Efficient Boolean Characteristic Function for Fast Timed ATPG

Yu-Min Kuo, Yue-Lung Chang, Shih-Chieh Chang
Department of CS, National Tsing Hua University, Hsinchu, Taiwan
ymkuo@cs.nthu.edu.tw, ulchang@nthucad.cs.nthu.edu.tw, scchang@cs.nthu.edu.tw

ABSTRACT

Circuit timing analysis is important in various aspects of circuit optimization. The problem of finding input vectors achieving functional and temporal requirements is known as timed Automatic Test Pattern Generation (timed ATPG). A timed ATPG algorithm will return an input vector that satisfies functional and temporal requirements simultaneously when evaluated. Several previous works use timed ATPG as a core engine for solving problems related to timing analysis, such as crosstalk and maximum instantaneous current analysis. Despite the usefulness of timed ATPG, traditional timed ATPG is slow and unscalable for large circuits. In this paper, we present a very efficient way for timed ATPG. On average, our results are 8 times faster than the most recent work, and in some cases, up to 32 times faster.

1. INTRODUCTION

Advanced VLSI design depends on accurate analysis of circuit timing in circuit optimization. Most previous works in timing analysis attempt to pessimistically find the delay of a circuit by taking false paths into account [3][4][11][14][16][17]. However, applications such as [4][5][7][9][10] require input vectors that, when evaluated, can meet some functional and temporal requirements simultaneously. In addition, the temporal requirement is not necessarily limited to the longest delay of the circuit. The problem of finding input vectors satisfying functional and temporal requirements simultaneously is known as timed Automatic Test Pattern Generation or timed ATPG [4].

Several previous works use timed ATPG as a core engine to solve problems related to timing analysis. For example, crosstalk analysis requires knowing whether a transition at some time instant of a line might induce noise on another line. [9] proposed a timed ATPG approach for finding input vectors to activate and observe the crosstalk effects, while [7] reported a timed ATPG approach for finding input vectors that can activate maximum instantaneous current. Moreover, methods proposed in [4][5] find the delay of a circuit based on timed ATPG.

Traditionally, timed ATPG [4] is implemented as an extension of the PODEM algorithm. Backtracking is performed when logic or timing conflicts occur. Additionally, implications are conducted forward and backward based on timed calculus, which handles the logic and timing propagation simultaneously. However, several previous works reported that timed ATPG was unscalable using the PODEM method of implementation. In [1][8], they show that timed ATPG cannot handle large circuits even though the results are the best among non-timed ATPG approaches such as simulated annealing and genetic algorithms.

An alternative way of resolving timed ATPG problem is to first construct a Boolean function called the *Timed Characteristic Function* (*TCF*), whose on-set contains input vectors having a delay greater than a time instant when evaluated. Then, the problem of timed ATPG can be solved by running an SAT solver [12][13][15][18] or an ATPG engine on the corresponding TCF. [2] proposed to use ADD, a data structure similar to BDD, to represent the relation between each input vector and its corresponding output arrival time. The results show that the method is not usable for large circuits due to the high computation cost. [3][11][16] derive recursive equations based on sensitization criteria for the TCF. The functional and temporal requirements of a node are modeled by a TCF, which can be recursively constructed.

In this paper, we propose effective ways to model the TCF, which dramatically speed up timed ATPG. We also demonstrate that our algorithm can derive more compact TCFs than previously reported results. In addition, we extract the useful correlation information about TCFs so SAT solvers or ATPG engines can easily find solutions. Our experimental results show that these combined techniques are roughly 8 times faster than the most recent published results and in some cases, up to 32 times faster.

The remainder of this paper is organized as follows. Section 2 introduces the timed characteristic function and Section 3 provides the efficient implementation of the TCF. Then, Section 4 shows the overall algorithm and Section 5 shows the experimental results. Finally, Section 6 concludes this paper.

2. TIMED CHARACTERISTIC FUNCTION

In this section, we review the definitions of the TCF and a way to compute the Boolean function. Similar to most previous works, we use the floating mode of operation and apply all input vectors at time t=0. In addition, for simplicity of discussion, each node has a constant delay of 1 in all examples. The general delay model can be easily extended.

