Synthesis for Testability by Adding Transitions of Undefined States to State Transition Tables

Hiroyuki Yotsuyanagi, Masaki Hashizume, Takeomi Tamesada
Dept. of Electrical and Electronic Engineering, The Univ. of Tokushima
2-1 Minami-Josanjima, Tokushima 770-8506 Japan
Tel: +81-88-656-9183, Fax: +81-88-656-9183
E-mail: {yanagi4, tume, tamesada}@ee.tokushima-u.ac.jp

Abstract: In this paper we propose procedures to enhance testability by modifying state transition tables. In these procedures, transitions about undefined states, which are not described in state transition tables but exist in a synthesized gate level circuit, are added to a state transition table. Experimental results for MCNC benchmarks are shown.

1. Introduction

Test generation for sequential circuits is more difficult than that for combinational circuits since not only output function but also state transition must be considered for sequential circuits. Several methods to design easily testable circuits have been proposed. Some methods modify circuits after gate level circuit is obtained such as scan design. Some other methods have been proposed to enhance testability during logic synthesis. Such methods are called synthesis for testability. We introduce new method to enhance testability by adding transitions of undefined states in state transition tables.

We propose two procedures in this paper. One is to make undefined states distinguishable from the defined states. The other is to make undefined states reachable from the defined states.

In test generation for sequential circuits, to detect a fault, it is necessary to find an input sequence that can distinguish the faulty circuit from the fault free circuit by observing the output sequence starting from any state. When undefined states are not used during the normal operation of a circuit, we can identify the faulty circuit if after the initialization it produces the output sequence that is obtained only when the circuit starts from an undefined state. We propose a procedure to make undefined states distinguishable from the defined states.

The existence of unreachable states is one of the causes that makes test generation for sequential circuits difficult. We have shown many unreachable states exist especially in large sequential circuits[1]. We have also proposed the procedure to remove redundancy based on unreachable states and have shown that many redundancies of circuits due to unreachable states exist in sequential circuits[2]. In [3], it has been pointed out that the existence of many unreachable states degrades the testability of a circuit. We propose another procedure to avoid making undefined states unreachable.

This paper is organized as follows. In Section 2, the definitions of defined states and undefined states are shown. In Section 3, we show the main idea of our two procedures, and the details of the procedures are given in Section 4. Section 5 shows the experimental results for benchmarks and Section 6 concludes the paper.

2. Definitions

Sequential circuit M_C can be described by 5-tuple:

$$M_C = (X_C, Q_C, Z_C, \delta_C, \lambda_C)$$

where

X_C denotes a set of input vectors,
Q_C denotes a set of states,
Z_C denotes a set of output vectors,
$\delta_C : X_C \times Q_C \rightarrow Q_C$ denotes the next state function,
$\lambda_C : X_C \times Q_C \rightarrow Z_C$ denotes the output function.

In this paper, we modify sequential circuits described in state transition table such as shown in Table 1. Each transition T_i in a state transition table can be described by 4-tuple

$$T_i = (I_i, PS_i, NS_i, O_i),$$

where

$I_i \in I_C$ denotes an input vector,
$PS_i \in Q_C$ denotes a present state,
$NS_i = \delta_C(I_i, PS_i)$ and $O = \lambda_C(I_i, PS_i)$ denotes the next state and the output vevtor obtained by applying I_i to the circuit when its state is PS_i, respectively.

Table 1. An example of a state transition table

PS	NS / O	
	I = 0	I = 1
s0	s2 / 00	s2 / 00
s1	s0 / 01	s2 / 01
s2	s0 / 10	s1 / 11

If n states are given in a state transition table, $k = \lceil \log_2 n \rceil$ bits are needed to encode these states. Therefore, a gate level circuit obtained by logic synthesis has 2^k states. The state transitions about 2^k - n states are not described in the original state transition table. In this paper, we define *defined state* and *undefined state* as follows.

Definition 1: A state is said to be a *defined state* if it is described in a state transition table.

Definition 2: A state is said to be an *undefined state* if it is not described in a state transition table but exists in a synthesized gate-level circuit.

We assume that one binary code is assigned for each defined state. In this case, the number of undefined states is $(2^k - n)$. For example, in Table 1 three states (s0, s1, and s2) are described. Since at least two bits are needed for encoding these states, the number of undefined states is (2^2 - 3) = 1. Many undefined states may exist for larger circuits.

