Identifying Invalid States for Sequential Circuit Test Generation

Hsing-Chung Liang, Chung Len Lee, *Senior Member, IEEE*, and Jwu E. Chen, *Member, IEEE*

***Abstract*—For sequential circuit test pattern generation incorporating backward justification, we need to justify the values on flip-flops to activate and propagate fault effects. This takes much time when the values to be justified on flip-flops appear to be invalid states. Hence, it is desirable to know invalid states, either dynamically during the justification process or statically before proceeding to test generation.**

This paper proposes algorithms to identify, before test generation, invalid states for sequential circuits without reset states. The first algorithm explores all valid states from an unknown initial state to search the complete set of invalid states. The second algorithm finds the complete set of invalid states from searching the reachable states for each state. The third algorithm searches the invalid states which are required for test generation to help stop justification early by analyzing dependency among flip-flops to simulate each partial circuit. Experimental results on ISCAS benchmark circuits show that the algorithms can identify invalid states in short time. The obtained invalid states were also used in test generation, and it was shown that they improved test generation significantly in test generation time, fault coverage, and detection efficiency, especially for larger circuits and for those that were difficult to generate.

***Index Terms*—Invalid states, sequential test generation, VLSI testing.**

I. Introduction

TEST generation for digital circuits mainly consists of two processes, i.e., activating the target fault and then propagating the fault effects to primary outputs. Both processes involve justification of line values which are required for making fault activation and propagation be possible. For combinational circuits, the above step is relatively easy as it is performed only in one time frame. However, for sequential circuits, the justification may involve more than one time frame. This first increases the memory usage for test generation. For example, the sequential circuit test generators [1]–[4] which utilize both the *forward time processing* (FTP) and the *reverse/backward time processing* (RTP/BTP) to generate tests require much memory to memorize the information of time frames for propagating faults and justifying the values; and even for the test generators which use only the RTP approach [5]–[10] and process the circuit in one time frame, they still need to memorize the justified states and the information of previous justified time frames in order to speed up test generation. Second, the computation complexity is increased greatly. This is because, during the process of backward justification, there is a large searching space on the states of flip-flops. The test generator may justify the states that are, in fact, unjustifiable. These unjustifiable states are *invalid states* of the circuit, i.e., they cannot appear on flip-flops no matter what input sequences are applied to the circuit from the initial state. The test generator will search in vain for all the possible values of lines or may get into an infinite loop and finally abort the process. This degrades the test generation efficiency.

Hence, it is desirable to obtain the information on invalid states during or before test generation. Chen and Bushnell [11] have developed a test generator to dynamically identify invalid states during test generation, however, they only provided the results of small circuits. To identify invalid states before test generation, Long *et al.* [12] used implicit state enumeration based on binary decision diagrams to find some invalid states. In addition, the symbolic simulation method [13] was also proposed to find some invalid states of sequential circuits.

In this paper, three algorithms are proposed to identify invalid states before test generation. The obtained information on invalid states is applied to test generation to improve the efficiency of the test generator. The first two algorithms search the complete set of invalid states, and the third algorithm identifies only the invalid states which are required for test generation. The first one simulates the circuit to explore all of the valid states to find invalid states. The second algorithm makes use of the fact that valid states are the states that can be reached from all other states for an initializable sequential circuit. For the third algorithm, this is because, for test generation, only a partial set instead of the complete set of invalid states is required. Three algorithms have been applied to ISCAS benchmark circuits [14] to identify invalid states. The time spent was small, and the obtained information on invalid states was shown to improve test generation efficiently, especially for larger circuits and those circuits for which test generation was difficult.

II. Identification of Complete Set of Invalid States

The sequential circuits treated are at the gate level without reset states, i.e., with unknown initial states. First, some terminologies are defined.

For a sequential circuit, a state s_i is said to be able to *reach* another state s_j if there exists an input sequence to make the

Manuscript received October 11, 1996. This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC-86-2215-E009-054. This paper was recommended by Associate Editor T. Cheng.

H.-C. Liang is with the Department of Electronics Engineering, Van Nung Institute of Technology and Commerce, Chung-Li, Taiwan, R.O.C.

C. L. Lee is with the Department of Electronics Engineering, National Chiao Tung University, Hsin-Chu, Taiwan, R.O.C.

J. E. Chen is with the Department of Electronics Engineering, Chung Hua University, Hsin-Chu, Taiwan, R.O.C.

Publisher Item Identifier S 0278-0070(97)09004-0.

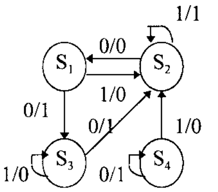


Fig. 1. State diagram example of a sequential circuit, where states s_1, s_2, and s_3 are valid states and s_4 is an invalid state.

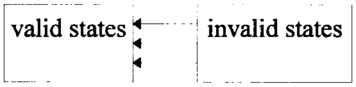


Fig. 2. Grouping of states as valid states and invalid states of a sequential circuit.

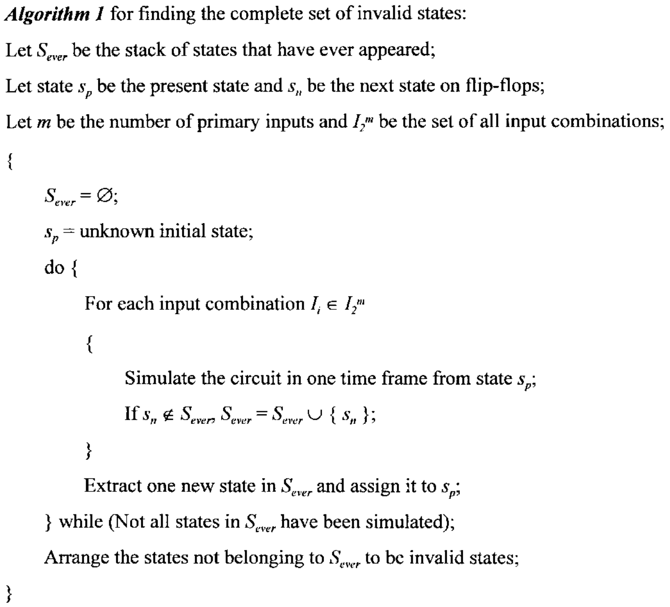
circuit go from state s_i to state s_j. A sequential circuit is called *initializable* [15] if it can be brought to a unique final state, starting with all memory elements in the unknown (X) state, through a *three-value*, i.e., [0, 1, x], *logic simulation* of an input sequence. Hence, if a sequential circuit with n flip-flops is initializable, each of its 2^n states can reach one specific state after it is applied with some input sequence. In an initializable sequential circuit, a *valid state* is a state that can be reached by any other states, including itself, while an *invalid state* is a state that cannot be reached by some states under whatever three-value input sequences. These definitions are more general than those of [16], which assumed reset states for the circuit. It is mentioned that the defined initializable here is under three-value logic simulation, and sometimes an uninitializable circuit, such as $s510$ in [14], is *functionally initializable* [17]. All states of this type of circuit will be considered as invalid states in this work.

