

Chapter 6 Digital System Testing

1. Goal

Separate the good chips from the bad ones.

2. Why?

The fabrication/manufacturing process of the VLSI chips is not a perfect process.

➔ A certain percentage of chips will not be manufactured correctly.

3. Some definitions:

- (i) A defect is the physical flaw that results in a bad chip.
- (ii) Defects are modeled as stuck-at-faults at the gate circuit level.
- (iii) An error has occurred when a circuit's output is opposite to its correct output.

4. Testing strategies

- (i) Exhaustive Testing
 - All input patterns are generated and tested!
 - Prohibitively expensive.
- (ii) Functional Testing
 - Circuit's verification vectors are used.
 - Incomplete coverage.
- (iii) Structural Testing
 - Model faults and generate test vectors for those faults.
 - Work in practice.

5. Fault Model in Structural Testing

- (i) Single stuck at faults
- (ii) Multiple stuck at faults
- (iii) Delay (transition) faults: slow to rise / fall.

§ 6.1 Combinational Logic Circuit Testing

1. Control input for gates

- (i) One input gate – Can't have a control input.
- (ii) Two input gates

AND gate – '1'	OR gate – '0'
XOR gate – 'X'	NAND gate – '1'

2. Example

		a	b	c	d	e	
Wire f	SA0	1	1	0	x	0 ✓	1 1 0 1 0 (SA0: f, h, i)
		1	1	x	0	0	
	SA1	0	x	0	x	0 ✓	0 1 0 1 0 (SA1: f, h, g)
		x	0	0	x	0	
		0	x	x	0	0	
		x	0	x	0	0	
Wire h	SA0	1	1	x	x	0 ✓	
		x	x	1	1	0	
	SA1	x	0	x	0	0	
		0	x	0	x	0 ✓	
Wire i	SA0	1	1	x	x	0 ✓	1 1 0 0 1 (SA1: i)
		x	x	1	1	0	
	SA1	1	1	x	x	1 ✓	
		x	x	1	1	1	
Wire g	SA0	0	x	1	1	0 ✓	0 1 1 1 0 (SA0: g)
		x	0	1	1	0	
	SA1	0	x	0	x	0 ✓	
		x	0	0	x	0	
		0	x	x	0	0	
		x	0	x	0	0	

Minimum (Optional) Test Vector = {11010, 01110, 11001, 01010}

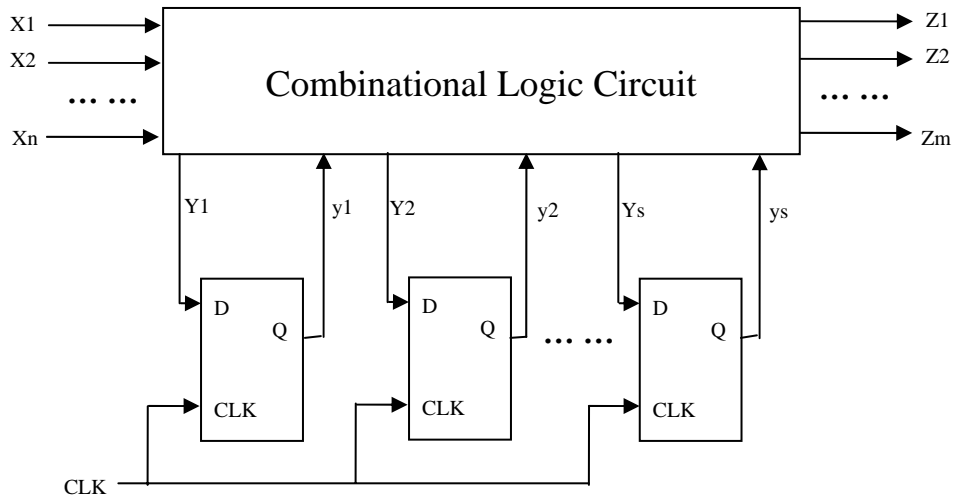
Maximum of Test Vectors = {00000, 00001,, 11111}

3. Sequential Logic Circuit Testing

(i) Basic concept of a scan chain

Any sequential circuits like this:

The outputs not only based on the inputs, but also the states (y_1, y_2, \dots, y_s)

