Verification of Floating-Point Adders **

Yirng-An Chen and Randal E. Bryant
yachen+@cs.cmu.edu, bryant+@cs.cmu.edu
Computer Science Dept., Carnegie Mellon Univ., Pittsburgh, PA 15213

Abstract. In this paper, we present a "black box" version of verification of FP adders. In our approach, FP adders are verified by an extended word-level SMV using reusable specifications without knowing the circuit implementation. Word-level SMV is improved by using Multiplicative Power HDDs (*PHDDs), and by incorporating conditional symbolic simulation as well as a short-circuiting technique. Based on a case analysis, the adder specification is divided into several hundred implementation-independent sub-specifications. We applied our system and these specifications to verify the IEEE double precision FP adder in the Aurora III Chip from the University of Michigan. Our system found several design errors in this FP adder. Each specification can be checked in less than 5 minutes. A variant of the corrected FP adder was created to illustrate the ability of our system to handle different FP adder designs. For each adder, the verification task finished in 2 CPU hours on a Sun UltraSPARC-II server.

1 Introduction

The floating-point (FP) division bug [10] in Intel's Pentium processor and the overflow flag erratum of the FIST instruction (FP to integer conversion) [12] in Intel's Pentium Pro and Pentium II processors have demonstrated the importance and the difficulty of verifying FP arithmetic circuits and the high cost of an arithmetic bug. FP adders are the most common units in FP processors. Modern high-speed FP adders [17] are very complicated, because they require many types of modules: a right shifter for alignment, a left shifter for normalization, a leading zero anticipator (LZA), an adder for mantissas, and a rounding unit.

Formal verification or exhaustive simulation can be used to ensure the correctness of FP adders. However, it is impossible to perform exhaustive simulations for a floating-point adder. Formal verification techniques such as theorem proving and model checking have been used to verify arithmetic circuits. Most of the IEEE FP standard has been formalized by Carreño and Miner [4] for the HOL and PVS theorem provers. To verify arithmetic circuits, theorem provers require users to guide the proof which is structured as series of lemmas describing the effect of circuit modules and their interactions [1]. Thus, the verification process is very tedious and implementation-dependent. After the famous Pentium division bug [10], Intel researchers applied word-level SMV [9] with Hybrid Decision Diagrams (HDDs) [8] to verify the functionality of the FP unit in one of Intel's processors [7]. Due to the limitations of HDDs, the FP adder was partitioned into several sub-circuits to be verified. The correctness of the overall circuit had to be ascertained manually from the verified specifications of the sub-circuits. This partitioning approach requires user intervention and thus could be error prone. Moreover, the specifications for the partitions are highly dependent on the circuit implementation.

** This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract number DABT63-96-C-0071.

To the best of our knowledge, only two types of arithmetic circuits can be verified by treating them as *black boxes* (i.e., the specifications contain only the inputs and outputs). First, an integer adder can be verified by using Binary Decision Diagrams (BDDs) [2]. Second, Hamaguchi *et al* [13] presented the verification of integer multipliers without knowing their implementations using Multiplicative Binary Moment Diagrams (*BMDs) [3]. However, their approach does not work for incorrect designs, because the *BMDs explode in size and counterexamples can not be generated for debugging. None of the previous approaches can verify FP adders without knowing their circuit implementations.

In this paper, we present a black box version of verification of FP adders. In our approach, a FP adder is treated as a black box and is verified by an extended version of word-level SMV with reusable specifications. Word-level SMV is improved by using Multiplicative Power HDDs (*PHDDs) [5] to represent the FP functions, and by incorporating *conditional symbolic simulation* as well as a *short-circuiting* technique. The FP adder specification is divided into several hundred sub-specifications based on the sign bits and the exponent differences. These sub-specifications are implementation-independent, since they use only the input and output signals of FP adders.

The concept of conditional symbolic simulation is to perform the symbolic simulation of the circuit with some conditions to restrict the behavior of the circuit. This approach can be viewed as dynamically extracting circuit behavior under the given conditions without modifying the actual circuit. Can we verify the specifications of FP adders using conditional symbolic simulation, avoiding any use of circuit knowledge? We identify a conflict in variable orderings between the mantissa comparator and mantissa adder, which causes the BDD explosion in conditional symbolic simulation. A short-circuiting technique to overcome this ordering conflict problem is presented and integrated into word-level SMV package. In general, this short-circuiting technique can be used when different parts of the circuit are used under different operating conditions.

