BIST Implementation

Viswanadh – 2019112011

Mani Teja - 2019102023

Sasanka – 2019102036

Siva Durga - 2019102038

Overview

Testing is the most time-consuming phase after manufacturing of any IC or product. To save the testing time of the product as well as make the product economically feasible, Built In Self-Test (BIST) is used. BIST can be referred to as the system or mechanism which is used to test any combinational circuit.

Pros and Cons of BIST

Pros:

1) BIST does not require any external hardware circuit for testing purposes, this reduces the cost required in the testing phase considerably.

2) BIST provides 90-95% fault detection coverage and sometimes it ranges up to 99%.

3) The testing is independent of future technology. So, there is no need to update our design of BIST repeatedly.

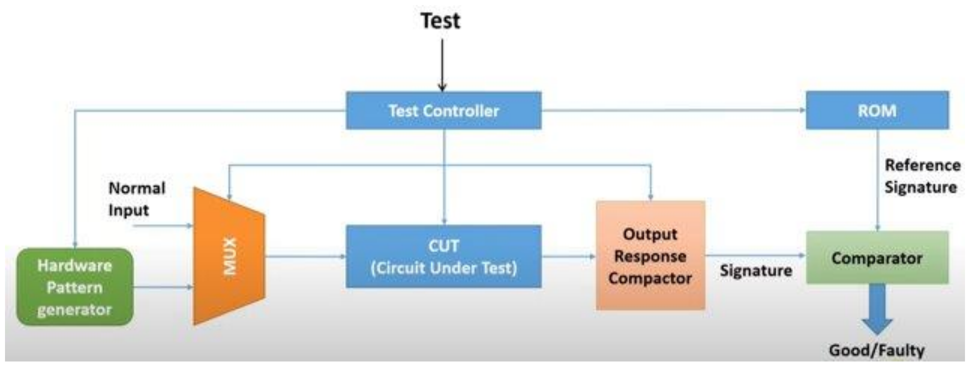
Cons:

1) BIST is separate testing circuit, and it occupies some of the area of the chip along with other parts of the circuits of the system in the chip thus the effective area of the chip will get reduced.

2) The relative cost of the hardware such as transistors and logic gates which are required to build BIST are declining but the relative cost of long wires is still slightly costly.

Implementation

We implement Online Non-Concurrent BIST. The term online indicates that the circuit may be made to test itself without being disconnected from the system. In the non-concurrent online BIST, normal operation is put off and only the testing would be carried out. The test process can be interrupted any time so that the normal operation can resume.



We have 5 major blocks in the circuit in addition to the test controller and ROM.

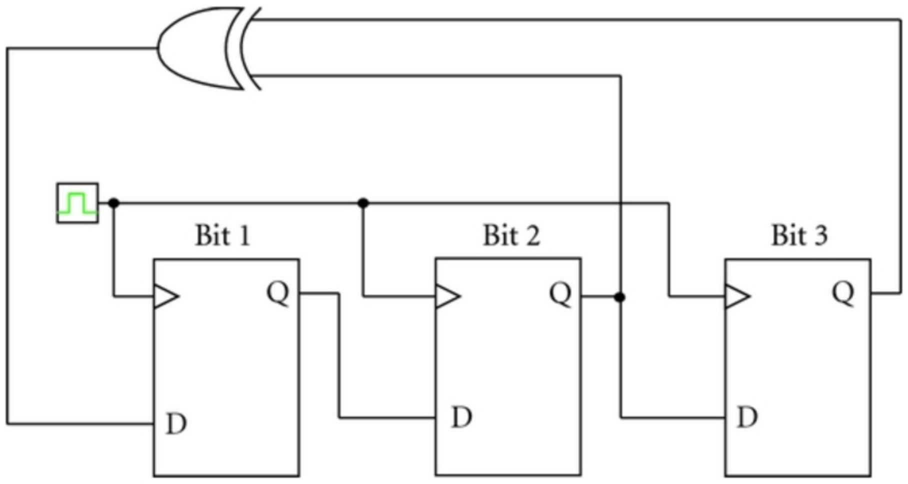
1) Mux

It decides whether the circuit should be in *test mode* or *normal working mode*. If the enable of the mux is 0, the circuit is in normal working mode. If the enable of the mux is 1, the circuit is in testing mode.

2) Pattern Generator (LFSR)

We use a pseudo random pattern generator to generate 2^n-1 inputs, where n is the number of inputs to the circuit under test. We have chosen a 3-input circuit. So, we design a pattern generator which randomly generates 7 input vectors (Each input vector has 3 values).

