Enabling Hardware Verification through Design Changes

Amr T. Abdel-Hamid[1], Sofiène Tahar[1], and John Harrison[2]

[1] Concordia University, Montreal, Quebec, H3G 1M8 Canada
{at_abdel,tahar}@ece.concordia.ca
[2] Intel Corporation, Hillsboro, OR 97124, USA
johnh@ichips.intel.com

Abstract. The IEEE-754 floating-point standard, used in nearly all floating-point applications, is considered as one of the most important standards. Deep datapath and algorithm complexity have made the verification of such floating-point units a very hard task. Theorem proving, offers a good solution to handle such verification tasks. In this paper, we stress on the design changes performed for the sake of formalizing and verifying the IEEE-754 table-driven exponential function in all abstraction levels of the design flow. While verifying the VHDL code implementation against a high-level abstract specification, we were faced by two main problems: (1) the large abstraction gap between the two models; and (2) the flatness of the VHDL code, making it intractable to model and formally verify. We have therefore proposed a hierarchical methodology to solve such modeling problem, and experimented it on our verification task using the HOL theorem proving environment.

1 Introduction

Designs are getting larger and larger, more complex every day. Simulation, although widely used, could never give the verification coverage needed. There are two full coverage approaches, brute-force and special-purpose simulation, which fail to give even a fair coverage ratio for a moderate design and also less accurate, as sometimes faults occur where they are least expected. It is becoming clear that the "quality" of the validation achieved by traditional simulation is rapidly deteriorating as the VLSI technology progresses.

Formal verification methods [10] have sometimes been accused of a lack of ability to get into a whole industrial product design cycle. Working on the same design path of most electronic products, we discuss in this paper the design changes performed for the sake of formalizing and verifying the IEEE-754 table-driven exponential function at all abstraction levels of the design flow. The Floating-point functions are a main building block in nearly any modern microprocessor. As an example of such circuits, we used the IEEE-754 exponential function with symmetric 32 bits for both input and output. The IEEE-754 exponential function was specified first formally in [7]. This behavioral specification was written in a high level *while language*, and was intended mainly to be verified against a more abstract mathematical description of the exponential function [12]. Starting from this behavioral specification Bui *et al.* [2] developed an RTL (Register Transfer Level) implementation of the design using VHDL. We were also interested in the development of a formal proof that a synthesized logic gate implementation also implies this RTL design.

In this exercise, we were faced by two problems: (1) the main intend of the specification, written in the *while language*, was to prove it against a more abstract arithmetic specification. This caused a large gap between this specification and the implementation it should be verified against. As the mathematical nature of the behavioral model was clearly unrelated to the lower level VHDL implementation. Unlike the implementation, we had to formalize the specification without making any kind of changes in it to insure system integrity. So, to solve such a problem we introduced a new level between the specification and the implementation, called *modular* behavioral specification, which will be discussed afterwards throughly in the paper; (2) the VHDL code provided was too flat to be modeled efficiently. Even if we succeed such a modeling, it would be too large to be verified. This problem was solved by reconstructing the RTL implementation in a more modular manner trying to minimize the interconnections between such modules.

In this work, we use the HOL theorem proving system [6] for specifying and verifying the floating-point design at hand. The HOL theorem prover is an interactive proof assistant for higher-order logic developed at Cambridge University by Gordon *et al.* [6]. It was explicitly designed for the formal verification of hardware, though it has also been applied to other areas including software verification and formalization of pure mathematics.

There exists very few related work in the open literature on the impact of design changes on the formal verification process. For instance, Curzon *et al.* [5] introduced this idea in the context of interactive proof. By using the HOL theorem prover, they suggested that the cost of verification of an ATM (Asynchronous Transfer Mode) switch in terms of time can be reduced by making appropriate design changes. For example, they redesigned the arbiter module to reset correctly if the cells and frame start occur close together. The authors noted the factors that were increasing the verification cost, where particular aspects of the behavioral specification were lengthening the verification task by a significant amounts. Zobair *et al.* [13] used a similar approach to investigate whether the same design changes would also reduce the verification cost while using the MDG tools [4]. Both of the above works have shown that design modifications and redesign for verifiability would have a significant effect on the speed and effort of the verification task. They also defined some design change rules similar to those discussed in this paper.

