Verification of Arithmetic Circuits by Comparing Two Similar Circuits

Masahiro Fujita
Fujitsu Laboratories of America, Inc.
3350 Scott Blvd., Bldg. #34, Santa Clara, CA95054
fujita@fla.fujitsu.com

Abstract. Recently there have been a lot of progress in technologies for comparing two structurally similar large circuits [2, 14, 13]. Circuits having more than 10,000 gates, whose BDD cannot be built, have been verified in several minutes. However, arithmetic circuit verification with respect to specification is still a hard problem. As shown in [16] some arithmetic circuits, such as multipliers, square function. cube functions, etc., must satisfy some recurrence equations, such as $f(x+1,y) = f(x,y) + y \ \ where \ f(x,y) = xy$, and those equations can be used for verification. In this paper, we use such recurrence equations in order to drive Boolean comparison problems of structurally similar circuits. That is, left hand sides and right hand sides of equations are realized as separated circuits and then compared. Using the recurrence equation properly, these circuits have many internal equivalent signals and many implications among signals, by which Boolean comparison programs, such as [2, 14, 13], can work very effectively. Using the proposed method, 16-bit multipliers, such as C6288 of ISCAS85 benchmark circuits, are verified within 12 minutes.

1 Introduction

Formal verification techniques have been paid much attention recently. There have been lots of works on formal hardware verification [11, 10], and among them, Binary Decision Diagram (BDD) [3] based verification techniques, such as [5, 8, 17, 6, 15], have given successful results for practical designs.

However, BDD may not work well for arithmetic circuits, such as multipliers. Therefore, several extensions are made on BDD, such as, BMD [4], HDD [7], OKFDD [9], etc. Although originally word-level verification is necessary in order to use BMD. by using the technique shown in [12] which compute BMD from outputs to inputs instead of inputs to outputs, we can now use BMD directly to verify arithmetic circuits, such as multipliers. However, if there are errors (bugs) in the circuits, BMD can easily blow up and the verification program may not terminate, since those circuits represent different logic functions from multiplier which can have exponential sizes of BMD. Of course this depends on each error but we actually observed this BMD explosion by randomly inserting logical errors to the multiplier circuits and generating BMDs.

In this paper, we show another approach to attack the verification of arithmetic circuits. Instead of directly generating BDD (or its extensions) from given circuits, we create circuits based on the recurrence equations that must be satisfied by the circuits. This idea was originally proposed by Ochi [16]. For example a recurrence equation for multipliers is:

$$f(x+1,y) = f(x,y) + y \quad \textit{where } f(x,y) = xy \textit{ and } x,\ y \textit{ are inputs}$$

Any circuits which satisfy the above recurrence equation are multipliers[1]. He used recurrence equations to verify arithmetic circuits by first generating BDD from the circuits and then check if that BDD satisfy the required recurrence equations. Clearly this method has a drawback that we have to build BDD for the circuits first, which is often impossible for large arithmetic circuits.

On the other hand, recurrence equations, such as shown above (for multipliers), may indicate a comparison problem of two circuits (or Boolean functions). That is, checking recurrence equation means comparing the left hand side and right hand side of the equation. So, basically we can use Boolean comparison techniques for such equivalence checking.

Recently there have been much progress in technologies for Boolean comparison of similar circuits. Here "similar" means that we can find many logical relationships, such as equivalences or implications, among the internal signals in the two circuits to be compared. In a practical design environment, designers want to check the equivalence of the two circuits which are very structurally similar. For example, it is often the case to check the equivalence between unoptimized circuits and manually optimized circuits. For such cases, there are lots of relationships among the internal signals in the two circuits. By utilizing these relationships, methods like [2, 14, 13] can verify much larger circuits than the circuits which can be verified just by BDDs. 10,000 gates or larger circuits can be verified within practical time.

Basically recurrence equations suggest two structurally similar circuits (left hand side and right hand side). Here we propose a new verification method

[1] Circuits must also satisfy boundary conditions, such as, $f(0,y) = 0$, which can be checked rather easily.

for arithmetic circuits based on recurrence equations. We first generate two circuits which correspond to left hand side and right hand side of the recurrence equations. Then apply Boolean comparison program for similar circuits to those circuits. Please note that we need only one circuit which should be verified. The two circuits to be compared are generated from that circuit by adding appropriate extra circuits, such as adders, incrementors, etc. Also note that we do not need specification in Boolean functions. Specification is fully described in the recurrence equations that we are using to generate two circuits.

