Induction-based Gate-level Verification of Multipliers

Ying-Tsai Chang and Kwang-Ting (Tim) Cheng
Department of Electrical and Computer Engineering,
University of California, Santa Barbara, CA 93106

Abstract

We propose a method based on unrolling the inductive definition of binary number multiplication to verify gate-level implementations of multipliers. The induction steps successively reduce the size of the multiplier under verification. Through induction, the verification of an n-bit multiplier is decomposed into n equivalence checking problems. The resulting equivalence checking problems could be significantly sped up by simple structural analysis. This method could be generalized to the verification of more general arithmetic circuits and the equivalence checking of complex datapath.

1. Introduction

Hardware verification has become increasingly important in assuring a timely successful design schedule. The complexity of VLSI design and specifications, the interoperation of design tools from different vendors, and the often inevitable manual tweaking to meet timing constraints all contribute to the uncertainties of the design process, and verification technique is crucial for assuring the correctness of the resulting design. A sound verification methodology will play not only the conservative role of increasing confidence in the design but also the more constructive role of facilitating and encouraging more speculative optimization and architecture exploration.

Ordered binary decision diagram (BDD) [1] based verification techniques have been successfully applied to practical designs, especially in symbolic model checking [2] and equivalence checking [3, 4]. However, it is incapable of handling some arithmetic circuits, such as multipliers [5]. The verification of multipliers is further complicated by the lack of a compact bit-level canonical representation [1] and the existence of many varieties of globally different architectures. Effective solutions to this problem would also very likely bring in general schemes that could enhance the existing methodology.

Among the various attempts to solve this multiplier verification problem [6, 7, 8, 9, 10], word level binary moment diagram (BMD) is one of the more successful ones. However, most approaches make certain assumptions regarding the multiplier under verification that might not apply to a true gate-level implementation. For example, the BDD-based approach in [6] requires the identification of product bits $a_i \wedge b_i$ and is unable to handle multipliers with recoding logic. The direct construction of BMD in [7] requires that the hardware implementation be specified in a modular form and that the grouping of adder modules into word modules be correctly specified. Although this approach is quite efficient, it is conservative in the sense that not all correct implementations can be proved, and certainly cannot be applied to a gate-level implementation. The assumptions made in these approaches amount to inserting internal observation points that correspond to desired intermediate signals in the circuit. A gate-level netlist has no easily identifiable internal observation point to facilitate the verification, a situation reminiscent of testing. The only true gate-level verification technique with no additional assumption is the backward-substitution-based construction of BMD [8], which has a polynomial complexity if the backward propagation of the support does not cut through many underlying adders simultaneously [11]. However, it is known that the memory usage in this construction process could easily blow up at the existence of any error.

2. The basic idea

The approach proposed in this paper is based on the observation that most multipliers are implemented as multi-operand addition trees. Figure 1 (a) shows the multiplication as the sum of partial products in dot notation that assimilates typical pencil-and-paper algorithms. This schematic representation also captures pretty well the actual circuit structure of typical implementations of multipliers. Typical multiplier architectures are represented by their different addition trees in Figure 1 (b). Notice also the operand asymmetry in the implementation of the symmetric multiplication function, where the *multiplicand a* is kept intact as a word in obtaining the partial products, and the *multiplier b* is decomposed into bits [2]. This *multiplicand/multiplier asymmetry* is one of the syndromes that plagued the verification of multipliers in an equivalence-checking setting. If we could decompose a multiplier circuit into the underlying partial-product addition tree, then the verification problem could be solved more easily.

The basic idea of induction-based verification of multipliers can be explained using the following equation:

[1] It is unlikely that there exists a canonical boolean function representation with polynomial complexity for reading out minterms that has a compact multiplier representation with polynomial construction time. If it did exist, it could be used to solve the integer factorization problem.

[2] By convention, the term *multiplier* is used to denote both a multiplication unit and the decomposed operand in such a unit. The exact meaning could always be inferred from the context without confusion.

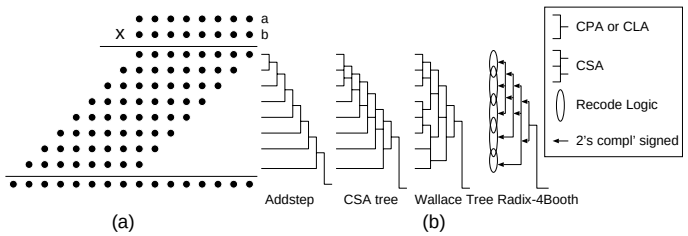

Figure 1. Schematic representations of various multiplier architectures.

