T; F; F; F; F; F; F; F; F; F; F; F ]
  else
   qam_out = WORD [ F; F; T; T; F; F; F; F; F; F; F; F; F; F; F ])))))))
```

The above model is based on the *wordTheory* [33]. The data types of VHDL can be modeled using this theory. The VHDL type *BIT* can be modeled using **T** and **F** where these represent **1** and **0** respectively. *BIT VECTOR* can be modeled using `WORD[...]` where the dots can be replaced with any sequence of **T** or **F** separated by *";"* as above. As an example, bit vector "110" can be modeled as `WORD[T;T;F]`. The above model is constrained using the condition `WORDLEN input = 3` since the input is always 3 bits and thus the model does not need to be generalized for *n* bits. Here, `WORDLEN` is a function that takes any `WORD` as input and returns the length of it. The model above now can be used (or in HDL terminology can be instantiated) as many times as required to model any

complex design. For our case, it is used two times to embed the port-mapped component in HOL, and named as *qam_mod2*. We stick to the same nomenclature used by the designer. Below is the corresponding HOL modeling.

```
⊢def  ∀ input out_qam_r out_qam_i.
      qam_mod2_imp (input out_qam_r out_qam_i) =
      (WORDLEN input = 6) ∧ (WORDLEN out_qam_r = 16) ∧
      (WORDLEN out_qam_i = 16) ∧ qam_imp (WSEG 3 0 input) out_qam_i ∧
      qam_imp (WSEG 3 3 input) out_qam_r
```

This model has the same characteristics as the one before except the *input* is now constrained to six bits since the input of *qam_mod2* will always be six.

Now that the modeling of the RTL block is completed it is time to model the specification of QAM in HOL. After that we will use the logical techniques of the tool to prove that the implementation is conformed to the specification. Since the design is based on IEEE 802.11a we have used the standard [19] itself as a specification in order to verify the QAM implementation. Accordingly, for every six bits entering the *qam_mod2* block, the bits are divided into three bits each, which acts as an input to the *qam* block. Then, as described above, the *qam_mod2* block outputs two vectors containing real and imaginary parts of the modulated input. Table 2 shows the encoding of bits for I and Q.

Input bits (b_0,b_1,b_2)	$I-out$	Input bits (b_3,b_4,b_5)	$Q-out$
000	-7	000	-7
001	-5	001	-5
011	-3	011	-3
010	-1	010	-1
110	1	110	1
111	3	111	3
101	5	101	5
100	7	100	7

Table 2
$64-QAM$ Encoding Table [19]

One point can be noticed from the two tables is the similarity of bit encoding both for I and Q and this helps us to model only one specification for both, while it is trivial to model them separately. Modeling a table in HOL can be done by using predicates as follows:

```
val TABLES_QAM =
  ⊢def  ∀ I_OUT.
        TABLES_QAM (I_OUT) =
        (I_OUT (F,F,F) = ¬7) ∧ (I_OUT (T,F,F) = ¬5) ∧
        (I_OUT (T,T,F) = ¬3) ∧ (I_OUT (F,T,F) = ¬1) ∧
        (I_OUT (F,T,T) = 1) ∧ (I_OUT (T,T,T) = 3) ∧
        (I_OUT (T,F,T) = 5) ∧ (I_OUT (F,F,T) = 7)
```

In the above model *I_OUT* is a triplet which will accept three bits similar to the left columns of Table 2. For each and every argument of *I_OUT*, a unique number will be

mapped as given in the tables and '∧' is used as a composition operator to construct all rows. Having covered all the pertinent details about the implementation and a very reliable means to extract the specification, *qam_spec* can be written in terms of TABLES_QAM as follows:

```
⊢def  ∀ b0 b1 b2 I_OUT.
      qam_spec (b0 b1 b2 I_OUT) =
      ∃OUT. TABLES_QAM OUT ∧ (I_OUT b0 b1 b2 = OUT (b0,b1,b2))
```

The specification *qam_spec* is mirrored, in the same way its implementation *qam_imp* is instantiated in *qam_mod2_imp*,

```
⊢def  ∀ input I_OUT_R I_OUT_I.
        qam_mod2_spec (input I_OUT_R I_OUT_I) =
        qam_spec (BIT 0 input) (BIT 1 input) (BIT 2 input) I_OUT_I ∧
        qam_spec (BIT 3 input) (BIT 4 input) (BIT 5 input) I_OUT_R
```

With the specification above we have finished all the groundwork to set the goal for verification of the QAM RTL block. Next we will discuss the verification in details and the proof strategies adopted to bolster the correctness of RTL implementation. The general goal is to prove that for all inputs and outputs the correctness theorem holds, under certain constraints, which can be stated as

```
∀ n inputs outputs. constraints ⟹ (implementation ≡ specification)
```

The equivalence can be replaced by implication which will set space for some allowance in the correctness theorem by proving only specific behaviors of the system, which will certainly weaken the sole purpose of verification. But, there are cases where the engineer (or anybody who is carrying out the proof work) can categorically exclude some cases given the certainty that those will never occur. For our case, it is an implication due to the constraints we have imposed in the definitions since we are certain that there can be no other combination occurring other than those. This justification leaves us only to state our goal, except we need one more definition to do so, which is as follows:

```
⊢def  ∀ x.
        TCOMP_VAL x =
        ¬& (BV ()) * 2 pow 3 + & (BV (BIT 2 x)) * 2 pow 2 +
        & (BV (BIT 1 x)) * 2 pow 1 + & (BV (BIT 0 x))
```

It is a simple definition based on *boolLibrary* of HOL to convert a `bool word` into its real number equivalent. The function `TCOMP_VAL` accepts a `bool word` and returns a real number. The "&" symbol is an overloaded HOL operator that converts any natural number to real number. And, `BV` is also a function, defined in theory *numTheory*, that uses another function to convert the boolean value into a natural number.

```
⊢def  ∀ b. BV b = (if b then SUC 0 else 0)
```

The function `SUC` takes a natural number and returns the consecutive natural number. So, `SUC 0` will return `1`. And, `BIT x input` chooses a particular bit positions from *input* defined in *x*. Now, we can state that our goal as - *for all input and output and constraints, the QAM implementation implies the QAM specification*

```
∀ n inputs outputs. constraints ⟹ (implementation ⟹ specification)
```

Formalized in HOL as

```
∀ input qam_out.
  qam_imp (input qam_out) ⟹
  qam_spec (BIT 0 input) (BIT 1 input) (BIT 2 input)
  (λ b0 b1 b2. TCOMP_VAL (WSEG 4 12 qam_out))
```

The definition `WSEG m k WORD` selects a portion of `WORD` from `k` to `k+m−1`. The function *qam_spec* takes three arguments and gives a corresponding output. One λ abstraction is used to convert the selected *qam_out* word into real number. Now, the stage is set to apply the tactics of HOL to prove the goal. We have used the existing theories of *wordTheory* and *realTheory* to build many helpful definitions and lemmas to prove the above goal and thus established the correctness of the RTL block formally. We prove the theorem and name it as *qam_imp_spec_correct*. Due to textual brevity, we do not include the whole proof procedure here line by line. But, proving this theorem just ensures us about the QAM block and we are yet to prove the implementation of *qam_mod2_imp*. In order to do so we set a goal as -

```
∀ input out_qam_r out_qam_i.
   qam_mod2_imp (input out_qam_r out_qam_i) ⟹
   qam_mod2_spec input (λ b1 b2. TCOMP_VAL (WSEG 4 12 out_qam_r))
     (λ b0 b1 b2. TCOMP_VAL (WSEG 4 12 out_qam_i))
```

We use the same libraries as before to prove this goal too. Below is the HOL proof steps.