By using case splitting appropriately, we can verify 16-bit multipliers, such as C6288 of ISCAS benchmark circuits, in less than 12 minutes on Sparc20. Moreover, different from the method in [12], the proposed method can finish verification in similar time, even if the circuits are not correct as shown in section 3.

In the next section, we introduce our verification method. Then section 3 gives preliminary results. Section 4 is our concluding remarks. Although we discuss only about multipliers for simplicity, the proposed method can be applied many arithmetic functions which have proper recurrence equations, such as, square functions, cube functions, etc.

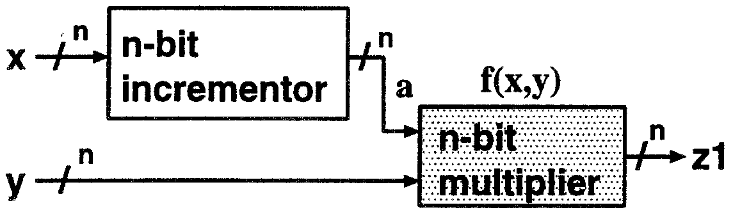
2 Verification algorithm

In this section we introduce our verification method. For simplicity, we use multipliers as examples all the time, although we can verify many other arithmetic circuits which have proper recurrence equations, such as, square functions, cube functions, etc. As long as there are proper recurrence equations, we can verify any circuits, including random circuits (assuming such recurrence equations are given)[2].

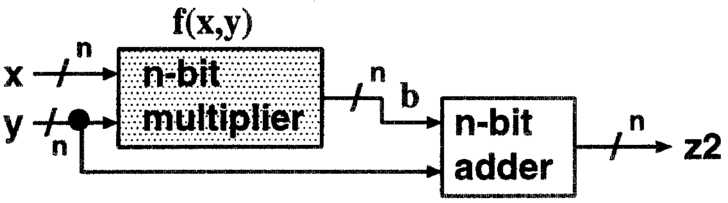
The basic idea for multipliers is illustrated in Figure 1. Since multipliers satisfy the following equation, two circuits which are derived from left hand side and right hand side of the equation respectively must realize the same Boolean function.

$$f(x+1,y) = f(x,y) + y \quad \textit{where } f(x,y) = xy \textit{ and } x,\ y \textit{ are inputs}$$

[2] In some sense, we can consider the proposed method is a kind of self-checking methods proposed in [1]. What we are doing in this paper can be described in the following way: by appropriatly using recurrence equations (self checking properties), we are reducing verification problems into Boolean comparison problem for similar circuits. Of course, if the reduced Boolean comparison problems are too large, we can use random simulation based checking just like in [1].



(a) Circuit corresponding to f(x+1,y)



(b) Circuit correponding to f(x,y)+y

Fig. 1. Circuit realization of the recurrence equation for multipliers

Please note that the two multipliers in Figure 1 are the same circuit which we want to verify. We are assuming here that incrementor and adder are given and they are guaranteed to be correct.

Please also note that the above equation together with boundary condition for $x = 0$ completely specify the function and that must be multiplier.

So, by comparing the two circuits shown in Figure 1, we can formally verify multipliers[3]. But this is not an easy Boolean comparison problem. Clearly we cannot build BDD for each circuit, if that is a large bit width multipliers, such as C6288 of ISCAS85 benchmark circuits. Although large portion of the two circuits are the same sub-circuits (multiplier), they are not similar circuits in the sense that we cannot find many equivalent signals between the two circuits. So, we cannot directly apply the Boolean comparison methods like [2, 14, 13].

However, if we consider only the case where the least significant bit of x, x_0 is 0, then the incrementor becomes just an inverter as shown in Figure 2[4]. Thus the two circuits become like the ones shown in Figures 3. There are many equivalent

[3] In this paper, we assume that extra circuits, such as incrementor, adder, subtracter, etc., are guaranteed to be correct. Or those should be verified first

[4] If $x_0 = 0$, then increment does not affect the values of $x_1, x_2, ..., x_{n-1}$.

signals between the two circuits, since most inputs are common and large portion of the circuits are the same. In fact there are a lot of functional relationships among internal signals of the two circuits.