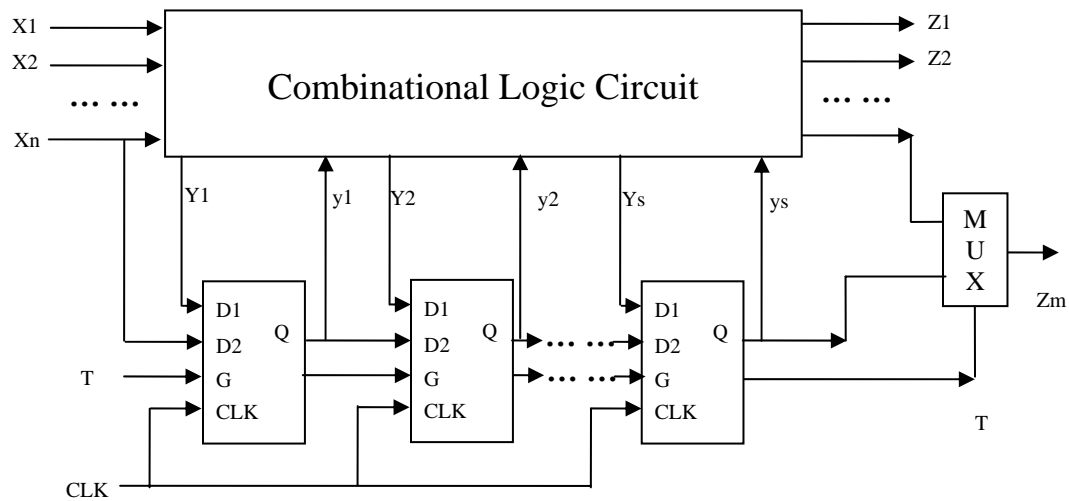


The basic idea: configure all memory elements as units in a shift register (scan) chain.

FFs are replaced by MUX-FFs

Extra pins are required: Scan_in, Scan_out, Test.

(ii) A way to test this: Scan-Path design



When $T=0$, D_1 input is the D-input, system operates as usual

When $T=1$, All FFs turn into one long shift register and the combinational logic is by-passed

Test Procedure:

1. Set $T=1$, shift in test pattern, y_i
2. Set X inputs as required
3. Set $T=0$, after sufficient delay, check Z
4. Apply a clock signal to CLK
5. Set $T=1$, shift out the FF outputs via Z_m , at the same time, shift in the next y_i

(iii) Level Sensitive Scan Design (LSSD) – (IBM approach)

Instead of using FFs, IBM use two latches build in a master-slave form and two phase clock (Non-overlapping clock)

Test procedure:

1. Scan in test vector via SDI by applying pulses alternatively to the test clock input TCK and the system clock input CK2
2. Set the corresponding test values on the X_i input. After sufficient propagation delay, check Z_j
3. Apply one clock pulse to system clock CK1 to enter new values of y_j into corresponding master latches
4. Scan out and check the y_j values by applying clock pulses alternatively to TCK and CK2

For most designs, scan-path / LSSD is a built-in part.

4. ATPG – Automatic Test Pattern Generators

(i) D – Calculus

Two steps:

Justify: to detect the fault

Propagate: set other unrelated pins to make it through

D (can be '0' or '1'), indicates that the fault free circuit should be 1 and the faulty circuit is 0. D' is the inverse.

For any gates, for example the two-input AND gate, we have

AND	0	1	D	D'	X
0	0	0	0	0	0
1	0	1	D	D'	X
D	0	D	D	0	X
D'	0	D'	0	D	X
X	0	X	X	X	X

Algorithm:

/* Generate a test for a fault s-a-v on line l */

```
{
    Set all input to  $x$ ;
    Justify (  $l, v'$  );
    If ( $v == 0$ ) Propagate ( $l, D$ );
    Else Propagate ( $l, D'$ );
}

--  $v$  = hypothesized faulty level
--  $l$  = point
```

Justify (l, Val);

/* Determine primary input (PI) that justify a line setting */

```
{
    If  $l$  is a PI, return;
     $c$  = gate's controlling value;
     $i$  = gate's inversion value;
     $inval := c \text{ xor } i$ ;
    If ( $val == inval$ ) select one of the gate inputs ( $j$ );
        Justify ( $j, c$ );
}
```

```

        Else for each  $j$  that is an input to the gate
            Justfy ( $j$ , val xor  $i$ );
    }

```

Propagate (l , err)

```

{
    Set  $l$  to err;
    If  $l$  is a PO return;
     $c$  = controlling value of the gate driven by  $l$ ;
     $l$  = inversion of the gate;
    For the gate's inputs ( $j$ )
        Justify ( $j$ ,  $c'$ )
     $k$  = the output line of the gate;
    Propagate ( $k$ , err xor  $i$ );
}

```

(ii) Some exceptional cases for this algorithm

- Wired logic should be avoided
- Dynamic logic should be avoided
- No one-shots, since these circuit are only external testable
- Redundant logic should be avoided
 - Since it is not testable, and can cause other faults to be masked
- Analog & digital mixed will cause problem
 - So, should be tested separately.

5. Examples

- (i) Example 1: Case with Fanout that make system intestable
 Stuck-at-1 at point F

	A	B	C	D
To Justify	0	0	X	
	0	X	0	
To Propagate		1	1	1

Problem: the fanout at point E

Justify needs E to be 0, while propagate needs E to be 1

Whenever you have fanout, this is likely to happen

Solution: Add more controllable or observable points to the circuit.

(ii) Example 2: Case with redundancy circuit

To eliminate static hazard, we need to add one more redundancy state

6. Three ways to solve ATPG problem

D-algorithm, PODEM, and FAM

Other than ATPG, there's another way to test the circuit:

RTPG: Random Test Pattern Generation (actually, pseudo-random binary sequence)

Usually, both ATPG & RTPG are used to test devices.