```
REPEAT GEN_TAC THEN
ARW_TAC [qam_mod2_spec,qam_mod2_imp]THEN
ARW_TAC[BIT_WSEG_input]THEN
ARW_TAC [qam_imp_spec_correct]THEN
ARW_TAC[BIT_WSEG_input]THEN
ARW_TAC [qam_imp_spec_correct]
```

We use only built in tactics. The `REPEAT GEN_TAC` tactic removes all the universal and existential quantifications. Next, `ARW_TAC` is a tactic defined using a rewriting tactic `RW_TAC` using simpset [16] `arith_ss`. This defined tactic is used to rewrite the the goal with the specifications and proved theorems as shown in the code segments above. We name this last proved theorem as *qam_imp_spec_correct*.

Having proved the correctness of *qam_mod2_imp* and *qam_imp* using the theorems *qam_imp_spec_correct* and *qam_mod2_imp_spec_correct* it can be concluded that the QAM is formally verified. The implementation conforms the specification given in the standard.

Following a similar approach we have proved the correctness of DQAM block. The details can be found in [1].

4.2 Verification of the S/P and P/S Blocks

In this section we will verify the serial to parallel block, later written as S/P, which is an indispensable part of the whole OFDM system. Most of the basics related to S/P are similar to those of the Parallel to Serial block, to be discussed later, and thus will cover almost all the important aspects of both blocks in this section. The concept of serial to parallel conversion is trivial. A long stream of data is divided into several equal or approximately equal length of chunks which can all be operated upon at the same time. From the mathematical point of view, it is the manipulation of a vector into several columns of a matrix. However, S/P conversion is very important in OFDM. The length of the blocks produced in S/P determine the number of spectral coefficients to be used by the IFFT, which is essential in choosing how many frequencies are to be used. Usually, the block length is a power of

2, which makes the IFFT and FFT algorithms most computationally efficient. Moreover, in OFDM, the data is divided among a large number of closely spaced carriers. Since the entire bandwidth is filled from a single source of data, it is necessary to transmit in a parallel way so that only a small amount of the data is carried on each carrier, and by this lowering of the bitrate per carrier, the influence of intersymbol interference is significantly reduced.

The S/P circuitry is very simple to implement. It has its presence both in the transmitter and receiver of the system. In the transmitter side, it is placed between *QAM* and *IFFT* block, and in the receiver side between *Guard Removal* and *FFT* block. The design at hand has the same functionality of of *"Bits to fixp"* block of SPW [5] in fixed-point model. It consists of a shift register and a latch, which are both clocked with the same rate as the input data. Six bits from input stream are serially shifted into a register. Then they are latched for six clock cycles. There are two control signals *enable* and *clear* to synchronize the whole process.

Modeling of the S/P block in HOL is done in a different way than what we have seen in Section 4.1. The modeling is not exactly one to one mapping because a VHDL *PROCESS* is involved here. In fact, a *PROCESS* never terminates itself, and it can only be controlled using *WAIT* statements and sensitivity lists. After executing the last statement, a *PROCESS* will be suspended only to be resumed later on an event in the sensitivity list. This last behavior poses a difficulty in modeling it in HOL due to non-termination problem. Higher order logic is a logic of total function and it does not allow the definition of any partial function. But, there are exceptions which motivates us to define our specification for S/P in a simpler way without resorting to complex definition. For example, the following is a total and non-recursive function that uses the expressive power of HOL [16]:

```
λ x. if (? n. P (FUNPOW g n x))
       then
           FUNPOW g (@n. P (FUNPOW g n x) ∧ (!m. m < n ==> ¬ P (FUNPOW g m x))) x
       else ARB
```

The function `FUNPOW` is a tail recursive function from the theory *arithmeticTheory* to define function iteration. The above function does a case analysis on the iterations of function g. The finite ones return the first value at which P holds and the infinite ones are mapped to a constant named `ARB` that holds all the arbitrary values. `ARB` is a way to convert partial-functions into total functions in HOL. But, using `ARB` will only complicate our model without any added benefit. A VHDL *PROCESS* is more than a simple loop and we have no cases to deal with infinity rather we only have finite sets of statements to be dealt with infinitely. This discussion is to justify why we did not use certain features of HOL to model our system which seems apparently helpful in doing so. The other aspect of the model is that three signals *clk, enable,* and *clear* are not used since we are verifying this module independently of other blocks, and there are no pipelining issues involved here. Having said that we introduce the implementation of S/P in HOL as follows

```
⊢def  ∀ cnt out_parallel input.
        Serial_Parallel_IMP (cnt out_parallel input) =
        ∃ shift_reg.
          (WORDLEN out_parallel = 6) ∧
          (shift_reg input = SHRN_bit cnt input out_parallel)
```

Apparently a simplification of the corresponding VHDL code but a little analysis will support its correct functionality. From the code, the variable `cnt` is a natural number whose type is defined as `num`; `out_parallel` is a `bool word` and `input` is of `bool` type.

The implementation takes three arguments where `cnt` is defined to keep track of the time or bit index which is a model of the `signal count`. The second variable has the same name of its VHDL counterpart and so is the last one - `input`. A function `shift_reg` is defined as `shift_reg:bool→bool word` to mimic the VHDL signal of the same name. Variable `out_parallel` is constrained to six using WORDLEN function as before because the design specifies so. Since the system will receive only one input at a time and then latches all till it fills the whole shift register, so we write another definition in HOL to manipulate every new bit entering the system and filling the empty places with zeros

```
⊢def  ∀ N M w.
        SHRN_bit (N M w) =
        WCAT (WORD (REPLICATE (WORDLEN w − (N + 1)) F),WORD [ M ])
```

This definition uses `WCAT` which concatenates two lists is defined in *word_baseTheory* [16] as

```
⊢def  ∀ l1 l2. WCAT (WORD l1,WORD l2) = WORD (l1 ++ l2)
```

The symbol '++' is an infix operator that appends two lists in the above definition. The recursive definition of `REPLICATE` is in the theory *rich_listTheory* which replicates any variable repeatedly as specified. It is defined as

