JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 15, 885-897 (1999)

A Compiled-Code Parallel Pattern Logic Simulator With Inertial Delay Model

KUO CHAN HUANG, CHUNG LEN LEE AND JWU E CHEN*
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan 300, R.O.C.
**Department of Electrical Engineering*
Chung Hwa University
Hsinchu, Taiwan 300, R.O.C.

This paper presents a parallel pattern compiled code logic simulator which can handle the transport delay as well as the inertial delay of the logic gate. It uses **Potential-Change Frame**, incorporating inertial functions, to execute event-canceling operation of gates, thus eliminating the conventional time wheel mechanism. As a result, it can adopt the parallel pattern strategy to increase the simulation speed. Furthermore, it is a compiled code simulator, which further improves its performance. Experimental results show that it significantly surpasses the conventional time wheel event-driven simulator in terms of simulation speed. In addition, it is also found that a significant percentage (27%) of hazards can be eliminated when the effect of the inertial delay is considered in the simulation.

Keywords: logic simulator, compiled code simulation, parallel pattern, inertial delay model, potential change frame

1. INTRODUCTION

Recently there has been considerable interest in studying use of the compilation technique to improve the performance of logic simulation [1-9]. The reason why the compilation technique had not previously received much attention was that the traditional levelized compiled code(LCC) simulator was based on the zero delay model, which is very unrealistic for treating circuit delays. The above situation has changed recently as some techniques have been proposed to generate codes for more complex timing models. For example, a threaded code technique was proposed to incorporate the unit delay model in a Tortle_c simulator [5], and a multi-delay parallel algorithm was developed to use the potential change set to handle a multi-delay model [6]. All the above techniques are based on the simple transition-independent transport delay model.

However, the transport delay model can not describe the transient behavior of the circuit properly. A simple CMOS NAND gate shown in Fig. 1 can be used to explain this. In Fig. 1(a), the propagation delay of the CMOS NAND gate, denoted by d, is 0.2 ns. For the simple delay model, any transition on inputs A or B at time t will propagate to output C at time t+0.2ns. Hence, as shown in Fig. 1(b), when input A has a rising transition at time t_1 and input B has a falling transition at time t_2, output C will have a falling transition at time t_1+0.2ns and another rising transition at t_2+0.2ns. However, when we use SPICE to simu-

Received February 21, 1998; accepted June 11, 1998.
Communicated by Youn-Long Lin.

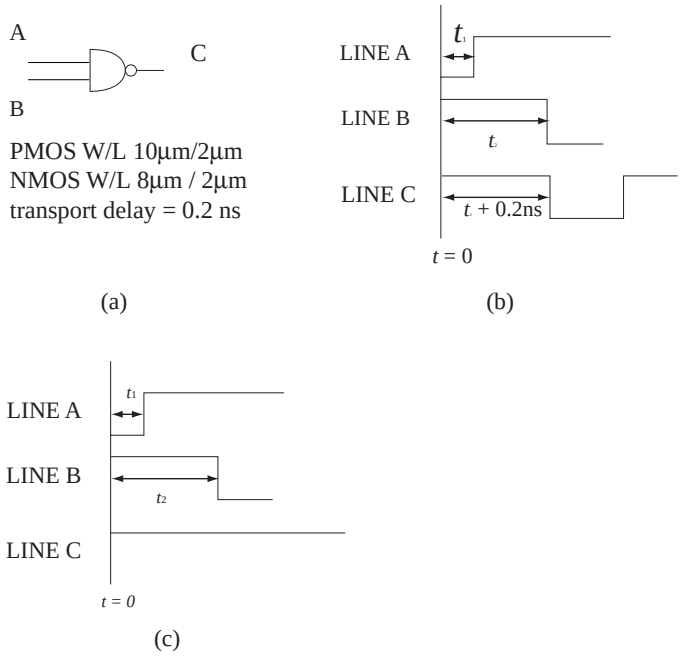


Fig. 1. (a) An example circuit of an NAND gate. (b) Waveforms of the lines obtained by using transport delay. (c) Waveforms of the line obtained by using the inertial delay model in which the input transitions on line C are suppressed.

late the transient behavior of the NAND gate while applying the above input stimuli, we find that the transitions on the input lines do not always propagate to the output line C. If the difference between t1 and t2 is not greater than a particular value, 0.2ns, for this example, the transitions will disappear (Fig. 1(c)). The particular value is the "inertial delay" of the gate, denoted by d_I, which is an essential characteristic of the logic gate. This is because switching a gate requires a certain amount of energy, which is determined by the size of the gate. Any input pulse whose duration is less than or equal to d_I will be automatically suppressed by the gate. Hence, if inertial delays of gates are neglected during the logic simulation, some invalid results will be obtained in predicting the transient behavior of a circuit. As a result, this may invalidate the results of analysis, for example, the number of switchings of a circuit, which is very important in determining the power of the circuit [10], or may invalidate the testing of delay faults.