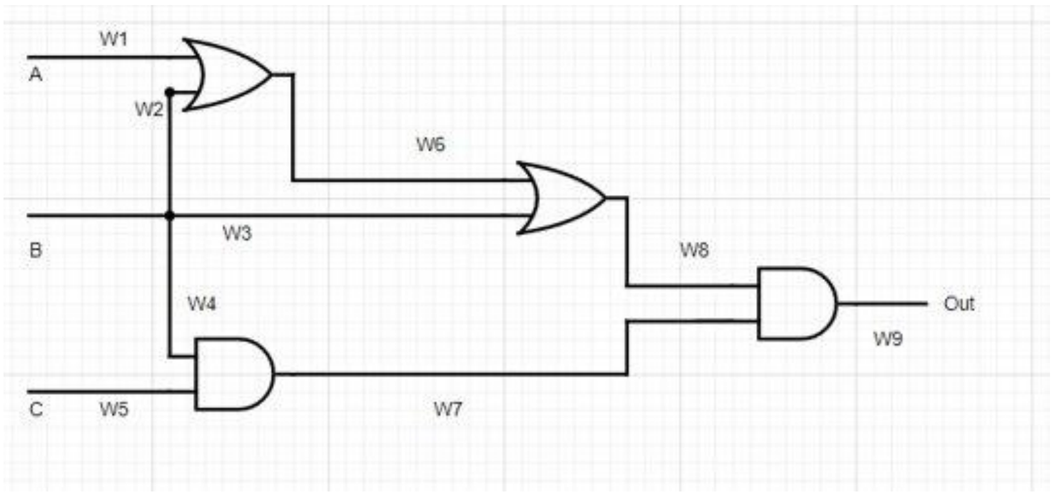
We use a *Linear Feedback Shift Register (LFSR)* for this process with characteristic polynomial $1+x^2+x^3$. The circuit is as follows.



We need to provide a random seed to this pattern generator as the initial condition. Let us say we provided 110 as initial seed. The patterns generated are tabulated.

Initial seed	1	1	0
	1	1	1
	0	1	1
	0	0	1
	1	0	0
	0	1	0
	1	0	1

3) Circuit Under Test (CUT)

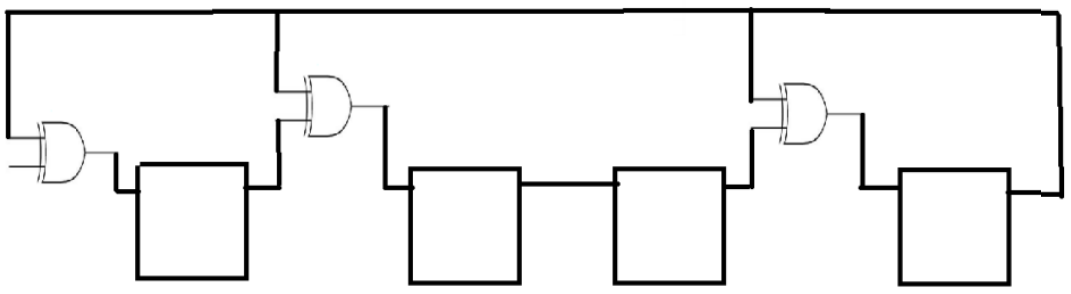


We take the following circuit and provide use of the patterns generated from the LFSR as inputs. Since we have 7 input vectors, after 7 clock cycles, we get 7 outputs based on the functionality of the circuit. In this case, we have the functionality to be *Out=B.C* So the outputs generated for the patterns generated are

A	**B**	**C**	**Out**
1	1	0	0
1	1	1	1
0	1	1	1
0	0	1	0
1	0	0	0
0	1	0	0
1	0	1	0

4) Signature Analyzer (SISR)

In this block, we generate a 4-bit signature value as the output when each of the 7 outputs obtained in the previous phase are given serially as the inputs. The signature Analyzer is a Serial Input Shift Register which takes in each of the inputs serially and generates a signature value. Note that when we pass the outputs of fault free circuit into the signature analyzer, we call the signature response generated to be the golden value which is stored and is used in the comparator block while testing faulty circuits. The circuit of the signature analyzer we used has a characteristic polynomial $1+x+x^3+x^4$



Initially all the outputs of the D-flipflops are set to be 0. Now we serially input 0110000 obtained in the previous block into it to generate the golden signature value.

Input	**Outputs of flipflops**
0	0000 [q3, q2, q1, q0]
1	1000
1	1100
0	0110

0	0011
0	1100
0	0110

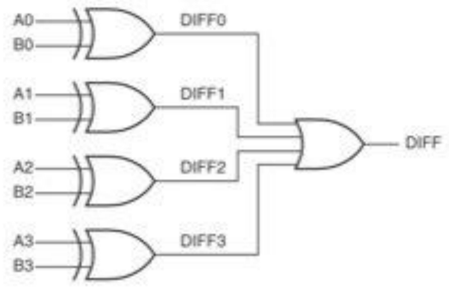
For the Fault free circuit, 0110 is the golden signature.

Let us say that our circuit is faulty, say we have B Stuck at 1. Then the output of the CUT is 0111001

Input	Outputs of flipflops
0	0000
1	1000
1	1100
1	1110
0	0111
0	1110
1	1111

5) Comparator

The signature value obtained for B stuck at 1 is 1111. Now comes our comparator block which compares the two obtained signatures with the golden signature and returns 0 if there is no fault and 1 if there is a fault. The circuit of the comparator block is as follows.