```
⊢def  (∀ x. REPLICATE 0 x = []) ∧
       ∀ n x . REPLICATE (SUC n) x = x::REPLICATE n x
```

Here the `REPLICATE` function fills the rest of the places of the shift register with 'F' depending on the current value passed to it by the function and then adds the `input` to it. In this way at the end of the iteration the whole register will be populated with serial data and will be ready to be latched out.

Having completed the modeling of implementation we describe the specification of the block so that we can explain the verification in the next section. We state the specification of the block as

```
⊢def  ∀ t out input.
        Serial_Parallel_SPEC (t out input) = (BIT t out = input)
```

It simply puts the relation between the input and output of the block in terms of bit position. At every time `t`, we have one input entering the block which goes in the bit position related to the current index of `t` of the output. A more general approach would be to use the modulo arithmetic to model the specification, but it is not required here due to the proof strategy we will follow next.

Unlike the verification strategy of QAM explained in Section 4.1, we adopt a case analysis approach to prove the goal. We can define the goal as following:

```
∀ out input t.
          (0 ≤ t ∧ t ≤ 5 ) ⟹
               Serial_Parallel_IMP (t out input) ⟹
                    Serial_Parallel_SPEC (t out input)
```

It has a very generic pattern like any other goal except the constraint which bounds t as, $0 \leq t \leq 5$. Bounding t helps to get over with the problem of looping which we stated earlier. We flatten one whole iteration which is enough to demonstrate the functional correctness of the given block. That is why we bound the variable only to check the cases starting from $t = 0$ to $t = 5$. Once we finish with case analysis we prove following trivial lemma

