設計自動化　５０－７
(1989. 12. 14)

共有二分決定グラフを用いた時間記号シミュレーション

石浦菜岐佐　　出口豊　　矢島脩三

京都大学工学部

あらまし　本稿では、新しいタイミング検証のツールとして、符号化時間記号シミュレーション（ＣＴＳＳ）を提案する。ここでは、遅延の最大値と最小値が与えられているゲートで構成される論理回路のふるまいを、正確にシミュレーションすることを考える。本手法は、それぞれのゲートにおいて、ゲートが取る可能性のある遅延の値のすべての場合を、２値で符号化して、ブール変数を導入して記述し、問題を記号シミュレーションに帰着して解くものであり、組合せ回路だけでなく、フィードバック・ループを含む論理回路も扱うことができる。われわれは、記号シミュレーションの内部表現に共有二分決定グラフを用いることにより、効率のよい処理系を開発した。本稿では、さらに本シミュレーション手法で得られるシミュレーション結果の解析を行なう手法についても述べる。

Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram

Nagisa ISHIURA, Yutaka DEGUCHI and Shuzo YAJIMA

Department of Information Science, Faculty of Engineering, Kyoto University
Kyoto 606, JAPAN

abstract　In this paper we propose a new timing verification technique named *coded time-symbolic simulation, CTSS*. We are concerned with simulation of logic circuits consisting of gates whose delay is specified only by its minimum and maximum values. We encode cases of possible values of delay of each gate by binary values with Boolean variables, and reduce the problem into the conventional symbolic simulation. This simulation technique can handle logic circuits containing feedback loops as well as combinational circuits. We implemented an efficient simulator by using *shared binary decision diagram (SBDD)* as an internal representation. We also propose novel techniques for result analysis of CTSS.

1. Introduction

Logic design verification is one of the most important processes for developing reliable digital systems, but it is, at the same time, one of the most laborious processes. Especially verification concerned with timing is often complicated and time consuming. In order to avoid difficulties, large part of a digital system is usually designed as a synchronous sequential circuit whose timing verification is relatively easy. However, there remain some portions, such as communication units, which are designed as asynchronous circuits. Even if the asynchronous part is much smaller than the synchronous part, as much (or more) efforts are required to achieve timing verification of the asynchronous part.

When behavior of a circuit depends on subtle timing relations, we have to consider the change of delay value which may be caused by the difference of of process conditions or the difference of usage environments. In logic simulation, which is currently one of the most effective method of dynamic timing analysis, we treat the uncertainty of delay value using the min/max delay model [1]. Although it enables us fast execution, it has been pointed out that simulation results are often too pessimistic [1] and that it is very difficult to know if a circuit under test has design errors.

As one of the solutions to this problem, time-symbolic simulation has been proposed in [3]. In this technique, the delay value of a gate is represented by a variable and simulation is executed with time represented by algebraic expressions. This technique gives us accurate simulation results without pessimism. It also enables us to derive the delay conditions under which the circuit behave expectedly. The algorithm proposed in [3], however, is based on time first evaluation algorithm [4] and can handle only combinational circuits without feedback loops. This restriction limits the applications of the technique.

In this paper we propose an alternative to the time-symbolic simulation which can handle circuits containing feedback loops. Instead of using the time variables, we *encode* possiblities of delay value of an uncertain delay unit using *Boolean variables.* Then the time-symbolic simulation is reduced into usual symbolic simulation [8], and we can simulate all kinds of circuits by the conventional simulation algorithm. We call our new technique *coded time-symbolic simulation (CTSS).* As a representation form of Boolean functions appearing in symbolic simulation, we use *shared binary decision diagram (SBDD)* [5] which is an improvement on the BDD [6]. The use of SBDD in CTSS drastically reduces storage requirement, and enables efficient execution.

It is also important to provide aids for analyzing simulation results, for the simulation results of CTSS are given in the form of Boolean expressions of coded time variables. In this paper we also propose a novel technique to specify an expected behavior, to compare it with result obtained by CTSS and to display the delay conditions for the expected behavior of the circuit under test.