Other related work to ours is dealing with the verification of the floating-point algorithms and designs in general. For instance, Miner [11] formalized the IEEE-854 floating-point standard in PVS. He used this formalization to verify abstract mathematical descriptions of the main real operations and their relation to the corresponding floating-point implementations. Carreno [3] defined and then formalized the same IEEE-854 standard in HOL. The most related work among these efforts, however, is the one of Harrison [8] who defined and formalized real numbers using HOL. Then he developed a generic floating-point library to define and verify the most fundamental terms and lemmas of IEEE-754 standard [7]. This former library was used by him to formalize and verify floating-point algorithms against behavioral specification of the square root [9] and the exponential function [7].

The organization of the paper is as follows. We will start in Section 2 by describing the Table-Driven exponential function algorithm, whose formal specification and implementation are discussed throughout this paper. Section 3 introduces our methodology and shows how hierarchical and modular techniques were used to ease the verification task. Section 4 shows the specification formalization, and discusses the new modular specification level added as well as the design changes done on the implementation for easing the verification. Section 5 describes the formal verification process and summarizes the experimental results. Finally, conclusions are drawn in Section 6.

2 The IEEE-754 Exponential Function

In this section, we give an introduction to the IEEE-754 exponential function whose formal specification and design are discussed in the rest of the paper.

Using an approximate polynomial expansion, Tang [12] has developed an algorithm for computing the floating-point exponential function using what he calls a *Table-Driven* approach. In this approach, the input is first reduced to a certain working range, $[-log\ 2/2^{L+1}, log\ 2/2^{L+1}]$, where L is an integer larger than or equal to 1, chosen beforehand, (for instance, L = 4 for single precision [12]). Then this input x is considered to be composed of:

$$x = \frac{(32 * m + j) * (log 2)}{32} + (r1 + r2)$$

where m and j are integers, and $r1$ and $r2$ are real numbers, $\mid r1 + r2 \mid < ((\log 2)/64)$.

Starting from this equation, the exponential function can be constructed as follows:

$$x = (m * log 2) + \frac{(j * log 2)}{32} + r$$

where $r = r1 + r2$. The exponential of x will hence be equal to

$$exp(x) = exp((m * log 2) + \frac{(j * log 2)}{32} + r)$$

$$= 2^m + 2^{\frac{j}{32}} + exp(r)$$

The main objective of the algorithm is to isolate m and j, and to evaluate the approximating polynomial. According to Tang, four steps are needed to compute this exponential function:

Step 1. Filter any out-of-bounds inputs that occur. As mentioned before, x should be a number between $[-(log2/32), (log2/32)]$.

Step 2. Start the computation by first calculating N,

$$N = INTEGER(X * INVL)$$

where $INVL$ is a floating-point constant approximately equal to $32/log2$ in our case, and INTEGER is the default IEEE-754 round-to-nearest mode. This N is composed of two parts,

$$N = N1 + N2$$

where $N1 = 32 * m$, and $N2 = j$

So, the variables m and j can be derived from the previous result as follows:

$$j = N2$$

$$m = \frac{N1}{32}$$

Given the value of N, $r1$ and $r2$ can be calculated as discussed in [12].

Step 3. Compute the polynomial $p(r)$, similar to the Taylor expansion showed above.

Step 4. The values of $2^{j/32}$, $j = 0, 1, ...32$, are calculated beforehand and are represented by two working-precision numbers (single precision in our case), $Slead$ and $Strail$. The sum approximates $2^{j/32}$ to roughly double the working precision. Finally $exp(x)$ is calculated as follows:

$$S = Slead(j) + Strail(j)$$

$$exp(x) = 2^m + (Slead(j) + (Strail(j) + S * p(r)))$$

3 Modeling and Verification Methodology

The verification process for the Table-Driven exponential function will be performed on many levels. Harrison [7] formalized and verified that a behavioral specification, an abstract algorithmic description he developed for the design, implies an abstract mathematical description of the IEEE-754 Table-Driven floating-point exponential function [12]. Starting from this behavioral specification, written in a *while-language*, Bui *et al.* [2] developed an RTL (Register Transfer Level) implementation of the design. The goal of this work is the modeling and verification of the latter implementation with respect to the behavioral specification designed by [7] using HOL. We were also interested in the development of a formal proof that the logic gate implementation, machine synthesized using the Synopsys tool, implies the RTL implementation.

