

# SWiTEST: A Switch Level Test Generation System for CMOS Combinational Circuits\*

K.J. Lee  
Dept. of Electrical Engineering  
National Cheng-Kung University  
Tainan 70101, Taiwan  
R.O.C.

C.A. Njinda, M.A. Breuer  
Dept. of EE-Systems  
University of Southern California  
Los Angeles, CA 90089-2560  
U.S.A.

## Abstract

For CMOS circuits, switch level test generation (SLTG) is potentially more powerful than conventional gate level test generation (GLTG). In this paper we present a SLTG system called SWiTEST. SWiTEST deals with bridging, breaking, stuck-open/on and stuck-at faults. It employs both logic and current (IDDQ testing) monitoring and takes into account the invalidation problem associated with stuck-open tests. The framework for SWiTEST is based on the PODEM algorithm.

## 1 Introduction

This paper describes SWiTEST, a system for automatically generating tests for CMOS combinational circuits. Conventional test generation (TG) algorithms usually deal with the detection of line stuck-at faults at the gate-level [1, 2, 3]. These algorithms have been applied to benchmark circuits containing more than ten thousand primitive gates, and have been shown to be quite efficient in generating tests for line stuck-at faults.

However, there exists several unique problems when testing CMOS circuits [4, 5]. These problems point out the inadequacy of gate level modeling and therefore a different form of representation is required. Judging from previous work [6, 7, 8, 9], switch level modeling [6] appears to be a more appropriate model to use for test generation of CMOS circuits. In this model the basic unit of a CMOS circuit is a transistor and each transistor is considered as an ON/OFF switch whose status depends on the logic value at its gate terminal. Most of the structural information of a circuit can be captured by this model. Realistic CMOS defects can be modeled as not only stuck-at faults, but also transistor stuck-on/open, bridging and breaking faults.

The inadequacy of gate level test generation (GLTG) in detecting CMOS faults does not imply that the concepts developed in GLTG should be discarded altogether. Test generation is basically a search problem where an appropriate test is sought. Many concepts used to search for tests at the gate level are also applicable at the switch level. Therefore we have based the implementation of SWiTEST on a well-known GLTG algorithm, namely PODEM [1]. To apply the concepts in PODEM at the switch level we divide the switch level test generation (SLTG) task into three subtasks: circuit manipulation, fault analysis and a

test generation framework. Circuit manipulation attempts to preprocess a CMOS circuit so that useful information for test generation can be obtained. Fault analysis aims at identifying what fault models are most realistic and how to efficiently deal with each of these faults. The test generation framework consists of the major components that can be shared by the test generators for each fault model. To improve fault detectability SWiTEST employs IDDQ testing [10, 11]. By monitoring the steady state current, this technique can detect many CMOS defects such as shorts between two metal lines, latch-up, gate-oxide short, and a floating node with a voltage that turns on both P- and N-type transistors. More information about IDDQ testing can be found in [12, 13].

The rest of this paper is structured as follows. Section 2 provides an overview of the SWiTEST system. In Section 3 some experimental results are presented and discussed. We conclude in Section 4.

## 2 System Overview

A block diagram of the major components of SWiTEST is presented in Figure 1. Next these components are described. Due to space limitation the reader should refer to [14] for more detailed information.

**Circuit Partitioning:** At the gate level the I/O relations between two gates are well defined, while at the switch level each transistor is a bidirectional device. To make use of PODEM concepts such as efficient backtracking, transistors must be clustered into groups so that during test generation each group can be considered as a basic unit and the I/O relations among groups can be defined. SWiTEST adopts a commonly-used partitioning scheme for switch level circuits which clusters a MOS circuit into a number of disjoint "channel-connected" components called *transistor groups (TGs)* [15]. The relationship between two TGs can be defined by the interconnections through the gate terminals of transistors in the original circuit. Each interconnection between two TGs can be considered as an input or output of the corresponding TGs.

**Complex Gate Identification:** Most switch level circuits consist of complex gates and pass transistors, where a complex gate is made up of a P-network and an N-network and contains only one output node. The output can never be treated as an "input" to the gate. Also to have a logic 1(0) at the output of a complex gate, a conducting path of P(N)-type transistors from VDD(GND) to the output must be turned on. Such knowledge can aid test

\*This work was supported by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract No. N00014-87-K-0861, and by the Federal Bureau of Investigation under Contract No. JFB190092.