Let y be a circuit node, *val* be a functional requirement, and t be a temporal requirement. The Boolean function $TCF(y=val, t+)$ characterizes the set of input vectors that, when evaluated, can cause node $y=val$ to be stable later than time t. In other words,

ICCAD'06, November 5-9, 2006, San Jose, CA

given an input vector *x*, the Boolean function *TCF*(*y*=*val*, *t*+) is equal to 1 if and only if node *y*=*val* is stable later than time *t* when input vector *x* is applied. The functional requirement *val* can be 1 or 0. The symbol "+" is to emphasize the concept of "*later*" than the temporal requirement *t*. For example, in Figure 1, suppose we are interested in finding input vectors that cause node *r*=0 to be stable later than time *t*=5. All input vectors satisfying the functional requirement, *r*=0, and the temporal requirement, stability later than time *t*=5, can be expressed as the on-set of the Boolean function *TCF*(*r*=0, 5+).

[16] uses sensitization criteria to obtain the TCF. For an OR gate whose output is *y*,

$$TCF(y=1,t+)=\sum_{z\in I(y)}TCF(z=1,(t-d)+)\cdot\prod_{w\in I(y)}\{TCF(w=1,(t-d)+)+[w=0]\}$$
$$TCF(y=0,t+)=\sum_{w\in I(y)}\{TCF(w=0,(t-d)+)\cdot\prod_{z\in I(y)\cap z\neq w}[z=0]\}$$

where *I*(*y*) denotes all inputs of node *y*, and *d* denotes the delay of node *y*. Similarly, for an AND gate,

$$TCF(y=1,t+)=\sum_{w\in I(y)}\{TCF(w=1,(t-d)+)\cdot\prod_{z\in I(y)\cap z\neq w}[z=1]\}$$
$$TCF(y=0,t+)=\sum_{z\in I(y)}TCF(z=0,(t-d)+)\cdot\prod_{w\in I(y)}\{TCF(w=0,(t-d)+)+[w=1]\}$$

Consider the example in Figure 1. Using the above equations, *TCF*(*r*=0, 5+) can be derived from its inputs' TCFs, which again are derived from their inputs' TCFs. During the recursive construction process, intermediate TCFs with different temporal requirements are constructed. When certain boundary conditions are met, an intermediate TCF can be determined without further recursive construction. Consider node *f* in Figure 1. We try to find *TCF*(*f*=0, 1+). First, there are three paths from the primary inputs to node *f*, i.e., paths {*a*, *c*, *d*, *f*}, and {*b*, *c*, *d*, *f*} with the path delay of 3 and path {*e*, *f*} with the path delay of 1. Since all paths from primary inputs to the node *f* have a path delay greater than 1, the temporal requirement, stability later than time *t*=1, is met for all input vectors. Since the temporal requirement of *TCF*(*f*=0, 1+) is met, we only need to consider the functional requirement *f*=0. Therefore, we have *TCF*(*f*=0, 1+) = *f* '.

3. EFFICIENT TCF IMPLEMENTATION

Using the procedure in the previous section, we can derive a TCF. However, the size of the TCF can be much larger than the original circuit. In Section 3.1, we show novel compact recursive formulations and then propose reduction techniques in Sections 3.2 and 3.3.

3.1. Recursive TCF Formulations

First, we discuss some properties of a TCF. Note that *TCF*(*y*=*val*, *t*+) defined previously uses the "+" symbol to express the concept of "*later*" than the temporal requirement *t*. We can also use the "–" symbol to express the concept of "*earlier*" than the temporal requirement *t*. For example, *TCF*(*r*=0, 5–) characterizes all input vectors which cause node *r*=0 to be stable earlier than time *t*=5. A TCF with "–" symbol is called an *earlier-timed* TCF and a TCF with "+" symbol is called a *later-timed* TCF.