As soon as we started the verification project, we noticed that there was a large abstraction gap between the high level behavioral specification and the RTL implementation. Trying to build the formal proof between the two levels, we found that such a gap cannot be bridged in one move. Another problem was that the flat behavioral specification itself cannot be modified or changed to facilitate the verification task.

On the other hand, it was noticed that the VHDL RTL implementation was very flat to be verified. The very high number of small modules and connections would make the modeling of such design in HOL a very long process, and it will nearly make the verification against the specification intractable. For these reasons, we rebuilt the RTL implementation of the design to a modular version. The code was redesigned into a smaller number of sub-modules to achieve the partitions needed for the modeling process. These modules were chosen to meet the following criteria:

1) *Self containing*: meaning the verification of one module does not affect the others in the same environment except by the input and output signals.

2) *Minimal communicating signals*: where the interfaces between such modules were minimized and kept as simple as possible. This would lead to extending the self containing property above, as well as less signals as possible in the heavy populated system level design.

Encountering the above problems, we introduced a new specification level, *the modular behavioral specification*. This level was placed between the modular RTL implementation and the abstract (flat) behavioral specification as shown in Figure 1 below. This new specification level was modeled using a closer modular composition as of the lower RTL design, yet its modules were described internally at a higher abstraction level as used in the original (flat) behavioral specification. It hence acted as the bridge that shorten the distance between the two design levels. Given the proposed design changes, the overall modeling and verification process is described in Figure 1, where the shaded boxes are the material provided by [7], and [2], while the white ones represent those developed in this work.

In Figure 1, we modeled the behavioral specification provided in [7] in HOL. We then proved that the modular behavioral specification implies the higher level abstract specification adopted from the one built by Harrison. We then built the modular VHDL code of the RTL level, after applying the design changes mentioned above. We then modeled this RTL implementation in HOL. Furthermore, we proved hierarchically that the synthesized gate level implementation implies the modular RTL description. Finally, all proof stages were linked in a single global proof covering the whole design cycle of the floating-point exponential function. In the next sections, we will describe the formal specification and verification of the IEEE exponential function design based on the proposed approach, respectively..

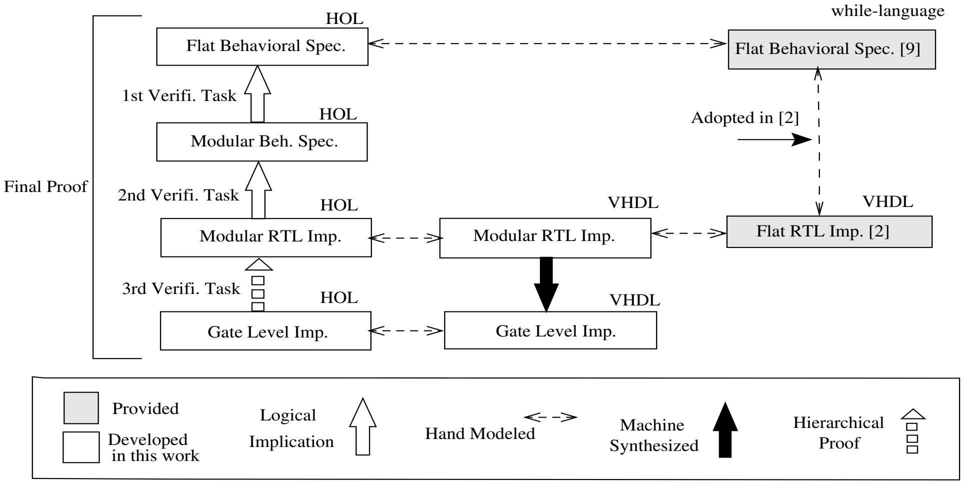

Fig. 1. The exponential function verification project

4 Formal Specification of IEEE-754 Exponential Function

In this section, we focus on the formal specification of the IEEE-754 exponential function through out the different abstraction levels. Due to space limitations, we will give a sketch of the main design change issues, while further specification details can be found in [1].