We used our system and these specifications to verify the FP adder in the Aurora III Chip [14] at the University of Michigan. This FP adder is based on the design described in [17], and supports IEEE double precision and all 4 IEEE rounding modes. In this verification work, we verified the FP adder only in the round-to-nearest mode, because we believe that this is the most challenging rounding mode for verification. Our system found several design errors. Each specification can be checked in less than 3 minutes or 5 minutes including counterexample generation. A variant of the corrected FP adder was created and verified to illustrate the ability of our system to handle different FP adder designs. For each FP adder, verification took 2 CPU hours. We believe that our system and specifications can be applied to directly verify other FP adder designs and to help find design errors.

The overflow flag erratum of the FIST instruction (FP to integer conversion) [12] in Intel's Pentium Pro and Pentium II processors has illustrated the importance of verification of the conversion circuits which convert the data from one format to another format (e.g., IEEE single precision to double precision). Since these circuits are much simpler than FP adders and only have one input operand, we believe that our system can be used to verify the correctness of these circuits.

2 *PHDD Overview

Chen and Bryant [5] introduced a representation, called Multiplicative Power HDDs (*PHDDs), to provide a compact representation for integer and floating-point functions. For expressing function g from Boolean variables to integer or floating-point values, *PHDDs use one of three decompositions of a function with respect to an input variable x:

$$g = \langle w, f \rangle = \begin{cases} c^w \cdot ((1-x) \cdot f_{\overline{x}} + x \cdot f_x) & (Shannon) \\ c^w \cdot (f_{\overline{x}} + x \cdot f_{\delta x}) & (Positive\ Davio) \\ c^w \cdot (f_x + (1-x) \cdot f_{\delta \overline{x}}) & (Negative\ Davio) \end{cases}$$

where $\langle w, f \rangle$ denotes $c^w \cdot f$, and $\cdot$, $+$ and $-$ denote multiplication, addition and subtraction, respectively. Term f_x ($f_{\overline{x}}$) denotes the 1- (0-) cofactor of f with respect to variable x, i.e., the function resulting when the constant 1 (0) is substituted for x. Term $f_{\delta x} = f_x - f_{\overline{x}}$ is called the linear moment of f with respect to x. This terminology arises by viewing f as a linear function with respect to its variables, and thus $f_{\delta x}$ is the partial derivative of f with respect to x. Similarly, Term $f_{\delta \overline{x}}$ is $f_{\overline{x}} - f_x$.

In general, the constant c can be any positive integer. Since the base value of the exponent in the IEEE floating-point (FP) format is 2, we will consider only $c = 2$ for the remainder of this paper. Observe that w can be negative, allowing the representation of rational numbers. The power edge weights enable us to represent functions mapping Boolean variables to FP values. To the best of our knowledge, *PHDD is the only decision diagram that can represent integer or floating-point functions efficiently. Readers can refer to [5] for more details of FP representation using *PHDDs. In this verification work, the output Boolean vector of a FP adder are converted into word-level functions represented by *PHDDs using a method similar to one described in [3]. Thus, the specifications of FP adders can be expressed in word-level functions using *PHDDs.

3 Floating-Point Adders

Let us consider the representation of FP numbers by IEEE standard 754. Double-precision FP numbers are stored in 64 bits: 1 bit for the sign (S_x), 11 bits for the exponent (E_x), and 52 bits for the mantissa (N_x). The exponent is a signed number represented with a bias (B) of 1023. The mantissa (N_x) represents a number less than 1. Based on the value of the exponent, the IEEE FP format can be divided into four cases:

$$\begin{cases} (-1)^{S_x} \times 1.N_x \times 2^{E_x - B} & If\ 0 < E_x < All\ 1\ \ (normal) \\ (-1)^{S_x} \times 0.N_x \times 2^{1-B} & If\ E_x = 0\ \ (denormal) \\ NaN & If\ E_x = All\ 1\ \&\ N_x \neq 0 \\ (-1)^{S_x} \times \infty & If\ E_x = All\ 1\ \&\ N_x = 0 \end{cases}$$

where NaN denotes Not-a-Number and ∞ represents infinity. Let $M_x = 1.N_x$ or $0.N_x$. Let m be the number of mantissa bits including the bit on the left of the binary point and n be number of exponent bits. For IEEE double precision, m=53 and n=11.

Due to this encoding, an operation on two FP numbers cannot be rewritten as an arithmetic function of the two inputs. For example, the addition of two FP numbers X (S_x, E_x, M_x) and Y (S_y, E_y, M_y) can not be expressed as $X + Y$, because of special cases when one of them is NaN or $\pm\infty$. Table 1 summarizes the possible results of the FP addition of two numbers X and Y, where F represents a normalized or denormalized number. The result can be expressed as $Round(X + Y)$ only when both operands have normal or denormal values. Otherwise, the result is determined by the case. When one

operand is $+\infty$ and the other is $-\infty$, the adder should raise an invalid arithmetic operand exception.

