Accurate Logic Simulation by Overcoming the Unknown Value Propagation Problem

Sungho Kang
Yonsei University
Seoul, Korea
shkang@yonsei.ac.kr

Stephen A. Szygenda
Southern Methodist University
Dallas, Texas

As circuits become larger and more complicated, logic simulation plays an important role in design verification and is widely used. However, unknown values in three-value simulation may cause the unknown value propagation problem that produces indeterminate output values. In this article, a new simulation algorithm is developed that can efficiently overcome the unknown value propagation problem. The algorithm is based on the partitioning approach. The experimental results using benchmark circuits prove the effectiveness of the new simulation algorithm.

Keywords: Logic simulation, unknown value propagation

1. Introduction

Two integral and important aspects of the total process of designing and manufacturing printed circuit boards or VLSI chips are verifying and validating a set of diagnostic tests for the products. It is clear that simulation [1, 2], at various levels, is essential and commonly used for verifying the design of digital systems. As circuits become larger and more complicated, the speed and accuracy of simulation are emphasized more. Efficient simulation should be able to produce the accurate outputs of the given circuit. Two-value simulation using logic values 0 (low) and 1 (high) can neither deal with the unknown values nor initialize circuits. Therefore, three-value logic simulation that includes X (unknown value), in addition to 0 and 1, is widely used. Also, five-value logic simulation that includes R (rising) and F (failing), in addition to three-value logic, is used for more accurate results. The unknown value (X) is used to represent the signals that cannot be changed to values other than 0 or 1, such as initialization of the logic net, spikes, hazards, and so forth. However, the unknown values may cause the unknown value propagation (X- propagation) problem [3, 4], which results in indeterminate outputs. Therefore, it is necessary to have a way of handling the unknown value propagation problem efficiently.

An example of the X-propagation problem is shown in Figure 1. In this figure, the output should be 1 since one of the two inputs of the OR gate is always 1. However, three-value simulation produces an output with value x instead of the correct output value of 1. The timing diagram for the X-propagation problem using the nominal delay model and the rise-fall delay model are shown in Figure 2 and Figure 3, respectively.

To solve the X-propagation problem, various methods [5-10] have been devised. Four-value simulation using 0, 1, X, and X **[IS THERE A WAY TO DISTINGUISH THIS FROM THE PREVIOUS X?]** (complement of X) is another method. This method can determine unknown values, as shown in Figure 4. However, this method is applicable only to special cases in which an unknown value and its complement coverage into one of the next gates along the path to the primary outputs. As shown in Figure 5, due to undiscriminated unknown values, this method can produce incorrect outputs. We can avoid the pessimistic cases [8] by discriminating unknown values and by determining outputs with the following Boolean algebra.

$$X_n + Xn = 1,$$

$$X_n \times Xn = 0.$$

If an unknown value passes through a gate that is not a *not* gate, then it is recognized as another unknown value different from the previous one, as shown in Figure 6. For large circuits, the number of unknown values becomes too large to be manipulated, so only a small number of unknown values can be determined. Since determining the outputs of a gate requires many different unknown values that increase during the simulation, this simulation method consumes a lot of time.