As mentioned earlier, the specification introduced in [7] was intended mainly to verify a higher level mathematical implementation of the design. This provided a large gap between this specification and the implementation it should be verified against. Unlike the implementation, we had to formalize the specification, written originally in *while language* (Figure 2),without making any changes in it to insure system integrity.

To introduce a modular behavioral specification, we have divided this specification into six intermediate blocks (modules), similar to those used for the RTL implementation, where the conjunction of these blocks (Figure 3) represents the full specification of the code described in Figure 2. It should be noted, that the mathematical operators (such as *, +, and /) in Figure 2 are bit vector operations and not mathematical real numbers operators. Trying to achieve maximum modularity for the design, we have tried to minimize the interfaces between different modules. This helped us to divide the verification tasks into well-defined smaller ones. Each of these blocks was also divided into smaller specifications giving us smaller sub-specifications clearly related to the goals needed to be proved.

The six modules composing the new level are completely responsible for checking the input X value and computing the result of the exponential function (see Figure 3). These blocks are:

```
Int_32 = Int(32)
Int_2e9 = Int (2 EXP 9)
Int_2e9 = Int (2 EXP 9)
Plus_one = float (0, 127, 0)
THRESHOLD_1 = float (0, 134, 6066890)
THRESHOLD_2 = float (0, 102, 0)
Inv_L = float (0, 121, 3240448)
L1 = float (0, 121, 3240448)
L2 = float (0, 102, 4177550)
A1 = float (0, 126, 68)
A2 = float (0, 124, 2796268)
var x:float, E:float, R1:float, R2:float, R:float, P:float,
Q:float,S:float, E1:float, N:Int, N1:Int, N2:Int, M:Int, J:Int;
```

/* Checking block */

```
if Isnan (X) then E:= X
else if X == Plus_infinity  then E:= Plus_infinity
else if X == Minus_infinity then E:= Plus_Zero
else if (abs(x) > THRESHOLD_1 then
if X > Plus_Zero then E:= Plus_infinity
else E:= Plus_Zero  else if abs(X) < THRESHOLD_2 then E:= Plus_one + X
else ( N:= INTRND (X * Inv_L);
```

/* m and j computing block */

```
N2:= N \% Int_32;
N1:= N - N2;  M:= N1 / Int_32;   J:= N2;
```

/* r1 and r2 block */

```
if abs (N) Int_2e9 then
R1:= (X-Tofloat(N1) * L1) - Tofloat (N2) * L1
else   R1:= X - Tofloat(N) * L1;
R2:= Tofloat(N) * L2;
R:= R1 + R2;
Q:= R * R (A1 + R * A2);
p(r) block P:= R1 + (R2 + Q);
```

/* Slead and Strail block */

```
S:= S_Lead(J) + S_Trail(J);
```

/* Exponent calculation block */

```
E1:= S_Lead(J) + (S_Trail(J) + S * P);
```

Fig. 2. Behavioral specification of the exponential function in *while language* [7]

1) *m and j computing block* (M_J_SPEC): responsible for half of Step 2 (cf. Section 2) by computing the values of m and j. Its input is the number X and its outputs are J, M, N, and $N1$.

2) *r1* and *r2* block ($R1\_R2\_SPEC$): responsible for the second half of Step 2, it computes the values of $r1$ and $r2$. Its inputs are X, $N1$, $N2$ (equal to J) and its outputs are the two floating-point numbers $R1$ and $R2$.

3) *p(r) block* (P_R_SPEC): computes the value of $p(r)$. It takes $R1$ and $R2$ and outputs P_R.

4) *Slead and Strail block* (Ge_J_SPEC): a floating-point multiplexer module, where the value of J decides which values for Strail and Slead should be chosen. Its input is just the number J and its outputs are Slead and Strail.

5) *Exponent calculation block* (Exp_Cal_SPEC): this is the main computational block where the exponential function is computed. It takes $Slead$, $Strail$, M and $P(R)$ as its inputs and outputs $EXP(X)$.

6) *Checking block* ($Compare\_SPEC$): the compare and decision module. According to the value of input X, this module decides whether to choose the computed value of the exponent or another output as NAN (Not-A-Number). It takes X and the computed exponent as its inputs and outputs $EXP(X)$.