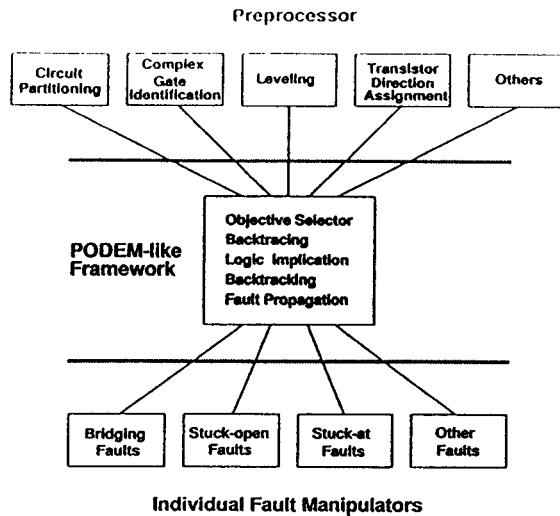


Figure 1: Organization of SWiTEST

generation processes such as backtracking and fault propagation. Moreover, many complex gates are in fact primitive gates (NAND, NOR or inverters) which have well defined I/O relationship. Thus further identifying primitive gates can potentially reduce the switch level test generation complexity to an extent that is close to that of gate level test generation.

**Transistor Group Levelling:** To determine the next objective to be achieved for backtracking and fault propagation, many gate level PODEM implementations use the “level” information, i.e., “how far” a gate is from its closest primary input or output. Similar level information based on transistor groups is employed in SWiTEST.

**Transistor Direction Assignment:** The partitioning or gate identification scheme only identifies the I/O relation among partitions or complex gates. In SWiTEST the signal flow directions of transistors [16] inside a partition or complex gate are determined and used to aid test generation. More information about this technique can be found in [14].

**Faults Considered:** SWiTEST can generate tests for stuck-open, stuck-on, bridging, breaking faults as well as stuck-at faults. A transistor stuck-on fault is considered as a bridging fault between the drain and source of the faulty transistor. For a line breaking fault a transistor is used to replace the line such that the line breaking fault can be modeled as a stuck-open fault of the transistor. It is assumed that 2-pattern tests are required for stuck-open faults while single pattern tests are enough for other faults. IDDQ testing is employed for bridging faults and for the invalidation problem of stuck-open fault testing due to charge sharing [14]. An efficient algorithm to generate stuck-open tests that cannot be invalidated by any circuit delay has been developed. The algorithm is based on deriving an intermediate vector, and then generating a two vector test set  $T_1$  and  $T_2$ . To guarantee that a test is not invalidated due to circuit delays, an algorithm that

enumerates all possible cutsets of an undirected graph is employed.

**Test Generation Framework:** Based on the gate level PODEM algorithm, the SWiTEST framework contains five major components as shown in Figure 1.

**Objective selection:** In the original PODEM algorithm only one objective is to be achieved at a time. When dealing with switch level faults it may be necessary to achieve more than one objective simultaneously. For example the initial objectives for detecting a bridging fault are to set the two shorted nodes to complementary logic values. Also when generating a test for a stuck-open fault it is necessary to turn off all but the faulty transistor in a cutset of transistors that separate a power source and an output node of a transistor group. SWiTEST uses an *objective array* to store the set of objectives that are achieved by each primary input assignment. Thus if backtracking is necessary, the objectives achieved by the latest PI assignment can be easily identified. Also to detect a bridging fault between two nodes  $x$  and  $y$ , two sets of initial objectives are possible, either  $x = 1$  and  $y = 0$ , or  $x = 0$  and  $y = 1$ . The correct selection of the set of objectives may have a large impact on the test generation performance since it is quite possible that only one set of objectives can be achieved. A set of guidelines is used in SWiTEST to determine the best objective for each fault.