		Y			
	+	$-\infty$	F	$+\infty$	NaN
X	$-\infty$	$-\infty$	$-\infty$	*	NaN
	F	$-\infty$	$Round(X+Y)$	$+\infty$	NaN
	$+\infty$	*	$+\infty$	$+\infty$	NaN
	NaN	NaN	NaN	NaN	NaN

Table 1. Summary of the FP addition of two numbers of X and Y. F represents the normal and denormal numbers. * indicates FP invalid arithmetic operands.

Figure 1.a shows the block diagram of the SNAP FP adder designed at Stanford University [17]. As an alternative to the SNAP design, the ones complementer after the mantissa adder can be avoided, if we ensure that input C (shown in Figure 1.a) of the mantissa adder is smaller than or equal to input A (shown in Figure 1.a), when the exponent difference is 0 and the operation of mantissa adder is subtraction. To ensure this property, a mantissa comparator and extra circuits are needed to swap the mantissas correctly. Figure 1.b shows a variant of the SNAP FP adder with this modification (the *compare* unit is added and the ones complementer is deleted). This *compare* unit exists in many modern high-speed FP adder designs and makes the verification harder as described in Section 5.2. Figure 2 shows the detailed circuit of the *compare* unit which generates the signal to swap the mantissas. The signal $E_x < E_y$ comes from the exponent subtractor. When $E_x < E_y$ or $E_x = E_y$ and $M_x < M_y$ (i.e., h =1), A is M_y (i.e. the mantissas are swapped). Otherwise, A is M_x.

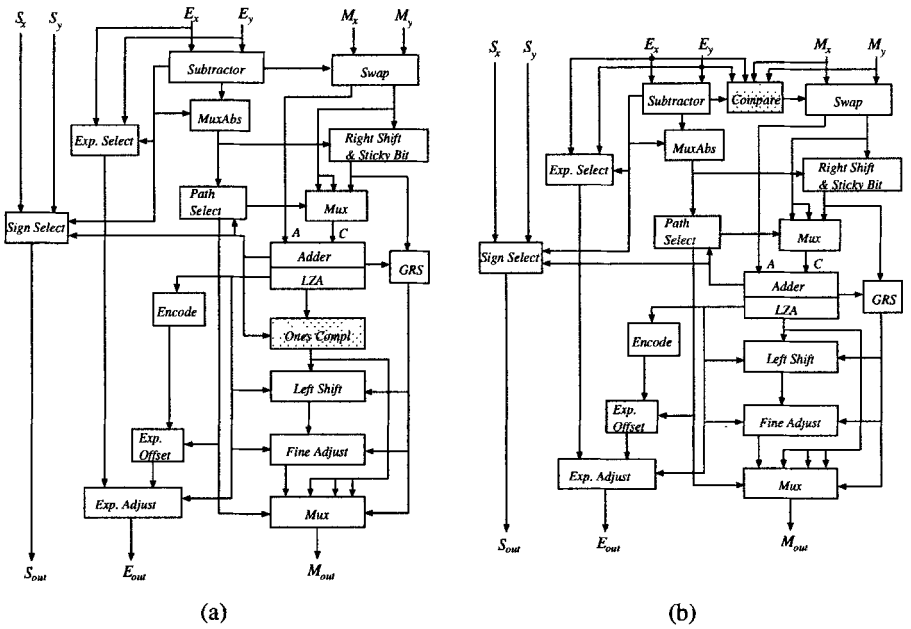

Fig. 1. The Stanford SNAP FP adder (a) and its variant (b).

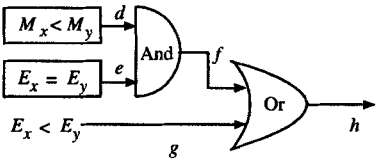

Fig. 2. Detailed circuit of the compare unit

4 Specifications of FP Adders

In this section, we focus on the general specifications of the FP adder, especially when both operands have denormal or normal values. In cases where at least one of the operands is a NaN or ∞, the specifications can be easily written at the bit level. For example, when both operands are NaN, the expected output is NaN (i.e. the exponent is all 1s and the mantissa is not equal to zero). This specification can be expressed as the "AND" of the exponent output bits is 1 and the "OR" of the mantissa output bits is 1.

