Chapter 1

Introduction

1.1 Introduction

In recent years, testing is more difficult because of increasing high complexity of integrated circuit. Many testing and design-for-testability methods for digital sequential circuits have been proposed and developed [1]. However, these methods usually involve long test generation time. A test methodology, Oscillation Ring Test (OR-Test) was proposed in the papers [2][3] to test the stuck-at and path delay faults for the sequential circuit. The methodology re-configures the feedback paths of the circuit under test (CUT) to be many oscillation rings (ORs) under the test mode. Under appropriate patterns applied to inputs of the CUT, ORs will oscillate. By observing the oscillation at the output of the CUT, we can tell that if the circuit is working properly. The methodology is very simple and effective since it needs not many patterns.

1.2 Characteristic of Tested Asynchronous Circuits

Digital sequential circuits can be divided into categories of the synchronous and the asynchronous circuit. The synchronous sequential circuit is synchronized with one or more clock signals. It is generally simple to be designed with only need to take clock skew and the worse case delay into account. For the asynchronous circuit, it dose not have a clock and the circuit works under a carefully designed timing consideration. As long as an input variable changes the value, the circuit will have response to that. It has the advantages of relatively high speed, low power consumption and no problem with the clock skew. However, it is more difficult to be designed since it needs to be considered the timing problems such as races and hazards [4]-[8].

To apply the OR-Test to the asynchronous circuit, we have assumed that the asynchronous circuit has the following characteristics.

- It operates in the fundamental mode
  - No input signals changed until the circuit is stable
  - Only one input signal changed at a time
- Its architecture is:
  - Combination circuit uses the static feedbacks used to maintain states
  - No explicit storage elements such as latches, flip-flops or C-elements
- It has a predetermined time period to give input vectors

1.3 Outline of The Thesis

In Chapter 2, we illustrate this OR-Test methodology and give the basic architecture of OR-Test for a asynchronous circuit. And we illustrate what kinds of fault detections to be used to detected stuck-at faults. Finally, we illustrate how to form oscillation rings and generate state patterns from the state transition table.

In Chapter 3, we show the overall procedure for applying the OR-Test for the asynchronous circuit. Examples are used to illustrate these procedures.

In Chapter 4, we give the experimental result on several benchmark circuits by applying this OR-Test methodology.

In Chapter 5, we propose a procedure to reassign the state for an asynchronous circuit for

OR-Test to improve its testability. An example is also given to demonstrate the procedure.

In Chapter 6, we give conclusion for the thesis.

Chapter 2
Application of OR-Test Methodology to the Asynchronous Circuit

2.1 OR-Test Methodology

Figure 2.1 is the architecture of the OR-Test applied to the synchronous sequential circuit, where the oscillation ring is formed, to detect stuck-at faults. If there is a stuck-at fault on the path of the oscillation ring or the sensitized path the oscillation will stop. So, a fault will be detected.

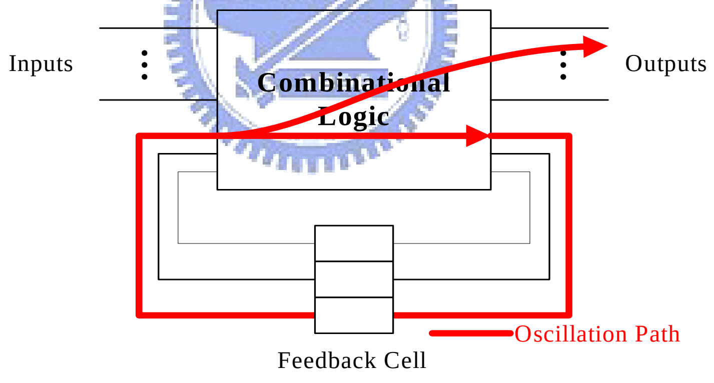
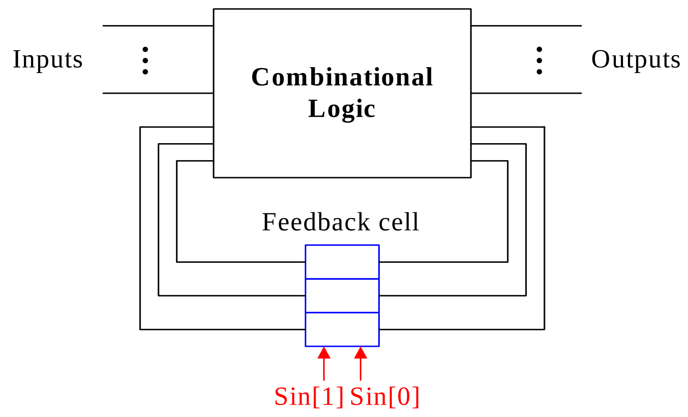


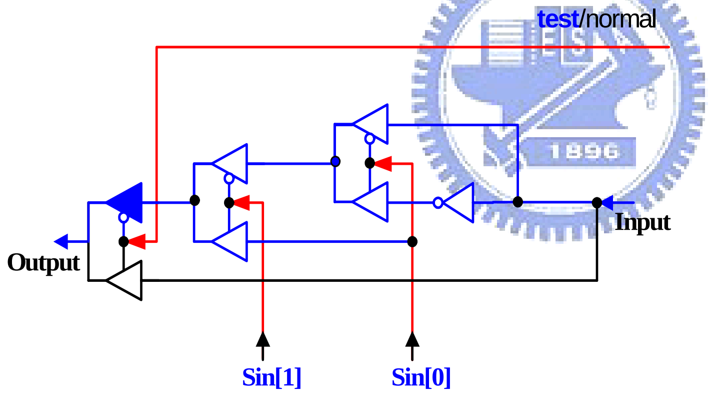
Figure 2.1 Simple illustration of OR-Test

2.2 Basic Architecture of OR-Test

The basic architecture of OR-Test is shown in Figure 2.2(a). In the architecture, feedback cells are added to the feedback paths. These feedback cells can be controlled to give the state patterns to achieve the oscillation condition. These state cells are shown in Figure 2.2(b). Two control pins are to control the state of the feedback cell and one common pin is added to control test/normal mode for OR-Test. An encoder is used to encode the control signal which will be explained later.



(a)



(b)

Sin[1]	Sin[0]	operation
0	0	Bypass
0	1	INV
1	0	Hold 0
1	1	Hold 1

(c)

Figure 2.2 (a) Basic architecture of the asynchronous circuit under test; (b) Feedback cell; (c) Operation of control signals

2.3 Fault Detection

Oscillation rings can be classified into two cases by its detection: **Case A** and **Case B**.

Case A Fault Detection

Case A fault detection is that oscillating signals of oscillation rings can be propagated to the output of the CUT. Stuck-at faults can be detected by observing the output if the output signals stop oscillating. An example circuit of such type is shown in Figure 2.3 which is in the fault-free condition. Under the input: XY = 01, the OR closed loop (c→ d→ e) will be oscillation, and the oscillating signal is propagated to the output Z. However, when a stuck-at-1 fault occurs at line ***c*** as shown in Figure 2.4, the output Z will stop oscillating and stay at 1. Thus, this fault is detected.