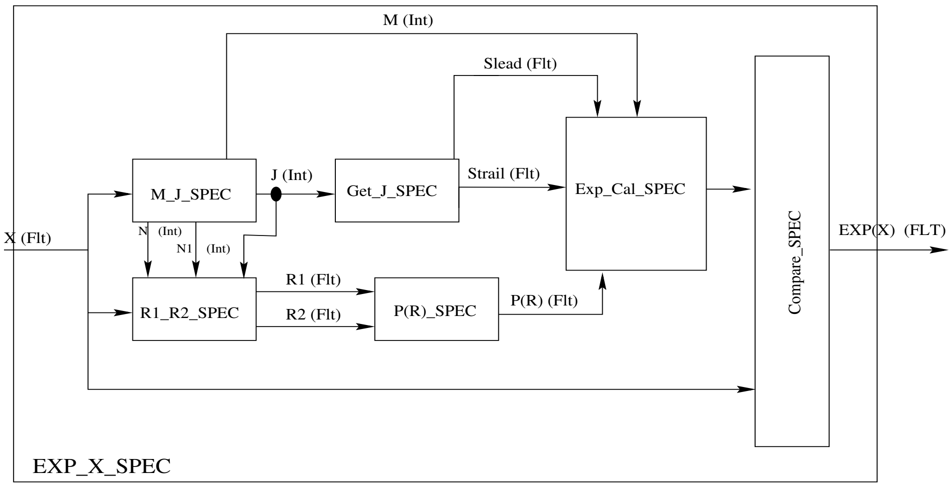

Fig. 3. Modular organization of the exponential function specification

The top level of this system specification in HOL looks as follows:

```
⊢def  IEEE_EXP_SPEC  Xs Xe Xm  EXP_s EXP_e EXP_m =
∃  Ns Ne Nm  Ms Me Mm  Js Je Jm  N1s N1e N1m  R1s R1e R1m...
(M_J_SPEC Xs Xe Xm  Ns Ne Nm  Ms Me Mm  Js Je Jm..)∧
(R1_R2_MOD_SPEC  Xs Xe Xm  Ns Ne Nm ..)∧
(Get_J_SPEC Js Je Jm  Strail_s Strail_e Strail_m ..)∧
(P_R_SPEC  R1_s R1_e R1_m R2_sR2_eR2_mPR_sPR_ePR_m)∧
(EXP_CAL_MOD_SPEC  Strail_s Strail_e Strail_m ..)∧
(Compare_SPEC  EXP_1_s EXP_1_e EXP_1_m  Xs Xe Xm  EXP_s EXP_e EXP_m)
```

In line with the behavioral specification, we also adopted a modular modeling of the IEEE-754 exponential function implementation. In fact, since the original VHDL implementation code developed by Bui *et al.* [2] was too flat as well, we performed a number of design changes, which keep the same code properties but make the design easier to model and verify. We have aimed mainly in our changes to reduce the logic complexity by introducing a hierarchical (modular) design architecture which reduced the complexity of the circuit. Also, in some modules the code has been changed to perform the same function, while being less complex.

Thanks to the above design changes, it became easier to move the new VHDL code into HOL notation. The HOL, top level model of our implementation is given as follows, which is now nearly a one-to-one mapping from the new VHDL model:

```
⊢def  IEEE_EXP_IMP Xs Xe Xm  EXP_s EXP_e EXP_m =
∃ Ns Ne Nm  Ms Me Mm  Js Je Jm  N1s N1e N1m...
(M_J_IMP Xs Xe Xm  Ns Ne Nm  Ms Me Mm  Js Je Jm  N1s N1e N1m) ∧
(R1_R2_MOD_IMP Xs Xe Xm  Ns Ne Nm  N1s N1e N1m ..) ∧
(Get_J_IMP Js Je Jm  Strail_s Strail_e Strail_m ..) ∧
(P_R_IMP R1_s R1_e R1_m  R2_s R2_eR2_m  PR_s PR_e PR_m) ∧
(EXP_CAL_MOD_IMP Slead_s Slead_e Slead_m ..) ∧
(Compare_IMP EXP_1_s EXP_1_e EXP_1_m Xs Xe Xm  EXP_s EXP_e EXP_m)
```

The above RTL modules show obviously one-to-one correspondence with the proposed modular behavioral specification. Moreover, there is a high level of regularity in these six modules where floating-point operations, like addition, multiplication, are the main sub-modules in all of them. This will help us in the reuse of the developed models and theories in building the lower levels of such specification. Each of the six modules was modeled further as a conjunction of lower level components.