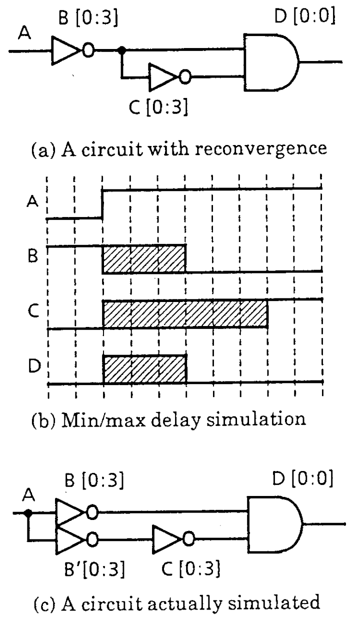
In the following section we discuss the modeling of the delay uncertainty. In section 3 we describe the details of CTSS using SBDD. In section 4 we show the techniques for result analysis. We discuss some issues on implementation of CTSS and show experimental results in section 5.

2. Modeling of Delay Uncertainty

2.1 Conventional Min/Max Delay Model

Actual gate delay is affected by process conditions or usage conditions. In order to guarantee the correct behavior of asynchronous logic circuits which are based on sophisticated timing relation, we have to examine the behavior taking the delay uncertainty into account.

In logic simulation, we treat the uncertainty by using the min/max (ambiguity) delay model [1]. It has been pointed out, however, that this model has serious shortcomings that the simulation results are often too pessimistic due to reconvergent fanouts [1]. For example, a timing chart in Figure 1(b) is the result of the min/max delay simulation for the circuit in Figure 1(a). The unknown states on line D indicate the possibility of a static hazard, which never occurs in an actual circuit because the rising edge on C never precedes the falling edge on B. This overpessimism comes from loss of the information that the uncertainty in the time of rising



(a) A circuit with reconvergence

(b) Min/max delay simulation

(c) A circuit actually simulated

Figure 1: Overpessimism in min/max delay simulation.

edge on C is partly due to that of falling edge on B, and we are actually simulating the circuit shown in Figure 1(c), instead of that in Figure 1(a).

There are simulators which detects reconvergences and can avoid the overpessimism to some extent. But it is considered to be impossible to solve this problem completely by a simple extension of the min/max delay simulation, because the problem of judging the possibility of hazards for a given combinational circuit and a pair of input vectors under the assumption of the uncertain delay model, has been shown to be *NP-hard* [2].

2.2 Time-Symbolic Simulation

Symbolic simulation is one of the approaches to an effective solution of this problem. In [3] they represents the delay time by a variable in the real number domain, and simulate logic circuits with time represented by algebraic expressions. This symbolic execution enables us to derive the delay conditions to make a circuit behave correctly by analyzing the expressions obtained as the simulation results, as well as it offers us accurate simulation results without pessimism. The algorithm shown in [3] is efficient but it supports only combinational circuits. This is because the algorithm is based on the time-first evaluation mechnism [4].

It is possible to adapt the time-symbolic simulation to the conventional space first evaluation algorithm, if we do not mind the efficiency. An algorithm is as follows. Events scheduled to occur in the future are maintained in a set Q.

1) Repeat 2)~4) until Q becomes empty.

2) Get an event e, out of Q, whose occurrence time is judged to be the smallest. If there are multiple candidates, investigate all the possibilities.

3) Compute the effect of e.

4) If there are new events as a result of 3) put them into Q.

In step 3), we do not have to investigate all the possibilities if the order of occurrences of some events does not affect the entire behavior of the circuit. However, we are forced to investigate almost all the possibilities as long as we take a pessimistic strategy, since it is almost impossible to know that in advance. As a result, simulation speed become so slow that we can not simulate even a small circuit in practical time. One of the keys to a breakthrough of this problem is an optimistic strategy such as in [9], which in turn makes the simulation control complicated.

Our approach, shown in the follwoing section, is completely different. We do not have to care about the order of the event occurences, and yet the useless branching are avoided.