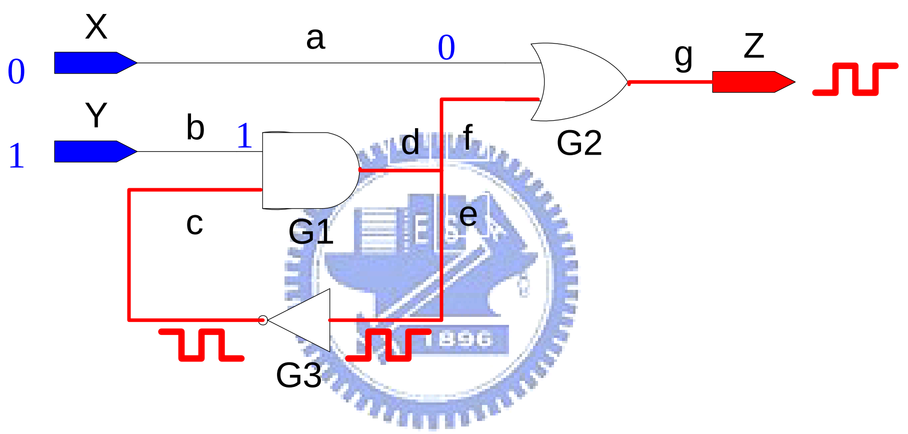


Figure 2.3 A fault-free circuit of Case A

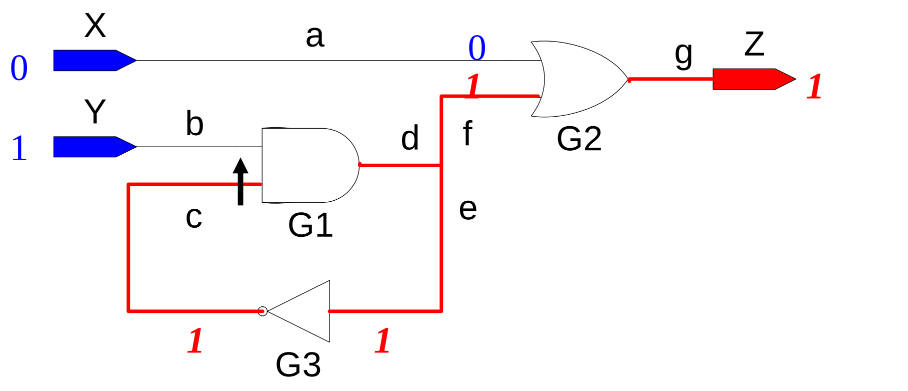


Figure 2.4 A faulty circuit of Case A

Similarly, c↑, c↓, d↑, d↓, e↑, e↓ faults on the closed loop, f↑, f↓, g↑, g↓ faults on lines used to propagate the oscillation signal into the output Z and a↑, b↓ faults on the lines used to sensitize the oscillation ring will be also detected.

Case B Fault Detection

Case B fault detection is that oscillating signals of oscillation rings are not propagated to the output at the fault free case but the output oscillates when stuck-at faults exist in the CUT. An example is shown in Figure 2.5. The circuit is the same as that in Figure 2.3. When input X changes from 0 to 1, the oscillation signal can not be propagated to output Z. However, if a stuck-at-0 fault exists at line ***a*** as shown in Figure 2.6, output Z will be oscillating. Thus the fault is detected.

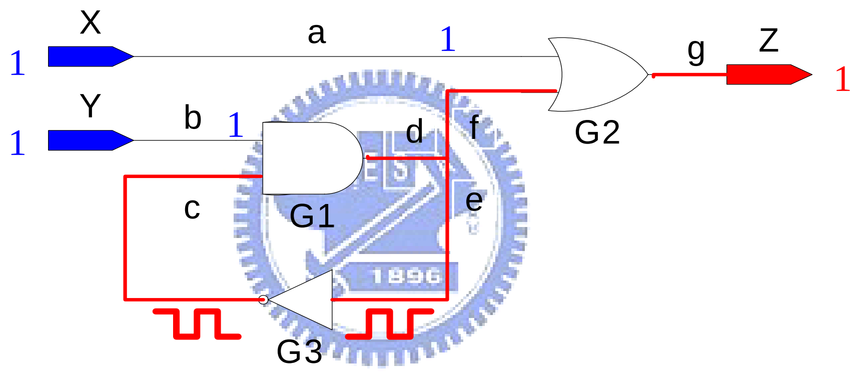


Figure 2.5 A fault-free circuit of Case B

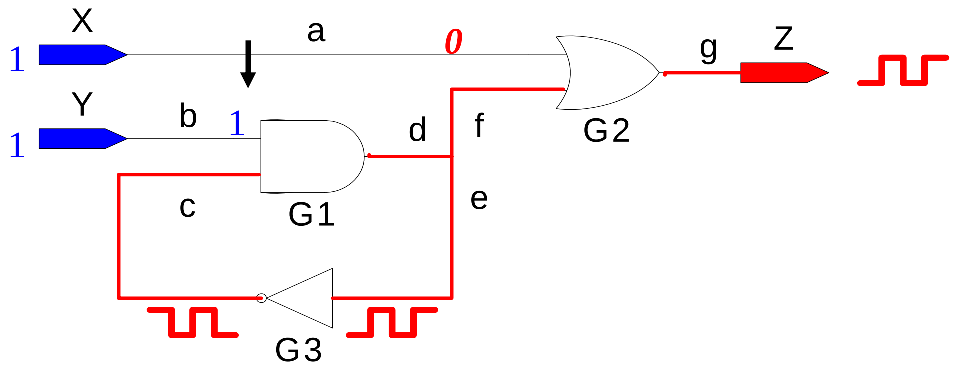


Figure 2.6 A faulty circuit of Case B

Both two cases have their advantages. Oscillation rings whose detection belongs to **Case A** can detect stuck-at faults on oscillation closed loops or on the paths that are used to propagate the oscillation signal to the output. Therefore, it usually can detect more faults than **Case B**. But oscillation rings whose detection belong to **Case B** can detect stuck-at faults which result in the propagation of the inner oscillation signal into the output and also can be used to detect stuck-at faults of some special circuits.

2.4 Formation of Oscillation Rings

This section introduces how oscillation rings are formed from the state transition table. It is assumed that the state transition table is available for the CUT from the circuit designer. From the transition table, state sets whose hamming distances are all 1 are first obtained. The following will show how to build the oscillation relation of these state sets and illustrate several tables to be used to generate state patterns.

2.4.1 Relation of State Sets for Oscillation Rings

Table 2.1 is a state transition table which is used to explain the procedure. Assume that the stable state **a** at XY = 01 is chosen first. In order to avoid race conditions in OR-Test, only present state **b** and **d** whose hamming distances are both 1 with respect to state **a at** the same input are considered.