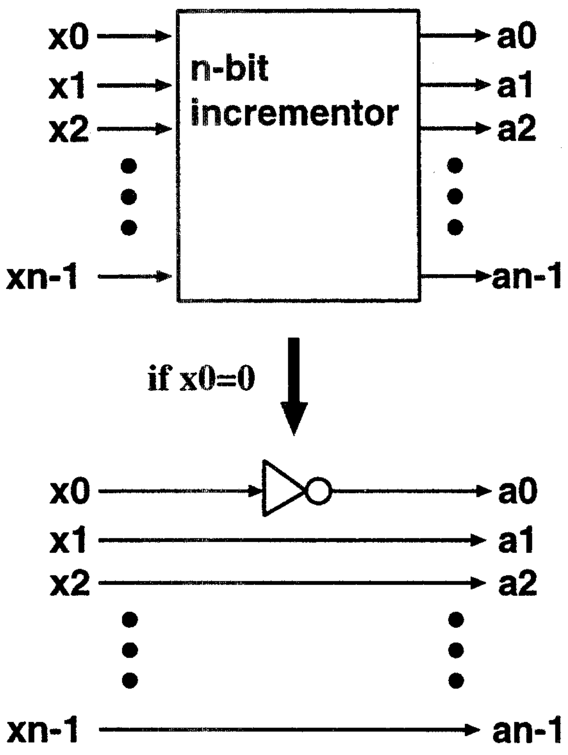


Fig. 2. If the least significant bit is 0, incrementor becomes just an inverter

By this case splitting, we can verify multipliers when $x_0 = 0$. Then how about the cases when $x_0 = 1$? We can proceed with the same idea: further case splitting with $x_1, x_2, \ldots$. For example, if $x_0 = 1$ but $x_1 = 0$, then the incrementor becomes just two inverters as shown in Figure 4. Again the two circuits generated are similar as shown in Figure 5.

The next splitting case is $x_0 = 1, x_1 = 1, x_2 = 0$ which needs three inverters for x_0, x_1, and x_2. This case splitting process can be continued until we reach the case where $x_0 = 1, x_1 = 1, x_2 = 1, \ldots, x_{n-2} = 1, x_{n-1} = 0$.

What we are doing here is just check the equation:

$$f(x+1, y) = f(x, y) + y \quad \textit{where } f(x, y) = xy \textit{ and } x,\ y \textit{ are inputs}$$

by case splitting the values of x_i.