3. Coded Time-Symbolic Simulation - CTSS

3.1 Coding of Uncertain Delay by Boolean Variables

We assume time to be descrete as is the usual with conventional logic simulation. Then we can enumerate the possibilities of actual delay value of a bounded uncertain delay. Let us take the circuit in Figure 1(a) as an example. Each of the inverters B and C, whose delay is specified as [0,3], will take one of the four delay values $\{0,1,2,3\}$. If we investigate the 16 cases, namely the 4 cases for B by the 4 cases for C, we can get the completely accurate simulation result. Total number of the cases to be examined will be exponential to the number of uncertain delay components in a circuit. This is inevitable because of the complexity of the problem as we told earlier. We focus our attention on how we can make the simulation process efficient.

Again in the exampel above, 16 possible signal values are associated with a signal line at a time period. If we code the delay of B and C using Boolean variables, such as $delay_B = (b_1, b_0)$ and $delay_C = (c_1, c_0)$, the 16 signal values can be seen as a *Boolean function* of the 4 input variables. Then the simulation with the uncertain delay is reduced into the usual symbolic simulation. That is the basic idea of our *coded time-symbolic simulation.*

For the convenience of explanation, we assume without loss of generality that a gate in a circuit is either a pure functional gate with delay 0 or a pure delay gate with a signle input and a signle output. Let us denote the signal value on a line s at time t as $\sigma_s[t]$. Then the signal value on the output line y of a functional gate g is computed by the following equation:

$$\sigma_y[t] = f_g(\sigma_{x_1}[t], \sigma_{x_2}[t], \cdots, \sigma_{x_k}[t]), \qquad (1)$$

where f_g is the Boolean function of g, and $x_1, x_2, \cdots, x_k$ are the signal lines which feed g.

As for a delay gate, we can interpret the coding of delay as shown in Figure 2. Namely the time variables b_1 and b_0 are the selection inputs to choose one of the four delay possibilities. Then the definition of a delay gate g is also straightforward. Let y and x be the output line and the input line of g, min_g and max_g be the minimum and maximum delay value of g, and $d_{g,0}, \cdots, d_{g,l} (l = \lceil \log_2(max_g - min_g + 1) \rceil)$ be the time variables coding the delay of g. Let us also define $d_{g,i}(k)$ and $D_g(k)$ as follows.

$$d_{g,i}(k) = \begin{cases} \overline{d_{g,i}} & \text{if } i\text{-th bit of the binary} \\ & \text{representation of } k \text{ is 0,} \\ d_{g,i} & \text{otherwise.} \end{cases}$$

$$D_g(k) = d_{g,0}(k) \cdot d_{g,1}(k) \cdot \cdots \cdot d_{g,l}(k).$$

The output value of a delay gate g is computed according to the equation :

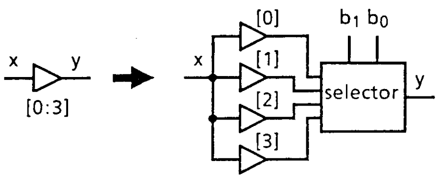


Figure 2: An interoretation of the coding of delay.

t	$\sigma_A[t]$	$\sigma_B[t]$	$\sigma_C[t]$	$\sigma_D[t]$
init	0	1	0	0
0	1	$b_1 + b_0$	$\overline{b_1 + b_0 + c_1 + c_0}$	0
1	1	b_1	$\overline{b_1 + c_1} \cdot \overline{b_0 \cdot c_0}$	0
2	1	$b_1 \cdot b_0$	$\overline{b_1 + c_1} + \overline{b_0 + c_0} \cdot \overline{b_1 \cdot c_1}$	0
3	1	0	$\overline{b_1 + c_1} + \overline{b_1 + b_0} + \overline{c_1 + c_0}$ $+\overline{b_0 + c_0} \cdot \overline{b_1 \cdot c_1}$	0
4	1	0	$\overline{b_1 + c_1} + \overline{b_1 \cdot c_1}$	0
5	1	0	$\overline{b_1 \cdot b_0 \cdot c_1 \cdot c_0}$	0
6	1	0	1	0

Figure 3: An example of the simulation.

$$\begin{aligned} \sigma_y[t] &= D_g(0) \cdot \sigma_x[t - min_g] \\ &+ D_g(1) \cdot \sigma_x[t - min_g - 1] \\ &\cdots \\ &+ D_g(max_g - min_g) \cdot \sigma_x[t - max_g]. \qquad (2) \end{aligned}$$