For an initializable sequential circuit, the valid and invalid states can be obtained from their state transition diagram if given. Fig. 1 shows an example of the state transition diagram of a sequential circuit. Assume the circuit is initializable, i.e., it has at least one valid state that can be reached from the unknown initial condition. In the diagram, states s_1, s_2, and s_3 can be reached by all four states and state s_4 can only be reached by itself. Therefore, states s_1, s_2, and s_3 are valid states and s_4 is an invalid state. In general, the states of a sequential circuit can be grouped into valid states and invalid states as shown in Fig. 2.

From the above definitions, two algorithms are proposed to find all invalid states of a sequential circuit without reset states. The first one simulates the circuit to explore all valid states to find and arrange invalid states. The second one identifies the invalid states by making use of the fact that every state can reach all of the valid states and every valid state cannot reach all of the invalid states. In addition to identifying invalid states, these two algorithms can also identify initializability of the circuit.

A. Algorithm 1 for Complete Set of Invalid States

As mentioned above, the first algorithm directly simulates the circuit to explore all valid states and then to find invalid states. The simulation starts from an unknown initial state at an arbitrary time frame with all of the possible input combinations. If all of the next states are still unknown, as happened for $s510$, the simulated circuit is referred to as uninitializable. Otherwise, if any new state appears at flip-flops, it is recorded and simulated later. The process continues until no new states appear. The states that never appear are invalid states. This method is similar to that of finding the initialization input sequence (or *synchronization sequence*) [17], [18] for a sequential circuit without reset states. The main difference is that the latter will stop when a specified state appears, but our method continues to simulate until no new state appears.



```
Algorithm 1 for finding the complete set of invalid states:
Let $S_{ever}$ be the stack of states that have ever appeared;
Let state $s_p$ be the present state and $s_n$ be the next state on flip-flops;
Let $m$ be the number of primary inputs and $I_2{}^m$ be the set of all input combinations;
{
    $S_{ever} = \varnothing$;
    $s_p$ = unknown initial state;
    do {
        For each input combination $I_i \in I_2{}^m$
        {
            Simulate the circuit in one time frame from state $s_p$;
            If $s_n \notin S_{ever}$, $S_{ever} = S_{ever} \cup \{ s_n \}$;
        }
        Extract one new state in $S_{ever}$ and assign it to $s_p$;
    } while (Not all states in $S_{ever}$ have been simulated);
    Arrange the states not belonging to $S_{ever}$ to be invalid states;
}
```

Fig. 3. Algorithm 1 for finding complete set of invalid states.

Fig. 3 shows the complete algorithm of Algorithm 1, and Fig. 4 shows the process of Algorithm 1 for finding the invalid states of the circuit in Fig. 1, where the states are supposed to have been encoded in a proper way for initializability [15]. This circuit has four states, and is implemented by using two flip-flops with one input. The simulation starts from an unknown initial state, i.e., all of the states as shown in the first node, with the single input assigned 0 or 1. For applying with 0, the next state of the circuit is still unknown, and therefore it is not necessary for it to be simulated anymore. For applying with 1, the next state will be the combination of s_2 and s_3, which represents a new state. This state is recorded and the simulation is continued from this state. This process is continued until no new state appears, as shown in the figure. The states that have ever appeared are s_1, s_2, and s_3. Therefore, s_4 is the invalid state of this circuit. In addition, this circuit is initializable because it has at least one input sequence to drive the circuit to a valid state. This algorithm, though, is simple yet very efficient when the number of invalid states of the circuit is much larger than the number of valid states since it only simulates valid states.

B. Algorithm 2 for Complete Set of Invalid States

The second algorithm makes use of the following facts.

1) For an initializable sequential circuit having n flip-flops, its state transition diagram is not disjointed in which the

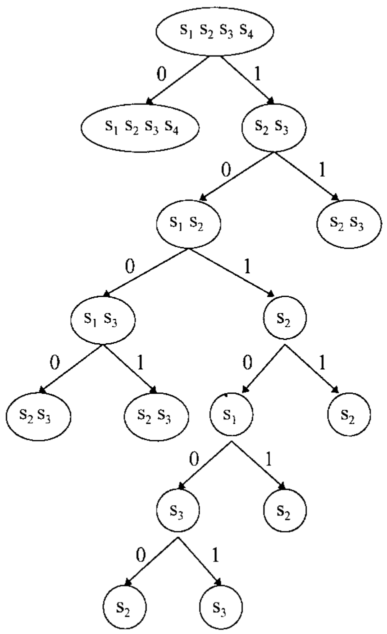


Fig. 4. The process of Algorithm 1 for finding the invalid states of the circuit of Fig. 1.

valid states are those states that all of the 2^n states can reach.

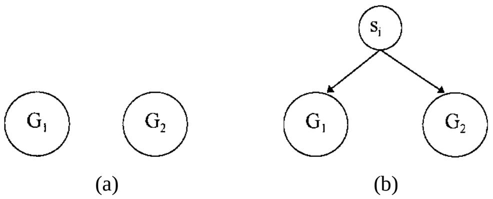
2) After it is applied with some initialization input sequence, the circuit will go to one of the valid states from the unknown initial condition.

