Word Level Model Checking – Avoiding the Pentium FDIV Error *

E. M. Clarke[†] M. Khaira[‡] X. Zhao[†‡]

Abstract

The highly-publicized division error in the Pentium has emphasized the importance of formal verification of arithmetic circuits. Symbolic model checking techniques based on binary decision diagrams (BDDs) have been successful in verifying control logic. However, lack of proper representation for functions that map boolean vectors into the integers has prevented this technique from being used for verifying arithmetic operations.

We have developed a new technique for verifying arithmetic circuits. The new technique, called *word level model checking*, has been used successfully to verify circuits for division and square root computation that are based on the SRT algorithm used by the Pentium. The technique makes it possible to handle both the control logic and the data paths in the circuit. The total number of state variables exceeds 600 (which is much larger than any circuit previously handled by other symbolic model checkers).

1 Introduction

Proving the correctness of arithmetic operations has always been an important problem. The importance of this problem has been recently emphasized by the highly-publicized division error in the Pentium. In order to verify such circuits, it is necessary to represent and manipulate functions that map boolean vectors to integer values. We have used **hybrid decision diagrams** (HDDs) [5] to represent the integer functions that occur in the arithmetic circuit verification. For the state variables corresponding to data bits, our representation behaves like a binary moment diagram (BMD) while for the state variables corresponding to control signals, it behaves like a multi-terminal BDD (MTBDD). By using this representation, we are able to handle circuits with both control logic and wide data paths.

Our representation for functions that map boolean vectors into the integers enables us to extend *temporal logic model checking* [3, 4] so that it can handle arithmetic circuits. In traditional model checking systems, specifications are expressed in a propositional temporal logic, and circuit designs and protocols are modeled as state-transition systems. An efficient search procedure is used to determine automatically if the specifications are satisfied by the transition systems. The main disadvantage of this approach is the state explosion which can occur if the system being verified has many components that can make transitions in parallel. The size of the transition systems that can be verified by model checking techniques has increased dramatically because of the use of BDDs [2]. Although such *symbolic model checking techniques* have been successful in verifying control logic, they cannot be directly used for verifying arithmetic circuits.

In the word level model checking system, propositions denoting nodes in circuits are represented as BDDs and are computed in exactly the same way as in the original symbolic model checking system. Words are arrays of propositions, each of which corresponds to a single bit. Expressions are composed of arithmetic operations applied to words. Hybrid decision diagrams can be computed for words and expressions using the algorithms for arithmetic operations. Atomic formulas can be relations between expressions, and their BDD representations can be computed by the algorithm that handles arithmetic relations. After the BDD representations for the atomic formulas are generated, the BDDs for static formulas and temporal formulas are computed in the same way as in ordinary model checking. In particular, the fixpoint computations are exactly the same in both cases.

By using the word level model checking system, we have successfully verified circuits for division and square root computation that are based on the SRT algorithm used by the Pentium. We are able to handle both the control logic and the data paths. All of the states in the finite state machine for the control logic have been verified. Moreover, we have proved invariant properties that guarantee the correctness of the data values and prevent overflows. The total number of state variables exceeds 600 (which is much larger than any circuit previously checked by SMV).

2 Word Level CTL

Symbolic model checking techniques based on Binary Decision Diagrams (BDDs) have been successful in verifying control logic [2]. However, lack of proper representation for functions that map boolean vectors into integers has prevented this technique from being used for verifying arithmetic circuits. We have experimented with the different representations that are introduced in previous sections. Unfortunately, there are fundamental problems with applying either the MTBDD or the BDD array representations for ver-

*This research was sponsored in part by the National Science Foundation under grant no. CCR-8722633, by the Semiconductor Research Corporation under contract 92-DJ-294, and by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330.

[†]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

[‡]Intel Development Labs., Intel Corporation, 5200 NE Elam Young Pkwy, Hillsboro, OR 97124

ification of arithmetic circuits. For the functions that arise in this type of application, the number of possible values is exponential in the number of bits. Therefore, the MTBDDs also have exponential size. On the other hand, arithmetic operations on BDD arrays are very expensive. In particular, since the BDD size for the middle bit of a combinational multiplier is exponential in the length of its operands, the BDD array representation is exponential for multiplication.

Bryant and Chen [1] have shown that BMDs give a compact representation for certain functions that have exponential size MTBDDs. They have used this representation to verify the data paths of some arithmetic circuits. They are able to conclude that a circuit is correct if the BMDs for the circuit and the specification are exactly the same. However, depending on the implementation and the control logic, there can be cases in which the circuits are correct but the BMDs are not identical. Moreover, since their technique cannot handle inequalities, it is impossible to check some of the properties that are needed in order to avoid the Pentium error.