```
∀ t.        (0 ≤ t ∧ t ≤ 5) ⟹
            (t = 0) ∨ (t = 1) ∨ (t = 2) ∨ (t = 3) ∨ (t = 4) ∨ (t = 5)
```

which simply states that when t is bound between 0 and 5, then the only values for which the correctness theorem needs to hold are $t = 0, 1, 2, 3, 4, 5$. We proved the goal and thus verified the functionality of the said RTL block.

Following a similar approach, we have proved the correctness of the P/S block. The details can be found in [1].

4.3 Discussion

The modeling, specification and verification done above for the OFDM RTL blocks demonstrate a way to incorporate formal methods in the verification of digital systems. We have described the implementation of the RTL blocks in HOL using formal logic. For the QAM block, it was straightforward to embed the if-then-else HDL code in HOL and the specification is obtained from IEEE 802.11 specification. Although the demodulator block has a similar implementation and its formal description was similar to the QAM block, but finding a specification to check the design could not be done using IEEE standard since this block resides in the receiver side and the designer has the freedom to choose any way to implement it. Both the specifications for QAM and demodulator are based on look-up tables and the implementations were proved against those. For the S/P and P/S blocks, the specifications and implementations were also formalized after much consideration about the VHDL *PROCESS*. The verification of all blocks were done using existing theories in HOL on real numbers, natural numbers, boolean logic, lists, words and others. Many lemmas were proved in order to aid the proof steps. Some lemmas were very trivial but HOL requires each and every proof step to be sound and complete and that is why there is no ambiguity in the HOL proof. The built-in rewriting tactics `RW_TAC` was heavily used with the powerful simplification sets augmented with the required lemmas and theorems. In most cases, the proof strategy starts with a rough proof sketch by hand and then formalized in HOL. But, some lemmas and intermediate theorems were simple enough to not resort to this approach.

The main purpose for using formal verification was to find bugs in the design. We did not find any bug in the blocks. But, some comments are in order. Namely, for the QAM block, it is given in the standard that the input for a 64-QAM modulation must follow the constellation diagram shown in Figure 3. The constellation gives output between -7 to 7 but the implementation used 16 bit 2's complement number to represent these numbers while 3 bits would have done the same job. If the standard is followed exactly, then this issue might have resulted in a bug in the design. But, the standard gives some flexibility to the designers in order to have more precise results from the IFFT block, as explained before in Section 4.1. As, we were aware about it at the time of verification, we constrained the implementation using the proper number of bits. The same comments are applied to the DQAM block. For the rest of the blocks we did not find any issue like this.

A pertinent question can be raised about the higher-order logic used for the modeling and verification of OFDM that - whether first-order logic can also be used for this purpose. The reason is of course automation of proofs and completeness in some cases. It is mentioned in Chapter 3 that higher-order logic is expressive and the variables in this logic can be functions and predicates those in turn can take functions and predicates as arguments and return them too. Whereas first-order logic can only quantify over objects and variables.

For the design verified none of the RTL blocks can be specified or verified using first-order logic fully. For instance, the QAM block cannot be modeled completely using first-order logic, although the implementation of the block—a pure combinational logic circuit—can be modeled in first-order since simple predicate logic is used. But, the instantiated specification of QAM block needs universal quantification on functions which were used to access the tables of the I and Q values for modulation. For, the S/P block, first-order logic cannot be used due to the use of existential quantification on the shift register in the formal modeling of the implementation. The same can be told for the P/S block. The implementation of the DQAM block can be modeled using first-order logic but the instantiation of the demapping function for modeling the decision region for specification needs to be universally quantified.

There are other blocks in the OFDM that we did not verify; namely, guard interval insertion and guard interval removal. The reason is that the RTL codes for those blocks were not available for the design at hand. The guard insertion block in the transmitter side has a portion of its behavioral code but the whole code mostly contains port-mapping [3] to the IP blocks. In general, the whole design contains many IP blocks and thus the verification of the design in its entirety is not practical using any theorem-proving tool like HOL. Still, this chapter demonstrates the scope and feasibility of formal methods in a comprehensive way in parts of the OFDM RTL blocks.

5 Formal Error Analysis

This section describes the error analysis of OFDM modem in a formal way. We first derive expressions for the accumulation of round-off error in the OFDM structure and then describe how we proved the corresponding theorems in HOL. Mainly we focus on the two computational blocks of the design—FFT and IFFT. Among all the blocks only FFT and IFFT are computational blocks doing arithmetic operation. Other blocks carry out merely mapping operations of bits from one domain to another. We take IFFT-FFT combination as the model for the error analysis of the OFDM modem. Figure 5 shows the block diagram of the IFFT-FFT combination.

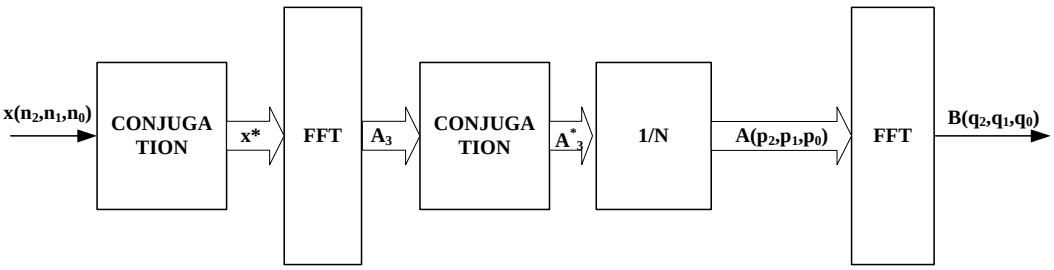


Fig. 5. Construction of IFFT-FFT

We first derive the equations for this system as [1]:

$$B(q_2, q_1, q_0) = \frac{1}{64} \sum_{\mathbf{p}} \sum_{\mathbf{n}} x(n_2, n_1, n_0)(W_{64})^{(L-M)} \tag{3}$$

where

$$\begin{aligned}
\sum_{\mathbf{p}} &= \sum_{p_0=0}^{3}\sum_{p_1=0}^{3}\sum_{p_2=0}^{3} \\
\sum_{\mathbf{n}} &= \sum_{n_0=0}^{3}\sum_{n_1=0}^{3}\sum_{n_2=0}^{3} \\
L &= 16q_0n_2 + (4q_1 + q_0)4p_1 + (16q_2 + 4q_1 + q_0)p_0 \\
M &= 16p_0n_2 + (4p_1 + p_0)4n_1 + (16p_2 + 4p_1 + p_0)n_0
\end{aligned} \tag{4}$$

Next we represent this mathematical model in real, floating-point and fixed-point domains. The signal $x(n)$ and twiddle factor W_{64} are complex numbers and can be written in terms of their real and imaginary components. In Equation (3) these two functions are multiplied with each other. We denote the real and imaginary parts of $x(n)$, $B(q)$, and W_{64} like C_0, D_0, C, D, U_{64}, and V_{64} and rewrite the Equation (3) as following

$$C(q_2, q_1, q_0) = \frac{1}{64}\sum_{\mathbf{p}}\sum_{\mathbf{n}} C_0(n_2, n_1, n_0)(U_{64})^{(L-M)} - D_0(n_2, n_1, n_0)(V_{64})^{(L-M)} \tag{5}$$

$$D(q_2, q_1, q_0) = \frac{1}{64}\sum_{\mathbf{p}}\sum_{\mathbf{n}} C_0(n_2, n_1, n_0)(V_{64})^{(L-M)} + D_0(n_2, n_1, n_0)(U_{64})^{(L-M)} \tag{6}$$

Mimicking the analysis of real numbers we ought to define the equations for floating-point and fixed-point number and state $fl(.)$ and $fxp(.)$ as floating-point and fixed-point, respectively. The characters prime and double primes are used to point to floating-point and fixed-point numbers and we will stick to this convention in the analysis set forth. Using these notations we denote the floating-point and fixed-point conversions of C and D as C', C'', D', D'', respectively.

In analyzing the effects of floating-point roundoff, the effect of rounding is presented multiplicatively. Let $*$ denote any of the operations $+$, $-$, $\times$, $\div$. It is known [32, 7] that if p represents the precision of the FP format, then

$$fl(x * y) = (x * y)(1 + \delta), \quad \text{where } |\delta| \leq 2^{-p}. \tag{7}$$

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

$$fxp(x * y) = (x * y) + \epsilon, \quad \text{where } |\epsilon| \leq 2^{-fracbits\ (X)} \tag{8}$$

and *fracbits* is the number of bits that are to the right of the binary point in the given FXP format X.

The real part of floating-point, C' , can be written with all the errors due to floating-point round-off as follows, where $\boldsymbol{\delta}$ accounts for the round-off error due to multiplication

of $C_0^{'}$ and $(U_{64})^{(L-M)}$ according to Equation (7).

$$
\begin{aligned}
C^{'}(q_2,q_1,q_0) \;=\; \frac{1}{64}\Bigg[\sum_{\mathbf{p}}\sum_{\mathbf{n}}\Big(\Big(C_0^{'}(n_2,n_1,n_0)(U_{64})^{(L-M)} \\
(1+\delta_{1024p_2+256p_1+64p_0+16n_2+4n_1+n_0})\Big)- \\
\Big(D_0^{'}(n_2,n_1,n_0)(V_{64})^{(L-M)} \\
(1+\epsilon_{1024p_2+256p_1+64p_0+16n_2+4n_1+n_0})\Big)\Big) \\
(1+\xi_{1024p_2+256p_1+64p_0+16n_2+4n_1+n_0}) \\
\prod_{\substack{i=1024p_2+256p_1\\+64p_0+16n_2\\+4n_1+n_0}}^{4095}(1+\lambda_i)\Bigg](1+\tau)(1+\rho)
\end{aligned}
\tag{9}
$$

The function $\boldsymbol{\epsilon}$ represents the error due to the round-off error after the multiplication of $D_0^{'}$ and $(V_{64})^{(L-M)}$. The error due to the subtraction of $[C_0^{'}(U_{64})^{(L-M)} - D_0^{'}(V_{64})^{(L-M)}]$ is represented using $\boldsymbol{\xi}$. Based on the errors due to one single iteration, the error due to the two summations $\sum_{\mathbf{p}}\sum_{\mathbf{n}}$ (which is actually an abbreviation for six summations $\sum_{p_0=0}^{3}\sum_{p_1=0}^{3}\sum_{p_2=0}^{3}\sum_{n_0=0}^{3}\sum_{n_1=0}^{3}\sum_{n_2=0}^{3}$ can be stated as products of $\boldsymbol{\lambda}$ where the upper index is set as 4095 due to six iterations each ranging from 0 to 3 giving $4\times4\times4\times4\times4\times4-1 = 4095$. It should have eclipsed all the rounding errors in the whole system of equation, but still the fraction $\frac{1}{64}$ incurs two round-off errors. One of them due to the division of 1 by 64, denoted as $\boldsymbol{\tau}$ and the other is for the multiplication thereafter with the rest of the equations, denoted as $\boldsymbol{\rho}$. These errors can be generalized on the same line of reasoning for the other equations.