3) Every valid state can reach all of the valid states, but cannot reach the invalid states; and every invalid state can reach all of the valid states, but may or may not reach the other invalid states.

Let n be the number of flip-flops, and let TS be the set of total states of a sequential circuit. The size of the set TS, i.e., $|TS|$, is equal to 2^n. Also, let VS and IVS be the sets of valid and invalid states, respectively. Obviously, $VS \cup IVS = TS$ and $VS \cap IVS = \emptyset$. For a state $s_i \in TS$, let RS_i be its *reachable* state set, i.e., the set of states that s_i can reach. We will have the following theorems.

Lemma 1: If a sequential circuit is initializable, under three-value logic simulation from the initial unknown condition, its state transition diagram is not disjointed, and its valid states can be reached from all of the states, including valid and invalid states.

Proof: From the previous definition, a sequential circuit is called initializable if the circuit can go to a unique state from the unknown initial condition after it is applied with some input sequence. This means that no matter what the initial state is, the circuit can go to a valid state after being applied with some input sequence. If the circuit has a disjointed state diagram, like that in Fig. 5(a), in which the states in G_1 can never reach those in G_2 and vice versa, we cannot find an input sequence that can promise to bring the circuit to a unique final state. Therefore, the circuit is not initializable. In addition, if the state diagram is connected, it may contain valid states that cannot be reached by some states, which also contradicts the definition of initializable circuit. Therefore, for an initializable circuit, all of the states, whether valid or invalid, are able to reach the valid states of the circuit. Fig. 5(b) is used to explain the second case. It shows a state diagram that is not disjointed, but the circuit is still uninitializable. Obviously, neither the states in G_1 nor those in G_2 can be valid states because they cannot be reached by some states. So if a circuit's state diagram contains the graphs in Fig. 5 or can be reduced to such types, it can be directly referred to as uninitializable. ■



(a) (b)

Fig. 5. Two examples of state transition diagram for uninitializable sequential circuits. (a) The circuit has two disjointed connected graphs. (b) Two groups of states cannot reach each other.

Theorem 1: For an initializable sequential circuit having n flip-flops, valid states are the states that can reach themselves and have the fewest reachable states, i.e.,

$$\begin{aligned} VS = \{s_i \mid s_i \in TS, s_i &\text{ can reach every} \\ s_j \in VS, &\text{ including itself} \\ |RS_i| = \min\{|RS_j|&, j = 1 \cdots 2^n\}\}. \end{aligned}$$

.

Proof: Since the circuit is initializable, there exists at least one input sequence that can initialize the circuit to one definite state, which is a valid state. From this valid state, the circuit can go through all valid states if applied with some input sequence. All of the valid states thus form a connected graph, say G_v, in which each node, i.e., each state, has at least one directed path to any node in the graph, including itself. Assume, for one state s_m not in G_v, that there exists one directed path from one node in G_v to s_m. This implies that the circuit can go to state s_m after being initialized and applied with some input sequence. Therefore, s_m is also a valid state, which contradicts the fact that G_v contains all of the valid states. This means that there are no directed paths from any node in G_v to those nodes outside G_v. Since the circuit is initializable from any state and its state transition diagram is not disjointed, each state outside G_v must have at least one directed path to the nodes inside G_v. The states outside G_v consequently are inclined to have more reachable states than those inside G_v, except for one special case when the state has a directed branch to one node in G_v but has no self-loop to itself. From this, we conclude that, in an initializable sequential circuit, the valid states are those states that can reach themselves and have the least number of reachable states. ■

In Fig. 1, TS is equal to $\{s_1, s_2, s_3, s_4\}$. The reachable state sets for states s_1, s_2, s_3 are the same, i.e., $\{s_1, s_2, s_3\}$, but that one of state s_4 is $RS_4 = \{s_1, s_2, s_3, s_4\}$. From the reachable state sets, each state of s_1, s_2, and s_3 can reach itself and the

```
Algorithm 2 for finding the complete set of invalid states:
Let $n$ be the number of flip-flops;
{
    $RS_i = \varnothing$ for $i = 1 .. 2^n$;
    For each state $s_i \in TS$, $i = 1 .. 2^n$,
    {
        If $RS_i == \varnothing$, simulate($s_i$, $RS_i$);
        For each $s_j \in RS_i$
        {
            If $s_j$ is invalid, $s_i$ is invalid, break;
            If $j < i$
            {
                If $s_j$ cannot reach $s_i$, $s_i$ is invalid;
                else $RS_i = RS_j$;
                break;
            }
            else if $j > i$
            {
                If $RS_j = \varnothing$, simulate($s_j$, $RS_j$};
                For each $s_k \in RS_j$ but $s_k \notin RS_i$,
                $RS_i = RS_i \cup \{s_k\}$;
            }
        }
        The final $RS_i$, $i = 1 .. 2^n$, is obtained;
        For any $s_i \in TS$, if $s_i$ cannot reach itself, $s_i$ is invalid;
    }
    The $VS$ contains those states that cannot be judged yet;
    In $VS$, the state having the minimum number of reachable states, i.e., $|RS|_{\min}$, is obtained
    If $VS = \varnothing$, the circuit is not initializable;
    else if $|VS| \neq |RS_i|_{\min}$, the circuit is not initializable;
    else if $|VS| = |TS|$, all $2^n$ states are valid states;
    else $IVS = TS - VS$;
}
```

Fig. 6. Algorithm 2 for finding complete set of invalid states.

other two states. Simultaneously, they have the least number of reachable states. Therefore, they are the valid states and state s_4 is the invalid state of the circuit.