We will show later that the recursive formulations of earlier-timed TCFs can lead to a compact representation for TCFs. First, we would like to show that a later-timed TCF can always be translated to an earlier-timed TCF. Consider a circuit node *y* and a temporal requirement *t*. We have the following equations:

TCF(*y*=1, *t*+) = *TCF*(*y*=1, *t*–)' • *y*. (EQ1)
TCF(*y*=0, *t*+) = *TCF*(*y*=0, *t*–)' • *y*'. (EQ2)

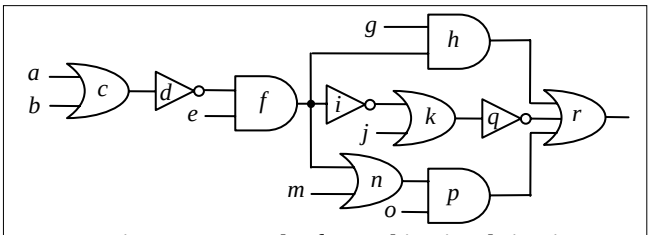

Figure 1: Example of a combinational circuit

We now illustrate the recursive earlier-timed formulations using a two-input gate whose output is *c*, inputs are *a* and *b*, and delay is *d*. For a two-input OR gate,

TCF(*c*=1, *t*–) = *TCF*(*a*=1, (*t*-*d*)–) + *TCF*(*b*=1, (*t*-*d*)–). (EQ3)
TCF(*c*=0, *t*–) = *TCF*(*a*=0, (*t*-*d*)–) • *TCF*(*b*=0, (*t*-*d*)–). (EQ4)

For a two-input AND gate,

TCF(*c*=1, *t*–) = *TCF*(*a*=1, (*t*-*d*)–) • *TCF*(*b*=1, (*t*-*d*)–). (EQ5)
TCF(*c*=0, *t*–) = *TCF*(*a*=0, (*t*-*d*)–) + *TCF*(*b*=0, (*t*-*d*)–). (EQ6)

The above equations show that an earlier-timed TCF of a node can be expressed by the earlier-timed TCFs of the node's inputs. In our algorithm, if a TCF is later-timed, it is translated into an earlier-timed TCF by (EQ1) or (EQ2). Then, we recursively derive the earlier-timed TCF by (EQ3), (EQ4), (EQ5) and (EQ6). In this way, all TCFs are earlier-timed, which is important for the sharing mechanism discussed in the next section.

3.2. Sharing Mechanism for TCFs

A TCF derived directly by the recursive formulation may have much larger size compared to the original circuit, mainly because all fanout branches of a node may require different TCFs. Consider node *f* in Figure 1, which has three fanout branches. During the recursive construction process of *TCF*(*r*=0, 5+), three intermediate TCFs, *TCF*(*f*=0, 1–), *TCF*(*f*=0, 2–), and *TCF*(*f*=0, 3–), are needed for node *f*.

In general, each fanout branch of a node may have its own functional and temporal requirements. Without careful planning, we may need to build TCFs for all fanout branches of a node. However, some TCFs are functionally equivalent, even with different temporal requirements. Before a TCF is finally constructed, the equivalence of two different intermediate TCFs can be difficult to recognize. For example, *TCF*(*f*=0, 2–) and *TCF*(*f*=0, 3–) have different temporal requirements but have equal functionality after construction.

We propose to collect useful arrival time information, which allows equivalent TCFs to be recognized during the recursive construction process. First, there can be many different paths from the primary inputs to a node *n*. Each path delay from a primary input to node *n* represents a possible delay or a possible arrival

time when an input vector is applied. For the same example, in Figure 1, there are three paths {*a, c, d, f*}, {*b, c, d, f*}, and {*e, f*} from the primary inputs to node *f* where {*a, c, d, f*} and {*b, c, d, f*} have the path delay of 3 and {*e, f*} has the path delay of 1. Therefore, there are two possible arrival times for node *f* after applying an input vector, i.e., 1 and 3.

Let $\{A_j\}$ be a sorted set of possible arrival times for node y, i.e. $A_k < A_{k+1}$. In addition, let $\{T_i\}$ be a set of the temporal requirements with which we must construct the earlier-timed TCFs on node *y*. We say the arrival time A_k is the next-largest arrival time of T_i if $A_{k-1} < T_i \leq A_k$. Consider the example in Figure 1. Node *f* has two possible arrival times: 1 and 3. In addition, node *f* has three temporal requirements: 1, 2 and 3. The next-largest arrival time of temporal requirement 2 is 3, and the next-largest arrival time of temporal requirement 3 is 3. Then, we have the following lemma:

Lemma 1: For two TCFs *TCF*(*y=val, t1–*) and *TCF*(*y=val, t2–*), if these two temporal requirements *t1* and *t2* have the same next-largest arrival time,

$$TCF(y=\text{val}, t1-) = TCF(y=val, t2-).$$

In our algorithm, the arrival time information can be derived by the same algorithm in [6]. After that, during the recursive construction process of a TCF, we use Lemma 1 to merge intermediate TCFs whose temporal requirements have the same next-largest arrival time.