SIMULATION, Vol. 79, Issue 2, February 2003 xxx-xxx

DOI: 10.1177/0037549703254811

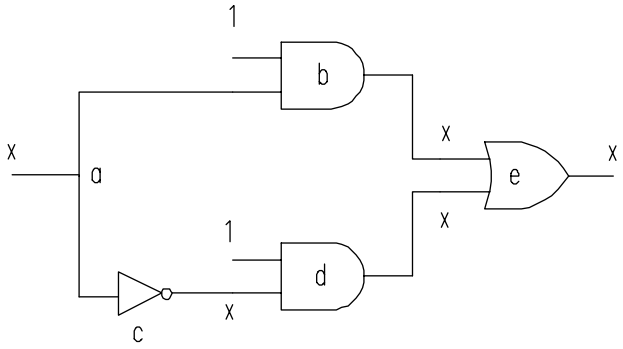

Figure 1. Example of the *X*-propagation problem

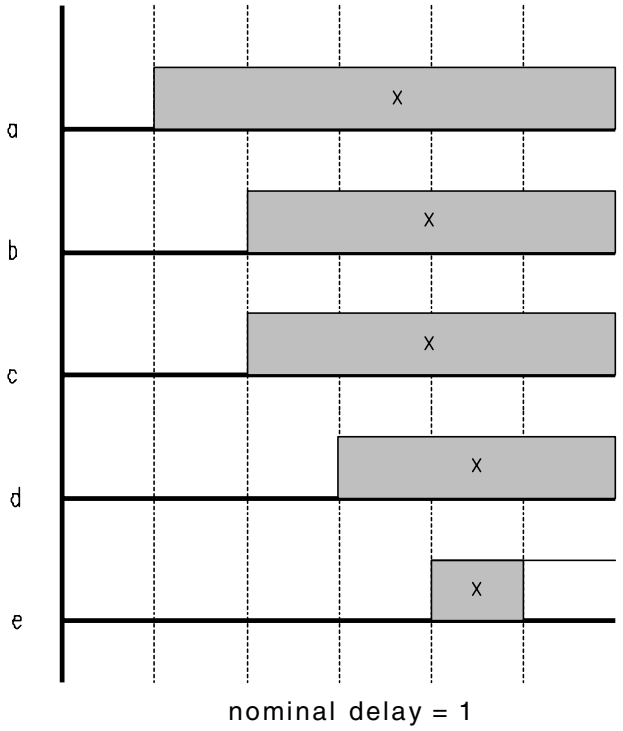

Figure 2. Example of the nominal delay model

In Chandra and Patel [9], a highly accurate simulation method is proposed using a Karnaugh map. Unknown values can be found and determined by finding all redundant prime implicants in the Karnaugh map of a circuit and by adding those terms to the circuits. Since finding all prime implicants is time-consuming and almost impossible for large circuits, the algorithm is identified as not practical [10]. Therefore, applying this method to large circuits could be inappropriate.

An efficient method should be reasonable in terms of time and accuracy, but there is a trade-off between the