The above theorems are used to judge initializability and identify valid and invalid states in an initializable sequential circuit. It is implemented according to the following. For each state s_i in TS, $i = 1 \cdots 2^n$, the algorithm simulates the circuit from s_i with all possible input combinations in one time frame to obtain the initial reachable state set RS_i. If any state s_j in RS_i is invalid, s_i is invalid because invalid states can only be reached by invalid states. Otherwise, it simulates the reachable state set RS_j in one time frame for s_j and adds the states in RS_j to RS_i. Finally, it obtains the reachable state sets for those states not yet judged to be valid or invalid. For these states, if one cannot reach itself, it is invalid. The VS is constructed from the states that pass the previous rules. The reachable state set having the least number of states is also obtained. If VS contains no states, the circuit is uninitializable. For the two cases in Fig. 5, VS will include the states both in G_1 and G_2, i.e., $|VS| = |RS_1| + |RS_2|$, but $|RS|_{\min} = \min\{|RS_1|, |RS_2|\}$. Therefore, it is easy to determine if the circuit is uninitializable or not by checking if $|VS| \neq |RS|_{\min}$. Having passed all of the rules, the circuit is initializable, and its valid and invalid state lists are therefore generated. The complete algorithm for Algorithm 2 is shown in Fig. 6.

C. Experimental Results for Algorithms 1 and 2

Algorithms 1 and 2 were applied to some of ISCAS benchmark circuits [14] to extract the complete set of invalid states. The results are shown in Table I, where the machine used was a SUN Sparc10 workstation with 192M memory. Two algorithms gave the same set of invalid states for a sequential circuit. The obtained invalid states are compressed into a "cube" form. For example, for $s27$, which has three flip-flops, its invalid states set $IVS = \{110, 111\}$ will be compressed into the set $C\_IVS = \{11\text{-}\}$, where "-" means don't care. The numbers of total states, invalid states, and compressed invalid cubes are given in the table. It can be seen that the two algorithms identified invalid states in reasonable time for these circuits. For the circuits which have more invalid states, Algorithm 1 usually used less time than did

TABLE I
EXPERIMENTAL RESULTS FOR ALGORITHMS 1 AND 2; THE MACHINE USED WAS A SUN SPARC10

circuit	inputs	flip-flops	total states	invalid states	invalid cubes	time for Algorithm 1	time for Algorithm 2
s27	4	3	8	2	1	0	0
s208	11	8	256	239	7	4	18
s298	3	14	16384	16166	137	43	12
s344	9	15	32768	31281	5040	766	38686
s349	9	15	32768	31281	5040	756	38741
s386	7	6	64	51	10	0.57	0.75
s420	19	16	65536	65519	15	21	32
s820	18	5	32	7	6	3160	1437
s832	18	5	32	7	6	2819	1443
s1488	8	6	64	16	5	11	5
s1494	8	6	64	16	5	11	5

Algorithm 2. The limitation on these two algorithms is that the memory used increases exponentially with the number of treated flip-flops. For a 192 M memory machine, it can handle at most up to 16 flip-flops in our experiment. Hence, only the benchmark circuits up to $s1494$ were applied with the two algorithms.

III. IDENTIFICATION OF INVALID STATES REQUIRED FOR TEST GENERATION

In justifying line values during backward justification in sequential test pattern generation, it rarely needs the information on complete invalid states because circuit lines usually depend on only a partial set of flip-flops. Therefore, it is sufficient enough to find on which flip-flops a circuit line depends, and then to find the required partial set of invalid states to assist the test generation. In order to find on which flip-flops a line depends, a *dependence graph* of flip-flops of a circuit is constructed.

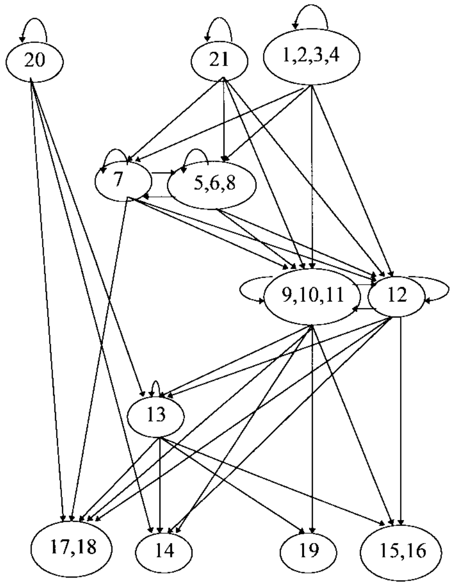


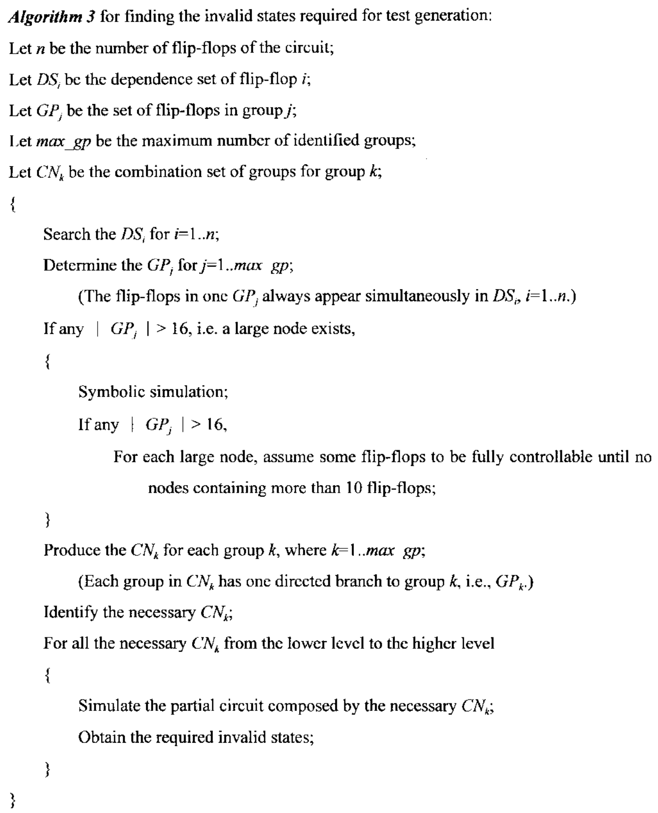
Fig. 7. Dependence graph of the circuit $s400$.

