ACCURATE LOGIC SIMULATION IN THE PRESENCE OF UNKNOWNS

Susheel J. Chandra

CrossCheck Technology
2001 Gateway Place
San Jose, CA 95110

Janak H. Patel

Coordinated Science Laboratory
1101 W. Springfield Ave.
Urbana, IL 61801

ABSTRACT

Logic simulation plays a very important role in the design of digital systems. A good logic simulator must be capable of accurately predicting signal values that occur in the actual circuit. However, this is extremely difficult in the presence of unknowns. Accurate logic simulation in the presence of *unknowns* has been shown to be an *NP*-complete problem. In this paper we present algorithms which use high-level descriptions to simulate the circuit exactly. The complexity of these algorithms, and the feasibility of their implementation is also discussed.

1.1. INTRODUCTION

Logic simulation plays a key role in the design of digital systems. For VLSI circuits, where design errors are very costly logic simulation is an invaluable tool. A good logic simulator must be capable of accurately predicting signal values that occur in the actual circuit. This is extremely difficult especially in the presence of *unknowns*. In most gate-level simulators, method of handling unknowns always lead to pessimistic results in that unknowns are erroneously assigned to some lines which should have a logic value of 0 or 1.

It has been shown [1] that accurate logic simulation in the presence of unknowns is an *NP* -complete problem. Chang and Abraham proposed the use of high-level descriptions to alleviate the problem. However, the algorithms presented in [1] resort to the use of heuristics and thus in the worst case can be of exponential complexity. They also present an approximate algorithm and compare its performance and accuracy to exact simulation. In this paper we will present linear time algorithms that use high-level descriptions to perform accurate logic simulation.

The remainder of this paper is organized as follows. Section 2 demonstrates the unknown propagation problem with the help of an example. In Section 3 we present three different algorithms for accurate logic simulation and prove their correctness. Section 4 deals with the complexity of these algorithms and other feasibility issues.

1.2. Unknown Propagation Problem

The existence of unknown values makes the problem of simulating a circuit precisely very difficult. In current gate-level simulation, methods of handling unknowns always lead to pessimistic results, because unknowns are erroneously assigned to some lines which should have logic value 0 or 1. As mentioned earlier, it has been shown [1] that accurate logic simulation in the presence of unknown values is *NP-complete*. Figure 1 illustrates the problem. In this case a gate-level simulation produces an output ×, however a more careful examination reveals that the output is always a 1. The circuit of Fig. 1. is just a two input multiplexer with both inputs set to 1 and an × on the control line. In this situation, no matter what value the control line takes on, the output of the multiplexer is always a 1.

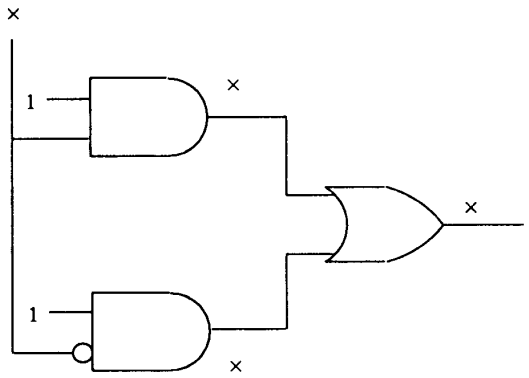

Fig. 1. Problem of unknown propagation in a gate-level simulation.

It is obvious that the problem occurs when two unknowns originating from the same point reconverge at some gate. The use of multiple unknown values and their complements alleviates the problem [2] however, the number of distinct values needed quickly becomes unmanageable. Our approach to the problem uses high-level representations of the function modules present in the circuit. In this paper we will consider different boolean representations such as, sum-of-products, product-of-sums, prime implicants etc., from a logic simulation stand point. Using these representations we will develop algorithms for accurate logic simulation in the presence of unknowns.