Figure 3 shows an example of the simulation of the circuit in Figure 1(a). We gave the rising edge at time 0. The Boolean function $\overline{b_1 + b_0 + c_1 + c_0}$ appearing on line C at time 0, for example, indicates that the value is 1 when $b_1 = b_0 = c_1 = c_0 = 0$, namely $delay_B = delay_C = 0$, and otherwise the value is 0. The signal value on D is always 0 which is the accurate result we expected. In CTSS, the variables of a delay does not appear in the formula to represent the signal value on a line s at time t, as long as the delay does not affect s at time t. So we can automatically avoid the useless comparison and branching.

3.2 Representation of Boolena Functions by Shared Binary Decision Diagram

Good representation of Boolean function is a key to efficient symbolic simulation. In our implementation we use *shared binary decision diagram (SBDD)* [5], which is an improvement on the binary decision diagram [6]. In the SBDD all the possible subgraphs are shared among multiple functions, as shown in Figure 4. The SBDD has the follwoing merits besides those of the BDD.

1) Many functions can be efficiently expressed *simultaneously.*

2) Many of the operations can be done much faster than in the BDD. Especially, equivalence of the two functions are checked by simply comparing the pointers while the isomophism should be examined in the BDD.

Those two merits are very much suitable for our purpose. Since in CTSS we have to represent many Boolean functions to express signal values on signal lines, the property 1) is very favorable. Figure 5 shows the representation of the signal values in the simulation of delay units connected in cascade. We can see how well the subgraphs are shared. The property 2) is also favorable in detecting changes of signal values in symbolic simulation.

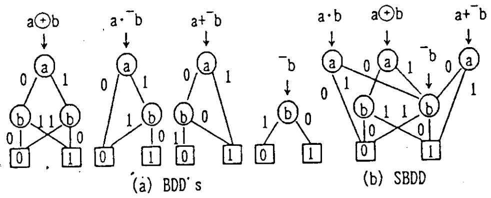
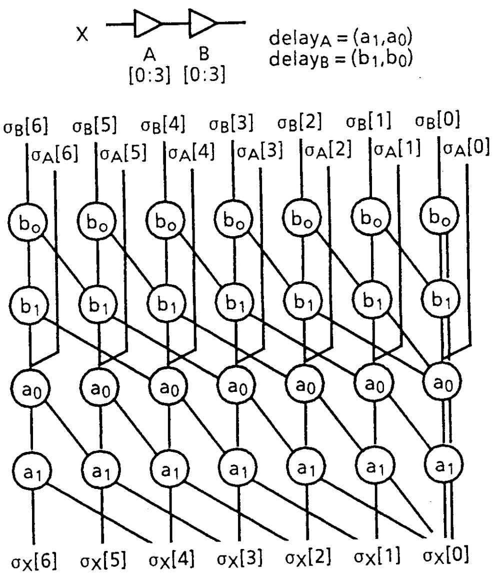


Figure 4: Shered binary decision diagram.



X
A [0:3]
B [0:3]
delayA = (a1,a0)
delayB = (b1,b0)
σB[6] σB[5] σB[4] σB[3] σB[2] σB[1] σB[0]
σA[6] σA[5] σA[4] σA[3] σA[2] σA[1] σA[0]
σX[6] σX[5] σX[4] σX[3] σX[2] σX[1] σX[0]

Figure 5: Representation of signal values by SBDD.

4. Result Analysis of CTSS

Although CTSS offers accurate simulation results, they are represented by Boolean functions and it is often hard to understand the meaning of the Boolean functions and to tell if there are errors. For example, in Figure 3 the signal values on C satisfies the following relation.

$$0 = \sigma_C[0] \subset \sigma_C[1] \subset \sigma_C[2] \subset \cdots \subset \sigma_C[6] = 1.$$

From this relation, we can conclude that there is always a single rising edge on C regardless of the combination of delay values. However, it seems to be hard to derive the fact only by looking at the expressions. It is important to prepare a mechanism to analyze simulation results and tell if simulation results match expected behavior of the circuit under test. In this section, we discuss methods of the result analysis. We propose a novel technique to match the simulation results with expected ones, and a technique to dispay the delay conditions for the expected behavior.