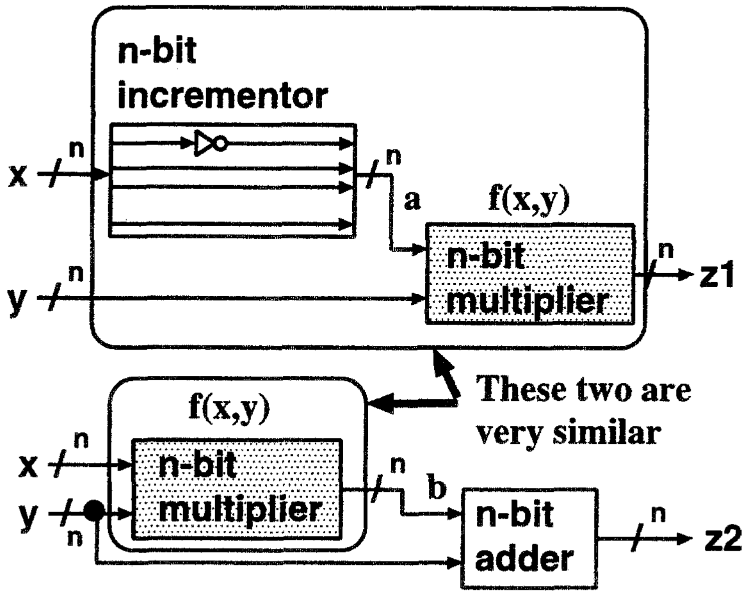


Fig. 3. By assuming $x_0 = 0$, the circuits become very similar

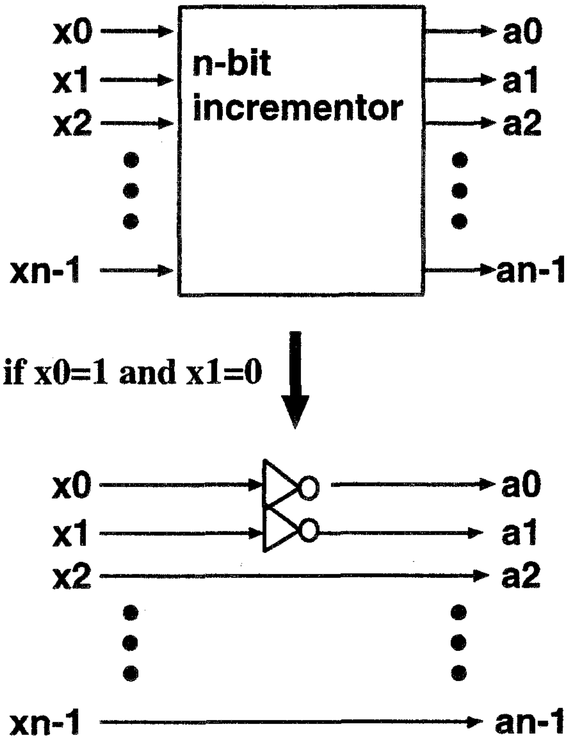


Fig. 4. The case where $x_0 = 1$ but $x_1 = 0$

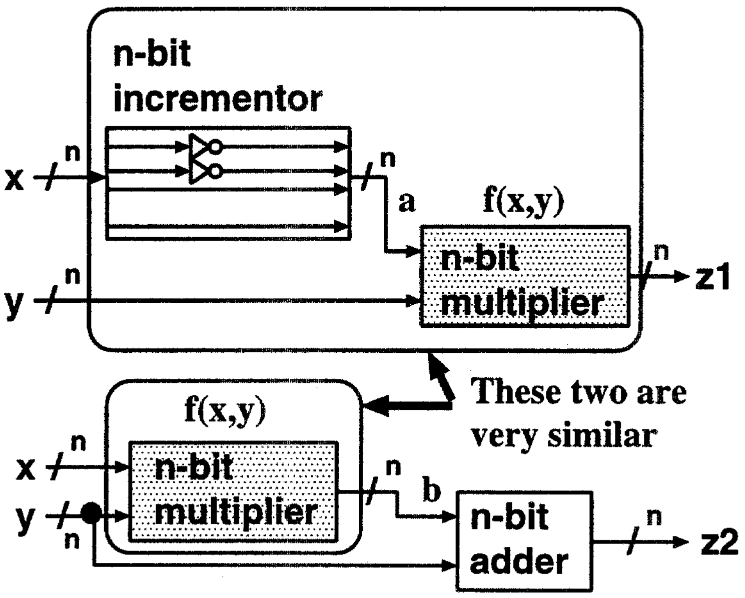


Fig. 5. By assuming $x_0 = 1, x_1 = 0$, again the circuits become very similar

As we have more number of inverters, the two circuits become less similar. However, as we have more number of inverters, we can fix the values of x_i more. That is, in the case of Figure 4, since this is the case where $x_0 = 1$ and $x_1 = 0$, we can fix the values of x_0 and x_1. So there are trade-offs in terms of difficulty and the most difficult case happens when there are two inverters as shown in Figure 4 according to our experiments for C6288 in the next section.

By using the above case splitting, we can keep the similarity of the two circuits. These circuits should be rather easy circuits for the Boolean comparison methods like [2, 14, 13]. In fact, as shown in the next section, we have found many equivalent signals which drastically reduce the verification time or complexity of the problem.

3 Preliminary experimental results

We did some preliminary experiments for multipliers. We plan to do more intensive experiments using other types of circuits, such as, square functions.

Our program first generates net-lists for the two circuits in Figure 2, 4, and others [5] from the given multipliers. Then apply our Boolean comparison programs to them.

[5] In the case of 16-bit multipliers, there are 16 cases in total. But some of them are trivial, since most of x_i are constants.

We verified C6288 of ISCAS85 benchmark circuits for its first 16 outputs, since as shown in Figure 1, all values should be the same bit-width (16bit in this case). The results are shown in Figure 6. We did two types of experiments. The first one is to just verify C6288 circuit, which is a correct multiplier. It took only less than 12 minutes in total to verify.

The program found 342 equivalent internal signals of the two circuits out of 3̃60 internal signals for the case of Figure 2. So large portion of the two circuits are equivalent and that is why verification can finish so quickly. The most time consuming case is the one shown in Figure 4 which took 8 minutes to finish. This is the case where the two circuits are similar but not so much and their circuit sizes are still large (only small number of x_i have fixed value). All the other cases are less than one minute.