To enable a logic simulator to handle the inertial delay, the event-driven method incorporating a timing wheel is usually used. An event-canceling mechanism is needed to implement the transition elimination, making the simulator very slow. This paper presents a compiled code logic simulator which utilizes an "Inertial_Function" technique to eliminate the above problem. Furthermore, the simulator incorporates the parallel pattern strategy to increase the simulation speed. As a result, the simulator is 300 times faster than the conventional simulator, which employs a timing wheel, and 8 times faster than the conventional interpreter type of simulator.

2. LOGIC SIMULATION BASED ON THE INERTIAL DELAY MODEL

In the conventional timing wheel method for transport delay model, the simulation is performed with respect to an increasing time step. That is, the events which occur at time, for example, t=1 are evaluated, and their response events are inserted into the timing wheel. Then, the events occurring at time t=2 are evaluated, and their corresponding response events are inserted into the timing wheel again. Evaluation and insertion are repeated until no event exists in the timing wheel. For this timing wheel technique to handle the inertial delay, an event canceling step is needed. The example shown in Fig. 2 can be used to explain this, where the d's and d_I's represent the transport delays and inertial delays of each gate, respectively. In the figure, the event lists of each time slot are listed in Figs. (b) and (c), where (A, ↑) and (C, ↓) mean a rising event occurring at line A and a falling event occurring at line C, respectively. If we do not consider the inertial delays of each gate, we obtain the event lists shown in Fig. 2(b). However, if the inertial delays are considered, for every evaluation of the transition event, we must check whether or not the event exists between time t-d_I and t. If such an event exists, both events needed to be canceled. In this example, at t=6, for the rising transition event at line E, since line E has a falling transition at t=3 and its dI is 3, both (E, ↓) and (E, ↑) need to be canceled. Consequently, (F, ↑) and (G, ↓) need to be canceled, and events (F, ↓) and (G, ↑) do not need to be inserted. This requires a check and cancellation procedure, which makes the computation overhead high.

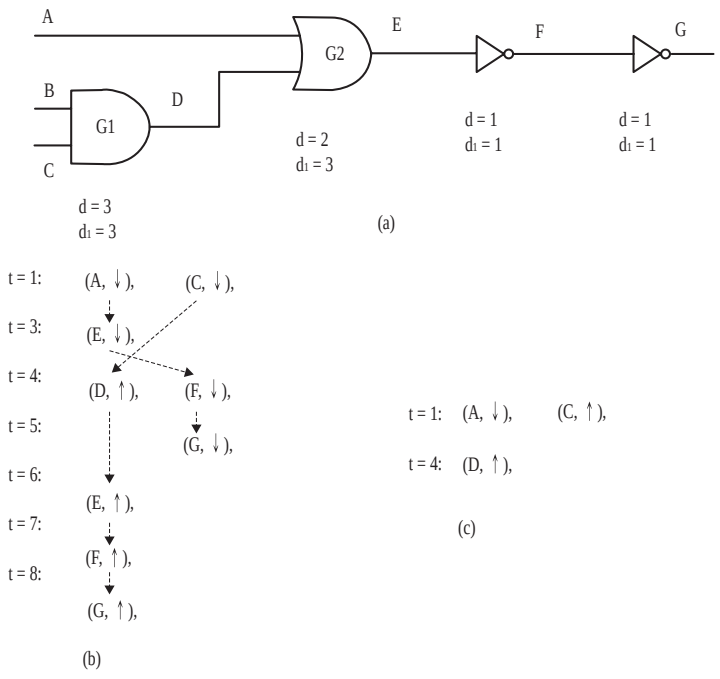


Fig. 2. (a) A example circuit to show event cancellation. (b) Event list before event cancellation. (c) Event list after event cancellation.