**Backtracking:** Backtracking always succeeds in the original PODEM algorithm, i.e., there always exist some unspecified primary input(s) so that by assigning logic values to these PIs the status of the objective line (or node) can be changed. This, however, is not always true at the switch level. One such example is shown in Figure 2(a) where node  $x$  cannot be assigned a logic value unless a backtracking process is performed. Also in the gate level PODEM algorithm it is assumed that loops do not exist in the circuit. Thus backtracking can be a *memoryless* process, i.e., when a new temporary objective is determined, the old objective can be simply discarded. However at the switch level the old objective must be stored because backtracking might lead to a *switch level loop* as illustrated in Figure 2(b) ( $y$  to  $x$  to  $z$  to  $y$ ). Another consideration is that the direction of backtracking at the switch level is not as clear as that at the gate level. We have developed a *hybrid, search-based* method to deal with these problems. This method makes use of the information about complex gates and transistor signal flow directions during backtracking. When possible, backtracking is performed at the gate level. Otherwise signal flow direction information is used to guide switch level backtracking. The problem of discovering that a floating node cannot be assigned a logic value is solved by a search-based mechanism.

**Logic implication:** The output of a transistor group can be set to 1 (0) only when a conducting path from  $VDD(GND)$  or a primary input with logic 1(0) to that node is established. Based on this observation, an efficient recursive, event-driven algorithm has been developed. This algorithm is basically a “depth-first search” starting with the node  $N$  that is to be assigned a logic value  $V$ . Any node that is connected to  $N$  through conducting transistors is assigned the logic value  $V$ . When the value of a

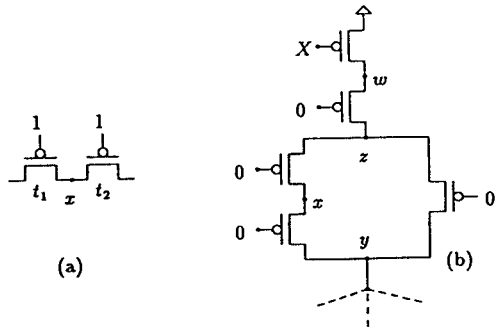


Figure 2: Problems at the switch level backtracing

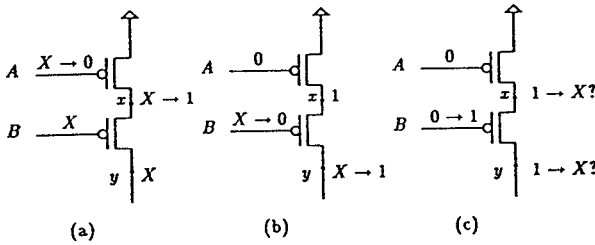


Figure 3: Difficulty in switch level backtracking

node is changed, the transistors that are controlled by the node are also processed so that the effect of assigning a logic value to a primary input can be propagated throughout the circuit in an event-driven manner.

**Backtracking:** Backtracking in the gate level PODEM algorithm is simply a forward implication starting with the primary input whose value has just been changed. It is not necessary to keep track of the information about which primary input assignment causes the status change of any internal line (or node) of the circuit. At the switch level this information is necessary as explained in the following example. In Figure 3(a)  $A$  is assigned 0. This results in  $x$  being assigned a value of 1. Then  $B$  is assigned a value of 0 as shown in Figure 3(b), which results in the assignment of value 1 to  $y$ . If later it is found that  $AB = 00$  cannot lead to a test, then backtracking must be performed and a value of 1 must be assigned to  $B$ , as shown in Figure 3(c). If the information that  $x$  drives  $y$  to 1 due to  $B = 0$  was not recorded, then unless a new simulation process on the entire transistor group containing  $x$  and  $y$  is done, it is impossible to go back to the status before  $B = 0$  was assigned.