1.3. Algorithms

Logic simulators widely use the three-valued algebra (0,1,×) to model the signal values in the circuit. More sophisticated simulators use multiple values to accurately model strengths of the signals. In this paper we will restrict ourselves to the three-valued logic (0,1,×), where × denotes the unknown

Acknowledgement: This research was supported by the Semiconductor Research Corporation under Contract 88-DP-109.

state i.e., it is uncertain weather the value should be a 0 or a 1. The following terms are used in the sequel.

DEFINITION 1: Let f be a boolean function of n variables, $x_1, \ldots, x_n$, where $x_i \in \{0,1,\times\}$. Any implicant of f is termed a *cube*. We use the notation $\alpha[i]$ to denote the i *th* coordinate of the cube α. This corresponds to the input x_i. □

DEFINITION 2: Let α and β be two cubes. Then their intersection known as the *cube intersection* $\alpha \cap \beta$ is defined using the coordinate intersection table below and the following rule. $\alpha \cap \beta = \varnothing$ if any coordinate intersection is $\varnothing$, else, $\alpha \cap \beta$ is the cube formed from the respective coordinate intersections.

$\cap$	0	1	×
0	0	$\varnothing$	0
1	$\varnothing$	1	1
×	0	1	×

□

DEFINITION 3: A cube α *covers* a cube β, denoted $\beta \subseteq \alpha$, if and only if $\alpha \cap \beta = \beta$. □

In this section we will first give an algorithm to perform forward implication using a sum-of-products or product-of-sums representation and then give an alternate boolean representation for a circuit to overcome the complexity of unknown value propagation. Note that the given sum-of-products expansion need not be in the simplest form known as a *minimal sum*, or the most complex form known as the *minterm expansion* of f. We will term this a *disjunctive form* (DF) of f. Let $\alpha_1, \ldots, \alpha_m$ be the product terms in the given DF of f. Let β be the cube corresponding to the current input assignment of f. Thus, if any product term α_i covers β, the output of f is 1. Also it is easy to see that if the intersection of β with all product terms is $\varnothing$, then the output of f is 0. If neither of the above two conditions are satisfied, then this algorithm cannot correctly determine the output and hence declares it as unknown. Algorithm 1 illustrates the above procedure. In the dual case, where we are given the product-of-sums expansion or *conjunctive form* (CF), α_i is a sum term or an *alterm* and the algorithm is very similar to Algorithm 1.

EXAMPLE 1: If each gate in the circuit of Fig. 1 is treated as a function module and Algorithm 1 is used to simulate the circuit, it will incorrectly determine the output to be unknown. □

Thus given a DF of f, Algorithm 1 is inexact. In fact, any exact algorithm using just a DF of f would have exponential complexity. However it is possible to reduce this complexity by keeping additional information. This is demonstrated by the following example.

EXAMPLE 2: Consider the following DF for f.

$$f(a,b,c,d) = ac\bar{d} + a\bar{c}\bar{d} + b\bar{c}d$$

The three product terms of f are shown in Table I.

TABLE I
Tabular form of f

a	b	c	d	cube
1	-	1	0	$ac\bar{d}$
1	-	0	0	$a\bar{c}\bar{d}$
-	1	0	1	$b\bar{c}d$

Given $a=1, b=1, c=0$ and $d=\times$, using Algorithm 1 $f(1,1,0,\times) = \times$. This is because the two cubes, $a\bar{c}\bar{d}$ and $b\bar{c}d$, which are consistent with the input cube, $ab\bar{c}$ (i.e., $\beta \cap \alpha_i \neq \varnothing$) do not independently cover the cube $ab\bar{c}$. □

ALGORITHM 1: Forward implication using a DF of f.

```
begin
 intersect ← null;
 for each i ∈ {1,2,...,m} do
  begin
   if β ⊆ α_i then return 1;
   else if β ∩ α_i ≠ ∅ then intersect ← notNull;
  end
 if intersect = null then return 0;
 else return ×;
end
```

If the values for a, b and c are substituted in the above DF of f, the resulting expression is $f = d + \bar{d} = 1$. However, Algorithm 1 erroneously propagates an unknown value at the output of f. We call this the *consensus problem*. Fig. 3 shows the product terms of f on a Karnaugh map. If the consensus term $ab\bar{c}$ is present in the given DF of f, Algorithm 1 will correctly determine the output of f under the constraints $a=1, b=1, c=0$. Thus if we have the following Theorem.