4.1 Matching Simulation Results Using Finite Automata

In [3] they compare simulation results with expected results by expressing the expected ones using the same data structure as the simulation results. Although we can apply this strategy in CTSS, it is also difficult to derive the Boolean expressions to represent the specifications in general. In this paper we propose an novel technique using a finite automaton. This technique is a generalization of the edge detection technique shown above.

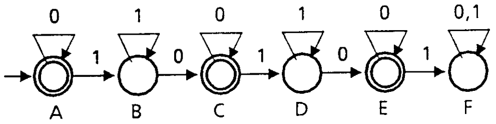


Figure 6: Automata A_η to represent a specification η.

At first we represent expected behavior of a circuit by a regular expression η. We construct a deterministic finite automaton A_η which accepts the same set of sequences as η. We design a sequential circuit M_η which inputs a sequence and outputs 1 if and only if A_η accepts the sequence. We simulate M_η along with the circuit under test. If the final output of the M_η is 1 then we can conclude the circuit satisfies the specification η regardless of the delay values. If the final output is a Boolean expression containing delay variables, the expression indicates the possible combination of actual delay values for the circuit to behave correctly.

For example, if we want to verify that not more than two 1-pulses are allowed on the output line x of the circuit C, the specification is writtten as

$$\eta = 0^* + 0^*1^*0^* + 0^*1^*0^*1^*0^*.$$

From this regular expression we can construct a deterministic automata A_η as shown in Figure 6. By the state assignment $A = (1,0,0), B = (0,0,0), C = (1,0,1), D = (0,0,1), E = (1,1,1), F = (0,1,1)$, we get a sequential machine M_η expressed by the following equation, where y_1, y_2 and y_3 are state variables and ok is the output of M_η:

$$\begin{aligned} y_1' &= \overline{x} \cdot \overline{y_2} + \overline{x} \cdot y_1 \cdot y_3, \\ y_2' &= y_2 \cdot y_3 + \overline{x} \cdot \overline{y_1} \cdot y_3, \\ y_3' &= y_3 + \overline{x} \cdot \overline{y_1}, \\ ok &= y_1. \end{aligned}$$

By simulating M_η along with the circuit C, we can know if C satisfies the specification η.

4.2 Extraction of Algebraic Expressions

As a result of the matching in the previous subsection, we get an Boolean expression indicating delay conditions for correct behavior. We can know a set of combinations of delay values imediately from this expression. However, it is much helpful if we can know the algebraic relation between delay values. Suppose the following expression is obtained, where $delay_A$ and $delay_B$ are coded by (a_0, a_1, a_2) and (b_0, b_1, b_2), respectively.

$$\begin{aligned} ok &= b2 \cdot \overline{a2} + b2 \cdot b1 \cdot \overline{a1} + b2 \cdot b1 \cdot b0 \cdot \overline{a0} \\ &+ b2 \cdot \overline{a1} \cdot b0 \cdot \overline{a0} + \overline{a2} \cdot b1 \cdot \overline{a1} \\ &+ \overline{a2} \cdot b1 \cdot b0 \cdot \overline{a0} + \overline{a2} \cdot \overline{a1} \cdot b0 \cdot \overline{a0}. \end{aligned}$$

It is hard to read by what condition the circuit behave correctly. However if we extract the following algebraic expression of $delay_A$ and $delay_B$, we can understand the meaning very well.

$$delay_A < delay_B.$$

Since target albegraic expressions include only addition and comparison of the delay variables [3], extraction, of the algebraic expressions is considered to be done efficiently by the technique in [7].

5. Implementation of the CTSS

5.1 Compiler Driven Implementation of CTSS

In order to enhance the performance of simulators, we usually adopt event driven simulation mechanism. In CTSS, however, we considered it not neccesarily advantageous to be based on the event driven simulation algorithm for the following reasons, and we decided to implement the first version of the simulator based on the compiler driven simulation algorithm.