3.3. Correlation Information Extraction

In this section, we illustrate how to extract the correlation information about TCFs to accelerate SAT solvers or ATPG engines. The correlation information, which can be easily recognized during recursive construction process of a TCF, describes the relationship between two intermediate TCFs of a node. We have the following two lemmas:

Lemma 2: For a node *y*, and two temporal requirements *t1* and *t2* where *t1<t2*,

$$TCF(y=val, t1-) \subseteq TCF(y=val, t2-), \text{ or}$$
$$TCF(y=val, t1-)' + TCF(y=val, t2-) = 1.$$

Lemma 3: For a node *y* and two temporal requirements *t1* and *t2*,

$$TCF(y=val, t1-) \cap TCF(y=val', t2-) = \varnothing, \text{ or}$$
$$TCF(y=val, t1-)' + TCF(y=val', t2-)' = 1.$$

When the condition in Lemma 2 is satisfied, we add an extra Boolean function $TCF(y=val, t1-)' + TCF(y=val, t2-) = 1$ to the overall TCF. When the condition in Lemma 3 is satisfied, we add an extra Boolean function $TCF(y=val, t1-)' + TCF(y=val', t2-)' = 1$ to the overall TCF. These extra Boolean functions do not change the on-set of the overall TCF, but add the correlation information when we solve the overall TCF.

5. EXPERIMENTAL RESULTS

We re-implement the algorithm shown in [16] and conduct experiments on a set of MCNC benchmark circuits. Since the main purposes in [16] are to find a circuit's delay, to fairly compare with their results, we generate 10 temporal requirements, from the largest to the smallest possible arrival time with an equal interval. Then, for each temporal requirement *t*, we attempt to find an input vector that causes at least one primary output to be stable later than time *t*. This can be modeled by the following equation.

$$TCF_C = \sum_{y \in PO(C)} [TCF(y=0, t+) + TCF(y=1, t+)].$$

Finally, TCF_C is solved using the Jerusat SAT solver [13].

In Table 1, we show the experimental results of C3540 under the unit delay model. Column 1 shows the temporal requirement. Columns 2, 3, and 4 show the number of the variables in the TCF, the run time of the TCF construction, and the run time of Jerusat by [16] while columns 5, 6, and 7 show the results by our approach. Columns 8, 9 and 10 show the ratio of the number of variables, the ratio of the run time of the TCF construction and the ratio of the run time of Jerusat by [16] to those by our approach respectively. For the highlighted temporal requirement 32, our approach is 40 times faster in the run time of Jerusat, and on average, our approach is 13 times faster.

Then, we perform experiments on a set of MCNC benchmark circuits under different delay models. For each circuit, we use the same method of generating temporal requirements as that in Table 1. Table 2 shows the results under the unit delay model and the results under the unit fanout delay model, and Table 3 shows the results using the TSMC 0.13 μm cell library. Column 1 shows the name of the circuit. Columns 2, 3, and 4 show the average number of variables (*Vars*), the average run time of the TCF construction, and the average run time of Jerusat for 10 temporal requirements by [16] while columns 5, 6, and 7 show the results by our approach. Columns 8, 9, and 10 show the ratio of the number of variables, the ratio of the run time of the TCF construction and the ratio of the run time of Jerusat by [16] to those by ours respectively. For the run time of Jerusat, on average, our approach is about 5 times faster than the approach in [16] under the unit delay model and about 8 times faster than the approach in [16] under the unit fanout delay model.

We also apply our approach to the TSMC 0.13 μm cell library in Table 3. Since the approach in [16] cannot be directly applied to a circuit with different rise and fall delays, we do not show the results. All experiments are performed using Sun Blade 2500 with 4GB physical memory.