**Fault propagation:** Fault propagation in SWiTEST can be divided into two stages: to propagate the fault effect from the fault site to an output of the faulty transistor group, and to propagate the fault effect from an output of the faulty transistor group to some primary output. Both stages can be done by setting objectives that will give a transistor group output different logic values in the

fault-free and the faulty circuits, and then trying to backtrace and assign some PI values to achieve these objectives. The fault propagation mechanism used in SWiTEST is similar to that used in the PODEM algorithm. The difference is that in the PODEM algorithm both the fault-free and faulty circuits are processed concurrently, while in SWiTEST, they are processed separately. During fault propagation, if a node has different values in the fault-free and faulty circuits (including 1 and  $X$ , and 0 and  $X$ ), then it is considered to be in the D-frontier [1]. When one primary input is assigned a new value the simulation must be performed on the fault-free and faulty circuits separately, thereby apparently requiring twice the computation time when compared to the original PODEM algorithm. However, at the switch level if a node has complementary logic values in the fault free and faulty circuits, then the node turns on different type of transistors. Thus only half of the circuit will be processed. This implies that at the switch level, dealing with both circuits concurrently is not more efficient than dealing with them separately.

### 3 Experimental results

Nine circuits consisting of a 4-bit parity generator (parity), a combinational adder (adder) and seven other circuits including six ISCAS-85 benchmark circuits have been used for experiments on a SUN Sparc 1 Workstation. The gates in a ISCAS-85 circuit are flattened into fully complementary CMOS switch level implementation. All transistor stuck-open and stuck-on faults in each circuit are processed. For stuck-on faults (which are considered as bridging faults), it is assumed that IDDQ testing is used. For stuck-open faults, test invalidation due to circuit delay is considered. Charge sharing is not explicitly considered because it is assumed that IDDQ testing is used. Tests for all node stuck-at faults can also be generated. To compare SWiTEST to existing gate level generators, experimental results on stuck-at faults at the output of each transistor group are given. Each stuck-on, stuck-open or stuck-at fault in each circuit is processed independently. No fault collapsing or fault simulation is used. The experimental results are shown in Table 1. The maximum number of backtrackings allowed per fault is set to 1000. Each columns of this table is defined as follows: [ntran] – number of transistors in the circuit, which is also the number of transistor faults; [mem] – size of memory required in  $10^3$  bytes; [nd] – number of faults detected; [nu] – number of faults identified as undetectable; [%fd] – percentage of detected faults among all faults; [s/f] – average time (seconds) spent on each fault, which does not include the relatively small pre-processing time; and [df] – number of stuck-at faults under consideration.

All faults in the first three circuits are either detected or identified as undetectable. For other circuits there exists faults that cannot be detected or identified as undetectable within the limited backtrackings. This is expected since it is well known that the PODEM algorithm cannot efficiently identify all undetectable faults even at the gate level.

The total test generation time is dependent on circuit size. However it appears that the average test generation time for each type of fault does not totally depend on circuit size. This is due to the fact that some circuits (e.g., c3540) have a larger percentage of aborted faults than oth-

circuit	ntran	mem (K)	s-on				s-op				s-at				
			nd	nu	%fd	s/f	nd	nu	%fd	s/f	nf	nd	nu	%fd	s/f
cadder	28	128	28	0	100.	0.001	28	0	100.	0.002	10	10	0	100.	0.001
parity	32	132	32	0	100.	0.001	32	0	100.	0.002	16	16	0	100.	0.002
c60	110	140	108	2	98.18	0.002	102	8	92.73	0.004	86	84	2	97.67	0.003
c880	1802	400	1751	0	97.17	0.070	1802	0	100.	0.014	1178	1178	0	100.	0.010
c1355	2308	468	2126	0	92.11	0.180	2068	72	89.60	0.248	1290	1290	0	100.	0.062
c1908	3446	656	3132	0	90.89	0.275	3393	0	98.46	0.162	2226	2204	0	99.01	0.138
c2670	5668	1044	5555	29	98.01	0.085	5381	48	94.94	0.257	4088	3955	16	96.75	0.219
c3540	7504	1300	7325	12	97.61	0.257	6888	0	91.79	1.176	5020	4697	0	93.57	1.167
c7552	15396	2556	15100	26	98.08	0.173	14762	0	95.88	0.472	10330	10061	0	97.40	0.334

Table 1: Experimental results

ers.

The memory requirement for each circuit is clearly linear with circuit size. For the largest circuit which contains 15,396 transistors, the memory requirement is less than 2.6 Mbytes, which is significantly less than that reported in [7], which requires 15 Mbytes for a circuit containing 770 transistors.