A. Dependence Graph

For a sequential circuit, the *dependence graph* of flip-flops is a graph that describes the relationship of flip-flops when flip-flops are to be justified. To explain this graph, a circuit, $s400$ [14], is used as an example. Circuit $s400$ has 21 flip-flops, and its dependence graph is shown in Fig. 7, where one node represents one or a group of flip-flops having the same dependence source and destination of flip-flops, and the numbers in each node are the flip-flop's number. In the graph, one directed branch represents the dependence between two nodes, and the values of the source node determine those of the destination node. One node with a self-loop or a group of nodes with directed branches to each other is, in fact, a cyclic graph connecting a group of flip-flops. For example, in the graph, the node containing flip-flop 20 has a branch directed to itself, which means that the flip-flop's value is determined by itself. For the node containing flip-flops 5, 6, and 8, the value of each flip-flop is determined by the values of flip-flops 1, 2, 3, 4, 21, 7, 5, 6, and 8. Apparently, the nodes on this graph can be levelized. In the graph, the nodes on the top have the lowest level, i.e., level 1, and the nodes at the bottom of the graph have the highest level. With this graph, when the invalid combinations of the values, i.e., the invalid states, on the flip-flops of the source nodes of a flip-flop are known, it is not necessary to try these values (i.e., states) when it is to justify this flip-flop. This saves much effort and computation time.

B. Algorithm 3 for Required Set of Invalid States

Algorithm 3, as shown in Fig. 8, is proposed to find the partial but required invalid states for test generation. The algorithm first constructs the *dependence set* for each flip-flop. The dependence set of a flip-flop is the set of flip-flops that can affect the value of the flip-flop. Then the flip-flops appearing simultaneously in the dependence sets are grouped into one node. In this way, the dependence graph is constructed. After constructing the dependence graph, the algorithm determines the level and the *combination set* for each node. The *combination set* of a node is a set of nodes which have directed branches to that node. For the example



```
Algorithm 3 for finding the invalid states required for test generation:
Let $n$ be the number of flip-flops of the circuit;
Let $DS_i$ be the dependence set of flip-flop $i$;
Let $GP_j$ be the set of flip-flops in group $j$;
Let $max\_gp$ be the maximum number of identified groups;
Let $CN_k$ be the combination set of groups for group $k$;
{
    Search the $DS_i$ for $i$=1..$n$;
    Determine the $GP_j$ for $j$=1..$max\_gp$;
        (The flip-flops in one $GP_j$ always appear simultaneously in $DS_i$, $i$=1..$n$.)
    If any $|GP_j| > 16$, i.e. a large node exists,
    {
        Symbolic simulation;
        If any $|GP_j| > 16$,
            For each large node, assume some flip-flops to be fully controllable until no
                nodes containing more than 10 flip-flops;
    }
    Produce the $CN_k$ for each group $k$, where $k$=1..$max\_gp$;
        (Each group in $CN_k$ has one directed branch to group $k$, i.e., $GP_k$.)
    Identify the necessary $CN_k$;
    For all the necessary $CN_k$ from the lower level to the higher level
    {
        Simulate the partial circuit composed by the necessary $CN_k$;
        Obtain the required invalid states;
    }
}
```

Fig. 8. Algorithm 3 for finding required set of invalid states.

of Fig. 7, for the node of flip-flops (5, 6, 8), the combination set is the set of nodes {(1, 2, 3, 4), (21), (7), (5, 6, 8)} in the graph. This is the set of flip-flops whose values have to be simulated to determine if there are any invalid states which can be ignored during justifying the values for flip-flops (5, 6, 8). In simulating the values of the combination set {(1, 2, 3, 4), (21), (7), (5, 6, 8)}, Algorithm 1 or 2 can be used. It is mentioned that only the partitioned circuit containing the combination set needs to be simulated. Although the obtained invalid states are therefore only a partial set of total invalid states, they are complete for the partitioned circuit part.

In the graph, if there exist nodes which contain too many, for example, more than 16 flip-flops, the memory needed to simulate the states of these flip-flops will be too large (larger than 192M). A workstation of an ordinary size of memory cannot handle this simulation. For this case, the following approach can be adopted. First, the method of symbolic simulation in [13] to identify untestable faults can be first applied to the circuit to identify the flip-flops which cannot be set to 1 and/or 0. If a flip-flop has been found to be unable to be set to 1 and/or 0, it need not be considered in the dependence graph, and can be eliminated. This reduces the size of the associated node in the dependence graph. Second, if there still exist nodes whose sizes exceed 16 after the above treatment, in the graph, the following *cycle-breaking* method can be used to break these large nodes:

The method selects the flip-flops one after another to be assumed fully controllable, and reconstructs the dependence graph until the graph has no large node. It can apply the strategies of selecting flip-flops to break cyclic structures [19]–[23], but here it permits the new nodes to still be in cyclic with fewer than ten flip-flops. For each large node, assume one of the flip-flops, for example, flip-flop a, to be fully controllable, i.e., treat it as a primary input. This essentially breaks the cyclic structure of flip-flops of the node. For all other flip-flops of this node, a subdependence graph can be constructed. For this newly constructed subdependence graph, if there is no node which has a size greater than ten, the flip-flops of this subgraph are simulated by using Algorithm 1 or 2 to find invalid states of these flip-flops. If there are still nodes whose sizes are larger than ten, the above cycle-breaking procedure is applied again. After those invalid states associated with all the remaining flip-flops are obtained, the information will be used to simulate for flip-flop a to see if there is any invalid state associated with flip-flop a. If there is none, the invalid states obtained are true invalid states. If there are invalid states, for example, $a = 1$, existing for flip-flop a, flip-flop a will be set to 0 and Algorithm 1 or 2 is used again to simulate all remaining flip-flops to find the true invalid states. The invalid states obtained are true invalid states. Obviously, there may be some or many invalid states not found after assuming some flip-flops fully controllable, but the obtained invalid states are still very useful to improve test generation, as shown later.