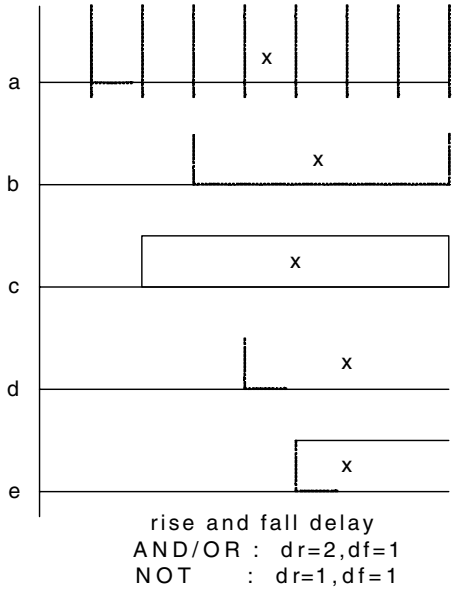

Figure 3. Example of rise-fall delay

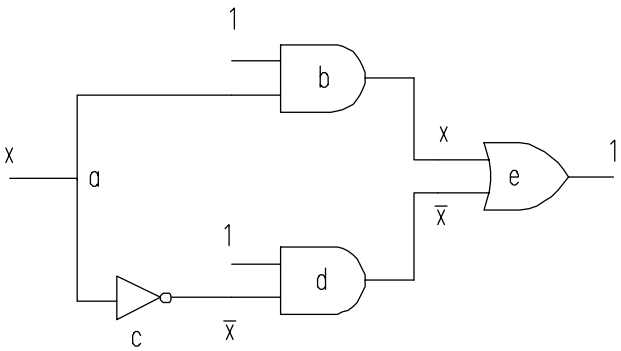

Figure 4. Four-value logic for the *X*-propagation problem

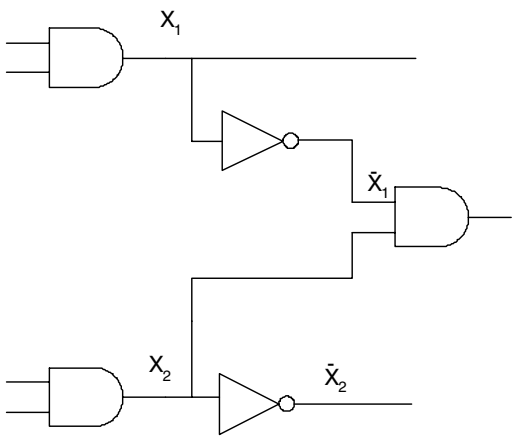

Figure 5. Counterexample of using four-value logic

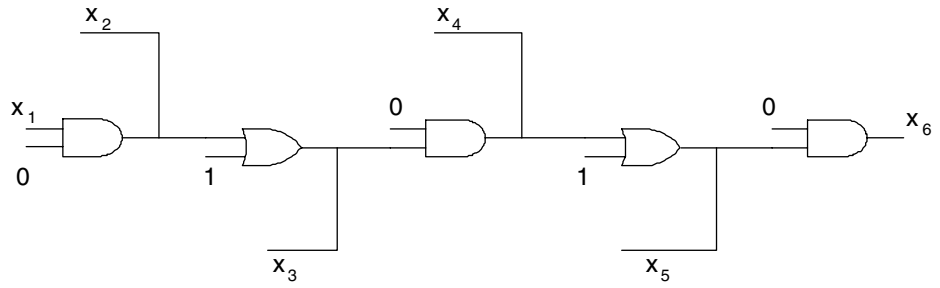

Figure 6. Cascading unknown values

two. Therefore, an optimized method should be used in terms of time and accuracy. The methods that solve the *X*-propagation problem can be roughly classified into two kinds: one is based on formal analysis, and the other is based on the logic simulator. The method that is based on formal analysis is very accurate but too time-consuming to be practical. On the other hand, logic simulation–based methods are efficient in terms of time, but the drawback of these methods is that they cannot always provide accurate results. Consequently, neither the former nor the latter methods can provide a complete solution for the *X*-propagation problem. In recent years, since timing is a critical issue in design verification, the increasing complexity of integrated circuits has made it possible to apply methods based on formal analysis. As a result, the methods based on logic simulation seem to be appropriate for this problem.

Therefore, to reduce the unknown output value in an efficient way, we present a practical method that is based on logic simulation. In this article, a partitioning method has been suggested to handle the *X*-propagation problem, and a newly developed simulation algorithm based on the partitioning method has been introduced.

2. Partitioning Approach

As shown in the previous section, there are many simulation methods in which unknown values can be determined in various ways. Partitioning has been used in simulation and test generation to reduce a complex problem into several small problems [11-13]. In this application, to simulate large circuits involving a lot of unknown values, we develop a new partitioning method in which the circuit is partitioned into several subcircuits and the logic values of primary outputs are determined on the basis of the simulation results of the subcircuits. When the circuit is partitioned, the circuit is modeled as a Huffman model, and only the combinational part of the circuit is considered. In order words, a partition is a combinational logic.

A large partition of an example circuit is represented in Figure 7. A partition begins at a node where the number of fan-outs of a gate is more than 1 and ends at a node into which the fan-outs reconverge. There may be many partitions in a circuit, and only the partitions where both the start gate and the end gate have unknown values can be modified. In Figure 7, three small partitions and one large partition that includes the three small partitions are illustrated. The three small partitions begin from gate 1, gate 2, and gate 3, respectively, and end at gate 13. There are two different simulation methods based on partitioning. One is to perform simulation for each small partition separately, changing the input value of each one. Then, the whole circuit is simulated using the simulation results of small partitions. The other is to determine the unknown logic value of a large partition by applying the exhaustive patters **[PATTERNS?]** to the inputs of the large partition. The simulation time of the large partitioning method increases as $O(2^n)$, where n is the number of small partitions that merge into the large partition. Accordingly, this method consumes more time than the small partitioning method. But the number of newly determined values is almost the same as that of the small partitioning method. Therefore, in this article, the small partitioning method is adopted to determine unknown values.

3. Simulation Algorithm

The simulation algorithm based on the small partitioning method is shown in Figure 8. The simulator parses the netlist, levelizes it, and then finds partitions in the circuit by find_partition(). Before performing the partitioning simulation, the circuit is simulated with routine simulate(), which is normal-mode simulation. Normal-mode simulation means three-value gate-level simulation with logic tables or equations. In the routine simulate(), the logic values of all the gates are determined, and the gates where unknown values can be generated are examined. The flowchart shown in Figure 9 explains the idea of the new simulation.

In the routine simulate_partition(), the unknown output value of each partition is determined by the routine p_simulation(). As shown in Figure 10, a small partition is simulated and the output value of the partition is determined in the routine p_simulate() using the small partitioning simulation method. To avoid useless simulation,