$$
\begin{aligned}
& (a_{n-1}a_{n-2}\ldots a_0) \times (\underbrace{0..0}_{i} b_{n-1-i}..b_0) \\
= \quad & (a_{n-1}a_{n-2}\ldots a_0) \times (\underbrace{0..0}_{i+1} b_{n-2-i}..b_0) \\
& + b_{n-1-i} \cdot (a_{n-1}a_{n-2}..a_0 \underbrace{0..0}_{n-1-i}) \qquad (1)
\end{aligned}
$$

where a and b are the n-bit operands of the multiplication function $\times$ with a being the multiplicand and b being the multiplier. The subscripts denote the bit number from the least significant bit (LSB), e.g. a_i is the i-th bit of the operand a and a_0 is the LSB. This equation, interpreted as a pure mathematical formula, serves as an inductive definition of binary number multiplication with the induction on the size of the multiplier. If the above equation could be proved for an implementation of a multiplier for all i from 0 to $n-1$, then the multiplier would be successfully verified. This scheme is most easily understood by comparing this equation with Figure 1 (a). This equation shows that the partial product addition tree can be successively reduced to shorter ones by proving that a suitable portion of the circuit does indeed implement another partial product addition. The inductive steps could be proved through the use of n equivalence checking procedures. The pseudocode for the verification of a multiplier is the following:

```
verify_multiplier(netlist,n){
  circuit2=netlist;
  for (i=n-1;i >= 0;--i){
      circuit1=circuit2;
      circuit2=set_ith_input_to_zero(circuit1,i);
      circuit_add=add_adder(circuit2,i);
      check_equivalence(circuit1,circuit_add);
  }
}
```

where `circuit1` implements the left hand side of the equation (1), and `circuit_add` implements the right hand side. *The most important reason this approach works well is that both circuits are derived from the same multiplier circuit, i.e. the gate-level multiplier under verification. Therefore, they have significant structural similarity that could be utilized during equivalence checking.* However, an efficient solution to the resulting equivalence checking problems requires additional structural information, as will be discussed in the following section.

3. Using structural information to speed up equivalence checking subproblems

State-of-art equivalence checkers utilize an incremental approach to explore the internal structural similarities between the circuits being compared. Starting from the equivalence of the primary inputs, candidate equivalent signal pairs are checked for equivalence, and the equivalent signals are merged. In the local BDD-based approach, merged equivalent signals form the support for building local BDDs of other candidate equivalent signal pairs, obviating the need for building a complete BDD for a huge portion of the circuit. Hence, the equivalence checking problem could be solved most efficiently if the compared circuits have similar structures. Since the equality of local BDDs is only a sufficient but not a necessary condition for the equivalence of the compared signals, false negatives might occur, i.e. differentiating patterns at the internal support are found even though the signals are equivalent. Moving the support backward is necessary to eliminate the signal dependency that causes these false negatives. The equivalence checking subproblems mentioned in the previous section could be solved much more efficiently by incorporating the following two crucial pieces of structural information:

1. **Identifying the multiplier.** The first structural information needed is to decide which operand, when decomposed, will lead to simpler equivalence checking problems, i.e. the two circuits under comparison have greater structural similarity. In other words, which operand should be used as the multiplier? The multiplier can be found by examining the number of gates/ signals in the fanout cone of the most significant bit (MSB) of the operands. An example in Figure 2 shows an 8-bit addstep multiplier with the inputs of the adder modules given by the product bits as arranged in Figure 1 (a). About 1/2 of the circuit gates/signals (left triangle) lie in the fanout cone of a_7, and about $1/7$ of the circuit gates/signals (lowest row of the addition tree) in the fanout cone of b_7. This difference is a consequence of the multiplicand/multiplier asymmetry discussed above. Clearly, decomposing the one whose MSB has the smaller fanout cone will result in simpler equivalence-checking problems. Using this criterion, the equivalence checking could proceed efficiently by first merging the isomorphic portions of two circuits under comparison from the primary inputs toward the primary outputs. This would result in difference portions of only about the size of $1/n$ of the n-bit multiplier circuit.

2. **Judicious selection of local support.** Existing equivalence checking techniques will still be inadequate for these subproblems due to the strategy used in handling false negatives. The source of false negatives in an addition tree could be understood from an implementation of a full adder as shown in Figure 3. If the local support cuts through a full

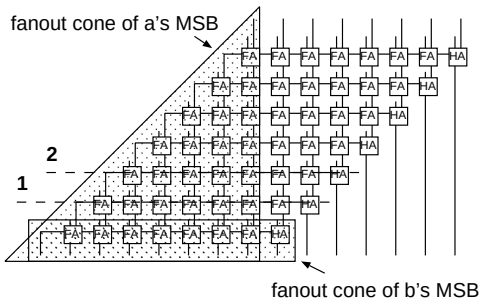

Figure 2. An implementation of an addstep multiplier to demonstrate employed structural analysis.