The error related with the imaginary part $D^{'}$ of the floating-point can be written as

$$
\begin{aligned}
D^{'}(q_2,q_1,q_0) \;=\; \frac{1}{64}\Bigg[\sum_{\mathbf{p}}\sum_{\mathbf{n}}\Big(\Big(C_0^{'}(n_2,n_1,n_0)(V_{64})^{(L-M)} \\
(1+\delta^{''}_{1024p_2+256p_1+64p_0+16n_2+4n_1+n_0})\Big)- \\
\Big(D_0^{'}(n_2,n_1,n_0)(U_{64})^{(L-M)} \\
(1+\epsilon^{''}_{1024p_2+256p_1+64p_0+16n_2+4n_1+n_0})\Big)\Big) \\
(1+\xi^{''}_{1024p_2+256p_1+64p_0+16n_2+4n_1+n_0}) \\
\prod_{\substack{i=1024p_2+256p_1\\+64p_0+16n_2\\+4n_1+n_0}}^{4095}(1+\lambda_i^{''})\Bigg](1+\tau^{'})(1+\rho^{'})
\end{aligned}
\tag{10}
$$

where the previous function symbols used in Equation (9) are modified with double/single prime, namely $\boldsymbol{\delta}^{''}$, $\boldsymbol{\epsilon}^{''}$, $\boldsymbol{\xi}^{''}$, $\boldsymbol{\lambda}^{''}$, $\boldsymbol{\tau}^{'}$, $\boldsymbol{\rho}^{'}$; but the meaning remains the same. A point to emphasize is that all error functions are in multiplication relation with the variable and this is what makes the floating-point round-off error much complicated. Similar formulas can be derived for the real and imaginary parts of fixed-point number, $C^{''}$ and $D^{''}$.

Adding the error parameters leaves us just one step away before we start to formalize the analysis after deriving the error that occurred in the conversion from one domain to another. We start with the real to floating-point conversion and the round-off error difference between the complex floating-point implementation and complex real implementation of IFFT-FFT denoted as $e(q_2, q_1, q_0)$. We derive the following equation that expresses the round-off error accumulated due to real to floating-point conversion,

$$e(q_2, q_1, q_0) = \frac{1}{64} \left[\sum_{\mathbf{p}} \sum_{\mathbf{n}} e_0(n_2, n_1, n_0)(W_{64})^{(L-M)} + \mathbf{f}(\mathbf{n}, \mathbf{p}) \right] \tag{11}$$

where we assume

$$e_0(q_2, q_1, q_0) = C_0^{'}(n_2, n_1, n_0) - C_0(n_2, n_1, n_0) + j\Big(D_0^{'}(n_2, n_1, n_0) - D_0(n_2, n_1, n_0)\Big) \tag{12}$$

and $\mathbf{f}(\mathbf{n}, \mathbf{p})$ is written according to Equations (9) and (10)

$$\begin{aligned}
\mathbf{f}(\mathbf{n}, \mathbf{p}) = {} & C_0^{'}(n_2, n_1, n_0)(U_{64})^{(L-M)} \big[(1 + \delta_{(p,n)})(1 + \xi_{(p,n)}) \prod_{i=(p,n)}^{4095} (1 + \lambda_i)(1 + \tau) - 1\big] \\
& - D_0^{'}(n_2, n_1, n_0)(V_{64})^{(L-M)} \big[(1 + \epsilon_{(p,n)})(1 + \xi_{(p,n)}) \prod_{i=(p,n)}^{4095} (1 + \lambda_i)(1 + \tau) - 1\big] \\
& + j \Bigg[C_0^{'}(n_2, n_1, n_0)(V_{64})^{(L-M)} \big[(1 + \delta_{(p,n)}^{''})(1 + \xi_{(p,n)}^{''}) \prod_{i=(p,n)}^{4095} (1 + \lambda_i^{''})(1 + \tau^{'}) - 1\big] \\
& - D_0^{'}(n_2, n_1, n_0)(U_{64})^{(L-M)} \big[(1 + \epsilon_{(p,n)}^{''})(1 + \xi_{(p,n)}^{''}) \prod_{i=(p,n)}^{4095} (1 + \lambda_i^{''})(1 + \tau^{'}) - 1\big] \Bigg]
\end{aligned} \tag{13}$$

The two variables n and p are used for the function as a short-hand for $n = n_2, n_1, n_0$ and $p = p_2, p_1, p_0$.

The above analysis can be adopted similarly to come at the following error function, $e^{'}(q_2, q_1, q_0)$, for the round-off error due to conversion from real to fixed-point domain

$$e^{'}(q_2, q_1, q_0) \; = \; C^{''}(q_2, q_1, q_0) - C(q_2, q_1, q_0) + j\big[D^{''}(q_2, q_1, q_0) - D(q_2, q_1, q_0)\big] \tag{14}$$

Denoting the error as $\mathbf{f}'(\mathbf{n}, \mathbf{p})$, the final error can be written as

$$e^{'}(q_2, q_1, q_0) = \frac{1}{64} \left[\sum_{\mathbf{p}} \sum_{\mathbf{n}} e_0(n_2, n_1, n_0)(W_{64})^{(L-M)} + \mathbf{f}'(\mathbf{n}, \mathbf{p}) \right] \tag{15}$$

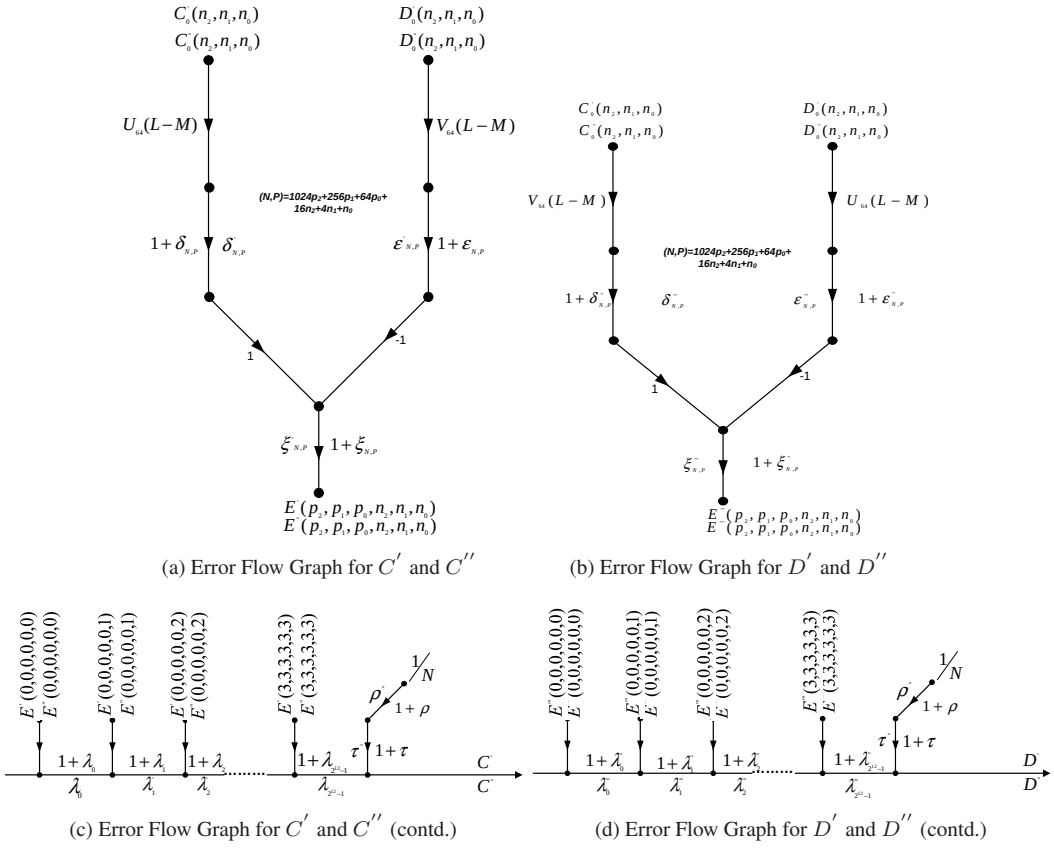
where $\mathbf{f}'(\mathbf{n},\mathbf{p})$ is constructed as follows

$$\begin{aligned}\mathbf{f}'(\mathbf{n},\mathbf{p}) =& \delta'_{(p,n)} + \epsilon'_{(p,n)} + \xi'_{(p,n)} + \sum_{i=(p,n)}^{4095} \lambda'_i + \tau' \\ &+ j\left[\delta'''_{(p,n)} + \epsilon'''_{(p,n)} + \xi'''_{(p,n)} + \sum_{i=(p,n)}^{4095} \lambda'''_i + \tau'''\right]\end{aligned} \tag{16}$$

Equation 16 is much simplified than its real to floating-point counterpart since this error is additive but not multiplicative. To derive the errors due to floating-point to fixed-point conversion, we do not resort to derive those mammoth equations as above, rather we use the previous derivations. If the two error results derived previously are subtracted then the result gives the error we are looking for. Denoting this error as $e''(q_2,q_1,q_0)$, it can be written as

$$e''(q_2,q_1,q_0) = e'(q_2,q_1,q_0) - e(q_2,q_1,q_0) \tag{17}$$

Figure 6 summarizes all the error analysis discussed so far in a flow-graph format. They refer to the errors incurred in the real parts of the floating-point and fixed-point model.



(a) Error Flow Graph for C' and C''

(b) Error Flow Graph for D' and D''

(c) Error Flow Graph for C' and C'' (contd.)

(d) Error Flow Graph for D' and D'' (contd.)

Fig. 6. Error Flow Graphs

Starting with $C'_0(n_2,n_1,n_0)$ from the left branch of Figure 6a, it is multiplied with

$U_{64}(L-M)$ as shown between the edge of the first two nodes. Next, the error occurring in the previous operation is multiplied in the edge between second and third node. In the same way we can reach in the similar calculation for $D_0^{'}(n_2, n_1, n_0)$. When the two branches meet in the bottom node, they are subtracted from each other due to multiplication of the $D_0^{'}$ with -1 and this operation adds to the next error which is expressed as $1+\xi_{N,P}$. Here, (N, P) refers to $1024p_2 + 256p_1 + 64p_0 + 16n_2 + 4n_1 + n_0$. Now, this error is labeled as $E^{'}(p_2, p_1, p_0, n_2, n_1, n_0)$.

If the same calculation is repeated for $C_0^{''}(n_2, n_1, n_0)$ and $D_0^{''}(n_2, n_1, n_0)$, the error at the end is labeled as $E^{''}(p_2, p_1, p_0, n_2, n_1, n_0)$. But, this time the error functions are all additive not multiplicative. We now look into Figure 6c which is the continuation of Figure 6a to define the errors related with the six summations each having four iterations. As stated before, the error here is denoted as $1 + \lambda_i$ for floating-point and λ_i for fixed-point. The error calculation starts from $E^{'}(0, 0, 0, 0, 0, 0)$ and $E^{''}(0, 0, 0, 0, 0, 0)$ and the corresponding errors are multiplied or added till $E^{'}(3, 3, 3, 3, 3, 3)$ and $E^{''}(3, 3, 3, 3, 3, 3)$. At this point, we are left with two more errors. The branch starting with node label $\frac{1}{N}$ adds errors due to division operation, which are denoted as $\rho^{'}$ and $1+\rho$ for fixed-point and floating-point respectively. And then the same constant is multiplied with the rest of the what is calculated so far and adds another error denoted as $\tau^{'}$ and $1+\tau$ for fixed-point and floating-point respectively. In the end $C^{'}$ and $C^{''}$ are found as we calculated earlier.

The above discussion can be applied as is for the calculation of the imaginary part of floating-point and fixed-point model using the flow-graphs in Figure 6b and Figure 6d.

5.1 Formal Error Analysis in HOL

For implementing the above error analysis in HOL, we first construct complex numbers on reals similar to [12]. We define 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 use convenient abbreviations for the real (*Re*) and imaginary (*Im*) parts of a complex number, and also define arithmetic operations such as addition, subtraction, and multiplication on complex numbers. We overload the usual symbols $(+, -, \times)$ for $\mathbb{C}$ and $\mathbb{R}$. Similarly, we construct complex numbers on floating- and fixed-point numbers. Then we define 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 [10]. We specify expressions in HOL for expansion of a natural number into a binary form in normal and rearranged order. The above enables us to specify the IFFT-FFT combination algorithm in real (*REAL_IFFT_FFT*), floating- (*FLOAT_IFFT_FFT*), and fixed-point (*FXP_IFFT_FFT*) abstraction levels using recursive definitions in HOL as described in Equations (3) and (4). Then we define the real and imaginary parts of the IFFT-FFT algorithm (*IFFT_FFT_RE*, *IFFT_FFT_IM*) and powers of the principal N-roots on unity (*OMEGA_RE*,*OMEGA_IM*). Later, we prove 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 (5) and (6). Then we prove lemmas to introduce an error in each of the arithmetic steps in real and imaginary parts of the floating- and fixed-point IFFT-FFT algorithms according to the Equations (9) and (10). We prove these lemmas using the fundamental error analysis lemmas for basic arithmetic operations [2]. Then we define in HOL the error of the pth element of the floating- (*REAL_TO_FLOAT_IFFT_FFT_ERROR*) and fixed-point

(*REAL_TO_FXP_IFFT_FFT_ERROR*) IFFT-FFT algorithms at step q, and the corresponding error in transition from floating- to fixed-point (*FLOAT_TO_FXP_IFFT_FFT_ERROR*). Thereafter, we prove lemmas to rewrite the errors as complex numbers using the real and imaginary parts. Finally, we prove the following lemmas to determine the accumulation of roundoff error in floating- and fixed-point IFFT-FFT combination algorithm by recursive equations and initial conditions according to the Equations (11) to (17).