The complete algorithm is described in the following by using the example of Fig. 7. For $s400$, Algorithm 3 traverses the circuit structure and obtains the following dependence sets for all flip-flops, where $DS_1 = \{1, 2, 3, 4\}$ means that the value on flip-flop 1 depends on those of flip-flops 1, 2, 3, and 4:

$$
\begin{aligned}
DS_1 &= DS_2 = DS_3 = DS_4 = \{1,\ 2,\ 3,\ 4\} \\
DS_5 &= DS_6 = DS_7 = DS_8 = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 21\} \\
DS_9 &= DS_{10} = DS_{11} = DS_{12} = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8, \\
&\qquad\qquad 9,\ 10,\ 11,\ 12,\ 21\} \\
DS_{13} &= DS_{14} = \{9,\ 10,\ 11,\ 12,\ 13,\ 20\} \\
DS_{15} &= DS_{16} = \{9,\ 10,\ 11,\ 12,\ 13\} \\
DS_{17} &= DS_{18} = \{7,\ 9,\ 10,\ 11,\ 12,\ 13,\ 20\} \\
DS_{19} &= \{9,\ 10,\ 11,\ 13\} \\
DS_{20} &= \{20\} \\
DS_{21} &= \{21\}.
\end{aligned}
$$

The algorithm groups the flip-flops that simultaneously appear in the dependence sets, e.g., 1, 2, 3, and 4, into a group to be a node. For $s400$, it obtains eight nodes from the dependence sets, i.e., (1, 2, 3, 4), (7), (5, 6, 8), (9, 10, 11), (12), (13), (20), and (21). The remaining flip-flops which are not present in dependence sets are also collected if they have the same dependence set; in this case, 15 and 16 are grouped together. Finally 12 nodes are obtained. Their respective dependence groups, which are called *combination* *sets*, are also shown as follows.

	Nodes		Combination Sets
Level 1:			
	(1, 2, 3, 4)	←	(1, 2, 3, 4)
	(20)	←	(20)
	(21)	←	(21)
Level 2:			
	(5, 6, 8)	←	(1, 2, 3, 4), (21), (5, 6, 8), (7)
	(7)	←	(1, 2, 3, 4), (21), (5, 6, 8), (7)
Level 3:			
	(12)	←	(1, 2, 3, 4), (21), (5, 6, 8), (7), (12), (9, 10, 11)
	(9, 10, 11)	←	(1, 2, 3, 4), (21), (5, 6, 8), (7), (12), (9, 10, 11)
Level 4:			
	(13)	←	(20), (9, 10, 11), (12), (13)
Level 5:			
	(15, 16)	←	(9, 10, 11), (12), (13)
	(19)	←	(9, 10, 11), (13)
	(14)	←	(20), (9, 10, 11), (12), (13)
	(17, 18)	←	(20), (7), (9, 10, 11), (12), (13)

Since no nodes have flip-flops more than 16 for this circuit, no symbolic simulation and cycle-breaking method need to be applied.

In finding the necessary combination sets to find required invalid states, we find that some combination sets include others, e.g., $CN_{(5,6,8)} \supset CN_{(1,2,3,4)}$ and $CN_{(5,6,8)} \supset CN_{(21)}$. This means that the invalid states found by simulating the partial circuit composed by flip-flops 5, 6, and 8 must include those found by simulating the partial circuit composed by flip-flops 1, 2, 3, 4 and the partial circuit composed by 21. Therefore, we only need to find the largest CN's which include all of the other smaller CN's. For s400, the two largest combination sets are obtained, i.e., {(1, 2, 3, 4), (5, 6, 8), (7), (9, 10, 11), (12), (21)} and {(7), (9, 10, 11), (12), (13), (20)}. The invalid states found for these two sets of flip-flops are sufficient for justifying the values on all flip-flops. It is seen that flip-flops 14, 15, 16, 17, 18, and 19 are not present in these two sets since they do not determine any flip-flop's value. They can be grouped into a set for searching more invalid states. These three sets are the necessary combination sets of the circuit. They have level 3, 4, and 5, respectively, and will be simulated one by one according to the order of levels. We need to follow the order of levels since, usually, the obtained invalid states for the combination sets at a lower level are to be used to find the invalid states for those at a higher level. The simulation is performed on the partial circuits composed of each set of flip-flops by using Algorithm 1 or 2. For s400, it took 412 s to simulate the first combination set, but took only 8 and less than 1 s to simulate the second and the third combination sets, respectively, to obtain 23 invalid cubes. In addition, to improve the speed of simulation, there are two strategies: 1) event-driven simulation and 2) Gray-code-type input patterns are adopted, which are used in order to reduce events in the primary inputs. Experimental results show that these two strategies can accelerate the simulation three–ten times.

TABLE II
EXPERIMENTAL RESULTS ON DETERMINING COMBINATION SETS AND REQUIRED INVALID STATES; THE TIME IS IN SECONDS

circuit	flip-flops	graph depth	necessary comb. sets	time for comb. sets	invalid cubes	percentage of invalid states	time for invalid cubes
s27	3	2	1	0	1	25.00	0.02
s208	8	8	1	0.03	7	93.36	0.08
s298	14	5	7	0.03	32	98.67	4
s344	15	3	1	0.03	5040	95.46	1179
s349	15	3	1	0.03	5040	95.46	1179
s382	21	5	3	0.05	23	98.09	412
s386	6	1	3	0.05	10	79.69	0.1
s400	21	5	3	0.05	23	98.09	420
s420	16	16	3	0.05	15	99.97	0.2
s444	21	5	3	0.05	23	98.09	432
s526	21	8	7	0.08	52	98.11	7027
s526n	21	8	7	0.07	52	98.11	6933
s641	19	2	2	0.13	26	98.99	4881
s713	19	2	2	0.12	26	98.99	5095
s820	5	1	1	0.03	6	21.88	743
s832	5	1	1	0.05	6	21.88	744
s838	32	32	7	0.23	31	> 99.99	0.3
s953	29	2	5	0.13	48	99.61	51
s1196	18	3	4	0.1	13	77.95	2421
s1238	18	3	4	0.08	13	77.95	2411
s1423	74	33	38	3.45	79	> 99.99	55462
s1488	6	1	1	0.05	5	25.00	1.7
s1494	6	1	1	0.07	5	25.00	1.7
s5378	179	20	117	21.02	60	> 99.99	7949
s9234	228	4	11	8.43	420	> 99.99	1858
s13207	669	22	72	176.68	1475	> 99.99	1080
s15850	597	84	208	166.45	755	> 99.99	3692

C. Experimental Results of Algorithm 3

Table II shows some results on ISCAS'89 benchmark circuits run by this algorithm on a Sun Sparc 10. For each circuit, the graph depth is the number of levels of its dependence graph. It is seen that the time spent on finding the necessary combination sets was negligible. In addition to giving the number of invalid cubes for each circuit, Table II also provides the percentage of the found invalid states to the total 2^n states. It can be seen that the found invalid states occupy a large percentage of the total states for most circuits. The last column, which is the time spent on finding these required invalid states for test generation in seconds, shows that this algorithm took moderate time in finding these states.