We propose a method which completely eliminates the use of a timing wheel. We use an array to record the timing information of each line of the circuit and conduct the simulation in a level by level fashion. For each line, we evaluate the timing array and check for transition elimination. For the pupose of illustration, we will consider the example circuit shown in Fig. 2, where the timing arrays for each line are shown in Fig. 3. Fig. 3(a) shows the timing arrays of all lines before transition elimination, and Fig. 3(b) shows the timing arrays after transition elimination. In the array Timing_E, since d of G2 is 2, Timing_E[t], the logic value of line E at time t, is equal to (Timing_A[0] OR Timing_D[0]) for t = 2. For t > 2, Timing_E[t] is equal to (Timing_A[t-2] OR Timing_D[t-2]). There are two transitions in Timing_E, i.e., a falling transition at t=3 and a rising transition at t = 6. Since the time interval of these two transitions is not larger than the inertial delay $d_I = 3$ of G2, the two transitions are eliminated , i.e., Timing_E[3], Timing_E[4] and Timing_E[5] are set to logic 1. The above operations can be performed using simple logic operations on a single timing array. This saves much computation overhead as compared to the timing wheel method. In the figure, only the underlined elements of timing arrays need to be simulated because transitions only occur at these time steps.

Timing_A :	1 0 0 0 0 0 0 0 0
Timing_B :	1 1 1 1 1 1 1 1 1
Timing_C :	0 1 1 1 1 1 1 1 1
Timing_D :	0 0 0 0 1 1 1 1 1
Timing_E :	1 1 1 0 0 0 1 1 1
Timing_F :	0 0 0 0 1 1 1 0 0
Timing_G :	1 1 1 1 1 0 0 0 1

(a)

Timing_A :	1 0 0 0 0 0 0 0 0
Timing_B :	1 1 1 1 1 1 1 1 1
Timing_C :	0 1 1 1 1 1 1 1 1
Timing_D :	0 0 0 0 1 1 1 1 1
Timing_E :	1 1 1 1 1 1 1 1 1
Timing_F :	0 0 0 0 0 0 0 0 0
Timing_G :	1 1 1 1 1 1 1 1 1

(b)

Fig. 3. (a) Timing arrays before transition elimination. (b) Timing arrays after transition elimination.

Our proposed simulator is a compiled-code simulator. An Inertial_Function is used to handle the above transition elimination, and a **potential change frame (PC-Frame),** which is derived from the **potential change set (PC-Set)**[7], is used to save memory and simulation time.

In order to simulate the timing information and the inertial characteristic of a digital circuit, the following four arrays: Time_Index, Logic_Value, Inertial_Index and Fanin_Index, denoted as G_{TI}, G_{LV}, G_{II} and G_{FI}, respectively, are defined for each logic gate G. If gate G has N time steps at which an event could possibly occur, and if gate G has M fanin gates, then G_{TI}, G_{LV} and G_{II} are three one-dimensional arrays whose sizes are N+1, and G_{FI} is a two-dimensional array whose size is (N+1)×M. The PC-Frame consists of the above four arrays, and each corresponding element within the four respective arrays are closely related during simulation. For example, the kth PC-Frame of a gate G contains $G_{TI}[k]$, $G_{LV}[k]$, $G_{II}[k]$ and $G_{FI}[k]$, where the first three terms are the kth element of the Time_Index array, Logic_Value array and Inertial_Index array, respectively, and the last term is the kth one-dimensional sub-array of the Fanin_Index array. $G_{TI}[k]$ records the time at which the kth possible transition can occur, $G_{LV}[k]$ stores the logic value of gate G in the interval of $[G_{TI}[k], G_{TI}[k+1]]$, and $G_{II}[k]$ stores the number of succeeding PC-Frames which will affect the

transition of the current PC-Frame due to the inertial delay. For the Fanin_Index array G_{FI}[k], the one-dimensional sub-array G_{FI}[k][j] stores the corresponding PC-Frames of the jth fanin gate of gate G for evaluation of G_{LV}[k], where j = 0 ~ (M-1).