Table 2.1 A state transition table example

PS	NS				OUT			
Q_2Q_1	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a 00	a 00	a 00	d 10	b 01	0	0	1	1
b 01	a 00	b 01	b 01	b 01	1	1	1	1
c 11	b 01	b 01	b 01	b 01	1	1	1	1
d 10	b 01	c 11	d 10	a 00	1	1	1	1

Consider state **b** first. For the same input XY = 01, the next state of present state **b** is **b**, therefore, it is a stable state. If we add an inverter to the feedback path of state variable $\mathbf{Q_1}$, the oscillation ring of state variable $\mathbf{Q_1}$ will be formed as shown in Figure 2.7(a). So, state variable $\mathbf{Q_1}$ of state **a** and **b** will interchange mutually with value 0 and 1. Since the output value of state **b** is opposite to state **a**, the detection of the oscillation ring belongs to **Case A**. Thus, the oscillation signal can be propagated into the output as shown in Figure 2.7(b).

		state	Q_2	Q_1
1st	Present state	**a**	0	0
	Next state	**a**	0	0
2nd	Present state	**b**	0	1
	Next state	**b**	0	1
			Bypass	**INV**

Figure 2.7 (a) Oscillation relation of state set [a, b]

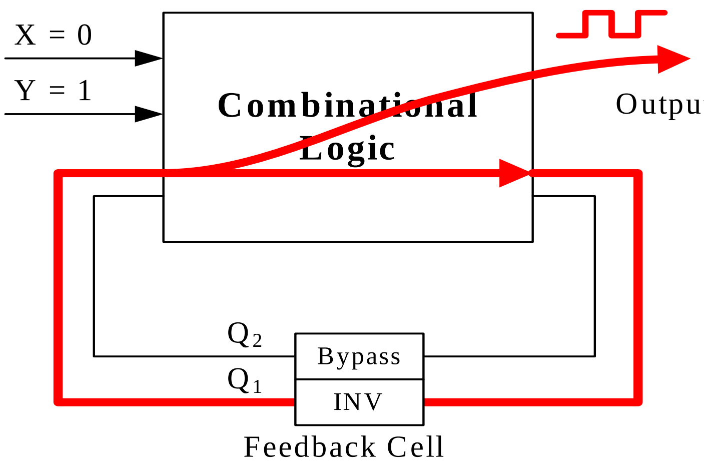


Figure 2.7 (b) Circuit diagram of state set [a, b]

Now consider state **d**. At the same input XY = 01, the next state of state **d** is **c**. Similarly, if we add an inverter to the feedback path of state variable **Q_2** and make the feedback path of state variable **Q_1** be always 0(This is Hold 0) , the oscillation ring of state variable **Q_2** will be formed as shown in Figure 2.8. The detection of the oscillation ring also belongs to **Case A**.

		state	Q_2	Q_1
1st	Present state	**a**	0	0
	Next state	**a**	0	0
2nd	Present state	**d**	1	0
	Next state	**c**	1	1
			INV	**Hold 0**

Figure 2.8 (a) Oscillation relation of state set [a, d]

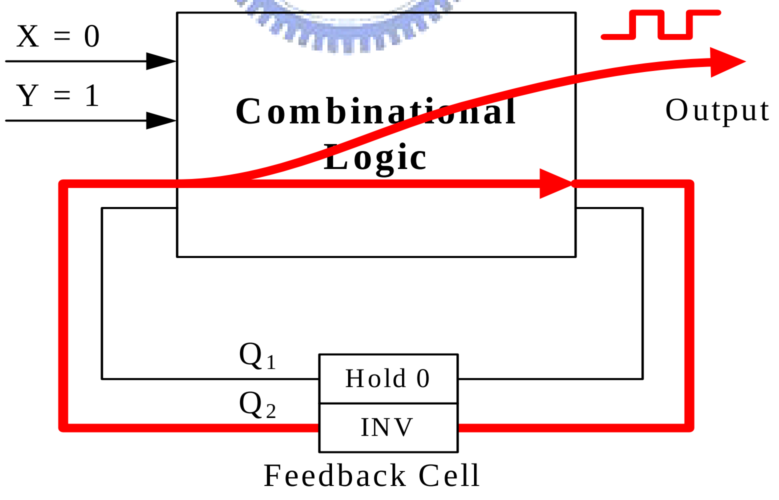


Figure 2.8 (b) Circuit diagram of state set [a, d]

2.4.2 True Table of State Variable and Function Table of Feedback Cell

Through the state transition table and the constraint of hamming distance = 1 between state sets, we can find total state sets which may be generated oscillation rings with valid states, and we use the true table of state variable and the function table of feedback cells to generate state patterns. As shown in Table 2.2(a), the true table of state variable is used to mark the change of state variables. For example, if the transition of the bit is from 0 to 0, we mark the transition **Low**. And Table 2.2(b) shows that the function table of the feedback cell is used to generate state patterns through the true table of the state variable. However, we should pay attention to the condition **Fail**. We can not provide simple logic circuit to achieve the condition **Fail**. Therefore, if a state pattern includes the condition **Fail**, it is invalid. In the next section, we will illustrate how to complete the function table of feedback cell.

Table 2.2 (a) True table of the state variable;

(b) Function table of the feedback cell

Change of Bit	OP Value
0 -> 0	Low
0 -> 1	Rising
1 -> 0	Falling
1 -> 1	High

(a)

OP Value		2nd			
		L	**H**	**R**	**F**
1st	**L**	Bypass	INV	Hold 0	Fail
	H	INV	Bypass	Fail	Hold 1

(b)

2.5 State Pattern

2.5.1 Type 1: Bypass State

Bypass state indicates that the function which is used to achieve the transition of present states without any logic devices added into the feedback cell. As shown in Figure 2.9(a) and Figure 2.10, state bits of two present states are both 0 and next states are also 0, too. Thus, nothing needs to be added to satisfy the condition that make 1st next state equal to 2nd present state and 2nd next state equal to 1st present state. This is **Bypass** state. In the same way, operation set {H, H} is the same as shown in Figure 2.9(b).

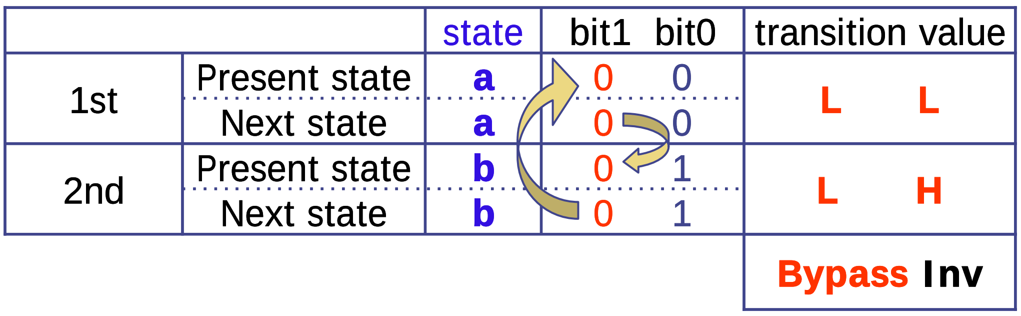
PS	NS	OP		MSR Cell		PS
0	0	L	->	Bypass	->	0
0	0	L	->		->	0

(a)

PS	NS	OP		MSR Cell		PS
1	1	H	->	Bypass	->	1
1	1	H	->		->	1

(b)

Figure 2.9 (a) Operation set {L, L}; (b) Operation set {H, H}



		state	bit1	bit0	transition value
1st	Present state	a	0	0	L L
	Next state	a	0	0	
2nd	Present state	b	0	1	L H
	Next state	b	0	1	
					Bypass Inv

Figure 2.10 Example of operation set {L, L}

2.5.2 Type 2: INV State

INV state indicates that an inverse operation should be added for the feedback cell. As shown in Figure 2.11(a) and Figure 2.12, 1st present state and next state are both 0 and 2nd are both 1. Thus, we just add an inverter to make two present states can interchange mutually with 0 and 1. This is **INV** state. In the same way, operation set {H, L} is the same as shown in Figure 2.11(b).