4.1 Specifications

When both operands have normal or denormal values, the ideal specification is $OUT = Round(X + Y)$. However, the *PHDD representation of FP addition grows exponentially with the size of the exponent. Thus, the specification must be divided into several sub-specifications for verification. According to the signs of the operands, the function $X + Y$ can be rewritten as Equation 1. Similarly, for FP subtraction, the function $X - Y$ can be also rewritten as true addition when the operands have different signs and true subtraction when the operands have the same sign.

$$X + Y = (-1)^{S_x} \times \begin{cases} (2^{E_x - B} \times M_x + M_y \times 2^{E_y - B}) & S_x = S_y (true\ addition) \\ (2^{E_x - B} \times M_x - M_y \times 2^{E_y - B}) & S_x \neq S_y (true\ subtraction) \end{cases} \quad (1)$$

The *PHDDs for the true addition and subtraction still grow exponentially. Based on the sizes of the two exponents, the function $X + Y$ for true addition can be rewritten as:

$$X + Y = (-1)^{S_x} \times \begin{cases} 2^{E_x - B} \times (M_x + (M_y >> i)) & E_y \leq E_x \\ 2^{E_y - B} \times (M_y + (M_x >> i)) & E_y > E_x \end{cases}, \text{where } i = |E_x - E_y|.$$

When $E_y \leq E_x$, the exponent is E_x and the mantissa is the sum of M_x and M_y right shifted by i bits ($M_y >> i$ in the equation). $|E_x - E_y|$ can range from 0 to $2^n - 2$, but the number of mantissa bits in FP format is only m bits.

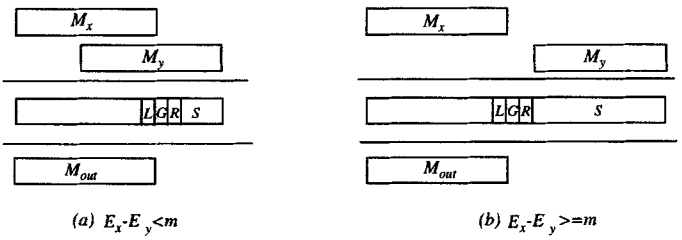

Fig. 3. Cases of true addition for the mantissa part.

Figure 3 illustrates the possible cases of true addition for $E_y \leq E_x$ based on the values of $E_x - E_y$. In Figure 3.a, for $0 \leq E_x - E_y < m$, the intermediate (precise) result contains more than m bits. The right portion of the result is denoted as L, G, R and S, where L is the least signification bit of the mantissa. The rounding mode will use

these bits to perform the rounding and generate the final result (M_{out}) in m-bit format. When $E_x - E_y \geq m$ as shown in Figure 3.b, the right shifted M_y only contributes to the intermediate result in the G, R and S bits. Depending the rounding mode, the output mantissa will be M_x or $M_x + 1 * 2^{-m+1}$. Therefore, we only need one specification in each rounding mode for the cases $E_x - E_y \geq m$. A similar analysis can be applied to the case $E_y > E_x$. Thus, the specifications for true addition with rounding can be written as:

$$\begin{cases} C_{a1}[i] \Rightarrow OUT = Round((-1)^{S_x} \times 2^{E_x - B} \times (M_x + (M_y >> i))) & 0 \leq i < m \\ C_{a2} \Rightarrow OUT = Round((-1)^{S_x} \times 2^{E_x - B} \times (M_x + (M_y >> m))) & i \geq m \\ C_{a3}[i] \Rightarrow OUT = Round((-1)^{S_x} \times 2^{E_y - B} \times (M_y + (M_x >> i))) & 0 < i < m \\ C_{a4} \Rightarrow OUT = Round((-1)^{S_x} \times 2^{E_y - B} \times (M_y + (M_x >> m))) & i \geq m \end{cases}$$

where $C_{a1}[i]$, C_{a2}, $C_{a3}[i]$ and C_{a4} are the conditions $Cond\_add\&E_x = E_y + i$, $Cond\_add\&E_x \geq E_y + m$, $Cond\_add\&E_y = E_x + i$, and $Cond\_add\&E_y \geq E_x + m$, respectively. $Cond\_add$ represents the condition for true addition and exponent range (i.e. normal and denormal numbers only). OUT is composed from the outputs S_{out}, E_{out} and M_{out}. While building BDDs and *PHDDs for OUT from the circuit, the conditions on left side of the $\Rightarrow$ will be used to simplify the BDDs automatically by conditional symbolic simulation.

The number of specifications for true addition is $2m + 1$. Since the value of m for IEEE double precision is 53, the number of specifications for true addition is 107. Since the specifications are very similar to one another, they can be generated by a looping construct in the word-level SMV specification language.