adder as shown by the dashed line in Figure 3, this support ignores the one hot property of the outputs of the upper half adder. From another perspective, the XOR gate in this design could be replaced by an OR gate to implement an equivalent full adder. When faced with negatives (true or false), present practice is to move the support backward one to several logic levels. However, such moving of support could continue to include a set of signals with strong correlations and could eventually reach all primary inputs in our setting, thus losing the advantage of using any internal structure similarity. We could avoid the undesired false negatives by carefully selecting the next support by some special structural information. We can use the adjacent adder in the merged part of the two circuits to push the support backward. This amounts to using the boundary of the fanout cone of the one less significant bit as the next backward support. This process is illustrated in Figure 2, showing an equivalence checking step used to reduce the multiplier size from 7 bits to 6 bits. Suppose the current support using the boundary of the merged isomorphic circuit (shown as dashed line 1) produces false negatives. The false negatives could be eliminated by using the boundary of the fanout cone of b_6 as the support (shown as dashed line 2); this effectively moves the support backward with significantly lower probability of false negatives. Since the multiplier has an exponential BDD size, the support could not be pushed backward indefinitely. In practice, we find that using only one adjacent bit suffices to eliminate all false negatives in any designs we encountered. The existence of such special local supports seems to be characteristic of arithmetic circuits.

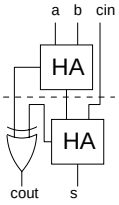

Figure 3. The appearance of a false negative when the support cuts through a full adder.

width (#bits)	addstep		csatree		cla		booth*	
	time	size	time	size	time	size	time	size
16	0.97	0.49	2.28	0.64	1.11	0.49	2.99	0.56
32	12	0.59	24	1.18	14	0.60	40	0.72
48	60	0.70	84	2.35	70	0.70	189	1.02
64	184	0.80	292	4.28	213	0.92	677	2.04
80	436	0.89	825	6.28	505	0.99	2223	2.90
96	937	1.37	1296	8.93	1045	1.95		
112	4213	3.05	3384	19.97	4440	3.05		
128	7450	3.39	10761	114.13	7014	4.59		

Table 1. Experimental results of induction-based verification of multipliers using local BDD-based equivalence checking procedure.

Another source of false negatives is the dependence among signals within the local support by shallow logic. This problem could be easily solved by learning the logical dependence of local support signals by one logic level backwards.

By employing these two pieces of structural information based on fanout cone analysis, we can significantly speed up the resulting equivalence checking.

Another crucial ingredient in the local BDD-based approach is variable reordering. The fact that BDDs are built upon a substantial local support for all primary outputs indicates that initial variable ordering and subsequent reordering are very important. In fact, variable reordering is the major limiting factor of this BDD-based approach. Experimental results show that variable ordering starting from the variable closest to the most significant bit of the output makes a good initial ordering and the sifting algorithm has the best performance.

One benefit of this approach is that it also greatly simplifies error diagnosis. Since the n-bit multiplier circuit is effectively decomposed into n portions with roughly equal sizes, the source of the error is limited to a small portion of the circuit in the case of inequivalence. The pseudocode in Section 2 is presented to proceed from the full circuit to the smaller subcircuits for clarity. In practice, for easier error diagnosis, the order of equivalence checking subproblems should proceed in the other direction, i.e. from null subcircuits to the full circuit. This order will result in the discovery of errors at simpler equivalence checking subproblems.

4. Experimental results

The experimental results are summarized in the following table with the verification time given in seconds and the peak memory usage given in Mb. The results are obtained on a Pentium III 733 Mhz computer with 256Mb memory running Linux OS. The verification of the C6288 netlist as a multiplier takes only 4.67 secs and uses 0.94Mb memory. In this table, the verifier is given implementations from four different architectures as illustrated in Figure 1 with operands of 16 to 128 bits, but the architecture information is not used during verification.

The results show that the verification of a gate-level multiplier is no longer bounded by memory limitations with this method. Due to the large number of variables in the local support, most of the execution time is spent on variable reordering. Notice the results for booth multipliers apply only to the verification of the nonzero bits. The leading zeros in the resulting equivalence check-

ing problems turn out to contain false negatives that could not be easily removed by the proposed heuristics. It seems to be a consequence of the characteristics of booth recoding, which has significant signal dependence on lower significant bits. The result for a Wallace tree multiplier is not available because we do not yet have a reasonable execution time at a specified width stated in the table. The Wallace tree architecture has a much larger local support variable size with its multiple-connected addition tree structure, so the time spent in reordering significantly degrades the result.

5. Generalizations to arithmetic circuits

Similar approaches to the verification of arithmetic circuits using functional equations have been proposed before. For example, the functional equation $(x+1)\times y = x\times y + y$ is employed in [9] to verify multipliers. However, this approach requires inserting circuits close to the primary inputs, making the resulting equivalence-checking problem much harder and more difficult to control than the approach described here. The idea of decomposing multipliers into roughly equal parts has also appeared in [10], where the circuit is decomposed according to the fanin cone of the primary outputs. Our research shows that the decomposition based on the fanout cone of the inputs is more efficient. However, the approach in [10] is capable of handling the Wallace tree architecture up to 32 bits, due to the smaller support variable size in that decomposition. Efforts to combine these two types of decompositions are currently under investigation.