Fig. 4 is an example of the PC-Frames of the output C of an AND gate with two inputs, A and B. There are five PC-Frames on output C, which is denoted by a dashed box, i.e., 0, 1, 2, 3, 4, where the 0th frame is only for reference, and the gate may have transitions occurring at four time steps: step 5, step 6, step 7, and step 9, as listed in CTI of the PC-Frame. For this gate, input A has a Time-Index, A_{TI}, of {0, 3, 4}; that is, it may have transitions at time steps 3 and 4; and input B has a Time-Index, B_{TI}, of {0, 2, 6}. The gate has a transport delay of d=3 units, and an inertial delay of d_I=2 units. The Time-Index, C_{TI}, is obtained as follows: First, A_{TI} and B_{TI} are combined to get C_{TI}', which is {0, 2, 3, 4, 6}. Since the propagation delay of the gate, d, is equal to 3, C_{TI} is obtained by adding 3 to each of the elements of C_{TI}' to be {0, 5, 6, 7, 9}. The Fanin_Index Array, C_{FI}, is obtained as follows: For C_{FI}[1][0], the CFI of input A at PC-Frame 1 since C_{TI}[1]-d = 2; that is, to cause output C to have a transition at time step 5 by means of input A, A should have a transition at time step 2. However, A_{TI}[1] = 3, which is greater than 2; i.e., a possible transition could occur only at time step 3; hence, only the logic value of PC-Frame 0 of input A should be used; i.e., C_{FI}[1][0] should be 0. For C_{FI}[1][1], since B_{TI}[2] = 6, which is greater than 2, and B_{TI}[1] = 2, the logic value of the first PC-Frame of input B should be used, i.e., C_{FI}[1][1] = 1. In other words, the Logic_Value C_{LV}[1] is evaluated by the following equation:

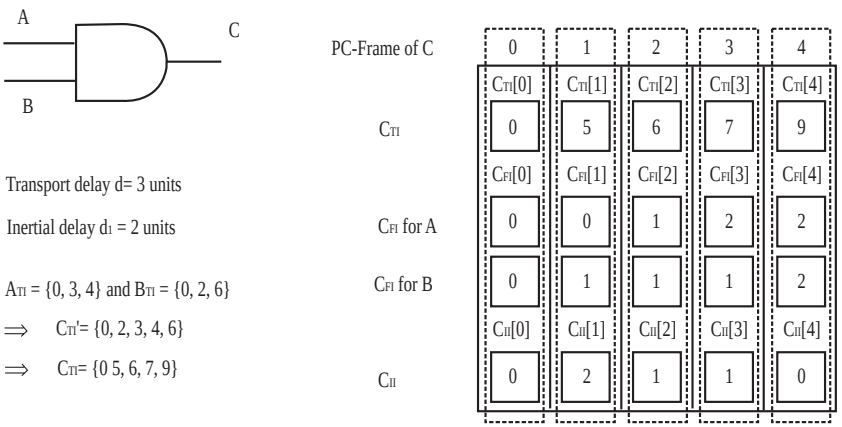


Fig. 4. An example circuit of an AND gate and the content of its PC-Frame.

$$C_{LV}[1] = A_{LV}[C_{FI}[1][0]] \,\&\, B_{LV}[C_{FI}[1][1]]$$
$$= A_{LV}[0] \,\&\, B_{LV}[1].$$

The Inertial_Index C_{II}, is obtained as follows: since (C_{TI}[3]-C_{TI}[1]) = 2 is equal to inertial delay d_I = 2 and (C_{TI}[4]-C_{TI}[1]) = 4, which is greater than inertial delay d_I = 2, C_{TI}[1] is assigned to be 2. This means that the transition on the two succeeding PC-Frames will affect the transition on the current PC-Frame. Similarly, all other elements of C_{FI} and C_{II} can be obtained in the same way, and they are listed in the figure.

The general steps in computing the PC-Frame are summarized as follows: First, for a primary input G, $G_{TI} = \{0,1\}$, $G_{FI} = \{\text{null, null}\}$ and $G_{II} = \{0,0\}$. Then, for a general gate G having a transport delay d, an inertial delay d_I and M fanin gates, I_0, I_1, ..., I_{M-1}, its G_{TI} is obtained by combining the sets I_{jTI} for every fanin gate I_j and by adding d to each element in G_{TI}. If the number of elements in the set G_{TI} is N+1, then G_{FI}, the Fanin_Index of gate G, is a two-dimensional (N+1)×M array, of which $G_{FI}[k][j]$ is an element, stores the corresponding PC-Frames of the jth fanin gate I_j of gate G for evaluation of $G_{LV}[k]$. The Fanin_Index array of first PC-Frame is an array containing 0, i.e., $G_{FI}[0][j] = 0$ for each j, since the 0th PC-Frame is the reference of time for every applied stimulus. Other $G_{FI}[k][j]$s are obtained using the following procedure (in C language):

for (p =0; $I_{jTI}[p] \le G_{TI}[k]-d$; p++);
$G_{FI}[k][j] = p - 1$.