```
∀ x q0 q1 q2. ∃ f. (IFFT_FFT_REAL_TO_FP_ERROR x q0 q1 q2 =
complex_64 * complex_sum (0,4) (λp0. complex_sum (0,4) (λp1.
complex_sum (0,4) (λp2. complex_sum (0,4) (λn0. complex_sum (0,4)
(λn1. complex_sum (0,4) (λn2.
   ERROR_0 x n2 n1 n0 * OMEGA n0 n1 n2 p0 p1 p2 q0 q1 q2 +
   f n0 n1 n2 p0 p1 p2 q0 q1 q2)))))))) ∧

∃ t l d e z t' l'' d'' e'' z''. f n0 n1 n2 p0 p1 p2 q0 q1 q2 =
complex ( Val (float_Re ((λn0 n1 n2. float_complex_round (x n0 n1 n2)) n0 n1 n2)) *
                  Val (FLOAT_OMEGA_RE n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      ((1 + d n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      (1 + z n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      mul (ER_K n0 n1 n2 p0 p1 p2,4097 − ER_K n0 n1 n2 p0 p1 p2)
                      (λi. 1 + l i) * (1 + t) − 1) −
Val (float_Im ((λn0 n1 n2. float_complex_round (x n0 n1 n2)) n0 n1 n2)) *
                  Val (FLOAT_OMEGA_IM n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      ((1 + e n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      (1 + z n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      mul (ER_K n0 n1 n2 p0 p1 p2,4097 − ER_K n0 n1 n2 p0 p1 p2)
                      (λi. 1 + l i) * (1 + t) − 1),
Val (float_Re ((λn0 n1 n2. float_complex_round (x n0 n1 n2)) n0 n1 n2)) *
                  Val (FLOAT_OMEGA_IM n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      ((1 + d'' n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      (1 + z'' n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      mul (ER_K n0 n1 n2 p0 p1 p2,4097 − ER_K n0 n1 n2 p0 p1 p2)
                      (λi. 1 + l'' i) * (1 + t') − 1) −
Val (float_Im ((λq0 q1 q2. float_complex_round (x q0 q1 q2)) q0 q1 q2)) *
                  Val (FLOAT_OMEGA_IM n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      ((1 + e n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      (1 + z n0 n1 n2 p0 p1 p2 q0 q1 q2) *
                      mul (ER_K n0 n1 n2 p0 p1 p2,4097 − ER_K n0 n1 n2 p0 p1 p2)
                     (λi. 1 + l i) * (1 + t') − 1))