3. Adding Transitions of Undefined States

3.1 A method to distinguish undefined states from the defined states

In general, logic synthesis tools assign state transition and output of the undefined states in order to achieve minimum circuit size or minimum delay of the circuit.

Undefined states do not appear during the normal operation of the synthesized circuit. However they may appear when a fault exists in a circuit.

To detect a fault, we must find an input sequence that makes the difference between the output sequence of the fault free circuit and that of the faulty circuit. However, if the output sequence starting from an undefined state is the same as the output sequence starting from some of the defined states, then the faulty circuit may not be distinguished from the fault free circuit. Since test generation for such faults is failed, it is desirable that undefined states can be distinguished from the other states.

We propose the procedure to modify state transition tables by adding new transitions about undefined states to a given state transition table in order that undefined states can be distinguished from defined states.

Figure 1 shows an example of transitions added by our procedure 1. This state transition graph includes all state transitions described in Table 1. State s0, s1 and s2 are defined states. Since two bits are required to encode these states, the gate level circuit obtained by logic synthesis has four states. We can add transitions about one state, denoted by s_U in the figure, without changing the function.

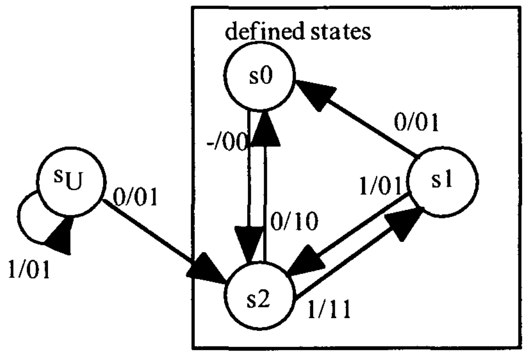

Figure 1. An example of the state transition modified by Procedure 1.

First we find the defined state S_R that has most incoming transitions. We add the transitions from undefined states to only this selected state in order to avoid making the circuit uninitializable. In this example, we set S_R be state s2 that has two incoming edges. Next, we select the input vector and the output vector that are unused or most rarely used in the transitions into the selected defined state. In this example, we select input 0 and output 01. And then the transition is added in order that the selected input vector brings the circuit from the undefined state into the defined state and outputs the selected output vector. In this example, transition (0, s_U, s2, 11) is added. The transitions for the other input vectors are also added to bring the circuit from s_U to s_U as shown in Figure 1.

The undefined states can be distinguished from the defined states in the state transitions modified by this procedure. The detail of this procedure is described in Section 4.

3.2 A method to avoid making undefined states unreachable

We propose another procedure to modify state transition tables to reduce the number of unreachable states. An unreachable state is a state that has no incoming transitions.

It has been known that many unreachable states exist for large sequential circuits[1]. This is because logic synthesis tools tend to make undefined states unreachable. However, it has been reported that existence of many unreachable states causes test generation difficult[3]. Since there exist no input sequences to bring a state of a circuit to an unreachable state, test generation always fails for the faults that require setting the state of the circuit to an unreachable state.

To avoid making undefined states unreachable, we change some transitions between two defined states and make undefined state reachable from the defined states.

Figure 2 shows an example to describe the procedure. This state transition graph is also derived from Table 1. The transitions of undefined state s_U are determined as follows.

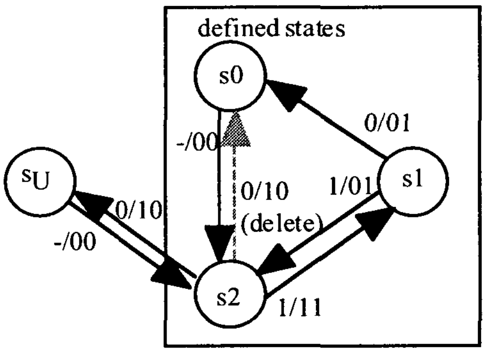

Figure 2. An example of the state transition modified by Procedure 2.

First, we find the defined state S_R that has most incoming transitions. Next, we find another defined state S_{R2} that has most outgoing transitions among the states that has a transition from state S_R. In this example, we set S_R = s2 and S_{R2} = s0. And then the transition from S_R to S_{R2} is replaced by the transition from S_R to an undefined state. We determined S_{R2} as the state with many incoming transitions to avoid that the replacement of the transition makes the defined state unreachable.