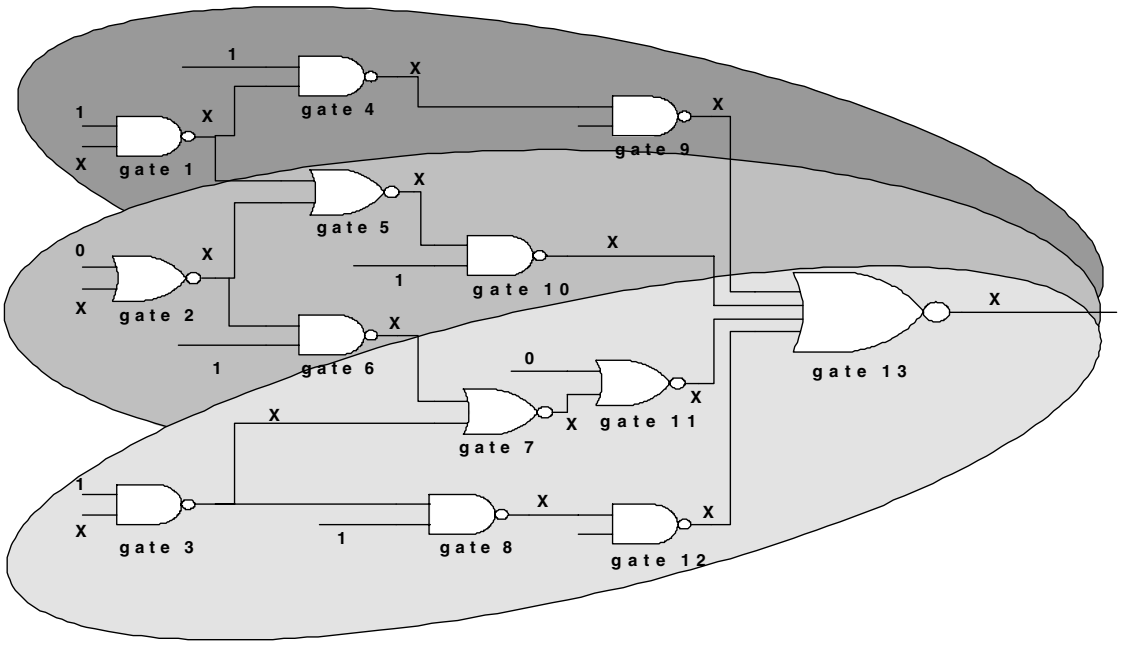

Figure 7. A large partition including three small partitions

```
main()                                  //partitioning mode simuation
{
        parse();                        //parse the input circuit
        levelize();                     //levelize the circuit
        find_partition();               // find partitions in the circuit

        while(get_input_value())
        {
                simulate();             //normal mode simulation
                simulate_partition();   //small partition simulation
                simulate();             // simulation with modified values
        }
}
```

Figure 8. Algorithm of partitioning-mode simulation

we search for the partitions, where both the input and the output values are unknown values, by p_simulated() before the output value of each partition is determined. Then we examine whether the output value of the partition is changed while the input value is changed from logic value 0 to logic value 1 by the routine p_simulate(). If the output value is not changed, then the unchanged value is fixed as the newly determined value. For example, in Figure 7, since logic value 0 and logic value 1 are applied to the inputs of the three small partitions, alternatively, it needs to be simulated $6(2 \times 3)$ times to determine the unknown value of Figure 7 in p_simulate(). By means of this procedure, we can determine some unknown values of the outputs of the small partitions and then, with those values fixed, carry out one more normal-mode simulation to determine the logic values of primary outputs of the whole circuit.

In partitioning-mode simulation, the degree of the limitation of the partitioning depth is an important factor for the simulation performance. The number of newly determined values from unknown values is in proportion to the degree of the limitation of the partitioning depth. But the large limitation of the partitioning depth makes simulation more complicated. Also, it requires a lot of simulation time. The simulation method that requires a large amount of time

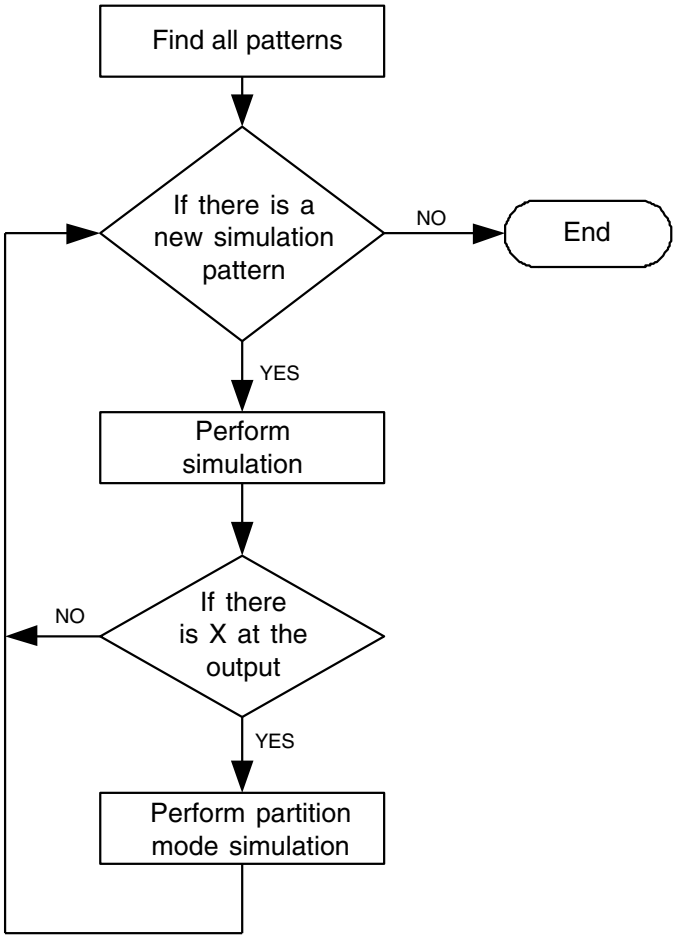

Figure 9. Flowchart of partitioning-mode simulation

cannot be practical because the time to market the chips is a crucial point. Therefore, the degree of the limitation of the partitioning depth should be optimized so that the algorithm can be a practical suggestion, with the performance of the system taken into consideration. The goal of this article is to suggest a practical algorithm that can be used in place of other time-consuming methods. Therefore, we limit the partitioning depth to a reasonable level according to diverse circuits.