The evaluation formula for Logic_Value $G_{LV}[k]$ derived from $G_{FI}[k]$ is as follows:

$$G_{LV}[k] = F_G(I_{0\,LV}[G_{FI}[k][0]],..., I_{j\,LV}[G_{FI}[k][j]],..., I_{(M-1)LV}[G_{FI}[k][M-1]]),$$

where F_G is the logic function of gate G.

Inertial_Index, $G_{II}[k]$, the number of succeeding PC-Frames which influence the Logic_Value of the current PC-Frame, is computed from Time_Index and d_I as follows:

1. Begin with n = k + 1,
2. If ($G_{TI}[n] - G_{TI}[k]$) is greater than d_I, then assign $G_{II}[k] = n - k - 1$ and stop.
3. If n is equal to N, the number of elements of G_{TI}, then assign $G_{II}[k] = n - k$ and stop.
4. Assign n = n + 1, and go back to step 2.

As mentioned previously, Inertial_Function is used to check the existence of other transitions. If a gate G with an inertial delay dI to have a transition at time t, then the transition may be influenced by other transitions at this gate in the interval $[t - d_I, t + d_I]$. Hence, Inertial_Function checks the existence of other transitions in the interval of $[t - d_I, t + d_I]$. Since this checking is performed from time t = 0, the transitions in the interval of $[t - d_I, t]$ for each gate has been checked during checking its preceding PC-Frames; hence it only needs to check for transitions for the interval $[t, t + d_I]$.

For each gate, the Logic_Value of each PC-Frame is first evaluated by means of logic simulation, taking into account its transport delay. For each $G_{LV}[k]$, if it is not equal to $G_{LV}[k-1]$, i.e., there is a transition at time $G_{TI}[k]$, then the transition elimination procedure is executed. Then, the logic value of the succeeding PC-Frame of $G_{II}[k]$ is checked. If it is not equal to $G_{LV}[k]$, then $G_{LV}[k]$ must be inverted to eliminate the transition at time $G_{TI}[k]$ since there is another transition in the interval $[G_{TI}[k], G_{TI}[k] + d_I]$. For each PC-Frame, simulation codes are added to execute the above procedure.

In the above, since the codes of the transition elimination procedure for PC-Frames with the same Inertial_Index are similar, we employ the subroutine call method to reduce the code size. Before generating the compiled codes to perform the logic simulation, we create a set of subroutines to eliminate transitions for the Inertial_Index ranging from 1 to

the maximum number of gates in the circuits. For example, a C-language subroutine for computing inertial for Inertial_Index equal to n is shown in Fig. 5. This reduces the size of the code and the compilation time because only the set of inertial subroutines is added and compiled. In our simulator, the generated mulation codes are assembly codes. The simulation codes for the example circuit shown in Fig. 4 are shown in Fig. 6, where the Logic_Value of each PC-Frame is first evaluated and then transition elimination for each PC-Frame is executed according to the associated Inertial_Index. In this example, the maximum value of Inertial_Index is 2; hence, subroutines Inertial_1 and Inertial_2 are generated.

In this simulator, since all operations in the final compiled codes are logical operations and comparisons, which are bit-wise instructions, the parallel pattern strategy is incorporated into the logic simulation to increase the simulation speed. The number of pattern pairs which can be simulated simultaneously depends on the word length of the machine.

```
int Inertial_n (target)
int *target;
{
   int mask, mask1, mask2;
   mask1 = (*(target+1))|(*(target+2))|.....|(*(target+n));
   mask2 = (*(target+1)) & (*(target +2)) &.....& (*(target +n));
   mask = (mask1 & (*(target-1)))|(~((*(target-1))|mask2));
   return ((mask & (*(target-1)))|(~(mask) & (*(target))));
}
```