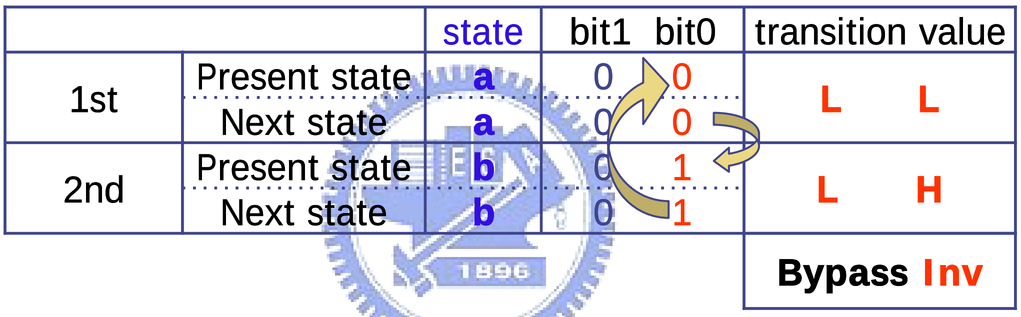
PS	NS	OP		MSR Cell		PS
0	0	L	->	INV	->	1
1	1	H	->		->	0

(a)

PS	NS	OP		MSR Cell		PS
1	1	H	->	INV	->	0
0	0	L	->		->	1

(b)

Figure 2.11 (a) Operation set {L, H}; (b) Operation set {H, L}



		state	bit1	bit0	transition value	
1st	Present state	a	0	0	L	L
	Next state	a	0	0		
2nd	Present state	b	0	1	L	H
	Next state	b	0	1		
					Bypass Inv	

Figure 2.12 Example of operation set {L, H}

2.5.3 Type 3: Hold State

Hold state indicates that no matter what the next state value is, the feedback cell is a fixed value. As shown in Figure 2.13(a) and Figure 2.14, we must make 1st next state equal to 2nd present state and 2nd next state equal to 1st present state. But **Bypass** state and **INV** state both do not apply to this condition. Thus, we only give the feedback cell a 0. That is **Hold 0** state. In the same way, operation set {H, F} is **Hold 1** state like **Hold 0** as shown in Figure 2.13(b).

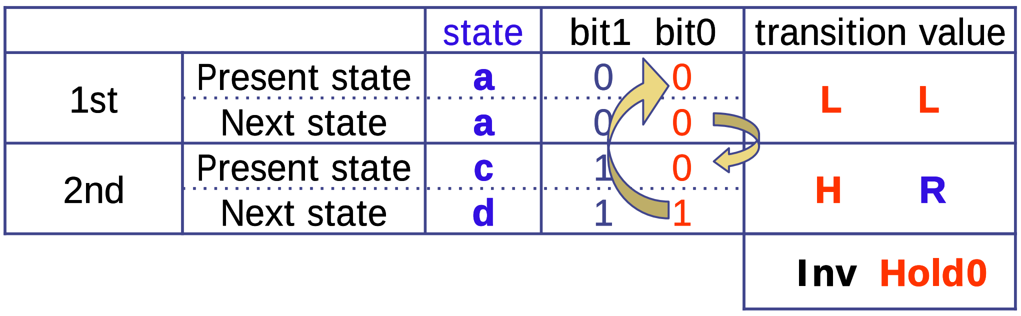
PS	NS	OP		MSR Cell		PS
0	0	L	->	Hold 0	->	0
0	1	R	->		->	0

(a)

PS	NS	OP		MSR Cell		PS
1	1	H	->	Hold 1	->	1
1	0	F	->		->	1

(b)

Figure 2.13 (a) Operation set {L, R}; (b) Operation set {H, F}



		state	bit1	bit0	transition value
1st	Present state	a	0	0	L L
	Next state	a	0	0	
2nd	Present state	c	1	0	H R
	Next state	d	1	1	
					Inv Hold0

Figure 2.14 Example of operation set {L, R}

2.5.4 Type 4: Fail State

Fail state indicates that no matter what we add into the feedback cell, the condition can not be achieved. As shown in Figure 2.15(a) and Figure 2.16, we must make 1st next state equal to 2nd present state and 2nd next state equal to 1st present state. But **Bypass** state, **INV** state and **Hold** state are all un-applicable for this condition. Thus, this state is an invalid state. This is **Fail** state. In the same way, operation set {H, R} is also **Fail** state as shown in Figure 2.15(b).

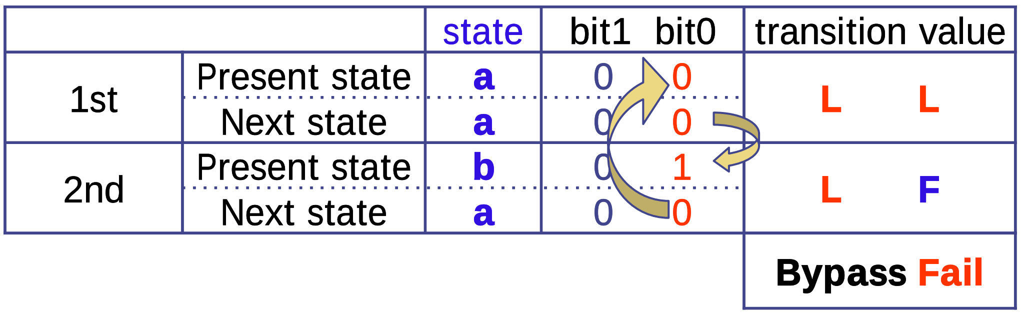
PS	NS	OP		MSR Cell		PS
0	0	L	->	Fail	->	1
1	0	F	->		->	0

(a)

PS	NS	OP		MSR Cell		PS
1	1	H	->	Fail	->	0
0	1	R	->		->	1

(b)

Figure 2.15 (a) Operation set {L, F}; (b) Operation set {H, R}



		state	bit1	bit0	transition value
1st	Present state	a	0	0	L L
	Next state	a	0	0	
2nd	Present state	b	0	1	L F
	Next state	a	0	0	
					Bypass Fail

Figure 2.16 Example of operation set {L, F}

2.6 Condition of State Patterns

Through the detailed illustration about types of state patterns added into the feedback cells in the last section, we illustrate two conditions of state patterns in this section. The first condition, which is termed as Condition Ⅰ, is composed of only one **INV** state on a feedback cell and **Bypass** states on all other cells as shown in Figure 2.17(a). The second condition, which is termed as Condition Ⅱ, is composed of only one **INV** state on a feedback cell and **Bypass** states and **Hold** states on all other cells as shown in Figure 2.17(b).