The partitioning example is shown in Figure 11, which starts at gate 1 and ends at gate 9, illustrating a simple example of finding a partition whose circuit level is 3. The find_partition() algorithm is represented in Figure 10. At first, all the possible partitions in the circuit are searched for by the find_partition() routine, which calls the test_partition() routine. The reconvergent fan-outs are searched for, and the detected partitions are included in partition_list.

To reduce the huge size of the search space, the user can limit the search depth of the tree. The routine test_partition() is a recursive function that checks whether the multiple fan-outs of a gate reconverge into a fan-in of another gate, using check_index. The variable check_index is initialized as the number of fan-outs of the start gate from which the partition begins. In search_partition(), check_index of the start gate is copied onto the following gates, which are connected with it through the first fan-out. Next, check_index of the start gate is decreased by 1, and the decreased value is copied onto the following gates, which are connected with it through the second fan-out. For copying the check_index, if the order check_index value already exists and s is bigger than the check_index value to be copied, then a partition is found. If the partition is found, the gates included in them are stored in the partition_list in sequence according to the gate level. For the reminding fan-outs, this procedure is repeated. If the previous check_index value and the check_index value to be copied are the same, then a small partition is included in a bigger one, as shown in Figure 11. This case can be discarded since this can be checked later.

```
find_partition() {
  temp=head of simulation list;                          // initialize
while (temp!=NULL) {
      if (gate[temp->index].num_fanout > 2)              // find gate with more than 2 fanouts
          test_partition();                              // lool for partition
   temp=temp->next;                                      // set temp as next list
   }
}

test_partition() {
    if (gets[index].value > check_index)
 add_partition_list(index);                              // store founded partitions
 else
      if (gets[index].value < check_index)
          gate[index].value=check_index;                 // set value as check_index
    if (depth_limit >0) {
depth_limit--;
     test_partition();                                   // find partitions recursively
   }
}

simulate_partition() {                                   // simulation for all partitions
  temp=head of partition linked list;                    // initialize
while (temp!=NULL) {
      if ((gate[temp->start].value==DONOT_CARE)
&&          (gate[temp->end].value==DONOT_CARE)) {
          suspected++;                                   // increase suspected
          p_simulation(te mp);
       }
   temp=temp->next;
   }
}

p_simulation(*te mp) {                                   // simulation for each partition
p=temp->p_sim_list;                                      // pointer for each partition
 gate[temp->start].value=0;                              // initialize start gate of partition
  value_0=operate(p);                                    // simulate only suspected partitions
 gate[temp->start].va;ue =1;                             // change start gate
  value_1=operate(p);
   if (((value_0==0)&&(value_1 ==0)) ||
  ((value_0==1) && (value_1 ==1))) {
detected++;                                              // increase detected
gate[temp->end].flag = value_0;                          // save the value
   }
}
```

Figure 10. Algorithms of find_partition() and simulate_partition()

4. Results

We have implemented the new simulation algorithm as part of a high-performance logic simulator. The simulator was implemented in approximately 13,000 lines of C. The logic simulator is based on an event-driven simulator and uses three logic values. The timing model of the logic simulator is nominal delay. The procedure of the logic simulation is illustrated in Figure 12. The circuit description of the ISCAS 85 bench format is parsed, and then the data structure is set up accordingly. After input vectors for logic simulation are translated, logic evaluation is performed using the library of logic primitives, in which the new simulation algorithm is applied. The user interface is command driven and has the functionality that any other simulators provide. A timing wheel is used as a scheduler that schedules the events at the right time by considering logic primitive delays. The simulation results are generated through post-processing.

The ISCAS 85 combinational benchmark circuits [14] were used to evaluate the performance of the logic simulation based on the partitioning simulation algorithm. All the