Fig. 5. The C language code for the Inertial_n subroutine.

```
C_LV[0] = A_LV[0] & B_LV[0];
C_LV[1] = A_LV[0] & B_LV[1];
C_LV[2] = A_LV[1] & B_LV[1];
C_LV[3] = A_LV[2] & B_LV[1];
C_LV[4] = A_LV[2] & B_LV[2];
if (C_LV[1] ! = C_LV[0])
   C_LV[1] = Inertial_2 (& (C_LV[1]));
if ( C_LV[2] !=C_LV[1])
   C_LV[2] = Inertial_1 ( & (C_LV[2]));
if (C_LV[3] ! = C_LV[2])
   C_LV[3] = Inertial_1 (&(C_LV[3]));
```

Fig. 6. The C language simulation code for the example AND gate shown in Fig. 4.

3. EXPERIMENTAL RESULTS

This proposed parallel pattern compiled code simulator was written in C language and run on a SUN SPARCClassic workstation with 96MB memory. In order to evaluate this simulator, we also implemented two other simulators, which were a conventional event-driven time wheel timing simulator and a software interpreted type of timing simulator. Also, to compare the simulation times for different timing models, the above three simulators were also implemented using only the transport delay model; i.e., the inertial delay for

every gate was zero. All the experiments were done on ISCAS benchmark circuits[11,12], where for the ISCAS89 sequential benchmark circuits, only the combinational parts of the circuits were simulated. Each circuit was simulated with 5120 random pattern pairs, and the transport delay of each gate was assumed to be equal to the number of fanin gates.

The results are shown in Table 1, where column T.W. represents the CPU time in seconds for the timing wheel simulator, Int. stands for the interpreted simulator and C.C. stands for the compiled code simulator, respectively. The columns T.W./C.C. and Int./C.C. are the ratios for the above times, respectively. Table 1(a) shows the results for the transport delay model simulation. In the table, it is seen that the compiled-code simulator needed far fewer simulation runs for all the circuits than did the other two simulators. On average, the compiled code simulator ran 359 times faster than the timing wheel simulator and 12 times faster than the interpreted type simulator. Table 1(b) shows the results for the inertial delay model simulation, where the inertial delay of each gate was assumed to be equal to the transport delay of the gate. Similar results are seen in this table; i.e., the compiled code simulator ran much faster than the other two simulators. On average, it ran 339 times faster than the timing wheel simulator and 8 times faster than the interpreted type of simulator. Also, it should be mentioned that a 96 M machine could handle the largest circuit among the benchmark circuits.

Table 1(a). The simulation times for these types of simulators for the transport delay model with 5120 random pattern pairs.

	CPU Time			Speed Up	
	T.W.	Interp.	C.C.	T.W./C.C.	Int./C.C.
c432	20.33	1.98	0.15	135.56	13.22
c499	22.15	1.83	0.13	166.13	13.75
c880	31.48	4.98	0.38	82.13	13.00
c1355	66.95	9.10	0.77	87.33	11.87
c1908	139.37	11.08	1.08	128.65	10.23
c2670	178.10	11.80	0.90	197.89	13.11
c3540	248.97	21.42	2.07	120.47	10.36
c5315	515.52	28.62	2.22	232.56	12.91
c6288	16795.10	112.63	13.97	1202.51	8.06
c7552	1163.85	40.68	3.52	330.95	11.57
s35932c	14236.00	199.23	12.37	1151.16	16.11
s38417c	9532.07	182.45	18.27	521.83	9.99
s38584c	4281.23	192.17	13.63	314.03	14.10
Average				359.32	12.18

Table 1(b). The simulation times for these types of simulators for the inertial delay model with 5120 random pattern pairs.

	CPU Time			Speed Up	
	T.W.	Interp.	C.C.	T.W./C.C.	Int./C.C.
c432	29.17	2.25	0.25	116.67	9.00
c499	44.82	2.02	0.17	268.90	12.10
c880	66.53	5.87	0.65	102.36	9.03
c1355	78.13	10.95	1.47	53.27	7.47
c1908	222.93	13.85	2.10	106.16	6.60
c2670	235.75	13.33	1.52	155.44	8.79
c3540	346.88	26.77	4.20	82.59	6.37
c5315	853.77	33.67	4.13	206.56	8.15
c6288	568.10	149.45	29.05	19.56	5.14
c7552	1870.20	48.68	6.62	282.65	7.36
s35932c	30483.83	218.35	18.77	1624.36	11.63
s38417c	21517.40	218.30	31.58	681.29	6.91
s38584c	15885.80	218.65	22.37	710.24	9.78
Average				339.23	8.33