THEOREM 1: Given the *complete sum* (sum of all *prime implicants*) of f, Algorithm 1 correctly determines the output of f.

PROOF: Since all prime implicants of f are available to us, any input combination which makes f a 1 must be covered by some prime implicant. Also, if some input combination reduces all the product terms in the complete sum of f to 0, then the output of f must be a 0 since f does not have any other implicant. □

However, the complete sum is not the only way to reduce the complexity of the problem. Let us consider an alternate form of representation. Suppose both, DF's of f and $\bar{f}$ are available to us. Let $\gamma_1, \ldots, \gamma_p$ be the product terms in the DF of $\bar{f}$. If after using Algorithm 1 on the DF of f, the value of $f = \times$, we apply Algorithm 1 on the DF of $\bar{f}$. This may determine the value of f to be 0 or 1, and the value of f can be deduced from it. However, if the value of $\bar{f} = \times$, then the value of f is truly $\times$. Thus we have the Algorithm 2, and a theorem which proves its correctness.

ALGORITHM 2: Exact forward implication using a DF of f and $\bar{f}$

```
begin
 intersect ← null;
 for each i ∈ {1,2,...,m} do
  begin
   if β ⊆ α_i then return 1;
   else if β ∩ α_i ≠ ∅ then intersect ← notNull;
  end
 if intersect = null then return 0;
 else
  begin
   for each i ∈ {1,2,...,p} do
    if β ∩ γ_i ≠ ∅ then return x;
   return 1;
  end
end
```

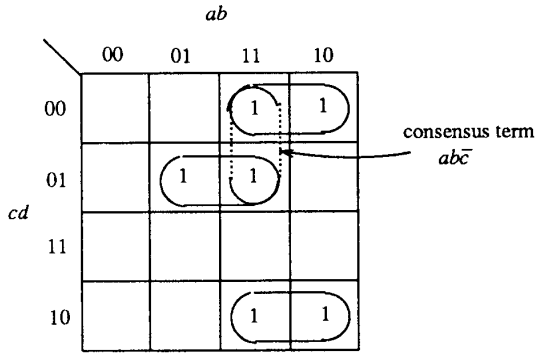

Fig. 3. Karnaugh map illustrating the consensus term.

THEOREM 2: Algorithm 2 correctly determines the output of f

PROOF: If f is null, all terms in a DF of f must evaluate to zero i.e., `intersect = null` in the above algorithm. Similarly if f is a tautology, all terms in a DF of $\bar{f}$ must evaluate to zero. In Algorithm 2 we compare the cube β with both f and $\bar{f}$, this will accurately determine if f is null, tautology or unknown. □

EXAMPLE 3: For the function in Example 2, a DF for $\bar{f}$ is

$$\bar{f} = \bar{a}\bar{b} + cd + \bar{a}\bar{d} + \bar{b}\bar{c}d$$

The product terms of f and $\bar{f}$ are shown in Table II. The first `for` loop of Algorithm 2 is exactly the same as Algorithm 1. However, at the end of the loop if the value of the variable `intersect` is `notNull`, Algorithm 1 returns an x, whereas Algorithm 2 goes on to examine $\bar{f}$. In this example, the intersect $\beta \cap \gamma_i = \emptyset$ for each cube γ_i of $\bar{f}$. Thus Algorithm 2 will return the correct output $f = 1$. □

From Theorems 1 and 2 we know that it is possible to efficiently deduce the correct output of f given either the complete sum of f or any DF of both f and $\bar{f}$. Also, given any DF of f, there exist algorithms [3] to determine both, $\bar{f}$ and the complete sum of f. Therefore the choice of representation is based primarily on two issues. First, the computation time required to compute $\bar{f}$ or the complete sum of f, and second, the size of the complete sum compared to the size of f and $\bar{f}$ taken together. Some of these issues will be discussed in more detail in the following section.