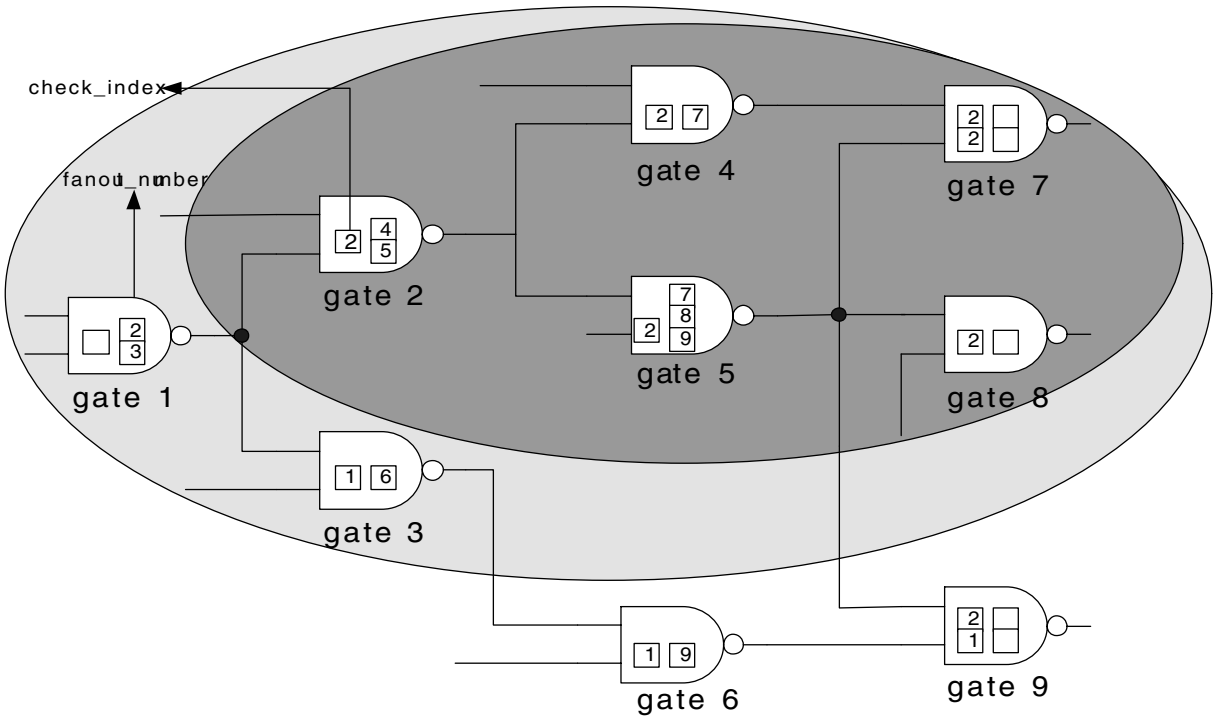

Figure 11. Example of finding partition

results were run on a 233-MHz Pentium chip system. The results demonstrate the high performance of the new algorithm and show the relationship between the partitioning depth and the required memory size.

The logic simulation methods suggested in Jea and Szygenda[5], Szygenda and Thomson [6], and Abramovici, Breuer, and Friedman [8] cannot solve the *X*-propagation problem properly, so comparisons are not possible with our simulation techniques. On the other hand, the logic simulation method presented in Chandra and Patel [9] guarantees the simulation result to be highly accurate. The method is identical to the partitioning simulation method in which the degree of the limitation of the partitioning depth is the maximum depth of the whole circuit. Consequently, the amount of calculations is so large that it is very time-consuming. It is clear that the new method is more efficient in terms of time and is less efficient in terms of accuracy than the method suggested in Chandra and Patel.

The comparison between the large partitioning simulation method and the small partitioning simulation method is given in Table 1. As conjectured, the inefficiency of the large partitioning simulation has been proved. As small partitions merge into large partitions, the simulation time increases as the order of $O(2^n)$), where n is the number of the merge **[MERGING?]** small partitions, and the probability of the unknown output values of the large partition being determined is inversely proportional to it. Therefore, the small partitioning method is used in the new partitioning simulation method.

The comparison between normal-mode simulation and partitioning-mode simulation is represented in Table 2. Normal-mode simulation means three-value gate-level simulation using logic tables or logic equations. One thousand random patterns are applied, and one-third of the total input values are unknown values. The normal-mode simulation time is shown in the sixth column of Table 2, and the partitioning-mode simulation time is shown in the seventh column. In the last column, the number of newly determined values from known values is presented. A large number of unknown values were newly determined with the specified logic values. If conventional approaches had been used, these unknown values would not have been determined. In order words, if the new simulation approach is not used and conventional simulation is performed, the outputs of simulation include lots of *X* values that should be deterministic values. Therefore, it has been proved that the accuracy of logic simulation increases by adopting the partitioning. From the results, we may infer the capability of the partitioning-mode simulation in solving the *X*-propagation problem. The partitioning-mode simulation consumes four to eight times more time than the normal-mode one, except for the C499 circuit. The simulation time linearly increases proportional to the partitioning depth. However, the number of newly determined values from