It is well known that for a compiled code simulator, the preprocessing time for a circuit is an important factor related to performance. Tables 2 lists the preprocessing times for the compiled code simulator and the interpreted type of simulator. Table 2(a) shows the results for the transport delay model, and Table 2(b) the results for the inertial delay model. Preprocessing of the compiled code simulator consists of two procedures: one is simulation code generation, and the other is simulation code compilation. From both tables, it is seen that the preprocessing time for the compiled code simulator was much larger than the simulation time for each circuit for simulation of 5120 pairs of patterns. However, if the number of simulated patterns is increased to, for example, five million, which is a number very commonly encountered in simulation of practical circuits, this time will become less significant. Also, comparing Table 2(a) and Table 2(b), we find that the preprocessing time needed if the inertial delay is considered is about 40% more than that if the inertial delay is not considered. In the tables, the last column, C.P.No., is the Critical Pattern Number, which was calculated using the following equation:

$$\textit{Critical Pattern Number} = \frac{((T_{P,CG} + T_{P,CC}) - T_{P,SI})}{(T_{S,SI} - T_{S,CC})} * 5120,$$

where $T_{P,CG}$, $T_{P,CC}$ and $T_{S,CC}$ are the code generation time, code compilation time and simulation time for the compiled code simulator for each circuit, respectively, and $T_{P,SI}$ and $T_{S,SI}$ are the preprocessing time and simulation time for the interpreted simulator, respectively.

Table 2(a). The preprocessing times for the interpreted type of simulator and the parallel pattern compilation type of simulator for the transport delay model.

	Interp.	Compile Code		C.P. No.
		Generate	Compile	
c432	0.12	0.37	7.70	22202
c499	0.12	0.33	7.20	22337
c880	0.22	0.87	10.00	11854
c1355	0.32	1.75	14.40	9728
c1908	0.47	2.78	18.80	10812
c2670	0.67	2.25	16.00	8259
c3540	0.85	5.40	30.10	9168
c5315	1.22	5.82	32.00	7098
c6288	1.50	31.77	163.60	10060
c7552	1.83	9.63	48.10	7701
s35932c	8.72	35.78	153.70	4953
s38417c	11.20	51.12	226.20	8299
s38584c	10.00	40.50	174.40	5876

Table 2(b). The preprocessing times for the interpreted type of simulator and the parallel pattern compilation type of simulator for the inertial delay model.

	Interp.	Compile Code		C.P. No.
		Generate	Compile	
c432	0.12	0.4	10.1	26667
c499	0.12	0.4	8.1	23109
c880	0.22	1.2	13.3	14002
c1355	0.32	2.4	21.0	12445
c1908	0.47	3.7	28.8	13951
c2670	0.67	2.8	21.3	10139
c3540	0.85	7.0	47.6	12184
c5315	1.22	7.3	48.5	9460
c6288	1.50	45.2	278.2	13689
c7552	1.83	11.9	72.5	10053
s35932c	8.72	40.8	205.9	6104
s38417c	11.20	62.8	331.3	10501
s38584c	10.00	46.9	238.9	7195

The meaning of the C.P.No. is that when the number of simulated patterns is larger than C. P.No., the total CPU time of the compiled code simulator is less than that of the interpreted simulator. It is seen that C.P.No. decreases with the size of the circuit. This means that the compiled code simulator is the better choice when a large circuit is to be simulated.

In Table 3, the average numbers of transitions per 32 random pattern pairs for simulating each circuit using only the transport delay model and using the inertial delay model are listed. It is seen that a significant percentage, i.e., 27%, of the transitions are eliminated if simulation is performed using the inertial delay model.

Table 3. The average numbers of transitions per 32 random pattern pairs under consideration the transport delay model and under consideration of the inertial delay model.