The transitions from the undefined state are added by duplicating the outgoing transitions of S_{R2}. This is necessary to keep the original function.

The undefined states are not unreachable in the state transitions modified by this procedure. The detail of this procedure is described in Section 4.

4. Procedures

In this section, we propose two procedures to add transitions to a state table. Procedure 1 adds transitions only from undefined states in order to make the difference between the output sequences starting from an undefined state and the output sequences starting from a defined state. Procedure 2 modifies some transitions in the defined state tables without changing its function in order to avoid making undefined states unreachable.

We assume that the input vectors and the output vectors are described in binary codes and state assignment for defined states in a given state transition table is obtained.

4.1 Procedure 1

In Procedure 1, transitions from undefined state are added to distinguish undefined states from the defined states.

[Procedure 1]

Step 1) Find binary codes unused in the given state transition table

Step 2) Count the number of incoming transitions for each state

Step 3) Set S_R be the state that has most incoming transitions

Step 4) Enumerate the input vectors and the output vectors used in the transitions to S_R

Step 5) Set I_R be the input vector that is unused or most rarely used in the transitions to S_R

Step 6) Set O_R be the output vector that is unused or most rarely used in the transitions to S_R

Step 7) For each undefined state (unused binary code) S_U, add the following two types of transitions: the transition from S_U to S_R (I_R, S_U, S_R, O_R) and the transitions (I, S_R, S_R, O_R) for $I \notin I_R$

In Step 2, we do not count the number of self loops since the state with most incoming transitions from the other states is likely to be a reset state.

4.1 Procedure 2:

In Procedure 2, one of the transitions between two defined states is changed to the transition from the defined state to an undefined state to avoid making undefined state unreachable. The transitions from the undefined state are also added to keep the original function.

[Procedure 2]

Step 1) Find binary codes unused in the given state transition table

Step 2) Select one of the undefined state (unused binary code) S_U

Step 3) Count the number of incoming transitions for each state, and set S_R be the state that has most incoming transitions

Step 4) Count the number of incoming transitions for each state that has incoming transitions from S_R

Step 5) Set S_{R2} be the state with most incoming transitions among the states that has incoming transitions from S_R

Step 6) Replace one of the transition from S_R to S_{R2}, $T_r = (I_r, S_R, S_{R2}, O_r)$, with the transition from S_R to S_U, $T_r' = (I_r, S_R, S_U, O_r)$

Step 7) Add the transitions from S_U to S_{R2}, $T_a' = (I_a, S_U, S_a, O_a)$, that is determined from each transitions from S_{R2}, $T_a = (I_a, S_{R2}, S_a, O_a)$

Step 8) If an undefined state remains unprocessed, goto 2.

In Step 3 and 4, we do not count the self loops to avoid the defined state become unreachable from the other states by setting the state with many self loops be S_R.

5. Experimental Results

We apply the procedures to MCNC benchmarks using the following steps.

[Overall Procedure]

Step 1) Apply state assignment program jedi, which is included in logic synthesis system SIS[4], to each state transition table

Step 2) Apply Procedure 1 or 2 to the obtained state table with state code

Step 3) Apply logic synthesis to the modified state table using SIS[4]

Step 4) Apply sequential ATPG tool HITEC[5] to the synthesized circuit

Table 2 shows the statistics of the original MCNC benchmarks. The column *input* and *output* show the number of inputs and outputs, respectively. The column *transition* shows the number of transitions described in the state transition table. The column *defined state* shows the number of states given in the state transition table. The column *undefined state* shows the number of undefined states given by $2^{\lceil \log_2 d \rceil} - d$ where d is the number of the defined states. Some benchmarks are removed from the table since they has 2^n states or the application of the logic synthesis failed.

Table 3 shows the experimental results. We estimate the effectiveness of the procedures by stuck-at fault coverage and fault efficiency of the test generation. The column *fault* shows the number of total stuck-at faults in the gate level circuits obtained by logic synthesis. The fault efficiency and the fault coverage obtained by HITEC are shown in the table. The column *Orig* shows the results obtained for the circuits synthesized from the original benchmark. The columns *Proc1* and *Proc2* are the results obtained for the circuits synthesized from the modified state transition tables by Procedure 1 and by Procedure 2, respectively.