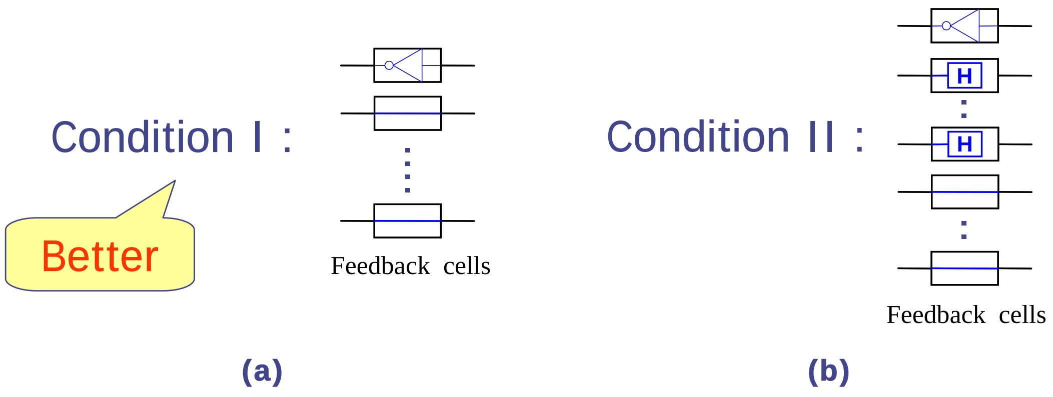


Figure 2.17 (a) Condition Ⅰ of state patterns (b) Condition Ⅱ of state patterns

Condition Ⅰ is better than Condition Ⅱ on the capability to detect stuck-at faults. As shown in Figure 2.18, the s-a-1 fault on the wire **W** can not be detected by Condition Ⅱ of state patterns. No matter what the value of the wire **W** is, Cell 1 always provides the value 1 in Figure 2.18(a). However, Cell 1 will be affected by the s-a-1 fault on the wire W in Figure 2.18(b). Thus, the s-a-1 fault on the wire **W** may be detected by Condition Ⅰ of state patterns.

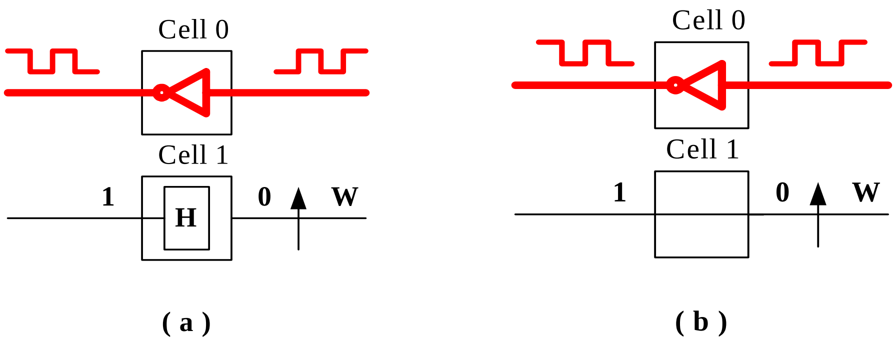


Figure 2.18 (a) W↑ in Condition Ⅱ (b) W↑ in Condition Ⅰ

Chapter 3

Procedure to Generate OR-Test Test Patterns

3.1 Flow of OR-Test by State Transition Table

OR-Test for asynchronous circuits by state transition table can be divided into two parts. First part is to obtain oscillation rings with their state patterns and input sequences from the state transition table, then to use synthesis tools to synthesize the state transition table into the circuit netlist in order to make the procedure more convenient. And second part is fault simulation. Figure 3.1 is the total flow chart of OR-Test by the state transition table.

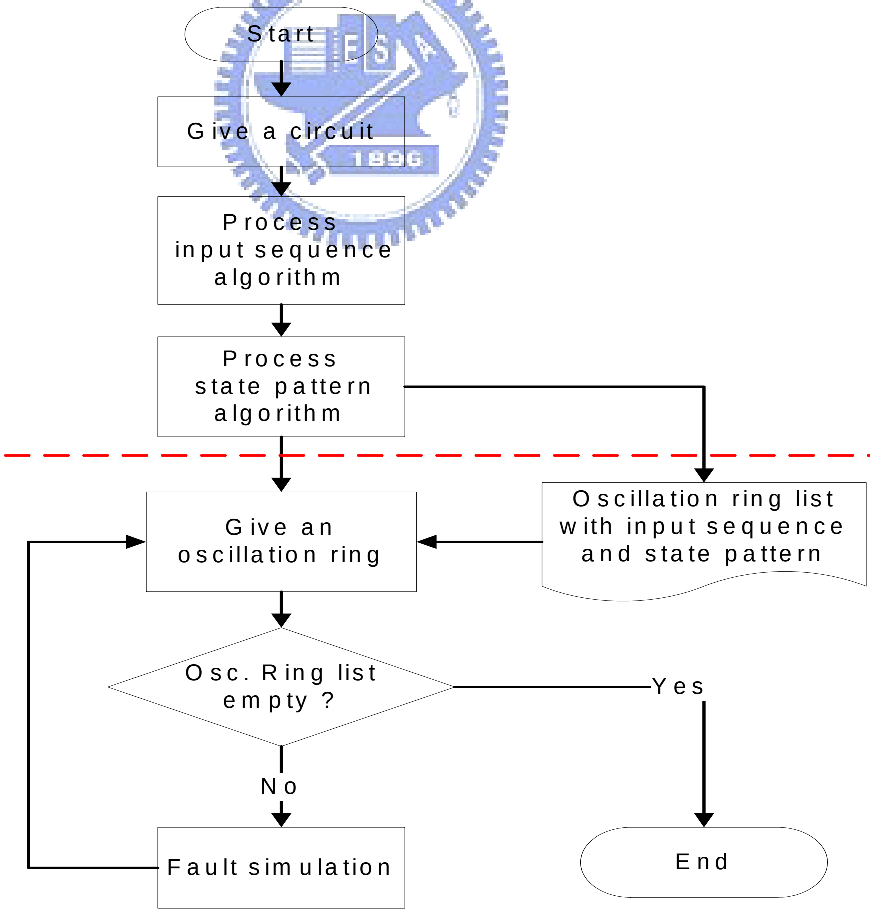

Start
Give a circuit
Process input sequence algorithm
Process state pattern algorithm
Oscillation ring list with input sequence and state pattern
Give an oscillation ring
Osc. Ring list empty ?
Yes
No
Fault simulation
End

Figure 3.1 Flow chart of OR-Test by state transition table

For this procedure, the data and netlist of the circuit is given as the input and the output will be the final fault coverage. The following sections will illustrate how to find state patterns and input sequences of oscillation rings and to do fault simulation.

3.2 Algorithm of Input Sequence Finding

Before doing OR-Test for asynchronous circuits, the most important thing is to find all input sequences of states. For the asynchronous circuit, invalid states are unstable states and only valid states have input sequences. An input sequence can make the inner state reach the starting state of an oscillation ring. So we want to find valid states and their input sequences.

As shown in Figure 3.2, we have the initial state and input first. And we must decide the length of input sequences. Then, use binary search to find them. Only one of input variables can be changed one time. When input changes, the state maybe changes. Until the new state is stable, diagnose if the new state and its corresponding input can be found in the valid state list. If not, save the state and input into the valid state list and continue to change input. On the contrary, if the state can be found in the list, diagnose if the length of the state found now is shorter than that in the list. If yes, replace that in the list with the state found now. If no, go back to the last node and search another path. Until all the paths are looked for, the valid state list with input sequences can be obtained and the maximum length of input sequences doesn't exceed that we decided before.