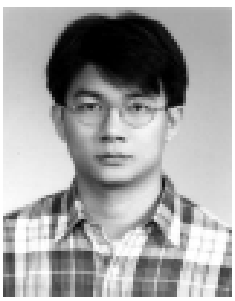
	No. of Transitions per 32 patterns		
	Transport	Inertial	elimination(%)
c432	4225	3214	24
c499	4166	3972	5
c880	7757	6889	11
c1355	14916	9186	38
c1908	30594	20885	32
c2670	36705	22914	38
c3540	63523	40237	37
c5315	92463	60453	35
c6288	1043320	715324	31
c7552	152779	101907	33
s35932c	576185	405250	30
s38417c	465550	371836	20
s38584c	318939	279309	12
Average			27

4. CONCLUSIONS

In this paper, we have proposed a parallel pattern compiled code logic simulator for the inertial delay model. The simulator uses the PC-Frame, incorporating an Inertial Function calculation technique, to eliminate the use of a time wheel, which is usually used in the conventional timing logic simulator. Also, for this reason, the parallel pattern strategy can be used, which further enhances the simulation speed. Experimental results on the ISCAS benchmarks show that this simulator enjoys significant speed improvement over the timing wheel event-driven simulator and the interpreted type of simulator. In addition, it has been found that significant percentage (27%) of transient transitions were eliminated when the inertial delay model is used, as compared to use of only the transport delay model, to simulate the timing waveform of the logic circuit.

REFERENCES

1. R. E. Bryant, D. Beatty, K. Brace, K. Cho and T. Sheffler, "COSMOS: A compiled simulator for MOS circuits," in *Proceedings of 24th Design Automation Conference*, 1987, pp. 9-16.
2. Z. Barzilai, J. L. Carter, B. K. Rosen and J. D. Rutledge, "HSS – A high-speed simulator," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 6, No. 7, 1987, pp. 601-617.
3. Z. Wang and P. M. Maurer, "LECSIM: A levelized event driven compiled logic simulator," in *Proceedings of 27th Design Automation Conference*, 1990, pp. 491-496.
4. M. Chiang and R. Palkovic, "LCC simulators speed development of synchronous hardware" in *Proceedings of IEEE International Conference on Computer Design*, 1986, pp. 87-91.
5. D. M. Lewis, "A hierarchical compiled code event-driven logic simulator," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 10, No. 6, 1991, pp. 726-737.
6. Y. S. Lee and P. M. Maurer, "Parallel multi-delay simulation," in *Proceedings of International Conference on Computer-Aided Design*, 1993, pp. 759-762.
7. P. M. Maurer, "Two new techniques for unit-delay compiled simulation," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 11, No. 9, 1992, pp. 1120-1130.
8. Y. S. Lee and P. M. Maurer, "Bit-parallel multidelay simulation," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 12,1996, pp. 1547-1554.
9. P. M. Maurer, "The inversion algorithm for digital simulation," *IEEE Tranactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 7, 1997, pp. 762-769.
10. F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Transactions on VLSI Systems*, Vol. 2, No. 4, 1994, pp. 446-455.
11. F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target simulator in Fortran," in *Proceedings of International Symposium on Circuits and Systems*, 1985, pp. 695-698.
12. F. Brglez, D. Bryan and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proceedings of International Symposium on Circuits and Systems*, 1989, pp. 1929-1934.



Kuo Chan Huang (黃國展) was born in Tainan, Taiwan, in 1968. He received the B.S. degree in electronics engineering from National Chiao Tung University, Taiwan, R.O.C., in 1990. He is a Ph.D. student in the Institute of Electronics, National Chiao Tung University, Taiwan. He engages in research on VLSI testing and design for testability.

Chung Len Lee (李崇仁) obtained his B.S. from National Taiwan University in 1968 and M.S. and Ph.D. degrees from Carnegie Mellon University in 1971 and 1975, respectively, all in Electrical Engineering. He has been with the Department of Electronics Engineering, National Chiao Tung University, since 1975, engaging in teaching and research in the fields of semiconductor devices, integrated circuits, VLSI, computer aided design and testing. He has supervised over 100 M.S. and Ph.D. students who completed their theses and has published over 200 papers in the above areas. He has been involved in various technical activities in the above areas in Taiwan as well as in Asia. He is on the editorial board of JETTA.

Jwu E Chen (陳竹一) received the B.S., M.S., and Ph.D. degrees in electronics engineering from National Chiao Tung University, Taiwan, R.O.C. He has been an Associate Professor in the Department of Electrical Engineering, Chung Hwa University, Taiwan, since 1990. His research interests include multiple-valued logic, VLSI Testing, synthesis for testability, reliable computing, yield analysis, and test management. He is a member of IEEE, the Computer Society, AAAS, and NYAS.