In the results, circuits s1423, s5378, s9234, s13207, and s15850 were found containing nodes larger than 16 in the initially generated dependence graphs. The symbolic simulation and the cycle-breaking methods were applied to these circuits. Table III shows the results after applying the above methods. In the table, the levels of the graph depth and the numbers of large nodes in the original dependence graphs are given for comparison. The invalid cubes, time spent, and the number of remaining large nodes after symbolic simulation are included. It can be seen that many more invalid cubes were obtained after the symbolic simulation, and the time spent was minimal. For circuits s9234 and s13207, no node had a size larger than 16 after the symbolic simulation, i.e., for these two circuits, the cycle-breaking method needed not to be applied. For circuits s1423, s5378, and s15850, the cycle-breaking method was

TABLE III
RESULTS FOR RECONSTRUCTING DEPENDENCE GRAPHS BY USING SYMBOLIC SIMULATION AND ASSUMING SOME FLIP-FLOPS IN LARGE NODES TO BE FULLY CONTROLLABLE

	original graph		symbolic simulation		cycle- breaking method			
circuit	depth	large nodes	invalid cubes	time	large nodes	assumed FFs	final depth	time
s1423	6	1	0	0.02	1	17	33	1.05
s5378	9	1	37	0.58	1	22	20	12.92
s9234	18	1	344	2.98	0	0	4	0
s13207	28	4	917	50.20	0	0	22	0.15
s15850	22	3	654	24.85	3	48	84	16.43

TABLE IV
COMPARISON OF APPLYING AND NOT APPLYING INVALID STATES INFORMATION TO TEST GENERATION PROCESS

		with				without			
circuit	total faults	fault coverage	generator efficiency	test patterns	run time	fault coverage	generator efficiency	test patterns	run time
s27	32	100.00	100.00	12	0	100.00	100.00	12	0
s208	215	63.72	100.00	179	**0.93**	63.72	100.00	179	**1.08**
s298	308	85.71	99.68	208	**27.87**	85.71	99.68	208	**33.03**
s344	342	96.20	99.71	63	**13.18**	96.20	99.71	63	**28.12**
s349	350	95.71	99.43	61	**18.12**	95.71	99.43	61	**32.40**
s382	399	88.22	96.24	3289	**1025.53**	88.22	96.24	3289	**1530.23**
s386	384	81.77	100.00	**209**	**15.57**	81.77	100.00	**302**	**20.85**
s400	424	87.03	97.17	3247	**769.03**	87.03	97.17	3247	**1145.87**
s420	430	41.63	100.00	143	**20.42**	41.63	100.00	143	**37.68**
s444	474	89.45	99.79	2388	**877**	89.45	99.79	2388	**1506**
s526	555	76.04	**96.22**	**2735**	**2891**	76.04	**93.15**	**2486**	**21957**
s526n	553	**79.39**	**97.84**	**3493**	**2312**	**77.58**	**91.68**	**1216**	**17315**
s641	467	86.51	99.79	324	**8.77**	86.51	99.79	324	**36.92**
s713	581	81.93	100.00	365	**6.45**	81.93	100.00	365	**25.97**
s820	850	95.06	96.71	**889**	**27196**	95.06	96.71	**902**	**27432**
s832	870	**93.10**	**94.71**	**948**	**54780**	**92.76**	**94.37**	**942**	**73318**
s838	857	29.64	100.00	208	**13.52**	29.64	100.00	208	**16.18**
s953	1079	8.25	100.00	20	**1.48**	8.25	100.00	20	**1.50**
s1196	1242	99.76	100.00	415	**10.90**	99.76	100.00	415	**11.27**
s1238	1355	94.69	100.00	427	**13.68**	94.69	100.00	427	**15.17**
s1423	1515	**24.62**	**31.42**	**242**	**71221**	**12.15**	**13.86**	**29**	**99469**
s1488	1486	**53.84**	**55.52**	**49**	**9151**	**37.01**	**38.69**	**33**	**10346**
s1494	1506	**55.91**	**57.84**	**55**	**7328**	**54.32**	**56.24**	**54**	**8595**
s5378	4603	**67.37**	**91.11**	**1287**	**170644**	**66.83**	**90.66**	**1057**	**471903**
s9234	6927	**0.26**	**100.00**	5	**381**	**0.14**	**99.77**	5	**1344**
s13207	9967	**6.56**	**96.01**	**311**	**20715**	**5.55**	**65.01**	**28**	**45780**
s15850	11753	**0.72**	**100.00**	**12**	**2949**	**0.65**	**9.18**	**2**	**15037**

applied and the numbers of the flip-flops were assumed to be fully controllable; the final depth after appling this cycle breaking and the time spent for this method are shown in the table. It can be seen that for $s15850$, as many as 48 flip-flops were assumed, and the time spent was still small. The final depth of the dependence graph was usually much longer than that of the initial dependence graph for these circuits.