We have used hybrid decision diagrams (HDDs) [5] to represent the integer functions that occur in the arithmetic circuit verification. In particular, for the state variables corresponding to data bits, we use the inverse Reed-Muller transform while for the state variables corresponding to control signals, we use the identity transform. Therefore, for data variables, this representation behaves like a BMD while for control variables, it behaves like an MTBDD. By using this representation, we are able to handle circuits with both control logic and wide data paths. Since this representation is a special case of the hybrid decision diagrams, all the algorithms mentioned in previous sections can be applied.

By using this representation, we have extended the symbolic model checking system SMV [7] so that it can also handle properties involving relationships among data words. In the original SMV system, atomic formulas can only contain state variables. In the extended system, we allow atomic formulas be equations or inequalities between expressions as well. These expressions are represented as hybrid BDDs. The logic that we use is the following:

- Atomic propositions: $Ap = \{p_1, \ldots, p_k\}$
- Propositional formulas: $Prop ::= Ap \mid Prop \wedge Prop \mid \neg Prop$
- Words: $Word ::= (Prop, Prop, \ldots, Prop)$
- Expressions: $Exp ::= Constant \mid Word \mid \text{next}(Word) \mid Exp \odot Exp \mid \text{if } SF \text{ then } Exp \text{ else } Exp$, where $\odot$ can be $+$, $-$, or $\times$.
- Atomic Formulas: $AF ::= Ap \mid \{\mathbf{A} \mid \mathbf{E}\}(Exp \sim Exp)$, where $\sim$ can be $=$, $<$, or $\leq$. Because of the nondeterministic behavior of the system, there can be more than one possible next state for a given state. Therefore, a path quantifier is needed when the next state operator is used in the expressions.
- Static Formulas: $SF ::= AF \mid SF \wedge SF \mid \neg SF$
- Temporal Formulas: $TF ::= SF \mid TF \wedge TF \mid \neg TF \mid \mathbf{AX}\, TF \mid \{\mathbf{A} \mid \mathbf{E}\}[TF\ \mathbf{U}\ TF]$

A model is given by:

- States: $S = 2^{Ap}$.
- Transition relation: $R \subseteq S \times S$
- Initial states: $S_0 \subseteq S$
- Valuation mapping for atomic propositions $V : Ap \times S \rightarrow \{0, 1\}$

The semantics for the logic is given by:

- Propositional formulas: $P : Prop \times S \rightarrow \{0, 1\}$

$$\begin{aligned} P(p_i, s) &= V(p_i, s) \\ P(f_1 \wedge f_2) &= P(f_1, s) \wedge P(f_2, s) \\ P(\neg f, s) &= \neg P(f, s) \end{aligned}$$

- Words: $W : Word \times S \rightarrow N$

$$W((f_0, f_1, \ldots, f_n), s) = \sum_{i=0}^{n} P(f_i, s) 2^i$$

- Expressions: $E : Exp \times S \times S \rightarrow N$. The state s' is used to handle next-state operators that occur in expressions.

$$\begin{aligned} & E(e_1 \odot e_2, s, s') = E(e_1, s, s') \odot E(e_2, s, s') \\ & E(\text{if } f \text{ then } e_1 \text{ else } e_2, s, s') = \\ & \qquad \text{if } (s \models f) \text{ then } E(e_1, s, s') \text{ else } E(e_2, s, s') \\ & E(w, s, s') = W(w, s) \\ & E(\text{next}(w), s, s') = W(w, s') \end{aligned}$$

- Atomic formulas:

$$\begin{aligned} s &\models p_i \Leftrightarrow V(p_i, s) = 1 \\ s &\models \mathbf{A}(e_1 \sim e_2) \Leftrightarrow \\ & \qquad \forall s'.\, R(s, s') \rightarrow (E(e_1, s, s') \sim E(e_2, s, s')) \\ s &\models \mathbf{E}(e_1 \sim e_2) \Leftrightarrow \\ & \qquad \exists s'.\, R(s, s') \wedge (E(e_1, s, s') \sim E(e_2, s, s')) \end{aligned}$$

  The formula $\mathbf{A}(e_1 \sim e_2)$ is true in state s when $(e_1 \sim e_2)$ holds for all successor states. Likewise, $\mathbf{E}(e_1 \sim e_2)$ is true in state s when $(e_1 \sim e_2)$ holds for some successor state.

- The semantics of SF and TF are the same as in CTL.

This logic can naturally be divided into three layers. The top layer contains atomic formulas, static formulas and temporal formulas. The second layer contains words and expressions. The third layer contains atomic propositions and propositional formulas. All of the objects in the top and bottom layers are boolean functions while the objects in the second layer are functions that map boolean vectors into the integers. Therefore, in the word level model checking system, all of atomic propositions, propositions, atomic formulas, static formulas and temporal formulas are represented as BDDs; while words and expressions are represented as hybrid decision diagrams.