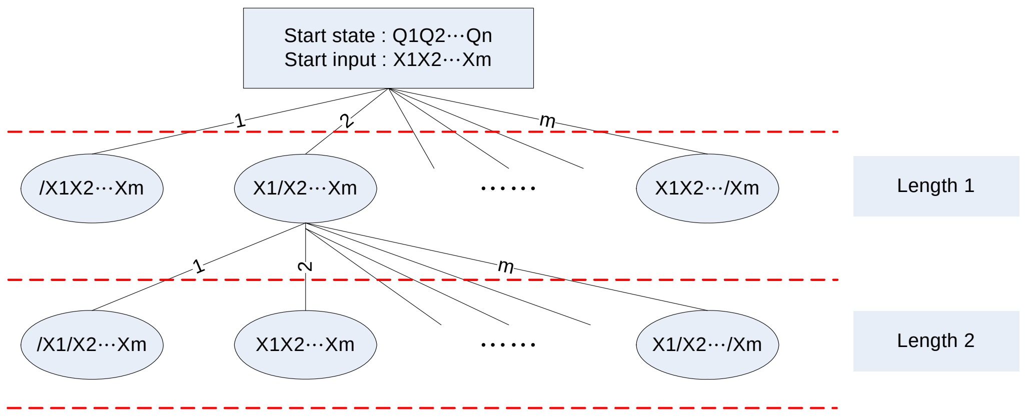


Figure 3.2 Illustration of finding input sequence

An example is shown in Table 3.1. First, the inputs XY and the state Q_2Q_1 are initially provided 00 and 00, respectively. The max length of input sequence is 2.

Table 3.1 Example of state transition table

PS Q_2Q_1	NS XY=00	XY=01	XY=11	XY=10
a 00	[✓ a] 00	[✓ a] 00	d 10	b 01
b 01	a 00	[✓ b] 01	[✓ b] 01	[✓ b] 01
c 11	-	b 01	-	-
d 10	-	c 11	[✓ d] 10	a 00

Thus, we can get the binary search figure like Figure 3.3. The symbol, X, in Figure 3.3 means that the state at that input was found before. Nevertheless, if the length of input sequence of the state found now is shorter than before, the state before is replaced by that found now and continue to do binary search until all paths are looked for. Although the valid states in the example are not found totally, we can increase the max length until the total valid states are found. So it is dependent on the actual situation to decide the max length of input sequences. Very often, we do not have to obtain the total valid states but still can achieve a high fault coverage.

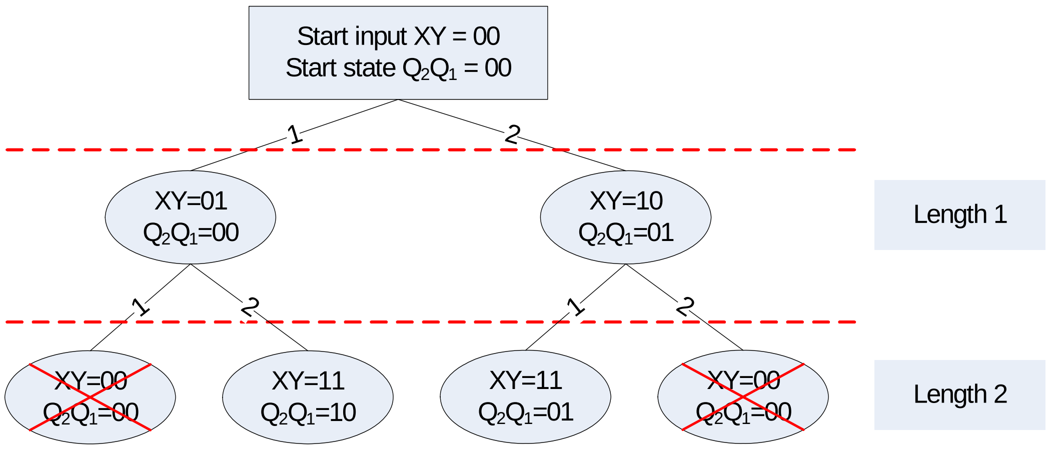


Figure 3.3 Binary search of Table 3.1

3.3 Algorithm of State Pattern

When finding valid states with their input sequence, observe other states whose hamming distances with these valid states is one. This is in order to avoid race condition. Then, observe the different bit between the two present states. If the bit changes the value at the next state, no oscillation rings are produced. If two bits are the same, an oscillation ring can be formed.

Now see Table 3.2. Assume there is a valid state A ($Q_3Q_2Q_1Q_0$ = 0000) at one certain input. Thus, state B, C, D and E will be observed because they are the states whose hamming distance with state A is one. And the different bits of state B, C, D and E against state A will be added INV state into. If different bits change values at their next states, there are no oscillation rings generated. So, in Table 3.2, only state B can not be used to generate an oscillation ring with state A because of the transition of Q_0. The fault detection of the state set [A, C] belongs to **Case B** fault detection and state set [A, D], [A, E] belong to **Case A** fault detection.

Table 3.2 State transition table in certain input

Present State	Next state	Output
A 0000	0000	0
B 0001	0000	0
C 0010	0010	0
D 0100	0100	1
E 1000	1001	1

Through the example we demonstrated, we can understand how to find state patterns. So, Figure 3.4 is the basic flow chart of finding state patterns.

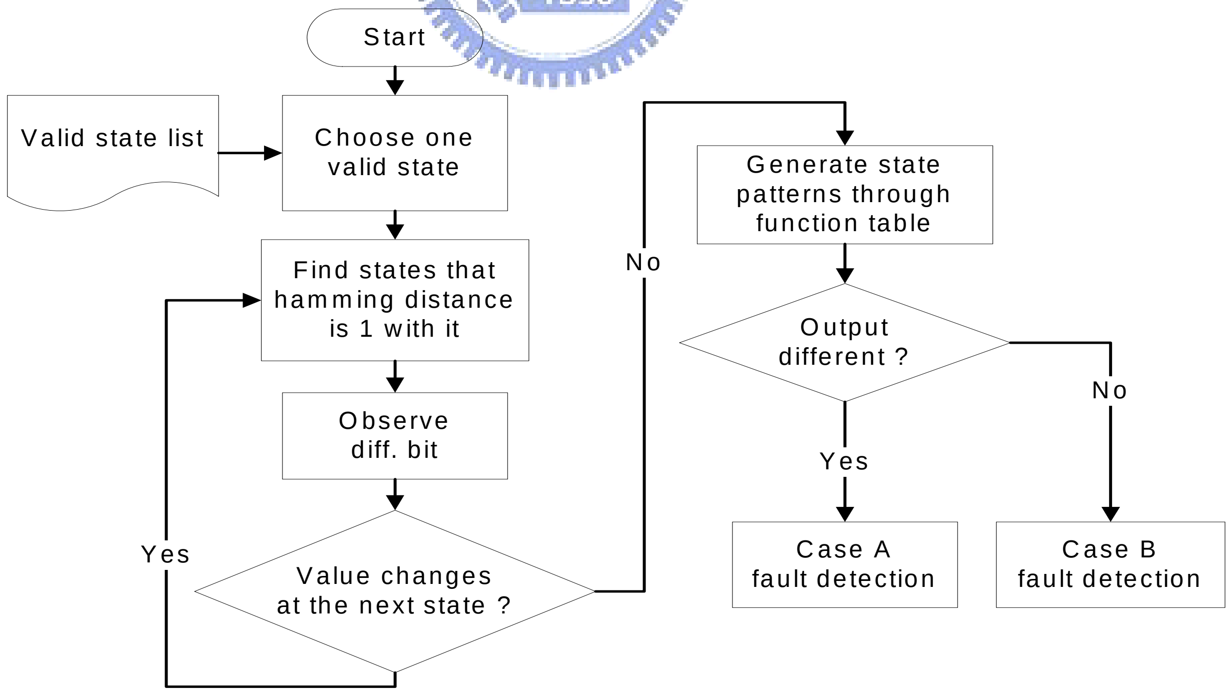


Figure 3.4 Flow chart of finding state patterns

3.4 Fault Simulation

After section 3.3, we have found oscillation rings with their state patterns and input sequences, and now we do fault simulation finally. Figure 3.5 is the simple flow chart of our fault simulation which is logic simulation written in C language.