Similarly, the specification of true subtraction can be divided into several hundred of sub-specifications. The specification of true subtraction is divided into two cases: *far* ($|E_x - E_y| > 1$) and *close* ($E_x - E_y$=0,1 or -1). For the *far* case, the result of mantissa subtraction does not require a massive left shift (i.e., LZA is not active). For the *close* case, the result of mantissa subtraction requires a massive left shift (i.e., LZA is active), which makes the verification harder. Thus, the specifications of the *close* case must be divided further based on the number of bits to be left shifted. Readers can refer to [6] for the details of these specifications.

4.2 Specification Coverage

Since the specifications of floating-point adders are split into several hundred sub-specifications, do these sub-specifications cover the entire input space? To answer this question, one might use a theorem prover to check the case splitting. In contrast, we propose a BDD approach to compute the coverage of our specifications.

Our approach is based on the observation that our specifications are in the form "$cond \Rightarrow out = expected\_result$" and $cond$ is only dependent on the inputs of the circuits. Thus, the union of the $cond$s of our specifications, which can be computed by BDD operations, must be TRUE when our specifications cover the entire input space. In other words, the union of the $cond$s can be used to compute the percentage of input space covered by our specifications and to generate the missing cases.

5 Verification System: Extended Word-Level SMV with *PHDDs

To verify integer arithmetic circuits, SMV [16] was extended using HDDs [8] to handle word level expressions in the specification formulas [9]. For verification of FP circuits,

we replaced the HDDs in word-level SMV with *PHDDs and introduced relational operators for FP numbers. As in word-level SMV, only the word-level functions are represented by *PHDDs while the rest of the functions are represented by BDDs.

5.1 Conditional Symbolic Simulation

We have introduced a conditional symbolic simulation technique into word-level SMV. Symbolic simulation performs a simulation with inputs having symbolic values (i.e., Boolean variables or Boolean functions). The simulation process builds BDDs for the circuits. If each input is a Boolean variable, this approach may cause a explosion of the BDD size in the middle of the process, because it tries to simulate the entire circuit for all possible inputs at once. The concept of conditional symbolic simulation is to perform the simulation process under restricted conditions, expressed as a Boolean function over the inputs.

In [15], Jain and Gopalakrishnan encoded the conditions together with the original inputs as new inputs to the symbolic simulator using a parametric form of Boolean expressions, but it is hard to incorporate this approach into word-level SMV. Our approach is to apply the conditions directly during the symbolic simulation process. After building the BDD for a circuit gate, the condition is used to simplify the BDD using the *restrict* [11] algorithm. Then, the simplified BDD is used as the input function for the gates connected to this one. This process is repeated until the outputs are reached. This approach dynamically extracts the circuit behavior under the specified condition without modifying the actual circuit.

5.2 Short-Circuiting Technique

Can we verify the specifications of FP adders by conditional symbolic simulation? In our experience, all the specifications for the FP adder design without a mantissa comparator, as in Figure 1.a, can be verified by conditional symbolic simulation, but not so for the FP adder containing a mantissa comparator, as in Figure 1.b. This is caused by a conflict in variable orderings for the mantissa adder and the mantissa comparator, which generates the signal $M_x < M_y$ (i.e. signal d in Figure 2). The best variable ordering for the comparator is to interleave the two vectors from the most significant bit to the least significant bit (i.e., $x_{m-1}, y_{m-1}, ..., x_0, y_0$). Table 2 shows the CPU time in seconds and the BDD size of the signal d under different variable orderings, where ordering offset represents the number of bits offset from the best ordering. For example, the ordering is $x_{m-1}, ..., x_{m-6}, y_{m-1}, x_{m-7}, y_{m-2}, ..., x_0, y_5, ..., y_0$, when the ordering offset is 5. Clearly, the BDD size grows exponentially with the offset. In contrast to the comparator, the best ordering for the mantissa adder is $x_{m-1}, ..., x_{m-k-1}, y_{m-1}, x_{m-k-2}, y_{m-2}, ..., x_0, y_k, ..., y_0$, when the exponent difference is k. We observed that the best ordering for the specification represented by *PHDDs is the same as the best ordering for the mantissa adder. Thus, extended word-level SMV can not build the BDDs for both the mantissa comparator and mantissa adder by conditional symbolic simulation, when the exponent difference is large.