3 Word Level Model Checking

Model checking is a technique of finding the set of states in a state-transition graph where a given CTL formula is true. There is a model checker called EMC that solves this problem using efficient graph-traversal techniques. If the model is represented as a state-transition graph, the complexity of the algorithm is linear in the size of the graph and in the length of the formula. The algorithm is quite fast in practice [3, 4]. However, an explosion in the size of the model may occur when the state-transition graph is extracted from a finite state concurrent system that has many processes or components. In *symbolic model checking systems* [2], BDDs are used to represent the transition relations and sets of states. The model checking process is performed by fixpoint operations on these BDDs. By using symbolic model checking techniques, the size of the transition systems that can be verified has increased dramatically. Although such techniques have been successful in verifying control logic, they cannot be directly used for verifying arithmetic circuits. This is because expressions that involve words with integer values cannot be handled properly.

In order to be able to perform model checking on the logic discussed in the previous section, it is desirable to implement various operations on hybrid decision diagrams. We consider scalar multiplication, addition and multiplication of two functions, and the if-then-else operation. Although the worst case complexity for these algorithms can be exponential, in practice, this algorithm works quite well. Model checking for word level properties also requires computing the set of assignments that satisfy $f_1 \sim f_2$, where $\sim$ can be one of $=, \neq, <, \leq, >$, or $\geq$. We have developed an efficient algorithm that can compute the BDD for the set of assignment satisfying an arithmetic relation. In particular, this algorithm has linear complexity for linear expressions [5].

Now that we are able to handle arithmetic operations and arithmetic relations, it is possible to extend the symbolic model checking algorithm so that it can verify word level properties. BDDs for the transition relation and all propositions are generated in exactly the same way as in the original symbolic model checking system. The hybrid decision diagram representation of a word $(f_0, f_1, \ldots, f_n)$ can be computed as

$$\sum_{i=1}^{n} (\text{if } f_i \text{ then } 2^i \text{ else } 0)$$

using operations mentioned above. The hybrid decision diagram representation of most expressions can be computed using similar operations. The only exception is the *next* operation, which can be performed by variable substitution. The substitution replaces all of the current state variables in the hybrid decision diagram for the word by their corresponding next state variables. The algorithm to obtain the BDD representing the set of variable assignments that make an algebraic relation true can be used to compute the BDD for atomic formulas. After the BDD representation for the atomic formulas is generated, the BDDs for static formulas and temporal formulas are computed in the same way as in ordinary model checking. In particular, the fixpoint computations are exactly the same in both cases.

Since we have used the same algorithm to compute the transition relation as in the ordinary model checking algorithm. The word level model checking algorithm does not work well when the transition relation does not have a concise representation. As an example, let's consider a multiplier. Let x and y be the input registers and z be the output register. Suppose the transition relation can be represented as follows:

$$Tr(x, y, z) = Tr'(x, y) \wedge (\text{next}(z) = x \times y)$$

Obviously, the BDD representation of the transition relation has exponential size since the BDD representation of the middle bit of a multiplier is exponential. This problem can sometimes be avoided by conjunctive decomposition of the transition relation. Let $\bar{x}, \bar{y}$, and $\bar{z}$ be the state variables that encode the current state value of x, y and z, respectively. Let $\bar{x'}, \bar{y'}$, and $\bar{z'}$ be the state variables that encode the next state value of x, y and z. Suppose that we want to verify a word level property of the form $f(x, y, z)$. There may be appearances of $\text{next}(z)$; if so, we can replace them by $x \times y$ at the word level and obtain a new formula. Hopefully, the resulting formula will be independent of z and the BDD representation of the formula can be denoted as $f'(\bar{x}, \bar{y})$. In this case, we can use Tr' as the transition relation to perform the fixpoint operations. Even if f' depends on some bits of z, we can often obtain a much simpler transition relation by eliminating the conjuncts that give the values of bits that are not needed.

4 Verification of an SRT division circuit

By using the word level model checking system, we have successfully verified circuits for division and square root computation that are based on the SRT algorithm used by the Pentium. We are able to handle both the control logic and the data paths. The division circuit that we investigated has 5 states, *idle, init, loop, last* and *rem*. This circuit can perform two different operations *division* and *remainder*. When the operation is *division*, the steps in the computation are

$$\textit{idle} \rightarrow \textit{init} \rightarrow \textit{loop}^* \rightarrow \textit{last} \rightarrow \textit{idle}$$

When the operation is *remainder*, the steps are

$$\textit{idle} \rightarrow \textit{init} \rightarrow \textit{loop}^* \rightarrow \textit{last} \rightarrow \textit{rem} \rightarrow \textit{idle}$$

Figure 1 gives the data path of the circuit at the loop state. All the words have 70 bits. However, only the leading bits of the partial remainder and multiples of the divisor are used to compute the quotient digit for the next cycle.