TABLE II
Tabular form of f and $\bar{f}$

	a	b	c	d	cube
f	1	-	1	0	$ac\bar{d}$
	1	-	0	0	$a\bar{c}\bar{d}$
	-	1	0	1	$b\bar{c}d$
$\bar{f}$	0	0	-	-	$\bar{a}\bar{b}$
	-	-	1	1	cd
	0	-	-	0	$\bar{a}\bar{d}$
	-	0	0	1	$\bar{b}\bar{c}d$

1.4. Complexity Issues

All the algorithms presented in the previous section involve searching tables representing different forms of the given function. Therefore it is crucial that the table sizes remain as small as possible. As mentioned earlier, it is possible to perform simulation using either the complete sum of f, or DF's of both f and $\bar{f}$. In general, it is not known which is the smaller of the two. There exists functions which are well behaved whose complements are of unreasonable size. Also, there exists functions having an extraordinarily large number of prime implicants. The following is an example of two such functions.

EXAMPLE 4: Consider the function

$$f = x_1x_2x_3 + x_4x_5x_6 + x_7x_8x_9 + x_{10}x_{11}x_{12}$$

The minimal DF of $\bar{f}$ has 81 terms whereas the complete sum of f has only 4 prime implicants. Now consider the function

$$g = x_1x_2x_4x_6x_8x_{10} + \bar{x}_2x_3x_4x_6x_8x_{10} + \bar{x}_4x_5x_6x_8x_{10} + \bar{x}_6x_7x_8x_{10} + \bar{x}_8x_9x_{10} + \bar{x}_{10}x_{11}$$

in case of g, the minimal DF of $\bar{g}$ has only 6 terms whereas the complete sum of g has 63 prime implicants. □

The above two functions are special cases of two known worst case functions. In the general form,

$$f = x_1x_2 \cdots x_m + x_{m+1}x_{m+2} \cdots x_{2m} + \ldots + x_{mn-m} \cdots x_{mn},$$

is a function of $m \cdot n$ variables and has n product terms. The minimal DF of f has m^n terms [3], whereas the complete sum of f has only n prime implicants. In case of g, the general form is defined inductively as follows [4].

$$g_1(x_1) = x_1$$

$$g_{m+1}(x_1, \ldots, x_{2m+1}) = x_{2m}\, g_m(x_1, \ldots, x_{2m-1}) + \bar{x}_{2m} x_{2m+1}.$$

Thus g_n is a function of $2n-1$ variables and has n product terms. It can be shown that the minimal DF of $\bar{g}_n$ also has n product terms whereas the complete sum of g_n has $2^n - 1$ prime implicants.

Another interesting class of functions is that of *parity functions*. In this case the complete sum consists of the entire truth-table of the function. Thus for large parity trees the table sizes will be considerable. However, due to the nature of the parity function an unknown at the input must always propagate to the output. Therefore, propagating values at the gate-level will result in the correct output.

Fig. 4 is a plot of the number of terms in f and $\bar{f}$ taken together, denoted as $|f| + |\bar{f}|$, versus the number of prime implicants for some ten input functions. These functions have been implemented in some of the Berkeley PLA's , [3] and can be considered to be *practical* functions. All points below the 45 degree line in the plot of Fig. 4 represent functions for which the number of prime implicants exceed $|f| + |\bar{f}|$. In all cases minimal DF's of f and $\bar{f}$ were used. Points above this line represent functions for which $|f| + |\bar{f}|$ is greater than the number of prime implicants. As is evident from the graph, for most of the functions considered, $|f| + |\bar{f}|$ is greater than the total number of prime implicants of f.