Table 1: Experimental results of C3540

Temporal requirement	[16]			Our approach			Ratio		
	Vars	TCF(s)	Jerusat(s)	Vars	TCF(s)	Jerusat(s)	Vars	TCF	Jerusat
52	2015	0.03	0.04	1440	0.01	0.03	1.40	3.00	1.33
47	9840	0.38	0.38	2989	0.08	0.1	3.29	4.75	3.80
42	21485	0.91	0.73	5179	0.19	0.15	4.15	4.79	4.87
37	33279	1.53	7.46	7564	0.31	0.21	4.40	4.94	35.52
32	39883	1.88	13.15	9008	0.37	0.33	4.43	5.08	39.85
27	45366	2.15	6.39	10173	0.42	0.32	4.46	5.12	19.97
22	48926	2.29	5.3	10503	0.43	0.31	4.66	5.33	17.10
17	38074	1.77	1.42	7924	0.32	0.22	4.80	5.53	6.45
12	16968	0.7	0.46	3924	0.12	0.1	4.32	5.83	4.60
7	3803	0.11	0.09	1897	0.03	0.04	2.00	3.67	2.25
Average	25964	1.18	3.54	6060	0.23	0.18	3.79	4.80	13.57

Table 2: Experimental results under the unit delay model and unit fanout delay model

Circuit	unit delay model									unit fanout delay model								
	[16]			Our approach			Ratio			[16]			Our approach			Ratio		
	Vars	TCF(s)	Jerusat(s)	Vars	TCF(s)	Jerusat(s)	Vars	TCF	Jerusat	Vars	TCF(s)	Jerusat(s)	Vars	TCF(s)	Jerusat(s)	Vars	TCF	Jerusat
C1355	5501	0.20	0.27	1779	0.05	0.05	2.77	3.81	5.58	8743	0.34	0.26	1774	0.05	0.05	4.11	5.65	4.19
C1908	9155	0.38	0.54	2879	0.10	0.22	2.78	3.53	2.70	23258	1.06	1.61	4218	0.16	0.47	4.34	5.19	3.29
C2670	5766	0.20	0.16	2028	0.04	0.05	2.61	4.59	2.90	8210	0.31	0.26	2046	0.04	0.05	3.39	6.26	4.16
C3540	25964	1.18	3.54	6060	0.23	0.18	3.79	4.80	13.57	87858	4.88	24.21	10724	0.53	0.38	6.14	7.03	32.16
C432	6268	0.25	0.26	1478	0.05	0.07	3.98	4.92	4.82	11590	0.50	0.47	1543	0.06	0.06	6.05	6.90	7.20
C499	8682	0.34	0.36	2386	0.08	0.07	3.32	3.97	4.61	14855	0.64	0.56	2289	0.08	0.07	5.01	5.76	5.93
C5315	13836	0.54	0.67	4647	0.12	0.16	2.64	4.16	3.50	30074	1.36	1.83	5530	0.18	0.20	4.47	6.30	6.96
C6288	307749	20.14	28.97	69124	3.99	11.45	3.99	4.72	3.46	1772018	343.49	452.07	326735	34.55	93.14	4.97	7.87	9.03
C7552	24194	1.01	0.87	7400	0.22	0.28	2.93	4.39	2.85	81900	6.77	3.81	7778	0.26	0.28	7.56	15.71	9.43
C880	3895	0.13	0.10	1287	0.03	0.03	2.80	3.76	3.01	5854	0.22	0.16	1414	0.04	0.04	3.53	4.63	3.65
des	24157	1.16	0.82	8099	0.24	0.23	2.77	6.82	3.22	25005	1.30	0.73	8274	0.35	0.24	2.59	3.95	2.66
i10	31232	1.56	1.19	9381	0.35	0.35	2.93	5.73	3.05	79761	4.57	3.99	14269	0.71	0.81	4.58	6.96	5.05
i8	8520	0.33	0.23	3025	0.08	0.09	2.74	3.86	2.54	8694	0.34	0.23	2945	0.08	0.09	2.81	4.16	2.61
my_adder	5821	0.22	1.17	1376	0.05	0.09	3.96	4.24	15.06	10381	0.43	1.95	1585	0.06	0.09	5.68	6.20	18.05
Average	34338	1.98	2.80	8639	0.40	0.95	3.14	4.52	5.06	154871	26.16	35.15	27938	2.65	6.85	4.66	6.61	8.17

6. CONCLUSIONS

In this paper, we present a very effective way for timed ATPG. Our contributions include a compact form to represent TCFs, a sharing mechanism to reduce the size of TCFs, and correlation information extraction techniques to decrease the run time of solving timed ATPG. On average, our algorithm is 8 times faster than [16] and in some cases, our algorithm is 32 times faster.