We have verified the circuit with both control logic and the data path. All states of the finite state machine have been checked. Let r be the partial remainder, q be the quotient, d be the divisor. We have checked the following properties:

- The expression $r + q \cdot d$ always equals the left-shifted dividend, i.e. $r + q \cdot d = 2^{2k} \cdot \text{dividend}$.
- The computation does not overflow. This is guaranteed by $-\frac{8}{3}d \leq r \leq \frac{8}{3}d$.

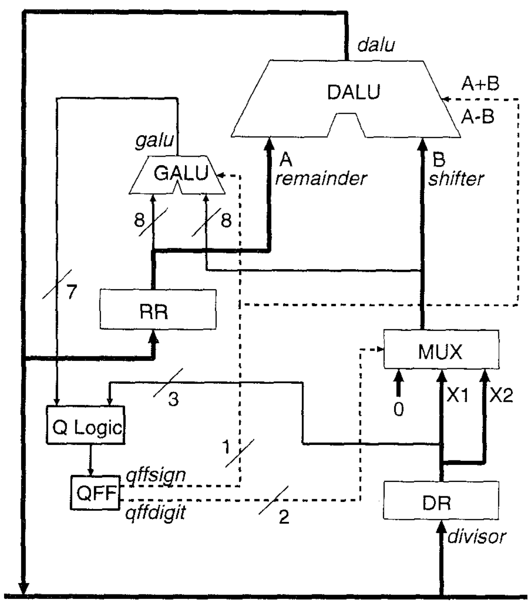

Figure 1: The data path for the division circuit

For example, we have proved that at init state, the remainder is the dividend and the quotient is zero. Therefore, the initial value for $r + q \cdot d$ equals the dividend. Moreover, the inequality mentioned above holds at the init state.

```
SPEC AG(state = init -> r = dividend & q = 0)
```

```
SPEC AG(state = init -> (-8) * d <= 3 * r <= 8 * d)
```

We have also proved that the inequality always holds in the loop states, and that $r + q \cdot d$ is invariant with respect to left shifting.

```
SPEC AG(state = loop ->
        A[((-8)*d <= 3*r <= 8*d) U state = last])
```

```
SPEC AG((state = loop & ((-8)*d <= 3*r <= 8*d)) ->
        A((r+q*r)*4 = next(r+q*r)))
```

The above properties are sufficient to guarantee that in the loop state, $r + q \cdot d$ always equals the dividend after left shifting. Similar properties are proved for the *last* and *rem* states. In addition, we have verified a circuit for computing square roots. The total number of state variables for the circuit that we verify exceeds 600 (which is much larger than any circuit previously checked by SMV).

5 Directions of Future Research

We have used word level symbolic model checking to replicate the Pentium FDIV bug and successfully verified the corrected circuit. In this paper, we have described the formal verification of a floating point division circuit based on the SRT algorithm using our word level model checker. We plan to experiment on more circuits. Possible applications include floating point multipliers, floating point adders, etc.

Our algorithm for solving arithmetic relations works extremely well for linear equations and inequalities. Although the current algorithm can handle some nonlinear equations and inequalities as well, it may be possible to extend this algorithm or to find a new algorithm that can handle more complicated nonlinear equations and inequalities.

There is still one problem with this technique. It can only be used for circuits that maintain the exact value of the data. When rounding occurs, the functions become less regular and the size of hybrid BDD representation is likely to explode. In these cases, the new value obtained after rounding can be described by a system of inequalities, and the verification process reduces to solving such systems. In another research project, we have built a theorem prover based on symbolic computation system Mathematica. The theorem prover is called *Analytica* [6] and is quite good at handling equations and inequalities. We believe that after some modification, Analytica will be useful for solving the inequalities that arise because of rounding in computer arithmetic.

References

[1] R. E. Bryant and Y. A. Chen. Verification of arithmetic functions with Binary Moment Diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 535–541. IEEE Computer Society Press, June 1995.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[3] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[5] E. M. Clarke, M. Fujita, and X. Zhao. Hybrid Decision Diagrams – overcoming the limitations of MTBDDs and BMDs. In *Proceedings of the 1995 Proceedings of the IEEE International Conference on Computer Aided Design*, pages 159–163. IEEE Computer Society Press, November 1995.

[6] E. M. Clarke and X. Zhao. Analytica: A theorem prover for Mathematica. *The Journal of Mathematica*, 3(1), 1993.

[7] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[8] G. S. Taylor. Compatible hardware for division and square root. In *Proceedings of the Fifth IEEE Symposium on Computer Arithmetic*, 1993.