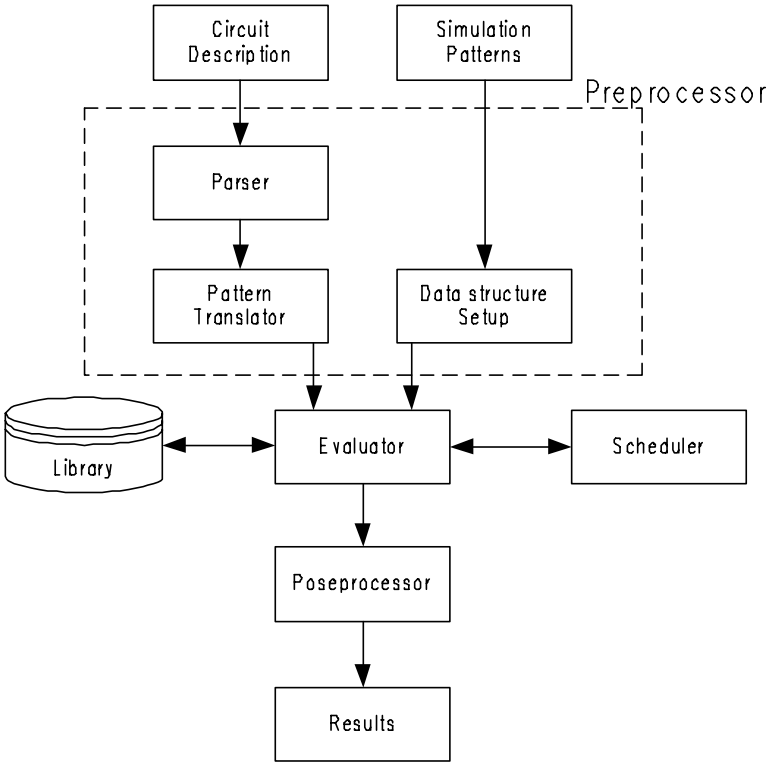

Figure 12. Block diagram of logic simulator

Table 1. An example of the *X*-propagation problem

Partition	C2670				C5315			
	Simulation Time (sec)		Determined Unknown Values		Simulation Time (sec)		Determined Unknown Values	
Depth	Small	Large	Small	Large	Small	Large	Small	Large
3	6.18	6.34	396	400	12.61	13.22	1418	1473
5	6.60	7.31	673	701	14.61	16.37	2019	2150
7	9019	12.04	1013	1027	18.29	22.42	2642	2892
9	11.57	2807	1214	1235	23.78	338.67	2893	2956
11	13.50	21.12	1591	1634	26.19	1780.31	3049	3160

unknown values does not linearly increase as the partitioning depth increases. This is because almost all the unknown values that may be newly determined have been already determined in a certain partitioning depth for each benchmark circuit. Consequently, we may assert that the performance of the partitioning is greatly affected by the partitioning depth.

Therefore, selecting a proper partitioning depth is an important factor for saving simulation time. According to the circuit topology, all partitions and the circuit level for each partition are determined. When the maximum among the circuit levels of all partitions is selected as the partition depth of the circuit for the simulation, no *X*-propagation problem exists. If the partition depth is the same as the maximum level of the circuit, it may take a plenty of simulation time. However, in most cases, many circuit levels of all partitions are far less than the maximum, and the maximum is not large enough. Therefore, the partition depth is usually decided less than the maximum. Also, the partition depth can be incrementally decided while performing simulation.

In Table 3, the relationship between the ratio of unknown values in the input patterns and the simulation time is represented. It has been observed that the partitioning-

Table 2. An example of a nominal delay model

Circuit	Number of Inputs	Number of Outputs	Number of Gates	Depth	Number Mode (sec)	Partitioning Mode (sec)	Determined Unknown Value
C432	36	7	160	10	0.52	3.11	126
C499	41	32	202	10	0.61	9.17	169
C880	60	26	383	10	0.90	3.18	279
C1355	41	32	880	10	1.18	6.46	163
C1908	33	25	546	10	1.48	8.01	634
C2670	233	140	1669	10	2.57	12.76	1406
C3540	50	22	2307	8	2.75	20.03	374
C5315	178	123	2307	10	4.38	25.65	2941
C6288	32	32	2416	10	4.29	33.20	377
C7552	207	108	3513	5	6.06	22.91	1342

Table 3. An example of rise-fall delay

Unknown Value Ratio	C2670 (sec)	C5315 (sec)	C6288 (sec)
1/3	11.97	24.98	29.32
1/7	8.15	17.18	23.55
1/11	6.76	14.17	20.86
1/15	5.77	12.47	17.86
1/19	5.23	11.62	17.34

mode simulation time decreases as the ratio of unknown value decreases, indifferent to the normal simulation time decreases as the ratio of unknown values decreases, indifferent to the normal simulation time, which does not change **[PLS. CLARIFY SENTENCE]**. This is due to the fact that this partitioning simulator simulates only the partition in which both the input and the output values are unknown. Therefore, the simulation time increases as the number of unknown values increases.