Multiplication Circuit	CPU time sec. on Sparc20	
C6288 (first 16 outputs)	Original (correct)	Error inserted
Case: x0=0	14.0	2.0-60.0 depending on errors inserted
Case: x0=1, x1=0	496.0	
All other cases	Less than 60.0	

Fig. 6. Results for 16-bit multipliers

Second experiment we did is to try to verify incorrect multipliers (verification fails) by intentionally inserting errors into C6288 (changing function of a gate, etc.[6]). Depending on changes, it took less than one minute (sometime in a couple of seconds) to prove the circuit is not a multiplier. Depending errors, the cases where verification fails are different, but mostly verification fails in multiple cases. Again this is extremely fast. Please note that the method in [12] may not work well for incorrect circuits.

4 Conclusions

We have shown a verification method for arithmetic circuits. We also demonstrated that C6288 can be verified in less than 12 minutes. Even if the circuits are not correct (there is a bug in the circuits), verification time remain similar or less. Also, different from BMD or HDD based methods, we do not need another BDD package, such as, BMD package. We can use existing BDD packages or Boolean comparison programs to verify arithmetic circuits. We believe that the proposed method has a significance in its applications.

Although in this paper we only discussed about combinational circuits, the proposed techniques can be applied to sequential circuits by deriving appropriate recurrence equations. Surely this is one of our future research topics.

Also, the proposed method can be considered to be a kind of self-checking methods proposed in [1]. What we are doing here can be described in the following way: by appropriately using recurrence equations (self checking properties), we are reducing verification problems into Boolean comparison problem for similar circuits. Of course, if the reduced Boolean comparison problems are too large, we can use random simulation based checking just like in [1]. We are planning to explore this area and study on extensions of the proposed method.

References

1. M. Blum, M. Luby, and R. Rubinfeld. "self-testing/correctig with application to numerical problems". In *Proc. of 22nd ACM Theory of Computing*, pages 73–83, 1990.
2. D. Brand. "verification of large synthesized designs". In *Proc. of ICCAD*, pages 534–537, Nov. 1993.
3. R.E. Bryant. "graph-based algorithms for boolean function manipulation". *IEEE Trans. Computer*, C-35(8):667–691, Aug. 1986.

[6] Even if we make many changes in the circuit, situations are the same. The two circuits we generate according to Figure 1 are very similar.

4. R.E. Bryant and Y.-A. Chen. "verification of arithmetic functions with binary moment diagrams". In *Proc. of 32nd DAC,* Jun. 1995.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. "symbolic model checking: 10^{20} states and beyond". In *Proc. of the Fifth Anual IEEE Symposium on Logic in Computer Science,* Jun. 1990.
6. H. Cho, G. Hachtel, S-W. Jeong, B. Plessier, E. Schwarz, and F. Somenzi. "atpg aspects of fsm verification". In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-90),* pages 134–137, Nov. 1990.
7. E.M. Clarke, M. Fujita, and Z. Zhao. "hybrid decision diagrams - overcoming the limitations pf mtbdds and bmds". In *Proc. of ICCAD,* pages 159–163, Nov. 1995.
8. O. Coudert and J.C. Madre. "a unified framework for the formal verification of sequential circuits". In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-90),* pages 126–129, Nov. 1990.
9. R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. Perkowski. "efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams". In *Proc. of 31st DAC,* Jun. 1994.
10. M. Fujita. "rtl design verification by making use of datapath information". In *Proc. of ICCD-92,* pages 592–597, Oct. 1992.
11. A. Gupta. "formal hardware verification methods: a survey". *Formal Methods in System Design,* Vol. 1(2/3), Oct. 1992.
12. K. Hamaguchi, A. Morita, and S. Yajima. "efficient construction of binary moment diagrams for verifying arithmetic circuits". In *Proc. of ICCAD,* pages 78–82, Nov. 1995.
13. J. Jain, R. Mukherjee, and M. Fujita. "advanced verification techniques based on learning". In *Proc. of 32nd DAC,* pages 420–426, Jun. 1995.
14. W. Kunz. "hannibal: An efficient tool for logic verification based on recursive learning". In *Proc. of ICCAD,* pages 538–543, Nov. 1993.
15. K.L. McMillan. "symbolic model checking: An approach to the state explosion problem". Technical Report CMU-CS-92-131, Carnegie Mellon University, May 1992.
16. H. Ochi and S. Yajima. "formal design verification of combinational circuits specified by recurrence equations". In *Proc. of SASIMI'95,* pages 101–105, Aug. 1995.
17. H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-Vincentelli. "implicit state enumeration of finite state machines using bdds". In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-90),* pages 130–133, Nov. 1990.