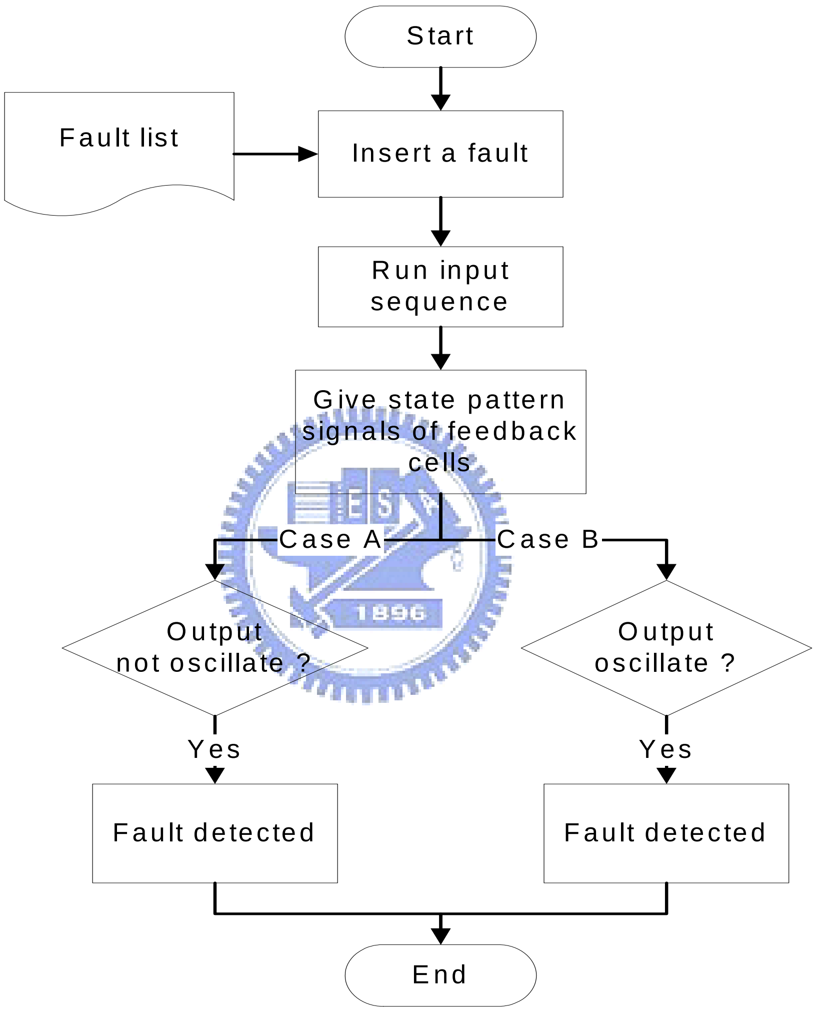


Figure 3.5 Simple flow chart of fault simulation

When doing fault simulation, the data of the oscillation ring will be given to the fault simulation program. Then, a fault is inserted into the circuit. Then, run input sequence of the fault-free circuit. When the predetermined cycle time goes by, give the next input vector of input sequence. Because of the inserted fault, race condition may happen when running input sequence. If more than two bits of state variables change at the same time, we presume that a race condition happens and the fault is undetected. After this, assign feedback cells with state patterns and start to do the OR-Test simulation. If the fault detection of the oscillation ring is **Case A**, and the output is stable, the fault is detected. Similarly, for the **Case B,** the output should be oscillating to detect the fault.

Chapter 4

Experimental Results on Benchmark Circuits

In the chapter, we apply the OR-test methodology derived above to several ISCAS89 benchmark circuits. These benchmark circuits are synchronous sequential circuits but in our experiment we took their circuits and then treat them as asychronous sequential circuits. The results are shown in Table 4.1. The circuits included are only small size circuits since for large size circuits, even the initial states can not be found and circuits are untestable by using this methodology. In the table, the # of gates, inputs, outputs, feedbacks, length of patterns, oscillation rings formed, and patterns, and fault coverages obtained for each circuit. For the column of the state patterns, the first number in the bracket is the number state patterns of OR rings of Condition Ⅰ and the second number of the bracket is the number of state patterns of OR rings of Condition Ⅱ.

From the table, we can see that for small size sequential circuits, a length of 3 for input sequence finding is enough for some circuits. In some circuits, we increased the length from 3 to 5 to obtain higher fault coverage. Thus, we can see that the fault coverage of the circuit, s3384, is changed from 75.99 % to 98.29 % when we change the length for input sequence finding from 3 to 5. Thus, if the input variables are more and more, we must give longer length to obtain higher fault coverage. And for circuits such as s208 and s838, the fault coverage are 0%, it is beause these circuits are counter-like cicuits whose state always changes during valid state searching. The number of valid states in those circuits are so few besides of the initial state, thus, no oscillation rings can be formed.

Table 4.1 Experimental results applying the OR Test methodology to benchmark circuits

Circuits	#Gates	#Inputs	#Outputs	#Feedbacks	#Length	#Osc rings	#Patterns	F.C.
s27	17	4	1	3	3	28	6 (2,4)	98.07%
s208	115	11	2	8	3	0	0	0
s349	185	9	11	15	3	1209	13 (12,1)	99.55%
s386	172	7	7	6	5	85	3 (3,0)	20.98%
s510	236	19	7	6	3	4940	10 (5,5)	29.51%
s713	447	35	23	19	3	22169	19 (3,16)	80.72%
s832	310	18	19	5	3	841	7 (4,3)	18.09%
s838	457	35	2	32	3	0	0	0
s953	440	16	23	29	3	1626	19 (1,18)	93.70%
s1269	624	18	10	37	3	13364	140 (24,116)	68.08%
s1269	624	18	10	37	5	193322	692 (24,668)	74.23%
s1494	661	8	19	6	3	68	4 (0,4)	18.51%
s1512	866	29	21	57	3	194012	64 (56,8)	53.60%
s3384	1911	43	26	183	3	239	27 (1,26)	75.99%
s3384	1911	43	26	183	5	3054	47 (1,46)	98.29%
s3330	1961	40	73	132	3	16285	19 (8,11)	30.90%
s4863	2495	49	16	104	3	0	0	0

Chapter 5

State Assignment to Improve OR-Testability for Asynchronous Circuits

In the prevous analysis, we have found that the state assignment is very important in forming the OR-Test rings which involve the fault coverage. In this chapter we will include some preliminary results on the state assignment of the asynchronous sequential circuits to improve their OR testability.

5.1 Assignment Flow

In the previous analysis, it was found that if the fault detections of the circuit belong to **Case B** or most state patterns belong to Condition Ⅱ, fault coverage will be bad. And if not all state variables have an oscillation signal in their closed loops, the detection result is also bad. Thus, we must assign states to make the circuit include more and more **Case A** fault detection or most state patterns belong to Condition Ⅰ.

The following is the steps to make the suitable state assignment.