After fully modeling the specification and implementation of the system, the next task was the verification process, which will be discussed in the next section along with the experimental results. The high similarity between the implementation at high levels and the specification considerably eased the verification task.

5 Formal Verification of the Exponential Function

So far, we modeled in HOL the IEEE-754 Table-Driven exponential function at different levels of abstraction, including the introduction of modular layers. The next step is to verify these different levels using a hierarchical proof approach in HOL.

Let X be the input and EXP the exponential output, our final goal is:

$$\begin{array}{l} \forall\ X\ \ EXP. \\ \qquad assuming\ \ \frac{-log2}{32} \leq X \leq \frac{log2}{32}. \\ \qquad (EXP\_GATE\_IMP\ \ X\ \ EXP\ \ \Longrightarrow\ \ EXP\_BEH\_SPEC\ \ X\ \ EXP) \end{array}$$

This goal implies that if X meets the boundry conditions specified, $\frac{-log2}{32} \leq X \leq \frac{log2}{32}$, then the gate implementation, EXP_GATE_IMP, should imply the high level behavioral specification, EXP_BEH_SPEC.

Obviously, this goal cannot be reached directly, due to the very high abstraction gap between the gate and behavioral levels as described above. Using the design changes done, the proof scheme was changed to hierarchically prove that the gate level implies the more abstract RTL. Then the new introduced modular specification was proved to imply the higher level abstract specification modeled in HOL. The RTL afterwards, was related, by a formal proof, to the modular behavioral specification. This latter proof gave us the ability to prove the whole design path of the circuit, starting from the gate level implementation of the design up to the behavioral specification proposed in [7]. The assumption that X in its boundaries is made by the boundry module within both the specification and the implementation. So the goal would be the following:

$$\begin{array}{l} \vdash_{thm}\ \forall X\ \ EXP.\ IEEE\_EXP\_GATE\_IMP\ \ X\ \ EXP \\ \qquad\qquad \Longrightarrow\ \ IEEE\_EXP\_RTL\_IMP\ \ X\ \ EXP \end{array} \quad (1)$$

$$\begin{array}{l} \vdash_{thm}\ \ \forall X\ \ EXP.\ IEEE\_EXP\_RTL\_IMP\ \ X\ \ EXP \\ \qquad\qquad \Longrightarrow IEEE\_EXP\_MOD\_BHV\_SPEC\ \ X\ \ EXP \end{array} \quad (2)$$

$$\begin{array}{l} \vdash_{thm}\ \ \forall X\ \ EXP.\ IEEE\_EXP\_MOD\_BHV\_SPEC\ \ X\ \ EXP \\ \qquad\qquad \Longrightarrow IEEE\_EXP\_BHV\_SPEC\ \ X\ \ EXP \end{array} \quad (3)$$

Finally using equations $(1), (2)$ and (3), we can reach the final goal stated again in equation (4):

$$\begin{array}{l} \vdash_{thm}\ \ \forall\ X\ \ EXP.\ IEEE\_EXP\_GATE\_IMP\ \ X\ \ EXP \\ \qquad\qquad \Longrightarrow IEEE\_EXP\_BHV\_SPEC\ \ X\ \ EXP \end{array} \quad (4)$$

A summary of the verification times for the whole system is given in Table 1. All experiments have been carried out on a Sun Ultra SPARC 2 workstation with a 296 MHz processor and 768 MB of memory. In the table, we have showed the verification time of two main building blocks, the floating-point adder and multiplier. These modules took a very high verification time, but due to the high reusability of pre-proven theorems and lemmas, other building blocks and even the main module required much less verification time. Also, it can be seen that the verification time for four of the six main modules are much smaller, as all these modules' building blocks were pre-proved, making the final task shorter. The sum of the times of the systems showed here will be less than the total verification time as there were some lemmas verified to achieve the final proof goal. The whole code was composed of nearly 4600 lines.