∀ X M V x q0 q1 q2. ∃ f'. (IFFT_FFT_REAL_TO_FXP_ERROR X M V x q0 q1 q2 =
           complex_64 *
            complex_sum  (0,4) (λp0. complex_sum (0,4) (λp1.
            complex_sum (0,4) (λp2. complex_sum (0,4) (λn0.
            complex_sum (0,4) (λn1. complex_sum (0,4) (λn2.
              ERROR'_0 X M V x n2 n1 n0 * OMEGA n0 n1 n2 p0 p1 p2 q0 q1 q2 +
              f' n0 n1 n2 p0 p1 p2 q0 q1 q2)))))) ∧
               ∃t' l' d' e' z' t''' l' d''' e''' z'''.
                f' n0 n1 n2 p0 p1 p2 q0 q1 q2 =
                   complex (d' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            e' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            z' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            sum (ER_K n0 n1 n2 p0 p1 p2,4096 −
                            ER_K n0 n1 n2 p0 p1 p2)(λi. l' i) + t',
                            d''' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            e''' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            z''' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            e' n0 n1 n2 p0 p1 p2 q0 q1 q2 +
                            sum (ER_K n0 n1 n2 p0 p1 p2,4096 −
                            ER_K n0 n1 n2 p0 p1 p2) (λi. l''' i) + t''')