Let us examine the *compare* unit carefully. We find that the signal d is used only when $E_x = E_y$. In other words, it is not necessary to build the BDDs for it, when $|E_x - E_y|$ is greater than 0. Based on this fact, we introduce a short-circuiting technique to eliminate unnecessary computations as early as possible. The word-level SMV system is modified to incorporate this technique. In the *PHDD package, the BDD operators,

such as *And* and *Or*, are modified to abort the operation and return a *special token* when the number of newly created BDD nodes within this BDD call is greater than a size threshold. In word-level SMV, for an *And* gate with two inputs, if the first input evaluates 0, 0 will be returned without building the BDDs for the second input. Otherwise, the second input will be evaluated. If the second input evaluates to 0 and the first input evaluates to a *special token*, 0 is returned. Similar rules are applied to *Or* gates with two inputs. *Nand*(*Nor*) gates can be decomposed into *Not* and *And* (*Or*) gates and use the same technique to terminate earlier. For other logic gates with two inputs, the result is a *special token* if either of the inputs evaluates to a *special token*. If the *special token* is propagated to the output of the circuit, then the size threshold is doubled and the output is recomputed. This process is repeated until the output BDD is built. For example, when the exponent difference is 30, the size threshold is 10000, the ordering is the best ordering of mantissa adder, and the evaluation sequence of the *compare* unit shown in Figure 2 is d, e, f, g and h, the values of signals d, e, f, g and h will be *special token*, 0, 0, 1, and 1, respectively, by conditional symbolic simulation. With these modification, the new system can verify all of the specifications for both types of FP adders by conditional symbolic simulation. We believe that this short-circuiting technique can be generalized and used when different parts of the circuit are used under different operating conditions.

Ordering Offset	BDD Size	CPU Time (Sec.)
0	157	0.68
1	309	0.88
2	608	1.35
3	1195	2.11
4	2346	3.79
5	4601	7.16
6	9016	13.05
7	17655	26.69
8	34550	61.61
9	67573	135.22

Table 2. Performance measurements of a 52-bit comparator with different orderings.

6 Verification of FP Adders

We use the FP adder in the Aurora III Chip [14], designed by Dr. Huff as part of his PhD dissertation at the University of Michigan, as an example to illustrate the verification of FP adders. This adder is based on the same approach as the SNAP FP adder [17] at Stanford University. This FP adder only handles operands with normal values. When the result is a denormal value, it is truncated to 0. This adder supports IEEE double precision format and the 4 IEEE rounding modes. Dr. Huff found several errors with the approach described in [17]. In this verification work, we verify the adder only in round to nearest mode, because we believe that the round to nearest mode is the hardest one to verify. All experiments were carried out on a Sun 248 MHz UltraSPARC-II server with 1.5 GB memory.

The FP adder is described in the Verilog language in a hierarchical manner. The circuit was synthesized into flattened, gate-level Verilog by Dr. John Zhong at SGI. Then, a simple Perl script was used to translate the circuit from gate-level Verilog to SMV format and to perform latch removal.

6.1 Latch Removal

Huff's FP adder is a pipelined, two phase design with a latency of three clock cycles. We handled the latches during the translation from gate-level Verilog to SMV format. Figure 4.a shows the latches in the pipelined, two phase design. In the design, the phase 2 clock is the complement of the phase 1 clock. Since we only verify the functional correctness of the design and the FP adder does not have any feedback loops, the latches can be removed. One approach is to directly connect the inputs and the outputs of latches. This approach would eliminate some logic circuits related to the latch enable signals as shown on the right side of the latches in Figure 4.a. With this approach, the correctness of these circuits can not be checked. For example, a design error in the circuit, always generated 0s for the enable signals of latches, can not be found if we use this approach to remove the latches.

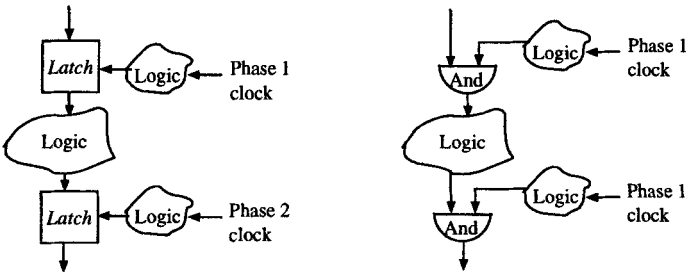

Fig. 4. Latch Removal. (a) The pipelined, two phase design. (b) The design after latch removal.

Our approach for latch removal is based on this observation: the data are written into the latches when the enable signals are 1. To ensure the correctness of the circuits for the enable signals, the latches can be replaced by *And* gates, as shown in Figure 4.b, without losing the functional behavior of the circuit. Since phase 2 clock is the complement of the phase 1 clock, we must replace the phase 2 clock by the phase 1 clock. Otherwise the circuit behavior will be incorrect. With this approach, we can also check the correctness of circuits for the enable signals of the latches.