The number shown by the italics mean that higher coverage or efficiency is obtained for the modified circuits than for the original benchmarks. For the circuits with high fault coverage, the application of our procedure causes the lower fault coverage than that of the original circuits. One of the explanations is that the additional transitions make a circuit larger and hence the modified circuit is hard to test. However, for the circuits with lower fault coverage such as bbara and dk14, fault coverage was increased by our procedure.

6. Conclusion

We presented two procedures to add transitions of undefined states to state transition tables to enhance testability. One is the procedure that adds the transitions from undefined states to the defined state. Another is the procedure that changes one of the transitions between defined states to avoid making undefined states unreachable.

As shown in the experimental results, for some of the benchmarks we could obtain higher fault coverage. Our procedure is simple enough to apply to large circuits and can enhance fault coverage for the circuits for which the fault coverage is low. However for the circuits with high fault coverage, our procedure decreases the fault coverage then is not effective. This is because the selection of the transition from the undefined state is not appropriate. To find better selection of these transitions is still left as a future work.

References

[1]H. Yotsuyanagi and K. Kinoshita, "Finding Unreachable States of Sequential Circuits," Technology Reports of the Osaka University, vol. 49, no. 2344, pp. 49-55, Apr. 1999.

[2]H. Yotsuyanagi and K. Kinoshita, "Undetectable Fault Removal of Sequential Circuits Based on Unreachable States," Proc. 16th VLSI Test Symp., pp. 176-181, Apr. 1998.

[3]T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski, "A Complexity Analysis of Sequential ATPG," IEEE Trans. CAD, vol. 15, no. 11, pp. 1409-1423, Nov. 1996.

[4]E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Electronics Reserch Laboratory Memorandum No. UCB/ERL M92/41, 1992.

[5]T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," Proc. of European Design Automation Conference, pp. 132-135, May 1994.

Table 2. MCNC benchmarks

circuit	input	output	transition	defined state	undefined state
bbara	4	2	60	10	6
bbtas	2	2	24	6	2
beecount	3	4	28	7	1
dk14	3	5	56	7	1
dk16	2	3	108	27	5
dk27	1	2	14	7	1
dk512	1	3	30	15	1
ex1	9	19	138	20	12
keyb	7	2	170	19	13
s1	8	6	107	20	12
s208	11	2	153	18	14
s27	4	1	34	6	2
s386	7	7	64	13	3
s420	19	2	137	18	14
s820	18	19	232	25	7
s832	18	19	245	25	7

Table 3. The experimental results for benchmarks

	fault			efficiency			coverage		
circuit	Orig	Proc1	Proc2	Orig	Proc1	Proc2	Orig	Proc1	Proc2
bbara	136	165	199	0.809	*0.849*	*0.663*	0.052	*0.709*	0.015
bbtas	61	72	85	0.885	*0.889*	*0.918*	0.033	0.014	*0.035*
beecount	94	121	106	1.000	1.000	1.000	1.000	0.926	0.991
dk14	185	202	202	1.000	1.000	1.000	0.022	*0.901*	*0.040*
dk16	451	479	528	1.000	0.789	1.000	0.000	0.000	0.000
dk27	61	74	75	1.000	1.000	1.000	0.000	0.000	0.000
dk512	127	131	123	1.000	1.000	1.000	0.000	0.000	0.000
ex1	484	498	565	1.000	1.000	0.752	0.060	*0.070*	*0.062*
keyb	381	406	482	0.987	*0.998*	*0.994*	0.958	0.941	0.927
s1	296	347	345	1.000	1.000	1.000	0.014	0.006	0.006
s208	206	226	245	1.000	1.000	1.000	0.927	0.898	0.918
s27	37	51	59	1.000	1.000	1.000	0.919	0.843	0.881
s386	228	260	265	1.000	1.000	1.000	0.952	0.069	0.049
s420	191	217	265	1.000	1.000	1.000	0.833	0.788	*0.834*
s820	573	597	594	0.993	*1.000*	*1.000*	0.962	*0.965*	*0.966*
s832	580	633	629	0.998	*1.000*	*1.000*	0.986	0.984	0.979