∀ X M V x q0 q1 q2. IFFT_FFT_FP_TO_FXP_ERROR X M V x q0 q1 q2
= right−hand side of [ REAL to FP  error theorem ] −
  right−hand side of [ REAL to FXP error theorem ]
```

5.2 Discussion

The error analysis done above covers the OFDM rounding error analysis thoroughly between different number domains. To establish the complete theory of error analysis we

proved three main theorems with the help of formalized real and imaginary part of FFT-IFFT expansion and also the theorems related to the error for arithmetic operations. All definitions were derived heavily from existing theories, e.g., *realTheory*, *boolTheory*, *ieeeTheory*, *floatTheory*, *fxpTheory*, *wordTheory*, etc. There is a very strong relationship between mathematical models and their formal counterpart which might have been observed above. The definitions built on top of established theories in turn helped to build the FFT and IFFT components; which build the theory for the FFT-IFFT combinations. Then this theory is extended and the operators are overloaded for establishing the real, floating-point and fixed-point counterparts of the design using the *floatTheory* and *fxpTheory*.

For all the theorems and assumptions in the whole error analysis work it is imperative that higher-order logic be used. The error analysis is based on the floating-point and fixed-point theory of HOL, which are two of the most important additions in HOL's rich theory base. Besides quantification over variables and objects, there are many theorems in both theories that make use of quantification over functions. Moreover, almost all the definitions required to model the FFT-IFFT combination needed higher-order logic for the same reason. The error analysis functions of both floating-point and fixed-point (δ, ϵ, ξ, λ ,τ, and ρ) are all existentially quantified in the main theorems proved and these theorems construct the core of the final result.

Throughout the proof of the theories built-in tactics and tacticals were used. In many of these proofs case analysis and induction were used. Our main approach to prove the theorems was to have a rough paper and pencil sketch of the approach and then formalize it using the techniques available in the HOL tool. Many times it happened that it was hard to prove the theorem as a whole in one shot and then we break the goal in manageable size to prove the parts separately to combine later. To accomplish this in a different way sometimes theorems are assumed in the proof to concentrate in the core goal and later the assumed theorem is proved. Thus we prove the theorems till the final error analysis between floating-point to fixed-point. Through the course of the modeling and proof, many lemmas are developed, some are trivial but essential and some are crucial to move to the next step in establishing a theorem. But, it is important to mention that the current theorems can be proved in a better way which is realized gradually as we moved to much complicated proofs and so the latter proofs are better and concise than the previous ones.

Another important issue needs to be addressed and also equally applicable for all the theorems proved in Chapter 4 is that how it can be assured that the definitions created by the user themselves are sound and really characterize what the system user intends to formalize. In short, there is no way to verify that the modeling in HOL done by the user reflects the hardware exactly. The tool can check all the type requirements based on the initial information of the system provided by the user, and if these information are wrong then the final formalization will also be wrong. The HOL system is based on five axioms and eight primitive inference rules. All the HOL theories are built on top of them and this is another reason of the lengthy installation time required since all the built-in theories are to be proved before becoming part of the initial system. This is why there is no chance to have ambiguity in the proof system of HOL. Although highly improbable, but a wrong implementation can be verified against a wrong specification. Each and every possible scenario can happen. The tool itself might not be free from bugs. That is, however, why a tool like HOL needs expert users who have good knowledge of formal methods and also of the system under verification. The same can be told about the real RTL design in simulation

where only the functionality of the system can be verified but it can never be assured completely that the final product will exactly behave as the specification due to manufacturing difficiencies or other factors.

Since HOL is an interactive tool where the user needs to guide every step of the proof, it is also possible that the theorem prover can be guided to falsely proof a system. But, HOL strongly checks the type of the terms and functions entered into the system. This particular constraint also makes it very difficult to make simple mistakes in defining wrong theorems thus also answers partially the concern mentioned in the previous paragraph. Still, if any user wants to trick the tool to generate proof arbitrarily, he/she has to use *oracle* [15] mechanism that enables arbitrary formulas to become elements of the `thm` type. By use of this mechanism, HOL can utilize the results of arbitrary proof procedures. To avoid unsoundness, a tag is attached to any theorem coming from an oracle. This tag is propagated through every inference that the theorem participates in and if falsity becomes derived, the offending oracle can be found by examining the tags component of the theorem. A theorem proved without use of any oracle will have an empty tag, and can thus be considered to have been proved solely by deductive steps in the HOL logic. Thus, the tool ensures its security against misuse.

6 Conclusion

This paper is mainly concerned to demonstrate the use of formal verification techniques, here theorem proving, to verify an implementation of an OFDM modem based on the IEEE 802.11a physical layer standard for wireless communication. The OFDM design is fairly complex and some important design blocks were chosen for verification purposes. We formally modeled and verified the RTL blocks such as QAM, DQAM, S/P, and P/S against the corresponding specifications in the standard. We were able to find a bug in the QAM modulation block which implementation was diverted from the constellation provided in the IEEE standard specification.

We also analyzed the errors in the OFDM system occurring at the time of converting from one number domain to the other, for all three domains—ideal real, floating-point, and fixed-point numbers. We used the IFFT-FFT combination as a model for the error analysis of the whole system. Then we derived new expressions for the accumulation of round-off error in the OFDM system and proved the corresponding theorems in HOL. This formalization can be considered as a large application of the formal error analysis framework described before and shows the viability of such analysis even for larger scale systems as the one analyzed.

The future work that can be carried out pertaining to this paper might elucidate new and interesting ideas and some suggestions are following:

- Verifying the RTL implementation of the OFDM blocks by taking into account clock transitions and other timing constraints.
- Development of a parameterized error analysis pattern for any FFT or IFFT design of arbitrary computing point and radix.
- Performing statistical error analysis for the OFDM modem to find average and mean square errors for IFFT-FFT combination. To perform such an analysis mechanically, we need to use a formal theory on the properties of random variables and random pro-

cesses [13, 18].

- Verifying the OFDM system using a combination of HOL and another powerful computer algebra system such as Maple [26] or Mathematica [27].

References

[1] A. N. M. Abdullah. Formal Analysis and Verification of an OFDM Modem. Master's thesis, Department of ECE, Concordia University, Montreal, QC, Canada, 2006.

[2] B. Akbarpour. *Modeling and Verification of DSP Designs in HOL.* PhD thesis, Department of ECE, Concordia University, Montreal, QC, Canada, 2005.

[3] P. J. Ashenden. *Designer's Guide to VHDL.* Morgan Kaufmann, 2001.

[4] 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*, pages 129–156, North-Holland, 1992.

[5] Cadence Design Systems Inc. *Signal Processing Worksystems (SPW) User's Guide*, July 1999.

[6] A. L. Cinquino. A Real-Time Software Implementation of an OFDM Modem Suitable for Software Defined Radios. Master's thesis, Department of ECE, Concordia University, Montreal, QC, Canada, 2004.

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

[8] F. Frescura, S. Andreoli, S. Cacopardi, and E. Sereni. An OFDM Radio Transmitter based on TMS320C6000 DSP for Telemetry Applications. In *Proceeding of International Conference Signal Processing Applications and Technology*, Dallas, Texas, USA, October 2000.

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

[10] J. Harrison. Constructing The Real Numbers in HOL. *Formal Methods in System Design*, 5(1-2):35–59, 1994.

[11] J. Harrison. Floating Point Verification in HOL Light: the Exponential Function. Technical Report 428, University of Cambridge Computer Laboratory, Cambridge, UK, 1997.

[12] J. Harrison. Complex Quantifier Elimination in HOL. In *Supplemental Proceedings of Theorem Proving in Higher Order Logics*, pages 159–174. Edinburgh, UK, September 2001.

[13] O. Hasan. *Formal Probabilistic Analysis using Theorem Proving*. PhD thesis, Department of ECE, Concordia University, Montreal, QC, Canada, 2008.

[14] S. Haykin. *Communication Systems*. Wiley, 1994.

[15] HOL Sourceforge Project. *The HOL System Description*. September 2005.

[16] HOL Sourceforge Project. *The HOL System Reference*. September 2005.

[17] M. Huhn, K. Schneider, T. Kropf, and G. Logothetis. Verifying Imprecisely Working

Arithmetic Circuits. In *Proceedings of Design Automation and Test in Europe*, pages 65–69, Munich, Germany, March 1999.

[18] J. Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, Cambridge, UK, 2002.

[19] IEEE 802.11 Working Group. *IEEE Std 802.11a-1999*. The Institute of Electrical and Electronics Engineers,Inc, 1999.

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

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

[22] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation, Modelling Techniques and Tools*, LNCS 2324, pages 200–204. Springer-Verlag, 2002.

[23] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of the IEEE802.11 Wireless Local Area Network Protocol. In *Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, LNCS 2399, pages 169–187. Springer-Verlag, 2002.

[24] B. Liu and T. Kaneko. Error analysis of digital filters realized with floating-point arithmetic. *Proceedings of the IEEE*, 57:1735–1747, October 1969.

[25] F. Manavi. Implementation of OFDM Modem for the Physical Layer of IEEE802.11a Standard Based on XILINX VIRTEX-II FPGA. Master's thesis, Dept. of ECE, Concordia University, Montreal, QC, Canada, 2004.

[26] Maplesoft. Waterloo Maple Inc. http://www.maplesoft.com, 2006.

[27] Mathematica. Wolfram Research Inc. http://www.wolfram.com, 2006.

[28] R. V. Nee and R. Prasad. *OFDM for Wireless Multimedia Communications*. Artech House Publishers, 2000.

[29] A. Roy and K. Gopinath. Improved Probabilistic Models for 802.11 Protocol Verification. In *Computer Aided Verification*, LNCS 3576, pages 239–252. Springer-Verlag, 2005.

[30] M. Tariq, Y. Baltaci, T. Horseman, M. Butler, and A. Nix. Development of an OFDM based High Speed Wireless LAN Platform using the TI C6x DSP. In *Proccedings of IEEE International Conference on Communications*, pages 522–526, New York, NY, USA, May 2002.

[31] 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.

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

[33] W. Wong. Modeling Bit Vectors in HOL: The Word Library. In *Higher Order Logic and Its Applications*, LNCS 780, pages 371–384. Springer-Verlag, 1994.

[34] Xilinx Inc. High-performance 64-point complex fft/ifft v2.0, product specification. http://www.xilinx.com/ipcenter, 2000.

[35] Xilinx Inc. Xilinx Coregen Library. http://www.xilinx.com/ipcenter/coregen, 2005.