6.2 Design with Bugs

During the verification process, our system found several design errors in Huff's FP adder. These errors were not caught by random simulations performed by Dr. Huff. The first error we found is the case when $A + C$ = 01.111...11, $A + C$ + 1=10.000...00, and the rounding logic decides to add 1 to the least significant bit (i.e., the result should be $A + C + 1$), but the circuit design outputs A+C as the result. This error is caused by incorrect logic in the *path select* unit, which categorized this case as a no shift case instead of a right shift by 1. While we were verifying the specification of true addition, our system generated a counterexample for this case in around 50 seconds. To ensure that this bug was not introduced by when translating the circuits, we have used Cadence's Verilog simulation to verify this bug in the original design by simulating the input pattern generated by our system.

Another design error we found is in the sticky bit generation. The sticky bit generation is based on the table given in page 10 of Quach's paper describing the SNAP FP

adder [17]. The table only handles cases when the absolute value of the exponent difference is less than 54. The design sets the sticky bit to 1 when the absolute value of the exponent difference is greater than 53 (for normal numbers only). The bug is that the sticky bit should not always be 1 when the absolute value of the exponent difference is equal to 54. Figure 5 shows the sticky bit generation when $E_x - E_y = 54$. Since N_x has 52 bits, the leading 1 will be the Round (R) bit and the sticky (S) bit is the OR of all of N_y bits, which may be 0. Therefore an entry for the case $|E_x - E_y| = 54$ is needed in the table of Quach's paper [17].

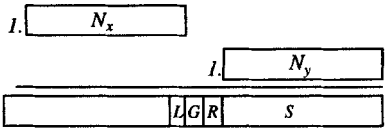

Fig. 5. Sticky bit generation, when $E_x - E_y = 54$.

6.3 Corrected Designs

After identifying the bugs, we fixed SMV version of the circuit. In addition, we created another FP adder by adding the *compare* unit in Figure 1.b to Huff's design. This new adder is equivalent to the FP adder in Figure 1.b, since the *ones complement* unit will not be active at any time.

To verify the FP adders, we combined the specifications for both addition and subtraction instructions into the specification of true addition and subtraction. We use the same specifications to verify both FP adders. Table 3 shows the CPU time in seconds and the maximum memory required for the verification of both FP adders. The CPU time is the total time for verifying all specifications. The FP adder II can not be verified by conditional symbolic simulation without the short-circuiting technique. The maximum memory is the maximum memory requirement of verification of these specifications. For both FP adders, the verification can be done within two hours and requires less than 55 MB. Each individual specification can be verified in less than 3 minutes. The verified specifications cover 99.78% of the input space for FP adders in IEEE round-to-nearest mode. The reason for uncovered input space (0.22%) is that the circuit does not implement the cases where either operand with denormal, NaN or ∞ values, and where the result of true subtraction is a denormal value.

Case	CPU Time (Sec.)		Max. Memory(MB)	
	FP adder I	FP adder II	FP adder I	FP adder II
True addition	3283	3329	49	55
True subtraction (*far*)	2654	2668	35	35
True subtraction (*close*)	994	1002	53	48

Table 3. Performance measurements of verification of FP adders. FP adder I is Huff's FP adder with the bugs fixed. FP adder II is FP adder I with the *compare* unit in Figure 1.b. For true subtraction, *far* represents the cases $|E_x - E_y| > 1$, and *close* represents the cases $|E_x - E_y| \leq 1$.

In our experience, the choice of decomposition type of the subtrahend's variables for true subtraction cases is very important to the verification time. The best decomposition type of the subtrahend's variables is *negative* Davio decomposition. If the subtrahend's variables use the *positive* Davio decomposition, the *PHDDs for OUT can not be built after a long CPU time (> 4 hours).

7 Conversion Circuits

The overflow flag erratum of the FIST instruction (FP to integer conversion) [12] in Intel's Pentium Pro and Pentium II processors has illustrated the importance of verification of conversion circuits [14] which convert the data from one format to another. These circuits perform conversions between any of the three number formats: integer, IEEE single precision, and IEEE double precision.

We believe that the verification of conversion circuits is much easier than the verification of FP adders, since these circuits are much simpler than FP adders and only have one operand (i.e. less input variables). For example, the specification of the double-to-single operation, which converts data from double precision to single precision, can be written as *"(overflow_flag = expected_overflow) & (not overflow_flag ⇒ (output = expected_output))"*, where *overflow_flag* and *output* are directly from the circuit, and *expected_overflow* and *expected_output* are computed in terms of the inputs. *Expected_output* is computed by $Round((-1)^S \times M \times 2^{E-B})$. Similarly, *expected_overflow* can be computed from the inputs. This specification covers double precision values which cannot be represented in single precision. For another example, the specification of the single-to-double operation can be written as *"output = input"*, since every number represented in single precision can be represented in double precision without rounding (i.e. the output represents the exact value of input).