The proposed method of induction-based multiplier verification can be extended to more general arithmetic circuits. The basic idea is to prove incrementally that the circuit corresponding to an operand does indeed implement the desired function and can remove the corresponding part of the circuit (operand) to simplify the verification problem. The operand to be removed at each step is the one closest to the primary output of the circuit, or the one with the smallest fanout cone, so that the resulting equivalence checking problem is the simplest. This *prove-and-remove* approach utilizes the locality of signals corresponding to the operands and incrementally reduces the circuit into simpler ones. Arithmetic circuits involving $+, -$, and $\times$ operations could be verified if the operands could be successively removed from the neighborhood of the primary outputs. At this point, the division operation could not be verified directly using this method due to the lack of observability of the intermediate remainders. The pseudocode of this scheme for general arithmetic circuits is the following:

```
arith_verify(netlist,expression,width,#operands){
 order=find_operand_order(netlist);
 circuit2=netlist;
 for (i=0; i < #operands;++i){
     circuit1=circuit2;
     circuit2=set_next_operand_zero(circuit1,order,i);
     switch(Op of the removed operand)
      case +:
       circuit_add=add_adder(circuit2,i);
       check_eq(circuit1,circuit_add);
      case -:
       circuit_sub=add_subtractor(circuit2,i);
       check_eq(circuit1,circuit_sub);
      case *:
       verify_multiplier(circuit1,width,i);
     simplify(expression,op,operand[i]);
 }
}
```

The verification of an arithmetic circuit with m n-bit operands with l multiplications could then be reduced to at most $(m-l)+n\times l$ equivalence checking problems. One benefit of this prove-and-remove methodology is the capability of verifying rescheduled arithmetic circuits. Datapath rescheduling based on arithmetic properties is a technique commonly employed in commercial synthesis tools. Due to the global nature of datapath rescheduling, typical equivalence-checking-based verification technique cannot verify the correctness of circuits under such transformation. For example, if $a\times b + c\times b$ is implemented as $(a+b)\times c$, the methodology proposed could be used to verify this circuit without the *a priori* knowledge of the underlying implementation.

6. Conclusion

We propose an induction-based technique that could successfully verify gate-level implementations of multipliers with different architectures. In implementing this scheme, we observe the need of different strategies for choosing local supports for random logic and datapath in the presence of false negatives in the equivalence checking problems. The proposed scheme could also be generalized to the more general arithmetic circuits. This method could also be integrated into RTL to gate-level equivalence checking tools for more efficient datapath verification.

Acknowledgment *The authors would like to thank K. -C. Chen for suggesting the practical importance of this problem. This work was supported in part by SRC Task 835.001 and NSF International Research Center for SoC.*

References

[1] R. E. Bryant, *Graph-based algorithm for Boolean function manipulation*, IEEE Transactions on Computers, Vol C-35, No. 8, pp. 677-691, 1986.

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

[3] D. Brand, *Verification of large synthesized designs*, Proc. of International Conference on Computer Aided Design (ICCAD), p 534-537, 1993.

[4] S. Y. Huang and K. T. Cheng, *Formal Equivalence Checking and Design Debugging*, Kluwer Academic Publishers, 1998.

[5] R. E. Bryant, *On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication*, IEEE Transactions on Computers, Vol 40, No. 2, pp. 205-213, 1991.

[6] J. Burch, *Using BDDs to verify multipliers*, Proc. of ACM/ IEEE Design Automation Conference (DAC), pp. 408-412, 1991.

[7] R. E. Bryant and Y. A. Chen, *Verification of arithmetic circuits with binary moment diagrams*, Proc. of the 32nd ACM/IEEE Design Automation Conference, pp. 535-541, 1995.

[8] K. Hamaguchi, A. Morita, and S. Yajma, *Efficient construction of binary moment diagrams for verifying arithmetic circuits*, International Conference on CAD, 1995.

[9] M. Fujita, *Verification of arithmetic circuits by comparing two similar circuits*, Proc. of the 8th International Conference on Computer Aided Verification (CAV), pp. 159-168, 1996.

[10] T. Stanion, *Implicit verification of structually dissimilar arithmetic circuits*, Proc. 1999 IEEE International Conference on Computer Design (ICCD), pp. 46-50, 1999.

[11] M. Keim, M. Martin, B. Becker, R. Drechsler, and P. Molitor, *Polynomial formal verification of multiplier*, IEEE VLSI Test Symposium, pp. 150-155, 1997.