If however, a larger sample of functions is considered, the trend seems to be exactly the opposite. Fig. 5 is a plot similar to Fig. 4 for 55 random 10 variable functions with number of minterms ranging from (0.1×2^{10}) to (0.9×2^{10}). It is interesting to note that for the sample of functions considered, there seems to be a cut-off point after which $|f| + |\bar{f}|$ is always less than the number of prime implicants of f.

For the eight outputs of the SN74181 ALU, $|f| + |\bar{f}|$ ranges from 24 to 376. Data on the number of terms in the complete sum is not available.

We now look at the complexities of the algorithms presented in the previous section. Let f be an n-input function with m product terms in its given DF. Let p be the number of product terms in the DF of f. It is easy to see that if the *cover* and *intersection* operations can be performed in constant time then forward implication is linear in the size of the table. Thus the complexity of exact simulation using Algorithm 2 is $O(m+p)$.

1.5. Conclusions

This paper addressed the problem of accurate logic simulation in the presence of unknowns. Algorithms to perform exact simulation using high-level descriptions are presented. The complexity of these algorithms is shown to be considerably less than that of existing algorithms. An analysis of realistic PLA's and some random functions is used to compare the complexity of the two algorithms presented. Data on the SN74181 ALU suggests that function blocks of similar size can be easily dealt with. The feasibility of the approach is demonstrated by a programmed implementation of the algorithms, as part of a high-level test generation system.

REFERENCES

[1] H. P. Chang and J. A. Abraham, "The complexity of accurate logic simulation," *Proc. International Conference on Computer-Aided Design*, 1987.

[2] M. A. Breuer, "A Note on Three-Valued Logic Simulation," *IEEE Transactions on Computers*, pp. 399-402, April 1972.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1985.

[4] C. McMullen and J. Shearer, "Prime implicants, minimum covers and the complexity of logic simplification," *IEEE Trans. Comput.*, vol. C-35, pp. 761-762, August 1986.

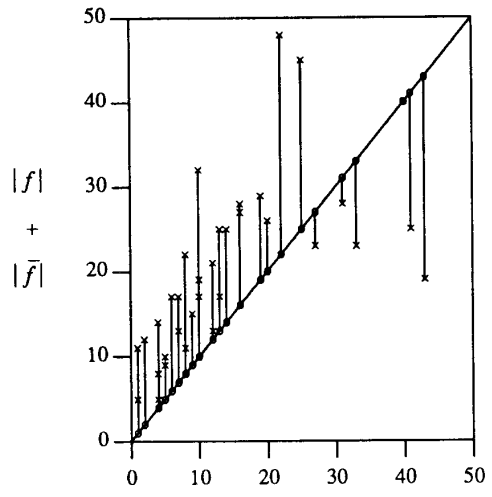

Fig. 4. $|f| + |\bar{f}|$ versus complete sum for 10 variable functions.

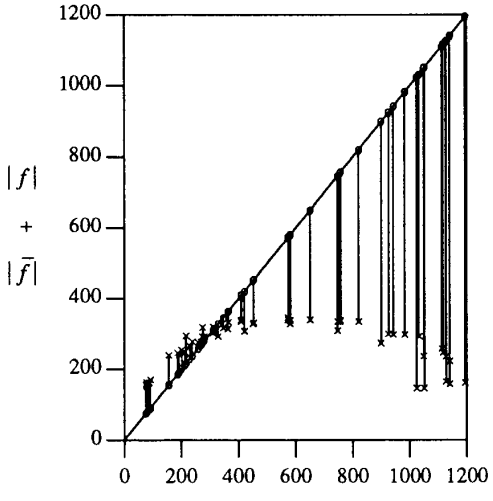

Fig. 5. $|f| + |\bar{f}|$ versus complete sum for random 10 variable functions.