8 Conclusions and Future Work

We presented a black box version of verification of FP adders with reusable specifications using extended word-level SMV, which was improved by using the Multiplicative Power HDDs (*PHDDs), and by incorporating conditional symbolic simulation as well as a short-circuiting technique. Based on case analysis, the specifications of FP adders are divided into several hundred implementation-independent sub-specifications. Conditional symbolic simulation and a short-circuiting technique make these specifications reusable in any implementation. We used our system and reusable specifications to verify a FP adder from the University of Michigan. Our system found several bugs in Huff's FP adder. Each specification was checked in less than 3 minutes or 5 minutes including counterexample generation. A variant of the corrected FP adder was created and verified to demonstrate the ability of our system to handle different implementations. For each FP adder, verification finished in 2 CPU hours on a Sun UltraSPARC-II server. We believe that our system and specifications can be applied to directly verify FP adders and to help find errors.

The overflow flag erratum of the FIST instruction [12] in Intel's Pentium Pro and Pentium II processors has illustrated the importance of verification of conversion circuits which convert data from one format to another. Since these circuits are much simpler than FP adders and have only one operand, we believe that our system can verify the correctness of these circuits. We plan to verify the conversion circuits in the Aurora III chip.

Acknowledgements

We thank Prof. Brown, Dr. Huff and Mr. Riepe at University of Michigan for providing us with Huff's FP adder and valuable discussions. We thank Dr. John Zhong at SGI for

helping us to synthesize the FP adder into flattened, gate-level Verilog. We also thank Bwolen Yang and Henry A. Rowley for proofreading this paper.

References

1. BROCK, B., KAUFMANN, M., AND MOORE, J. S. ACL2 theorems about commerical microprocessors. In *Proceedings of the Formal Methods on Computer-Aided Design* (November 1996), pp. 275–293.
2. BRYANT, R. E. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers* (August 1986), pp. 8:677–691.
3. BRYANT, R. E., AND CHEN, Y.-A. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference* (June 1995), pp. 535–541.
4. CARREÑO, V. A., AND MINER, P. S. Specification of the IEEE-854 floating-point standard in HOL and PVS. In *High Order Logic Theorem Proving and Its Applications* (September 1995).
5. CHEN, Y.-A., AND BRYANT, R. E. *PHDD: An efficient graph representation for floating point circuit verification. In *Proceedings of the International Conference on Computer-Aided Design* (November 1997), pp. 2–7.
6. CHEN, Y.-A., AND BRYANT, R. E. Verification of floating-point adders. Tech. Rep. CMU-CS-98-121, School of Computer Science, Carnegie Mellon University, 1998.
7. CHEN, Y.-A., CLARKE, E. M., HO, P.-H., HOSKOTE, Y., KAM, T., KHAIRA, M., O'LEARY, J., AND ZHAO, X. Verification of all circuits in a floating-point unit using word-level model checking. In *Proceedings of the Formal Methods on Computer-Aided Design* (November 1996), pp. 19–33.
8. CLARKE, E. M., FUJITA, M., AND ZHAO, X. Hybrid decision diagrams overcoming the limitations of MTBDDs and BMDs. In *Proceedings of the International Conference on Computer-Aided Design* (November 1995), pp. 159–163.
9. CLARKE, E. M., KHAIRA, M., AND ZHAO, X. Word level model checking – Avoiding the Pentium FDIV error. In *Proceedings of the 33rd ACM/IEEE Design Automation Conference* (June 1996), pp. 645–648.
10. COE, T. Inside the Pentium Fdiv bug. *Dr. Dobbs Journal* (April 1996), pp. 129–135.
11. COUDERT, O., AND MADRE, J. C. A unified framework for the formal verification of sequential circuits. In *Proceedings of the International Conference on Computer-Aided Design* (November 1990), pp. 126–129.
12. FISHER, L. M. Flaw reported in new intel chip. *New York Times* (May 6 1997), D, 4:3.
13. HAMAGUCHI, K., MORITA, A., AND YAJIMA, S. Efficient construction of binary moment diagrams for verifying arithmetic circuits. In *Proceedings of the International Conference on Computer-Aided Design* (November 1995), pp. 78–82.
14. HUFF, T. R. Architectural and circuit issues for a high clock rate floating-point processor. *PhD Dissertation in Electrical Engineering Department, University of Michigan* (1995).
15. JAIN, P., AND GOPALAKRISHNAN, G. Efficient symbolic simulation-based verification using the parametric form of boolean expressions. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (August 1994), pp. 1005–1015.
16. MCMILLAN, K. L. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
17. QUACH, N., AND FLYNN, M. Design and implementation of the SNAP floating-point adder. Tech. Rep. CSL-TR-91-501, Stanford University, December 1991.