Table 3: Experimental results using the TSMC 0.13 µm cell library

Circuit	Our approach		
	Vars	TCF(s)	Jerusat(s)
C1355	2866	0.13	0.09
C1908	7692	0.40	0.77
C2670	1904	0.05	0.05
C3540	15792	1.22	0.76
C432	2102	0.12	0.08
C499	2326	0.11	0.08
C5315	5557	0.22	0.19
C6288	1785688	794.29	1335.13
C7552	8154	0.32	0.27
C880	1307	0.04	0.03
des	7010	0.32	0.22
i10	14787	1.04	0.68
i8	2025	0.06	0.06
my_adder	2122	0.07	0.06
Average	132809	57.03	95.60

7. REFERENCES

[1] P. Ashar, and S. Malik, "Functional Timing Analysis Using ATPG," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 1025-1030, Aug. 1995.

[2] R. I. Bahar, H. Cho, G. D. Hachtel, E. Macii, and F. Somenzi, "Timing Analysis of Combinational Circuits using ADD's," in *Proc. of IEEE European Design Test Conference*, pp. 625-629, 1994.

[3] H.-C. Chen, and D. Du, "Path Sensitization in Critical Path Problem," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 196-207, Feb. 1993.

[4] S. Devadas, K. Keutzer, and S. Malik, "Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 12, pp. 1913-1923, Dec. 1993.

[5] J. L. Güntzel, A. C. M. Pinto, and R. Reis "A Timed Calculus for ATG-Based Timing Analysis of Circuits with Complex Gates," in *Proc. IEEE Latin American Test Workshop*, pp. 234-239, 2001.

[6] H. Kriplani, F. Najm, and I. N. Hajj, "Pattern Independent Maximum Current Estimation in Power and Ground Buses of CMOS VLSI Circuits: Algorithms, Signal Correlations, and Their Resolution," *IEEE Trans. on Computer-Aided Design*, vol. 14, no. 8, pp. 998-1012, Aug. 1995.

[7] Y.-M. Jiang, A. Krstic, and K.-T. Cheng, "Estimation for Maximum Instantaneous Current Through Supply Lines for CMOS Circuits," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 8, no. 1, pp. 61-73, Feb. 2000.

[8] Y.-M. Jiang, K.-T. Cheng, and A. Krstic, "Estimation of Maximum Power and Instantaneous Current Using a Genetic Algorithm," in *Proc. of IEEE Custom Integrated Circuits Conference*, pp. 135-138, 1997.

[9] R. Kundu, and R. D. Blanton, "Timed Test Generation for Crosstalk Switch Failure in Domino CMOS Circuits," in *Proc. of the IEEE VLSI Test Symposium*, pp. 379-385, 2002.

[10] R. Kundu, and R. D. Blanton, "ATPG for Noise-Induced Switch Failures in Domino Logic," in *Proc. of the ICCAD*, pp. 765-768, 2003.

[11] P. C. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vicentelli, "Timing Analysis and Delay-Fault Test Generation using Path-Recursive Functions," in *Proc. of the ICCAD*, pp. 180-183, 1991.

[12] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. of the DAC*, pp. 530-535, 2001.

[13] A. Nadel, "Backtrack Search Algorithms for Propositional Satisfiability: Review and Innovations," *Master's Thesis, the Hebrew University of Jerusalem*, 2002.

[14] J. P. M. Silva, and K. A. Sakallah, "Efficient and Robust Test Generation-Based Timing Analysis," in *Proc. of the International Symposium on Circuits and Systems*, pp. 303-306, 1994.

[15] J. P. M. Silva, and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.

[16] L. G. Silva, J. P. M. Silva, L. M. Silveira, and K. A. Sakallah, "Satisfiability Models and Algorithms for Circuit Delay Computation," *ACM Trans. on Design Automation of Electronic Systems*, vol. 7, no. 1, pp. 137-158, Jan. 2002.

[17] H. Yalcin, and J. P. Hayes, "Hierarchical Timing Analysis Using Conditional Delays," in *Proc. of the ICCAD*, pp. 371-377, 1995.

[18] H. Zhang, "SATO: An Efficient Propositional Prover," in *Proc. of International Conference on Automated Deduction*, pp. 272-275, 1997.