This partitioning-mode simulation is efficient in the case of a small number of unknown value inputs. Let T_n be the normal-mode simulation time for one input pattern, and let T_p be the partitioning-mode simulation time for one input pattern. Also, let P be the number of simulation patterns and p the number of the cases in which one or more unknown values are derived at primary outputs of a simulated circuit from applying P input patterns. The new simulation time, T_{new}, and the old simulation time, T_{old}, are given as follows:

$$T_{new} = P \times T_n + p \times T_p,$$

$$T_{old} = P \times T_p.$$

The new simulation time is much shorter than the old one, which is the simulation time when the partitioning-mode simulation is adopted for all the cases, regardless of whether the logic values of the primary outputs are unknown. We can see that T_p is much longer than T_n as a result of the experimental results. Hence, if p is so large, T_{new} may be too long, and the new simulation method based on partitioning would be less effective. Fortunately, in real cases, p is so small that T_{new} is reasonable from a practical point of view. Thus, this method is an efficient means of solving the X-propagation problem, especially when p is a small number.

The efficiency of the simulator is determined according to the accuracy. The accuracy means that the output of the simulation is the same as that of the physical implementation of the circuit. In logic simulation, due to the unknown value problem, there exist X values at the output of the simulator, while the real circuit provides 0 or 1. This makes the simulator inaccurate. To overcome this problem, we provide a new approach in this article. Using the new algorithm, the simulation time is longer than the conventional simulation. However, the new algorithm is more accurate. There is a trade-off between simulation speed and accuracy. The bottleneck is caused by the simulation of the small partition before the simulation of the entire circuit can be done. Despite the bottleneck, when accuracy is more important than speed in the simulation environments, the new approach is practical and valuable.

5. Conclusion

Since efficient simulation should be fast and accurate, the simulation output should be equivalent to the real physical implementation. However, logic simulation cannot provide accurate output in source cases due to the unknown

value propagation problem. To overcome this problem, we have developed a new simulation method using partitioning. The new algorithm is based on the fact that there is a trade-off between simulation time and accuracy. If an optimized partitioning depth is selected according to the computational resources, the time efficiency and accuracy of the simulation can be achieved. In addition to achieving maximum performance, the partitioning-mode simulation is performed only in cases when both the input and output values of a partition are unknown. The experimental results show the efficiency of the new simulation method. To achieve high-performance simulation, the hybrid method that uses BDD **[PLS. SPELL OUT]** instead of the small partition simulation should be investigated.

6. References

[1] Olukotun, K., M. Hemrich, and D. Ofelt. 1998. Digital system simulation: Methodologies and examples. *Proceedings of Design Automation Conference*, pp. 658-62.

[2] Kang, S., and S. Szygenda. 1991. The Simulation Automation System (SAS): Concepts, implementations and results. *IEEE Transaction on VLSI* 2 (1): 89-99.

[3] Miczo, A. 1986. *Digital logic testing and simulation*. New York: Harper & Row.

[4] Horbst, E. 1986. *Logic design and simulation*. Amsterdam: North-Holland.

[5] Jea, Y. H., and S. A. Szygenda. 1979. Mapping and algorithms for gate modeling on a digital simulation environment. *IEEE Transactions on Circuits and Systems* 26 (5): 304-15.

[6] Szygenda, S., and F. W. Thomson. 1976. Modeling and digital simulation for design verification and diagnosis. *IEEE Transactions on Computers* 325 (12): 1242-53.

[7] Breuer, M. A., and A. D. Friedman. 1976. *Diagnoasis and reliable design of digital systems*. **CITY?:** Computer Science Press.

[8] Abramovici, M., M. A. Breuer, and A. D. Friedman. 1990. *Digital system and testable design*. **CITY?**: IEEE Press.

[9] Chandra, S. J., and J. H. Patel. 1987. Accurate logic simulation in the presence of unknowns. *Proceedings of International Conference on Computer-Aided Design*, pp. 34-7.

[10] Chang, H. P., and J. A. Abraham. 1987. The complexity of accurate logic simulation. *Proceedings of International Conference on Computer-Aided Design*, pp. **000-000**.

[11] Schulz, M., Trischler, E., and Sarfert, T. 1987. SOCRATES: A highly efficient automatic test pattern generation system. *Proceedings of International Test Conference*, pp. 1016-26.

[12] Silva, J., and A. Sakallah. 1996. Dynamic search space pruning techniques in path sensitization. *Proceedings of Design Automation Conference*, pp. 705-11.

[13] Pomeranz, I., and S. Reddy. 1996. Fault location based on circuit partitioning. *Proceedings of International Conference on Computer-Aided Design*, pp. 242-7.

[14] Brglez, F., and H. Fujiwara. 1985. A neutral netlist of 10 combinational benchmark circuit and a target translation in FORTRAN. *Proceedings of the International Symposium on Circuit and Systems*, pp. 695-8.

Sungho Kang *is* **POSITION?***at Yonsei University, Seoul, Korea.*

Stephen A. Szygenda *is* **POSITION?***at Southern Methodist University, Dallas, Texas.*