Table 1. Verification times of different system modules

Module Name	**Verification Time (sec.)**
Floating-Point Addition	60.50
Floating-Point Multiplication	30.54
M_J Module	2.12
R1_R2 Module	5.62
P_R Module	3.42
EXP_Cal Module	1.97
IEEE_EXP Module	5.29
Total Verification Time	**214.20**

6 Conclusions

Most verification and testing tools will fail short to verify a circuit with a deep datapath. The IEEE-754 Table-Driven exponential function with its 32 bit input and 32 bit output implementation would be considered an impossible task for exhaustive simulation. Model Checking techniques will not go a lot further as the deep datapath means a huge state space causing a *state space explosion* problem. Also, the main module and most of its sub-modules properties cannot be covered easily with, e.g., CTL properties.

In this paper, we have demonstrated the use of HOL to model the behavioral and RTL specifications for the IEEE-754 Table-Driven exponential function in a modular form. We have showed that adding new levels in the design path, as in the case of the intermediate behavioral specification, or changing the design to modular blocks, would ease both the modeling and verification processes. Finally, using *hierarchical verification*, we have been able to develop a formal proof indicating the correctness of the implementation using the HOL tool.

The main lessons for the design changes enabling formal verification are:

1) Modules performing the same function or highly interacting should be placed in the same module to minimize system blocks interfaces.

2) Create a boundry at the system, where the number of sub-modules in the main modules should not exceed it. As this would complicate the verification of the blocks afterwards.

3) Create easily localized modules, so that the module could be redesigned with minimal changes in the system verification task, as well as minimizing the main connections to other modules.

One of the very important advantages of the hierarchical verification lies in the fact that the change of a module or more will not mean the re-proof of the whole system. It only means the re-proof that the new module meets the same specification that the older version did. This may mean a lot for a tight time-to-market in a fast moving technology like microelectronics. As an example, our proof can always be used with the changing technology as long as we prove that the lower modules, gates for instance, are still satisfying the same properties.

References

1. A.T. Abdel-Hamid. A Hierarchical Verification of the IEEE-754 Table-Driven Floating-Point Exponential Function using HOL. M.A.Sc. Thesis, Concordia University, Department of Electrical and Computer Engineering, Montreal, Canada, April 2001.
2. H.T. Bui, B. Khalaf, and S. Tahar. Table-Driven Floating-Point Exponential Function. Proc. IEEE Canadian Conference on Electrical & Computer Engineering, May 1999, pp. 450-455.
3. V.A. Carreno. Interpretation of IEEE-854 Floating-Point Standard and Definition in the HOL system. NASA Technical Memorandum 110189, September 1995.
4. F. Corella, Z. Zhou, X. Song, M. Langevin and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. Formal Methods in System Design, vol. 10, February 1997, pp. 7-46.
5. P. Curzon and I. Leslie. Improving Hardware Designs whilst Simplifying their Proof. In Designing Correct Circuits, Workshops in Computing, British Computer Society, 1996.
6. M.J.C. Gordon and T.F.Melham. Introduction to HOL: a Theorem Proving Environment for Higher-order Logic. Cambridge University Press, 1993.
7. J.R. Harrison. Floating Point Verification in HOL Light: The Exponential Function. Technical Report number 428, University of Cambridge Computer Laboratory. UK, June 1997.
8. J.R. Harrison. Theorem Proving with the Real Numbers. Technical Report number 408, University of Cambridge Computer Laboratory, December 1996.
9. J.R. Harrison. Verifying the Accuracy of Polynomial Approximations in HOL. In Theorem Proving in Higher Order Logics, LNCS 1275, Springer Verlag, 1997, pp. 137-152.
10. C. Kern, and M. R. Greenstreat. Formal verification in Hardware Design: A Survey. ACM transactions on Design Automation of Electronic Systems, vol. 4, no. 2, April 1999, pp. 123-193.
11. P.S. Miner. Defining the IEEE-854 Floating-Point Standard in PVS. NASA Technical Memorandum 110167, June 1995.
12. P.T.P. Tang. Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic. ACM Transactions on Mathematical Software, vol. 15, no. 2, 1989, pp. 144-157.
13. M. Hasan, S. Tahar, and P. Curzon. Impact of Design Changes on Verification Using MDGs. Proc. IEEE Canadian Conference on Electrical & Computer Engineering, Halifax, Nova Scotia, Canada, May 2000, pp. 173-178.