As expected, the percentage of detected stuck-open faults is in general smaller than the other two faults. However the difference is not significant. This result illustrates that the robust detection of stuck-open faults using a PODEM based algorithm is achievable.

The average time for SWiTEST to process a stuck-at fault for c7552 is 0.334 seconds. In [3], which is probably the fastest gate level test generation system currently available, it takes approximately 41.5 seconds to generate 380 test vectors. This also includes the time for fault simulation. From experimental data reported recently, fault simulation and test generation time are approximately the same [2, 3]. Therefore the average time to generate a test for a given fault is roughly  $(41.5/2)/380 = 0.0546$  seconds, which is within an order of magnitude of 0.334 seconds. This comparison is not very precise due to different hardware resources and program implementation, but it does illustrate that SLTG can be done in time compatible to that required for GLTG. Note that in [3] the number of backtrackings is close to zero due to the efficient heuristics employed. If similar heuristics are employed in SWiTEST, the performance of SLTG can be further improved.

## 4 Conclusion

Automatic test generation for CMOS circuits at the switch level is an important and complex problem. In this paper we have presented a switch level test generation system for CMOS combinational circuits. The experimental results on SWiTEST imply that switch level test generation can be done in CPU time that is within an order of magnitude of that required for gate level test generation. This is true even for stuck-open faults which have previously been considered as much more difficult to detect due to the test invalidation problem.

## References

- [1] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. on Computers*, C-30(3):215-222, Mar. 1981.
- [2] M.H. Schulz and E. Auth. Advanced automatic test pattern generation and redundancy identification techniques. In *Proc. Int'l. Symp. on Fault Tolerant Computing*, pages 30-35, 1988.
- [3] J.A. Waicukauski, P.A. Shupe, D.J. Giramma, and A. Matin. ATPG for ultra-large structured designs. In *Proc. Int'l. Test Conf.*, pages 44-51, Sept. 1990.
- [4] Y.K. Malaiya and S.Y. Su. A new fault model and testing technique for CMOS devices. In *Proc. Int'l. Test Conf.*, pages 25-34, 1982.
- [5] H.H. Chen, R.G. Mathews, and J.A. Newkirk. An algorithm to generate tests for MOS circuits at the switch level. In *Proc. Int'l. Test Conf.*, pages 304-312, 1985.
- [6] R.E. Bryant. A switch-level model and simulator for MOS digital systems. *IEEE Trans. on Computers*, C-33(2):160-177, Feb. 1984.
- [7] K. Cho and R.E. Bryant. Test pattern generation for sequential MOS circuits by symbolic fault simulation. In *Proc. Design Automation Conf.*, pages 418-428, 1989.
- [8] C.H. Chen and J.A. Abraham. Mixed-level sequential test generation using a nine-valued relaxation algorithm. In *Int'l Conf. on Computer-Aided Design*, pages 230-233, Nov. 1990.
- [9] M.K. Reddy. *Testable CMOS digital design and switch-level test generation for MOS digital circuits*. PhD thesis, Univ. of Iowa, Iowa City, 1991.
- [10] J.M. Soden and C.F. Hawkins. Test considerations for gate oxide shorts in CMOS ICs. *IEEE Design & Test of Computers*, 3(4):56-64, Aug. 1986.
- [11] W. Maly (editor). Built-in current testing — Part I. Technical Report CMUCAD-88-27, Carnegie-Mellon Univ., June 1988.
- [12] T.M. Storey and W. Maly. CMOS bridging faults detection. In *Proc. Int'l. Test Conf.*, pages 842-851, Sept. 1990.
- [13] F.J. Ferguson and T. Larrabee. Test pattern generation for realistic bridging faults in CMOS ICs. In *Proc. Int'l. Test Conf.*, pages 492-499, 1991.
- [14] K.J. Lee. *Switch level test generation for CMOS circuits*. PhD thesis, Univ. of Southern California, Los Angeles, August 1991.
- [15] R.E. Bryant. An algorithm for MOS LSI logic simulation. *LAMBDA*, pages 46-53, Fourth Quarter 1980.
- [16] N.P. Jouppi. Derivation of signal flow direction in MOS VLSI. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(3):480-490, May 1987.