D. Application to Test Generation

The obtained invalid states were applied to a test generator which was implemented in the BACK [7]-like algorithm. It can also dynamically find some invalid states during searching test patterns. Table IV gives the results of test generation using and without using the information of obtained invalid states. The generator efficiency is defined as (#detectable faults $+$ #untestable faults)/#total faults $*$ 100. It is seen that the test generation with the information of invalid states achieved higher fault coverages and efficiencies than that without using the information. For most circuits, especially for larger circuits, the time used, including that for finding required invalid states, was still less than that without using the information. In total, the used time is reduced by 40%, but the fault coverage and efficiency are improved by 34 and 168%, respectively, for these circuits. The results for some circuits like $s13207$ and $s15850$ are even better than those of STG3 [9]. Due to the limitation of the system required for our implemented test generation program, the results for large circuits, such as $s35932$ and $s38584$, were not available, and are needed to be solved in the future.

IV. CONCLUSION

In this paper, algorithms to search invalid states for sequential circuits have been proposed. The first algorithm explores all of the valid states, and the second one searches the reachable states to obtain the complete set of invalid states. Since the test generation usually does not need the complete set of invalid states, the third algorithm is proposed to find the partial invalid states required for test generation. It analyzes the circuit structure to obtain the dependence graph among flip-flops, and extracts the necessary combination sets for simulation by applying the first two algorithms. Experimental results show that the algorithm can find the required invalid states for test generation in moderate time as compared to the test generation time. When the obtained invalid states are applied to test generation, the test generation time, the fault coverage, and test efficiency are improved, especially for large circuits. In addition to improving test generation, the information on the obtained invalid states can be useful in resynthesizing circuits or for partial scan to make circuits more testable.

REFERENCES

[1] H.-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test generation for sequential circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1081–1093, Oct. 1988.

[2] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation for highly sequential circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1989, pp. 362–365.

[3] D. H. Lee and S. M. Reddy, "A new test generation method for sequential circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 446–449.

[4] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. European Design Automation Conf.*, Feb. 1991, pp. 214–218.

[5] R. Marlett, "EBT: A comprehensive test generation technique for highly sequential circuits," in *Proc. 15th Design Automation Conf.*, June 1978, pp. 332–339.

[6] ——, "An efficient test generation system for sequential circuits," in *Proc. 23rd Design Automation Conf.*, June 1986, pp. 250–256.

[7] W.-T. Cheng, "The BACK algorithm for sequential test generation," in *Proc. Int. Conf. Computer Design*, Aug. 1988, pp. 66–69.

[8] W.-T. Cheng and T. J. Chakraborty, "GENTEST: An automatic test generation system for sequential circuits," *IEEE Computer*, vol. 38, pp. 43–49, Apr. 1989.

[9] W.-T. Cheng and S. Davidson, "Sequential circuit test generator (STG) benchmark results," in *Proc. Int. Symp. Circuits Syst.*, June 1989, pp. 1939–1941.

[10] E. Auth and M. H. Schulz, "A test-pattern-generation algorithm for sequential circuits," *IEEE Design Test Comput.*, vol. 8, pp. 72–86, June 1991.

[11] X. Chen and M. L. Bushnell, "Sequential circuit test generation using dynamic justification equivalence," *J. Electron. Testing: Theory Appl.*, vol. 1, pp. 9–33, Feb. 1996.

[12] D. E. Long, M. A. Iyer, and M. Abramovici, "Identifying sequentially untestable faults using illegal states," in *Proc. 13th VLSI Test Symp.*, 1995, pp. 4–11.

[13] H.-C. Liang, C. L. Lee, and J. E. Jwu, "Identifying untestable faults in sequential circuits," *IEEE Design Test Comput.*, vol. 12, no. 3, pp. 14–23, 1995.

[14] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential Benchmark circuits," in *Proc. Int. Symp. Circuits Syst.*, June 1989, pp. 1929–1934.

[15] K.-T. Cheng and V. D. Agrawal, "Initializability consideration in sequential machine synthesis," *IEEE Trans. Comput.*, vol. 41, pp. 374–379, Mar. 1992.

[16] S. Devadas, H.-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "Irredundant sequential machines via optimal logic synthesis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 8–18, Jan. 1990.

[17] J. A. Wehbeh and D. G. Saab, "On the initialization of sequential circuits," in *Proc. Int. Test Conf.*, Sept. 1994, pp. 233–239.

[18] A. Lioy and M. Poncino, "On the resetability of synchronous sequential circuits," in *Proc. Int. Symp. Circuits Syst.*, June 1993, pp. 1507–1510.

[19] K.-T. Cheng and V. D. Agrawal, "An economical scan design for sequential logic test generation," in *Dig. Papers, 19th Fault-Tolerant Computing Symp.*, Aug. 1989, pp. 28–35.

[20] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan designs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1990, pp. 322–325.

[21] G. W. Smith and R. B. Walford, "The identification of minimal feedback vertex set of a directed graph," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 9–15, Jan. 1975.

[22] S. Park and S. B. Akers, "A graph theoretic approach to partial scan design by K-cycle elimination," in *Proc. Int. Test Conf.*, Sept. 1992, pp. 303–311.

[23] P. Ashar and S. Malik, "Implicit computation of minimum-cost feedback-vertex sets for partial scan and other applications," in *Proc. 31st Design Automation Conf.*, June 1994, pp. 77–80.

Hsing-Chung Liang was born in Tao-Yuan, Taiwan, R.O.C., in 1967. He received the B.S., M.S., and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsin-Chu, Taiwan, R.O.C., in 1989, 1991, and 1997, respectively.

He has been Assistant Professor in the Department of Electronics Engineering, Van Nung Institute of Technology and Commerce, Chung-Li, Taiwan, since August 1997. His research interests include VLSI testing and design for testability.

Chung Len Lee (S'70–M'75–SM'92) received the B.S. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1968 and the M.S. and Ph.D. degrees from Carnegie Mellon University, Pittsburgh, PA, in 1971 and 1975, respectively, all in electrical engineering.

He has been with Department of Electronics Engineering, National Chiao Tung University, Hsin-Chu, Taiwan, 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 to complete 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 (S'88–M'92) received the B.S., M.S., and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsin-Chu, Taiwan, R.O.C.

He has been Associate Professor in the Department of Electrical Engineering, Chung Hua University, Hsin-Chu, 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 the IEEE Computer Society, AAAS, and NYAS.