Step 1. Find state sets which may be formed oscillation rings in the flow table

Step 2. Assign to make every state variables have oscillation rings with **Case A** fault detections. (**Case A & Condition** Ⅰ, first)

Step 3. Add extra states to avoid race condition

Step 4. If numbers of state variables are not enough to solve race condition, expand state variables one bit and return to **Step 2**

Step 5. Make don’t care items of states stable to generate extra oscillation rings with **Case A** fault detection

i. Find don’t care items of states which can generate oscillation rings with valid states

ii. Make them stable and their outputs opposite to outputs of valid states

Step 6. Fill in suitable values into outputs which maybe result in static hazards

Step 7. end

Through the last steps, we use state sets which may be built oscillation rings in the flow table and don’t care items of states to generate oscillation rings with **Case A** fault detections and Condition Ⅰ state patterns on the feedback path of every state variable. And we also can expand state variables one bit to increase fault coverage.

5.2 Example and Result Analysis

Now, we will give an example to illustrate the steps in the above section. As shown in the following, Table 5.1(a) is an original flow table and Table 5.1(b) is its original state transition table of an asychronous machine. With the state transition table, fault coverage is around 77 %. Now we reassign the state by using the steps we proposed in the last section. In Table 5.1(a), we sort state sets which may be generated ORs by cases of fault detection and conditions of state patterns as shown in Table 5.2. We want to assign the states with the highest priority detection condition as **Case A** & Condition Ⅰ.

Table 5.1(a) Original flow table of an asychronous circuit

PS	NS				OUT			
	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a	$\textcircled{a}$	$\textcircled{a}$	c	b	0	0	-	0
b	a	$\textcircled{b}$	$\textcircled{b}$	$\textcircled{b}$	0	0	0	0
c	-	b	$\textcircled{c}$	b	-	-	1	-

Table 5.1(b) Original state transition table of an asychronous circuit

PS	NS				OUT			
Q_2Q_1	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a 00	$\textcircled{a}$ 00	$\textcircled{a}$ 00	c 10	b 01	0	0	-	0
b 01	a 00	$\textcircled{b}$ 01	$\textcircled{b}$ 01	$\textcircled{b}$ 01	0	0	0	0
d 11	-	b 01	-	-	-	-	-	-
c 10	-	d 11	$\textcircled{c}$ 10	a 00	-	-	1	-

Table 5.2 State sets of oscillation rings of Table 5.1

	XY = 01	XY = 11
Case **A** & Condition Ⅰ	-	[b, c]
Case **A** & Condition Ⅱ	[a, c]	[a, b]
Case **B** & Condition Ⅰ	[a, b]	-

Through **Step 2**, we assign state b $(Q_2, Q_1) = (1, 0)$ and state c $(Q_2, Q_1) = (1, 1)$. Thus, the state variable Q_1 has the oscillation signal on its feedback path. Since the state set [a, b] can be also found to be in the condition of **Case B** and Condition Ⅰ, thus we choose this state and assign $(Q_2, Q_1) = (0, 0)$ and, for the input XY = 11, assign the output of state a (0, 0) to be 1, which is opposite in value to the output of state b (1, 0). Thus, state variable Q_2 also has the oscillation signal on its feedback path. And continue this to assign other state sets. In the asssignment, we find that state set [a, c] can not be achieved our OR condition since the hamming distance needed for assignment will exceed one. Thus, after **Step 2** , the result will be as shown in Table 5.3.

Table 5.3 State transition table after executing Step 2

PS	NS				OUT			
Q_2Q_1	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a 00	$\textcircled{a}$ 00	$\textcircled{a}$ 00	c 11	b 10	0	0	1	-
b 10	a 00	$\textcircled{b}$ 10	$\textcircled{b}$ 10	$\textcircled{b}$ 10	-	0	0	0
c 11	-	b 10	$\textcircled{c}$ 11	b 10	-	-	1	-

In Table 5.3, when the present state is state a (0, 0) at XY = 01 and input XY will be changed from 01 to 11, race condition may occur. Thus, **Step 3** is executed to avoid this condition. Hence, state d (0, 1) is added to avoid race condition and the modified table is as shown in Table 5.4.

Table 5.4 State transition table after executing Step 3

PS	NS				OUT			
Q_2Q_1	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a 00	$\textcircled{a}$ 00	$\textcircled{a}$ 00	d 01	b 10	0	0	1	-
b 10	a 00	$\textcircled{b}$ 10	$\textcircled{b}$ 10	$\textcircled{b}$ 10	-	0	0	0
c 11	-	b 10	$\textcircled{c}$ 11	b 10	-	-	1	-
d 01	-	-	c 11	-	-	-	1	-

For this example, no extra state variables are needed to avoid race condition and **Step 4** is omitted. We continue to execute **Step 5**. **Step 5** is to make don't care entries of states stable at certain inputs in order to generate extra oscillation rings with **Case A** fault detections and Condition Ⅰ state patterns. When state d (0, 1) is stable and its output value is 1 at XY = 00 and 01 as shown in Table 5.5, two oscillation rings with **Case A** fault detections and

Condition Ⅰ state patterns can be generated. Thus, this step is very useful to generate extra oscillation rings.

Table 5.5 State transition table after executing Step 5

PS	NS				OUT			
Q_2Q_1	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a 00	ⓐ 00	ⓐ 00	d 01	b 10	0	0	1	-
b 10	a 00	ⓑ 10	ⓑ 10	ⓑ 10	-	0	0	0
c 11	-	b 10	ⓒ 11	b 10	-	-	1	-
d 01	ⓓ 01	ⓓ 01	c 11	-	1	1	1	-

Finally, **Step 6** is to fill in suitable values in don’t care entries of the outputs to avoid static hazards. And the finial state transition table obtained is that as shown in Table 5.6.

Table 5.6 Final state transition table

PS	NS				OUT			
Q_2Q_1	XY=00	XY=01	XY=11	XY=10	XY=00	XY=01	XY=11	XY=10
a 00	ⓐ 00	ⓐ 00	d 01	b 10	0	0	1	0
b 10	a 00	ⓑ 10	ⓑ 10	ⓑ 10	0	0	0	0
c 11	-	b 10	ⓒ 11	b 10	-	-	1	-
d 01	ⓓ 01	ⓓ 01	c 11	-	1	1	1	-

Result Analysis and Comparison

With the state assignment of Table 5.6, the fault coverage obtained is around 95% which is an 18% increase from the original 77%. The detail analysis results for two cases are shown below.

Table 5.7 Simulation result of two state transition tables

	Original State Transition Table	Modified State Transition Table
Numbers of Oscillation Rings	3 Case A (Condition Ⅱ) 1 Case B (Condition Ⅰ)	3 Case A (Condition Ⅰ) 1 Case A (Condition Ⅱ) 1 Case B (Condition Ⅰ)
Fault Coverage	**77 % (48 / 62)**	**95 % (72 / 76)**

Chapter 6

Conclusion

In the thesis, we propose an OR-Test methodology to test asynchronous circuits. We have investigated the conditions of forming oscillation rings and analyzed conditions of acihieving maximal detection. Due to un-availability of benchmark asychronous circuits, only a few circuits had been applied with this methodology as the experiment. However, it is found that generally a few number of state patterns will obtain a significant percentage of fault coverages. We have also proposed a state assignment procedure to reassign states of an asychronous circuit to obtain a higher OR testability. One good feature of this methodology is that it has the BIST capability. Although this is only a preliminary study, in view of the diffuculty in testing asychronous circuits encountered, this methodology is worthy of further studying.