1) Since an event on the input line of a delay gate g at time t affects the ouput line of g at time $t + min_g, t + min_g + 1, \cdots, t + max_g$, we have to handle much more events than in the usual logic simulation.

2) In order to accelerate symbolic operation in SBDD, we keep recent results of symbolic operations in a hash table [5], and we can execute the same symbolic operations as we executed recently by just looking up the table. Since the cost of the table look-ups is much smaller than that of the symbolic operations, we can not expect the drastic reduction of computation time by omitting the same operations according to the event driven simulation strategy.

In the compiler driven simulation, we have to pay attention to the order of gate evaluation, because we may have to evaluate a gate for many times until the circuit becomes stable, which brings the considerable drawback in computation time. We classify gates into the following 2 categories.

1) Delay gates whose minimum delay value is not 0.

2) Functional gates (whose dealy value is 0) and delay gates whose minimum delay value is 0.

Since the output value of a gate in the category 1) at time t does not depend on the input value at time t, we can evaluate the gate without waiting for evaluations of the other gates, but we have to be careful for the order of evaluations of gates in the category 2). In our implementation, we evaluate all the gates in the category 1) first and then evaluate the gates in the category 2) in the order of level number. We exclude the circuit which contains loops consisting of gates in the category 2) in the preprocessing stage.

5.2 Experimantal Results

We have implemented a simulator based on the techniques stated so far. The simulator is written in C language and runs on the Sun3/60 workstation.

We simulated an 8-bit ripple carry adder consisting of 48 gates, all of which has a bounded uncertain delay [1,4]. We gave an input stimulus of length 100 which contains 5 events at time 1, 5, 10, 20 and 35. It took about 28 seconds to execute simulation. Since the total number of gate evaluation is 4800, the simulation speed is about 170 gate evaluation/sec. The number of events, which we counted by the separate program, was 756, and simulation performance in this case is about 27 event/sec. This may be much slower than usual min/max delay simulators, but it is considerd to be amazingly fast because it simulated 4^{48} cases within a minutes! The number of the nodes required for simulation was 18180. Since we have not done any attempts on the ordering of variables, these figures (number of nodes and simulation speed) is expected to be improved.

Acknowledgment

The authors would like to thank to Mr. Shin-ichi Minato, who offered us the Boolean function manipulation subroutines using SBDD representation, which is indispensable for our implementation. We would like to express our sincere appreciation to the members of Yajima Laboratory at Kyoto University for their valuable discussions and comments.

References

[1] M. A. Breuer and A. D. Friedman: Diagnosis & Reliable Design of Digital Systems, Computer Science Press, (1976).

[2] N. Ishiura and H. Yasuura: On a Relation between Time-Models and Computation Time of Hazard Detection Problems, IEICE (the Institute of Electronics, Information and Communication Engineers of Japan) Technical Report, COMP88-21, pp. 45-52, in Japanese, (1988)

[3] N. Ishiura, M. Takahashi and S. Yajima: Time-Symbolic Simulation for Accurate Timing Verification of Asynchronous Behavior of Logic Circuits, Proc. 26th DAC, pp. 497-502, (1989).

[4] N. Ishiura, H. Yasuura and S. Yajima: Time-First Evaluation Algorithm for High-Speed Logic Simulation, Proc. ICCAD-84, pp. 197-199, (1984).

[5] S. Minato, N. Ishiura and S. Yajima: Fast Tautology Checking Using Shared Binary Decision Diagram - Benchmark Results -, Proc. IFIP International Workshop on Applied Formal Methods for Correct VLSI Design, (to appear in Nov. 1989).

[6] R. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., vol. C-35, no. 8, pp. 677-691, (1986).

[7] M. Ohmura, H. Yasuura and K. Tamaru: Functional Information Extraction from Combinational Circuits, submitted to the 27th DAC.

[8] W. C. Carter, W. H. Joyner and D. Brand: Symbolic Simulation for Correct Machine Design, Proc. 16th DAC, pp. 280-286, (1979).

[9] T. Yoneda, K. Nakade and Y. Tohma: A Fast Timing Verification Method Based on the Independence of Units Proc. FTCS-19, pp. 134-141, (1989).