We have 0110 as the golden signature and 1111 as the signature for B stuck at 1. If we send these two values into the comparator block, we get output as 1 confirming that there is a fault in the circuit.

BIST Implementation in Verilog

The above BIST architecture is implemented in Verilog and the following are the code snippets of it.

1 - LFSR

Pgen.v

```
`timescale 1ns / 1ps
module pgen (input clk, enable, reset, output reg [2:0] q);

reg temp;
initial q = 3'b101;

always @ (posedge clk) // start generating when enable = 1 by setting check
to 1
begin
      if (enable == 1'b1 && reset == 1'b0)
      begin
      temp = q[1] ^ q[2]; // store it temporarily to later assign without
```

```
clashing with cache update
      q[2] <= q[1];
      q[1] <= q[0];
      q[0] <= temp;
      end
      else if (reset == 1'b1) q <= 3'b110;
end

//$display("%b %b %b %b %b",a,b,c,d,y);
endmodule
```

2 - MUX

Mux.v

```
`timescale 1ns / 1ps

module mux(input [2:0] primary, input [2:0] test, input enable, output [2:0]
y);

wire [2:0] en;
assign en = {enable, enable, enable};
wire [2:0] enNot, out1, out2;

assign enNot = ~en;
assign out1 = primary & enNot;
assign out2 = test & en;
assign y = out1 | out2;

endmodule
```

3 - CUT

Cut.v

```
`timescale 1ns / 1ps
module cut (input [2:0] a,input [11:0] check,input value,output y); //atmost
one saf
wire aa,b,c,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13;
```

```
assign aa = (check[0]==1) ? value: a[0];
assign b = (check[1]==1) ? value: a[1];
assign c = (check[2]==1) ? value: a[2];
assign w1 = (check[3]==1) ? value: aa;
assign w2 = (check[4]==1) ? value: b;
assign w3 = (check[5]==1) ? value: b;
assign w4 = (check[6]==1) ? value: b;
assign w5 = (check[7]==1) ? value: c;
assign w6 = (check[8]==1) ? value: w1 | w2;
assign w7 = (check[9]==1) ? value: (w4 & w5);
assign w8 = (check[10]==1) ? value: (w3 | w6);
assign w9 = (check[11]==1) ? value: (w7 & w8);
assign y = w9;
//$display("%b %b %b %b %b",a,b,c,d,y);
endmodule
```

4 - Signature Analyser

Sir.v

```
`timescale 1ns / 1ps
module sir (input clk,enable, reset,y, output reg [3:0] q);

reg [2:0] temp;
initial q = 4'b0;

always @ (posedge clk) // start generating when enable = 1 by setting check
to 1
begin
      if (clk == 1) begin
      temp[0] = q[0] ^ y; // store it temporarily to later assign without
clashing with cache update
      temp[1] = q[0] ^ q[3];
      temp[2] = q[0] ^ q[1];

      q[1] <= q[2];
      q[2] <= temp[1];
      q[3] <= temp[0];
      q[0] <= temp[2];
end
      if (reset == 1)   q <= 4'b0;
end

//$display("%b %b %b %b %b",a,b,c,d,y);
endmodule
```

5 - Comparator

Comp.v

```
`timescale 1ns / 1ps

module comp(input [3:0] gSign, input [3:0] cutSign, input enable, output y);
wire [3:0] en;
assign en  = {enable, enable, enable, enable};
wire [3:0] x0;
wire x1;
assign x0 = gSign ^ cutSign;
assign x1 = x0[0] | x0[1] | x0[2] | x0[3];
and(y,x1,enable);
endmodule
```

Integrating module

Alu.v - main module of the BIST implementation

```
`timescale 1ns / 1ps
`include "pgen.v"
`include "mux.v"
`include "cut.v"
`include "sir.v"
`include "comp.v"

module alu(input [2:0] in, input clk, reset, enable, input [11:0] check,
input value, output y,e);
// 3 inputs,clk,reset , cl output, fault detector

// initiate Pattern Gen blocks
// Let output be [2:0]p
wire [2:0] p;
pgen p1 (clk, enable, reset, p);

// initiate muxes
// outputs ym1, ym2
wire [2:0] ym1, ym2;
mux m1 (in,p,enable,ym1); // to fault-free
mux m2 (in,p,enable,ym2); // to faulty ckt
```

```
// execute the combinational logic
// outputs: yc1, yc2
wire yc1, yc2;
cut c1 (ym2,12'b0,1'b0,yc1); // golden
cut c3 (in,12'b0,1'b0,y); // golden
cut c2 (ym1,check,value,yc2); // faulty

// signature register
// Let outputs be ys1, ys2
wire [3:0] ys1, ys2; // size should be changed. 7 values collected and 4
values are observed
sir s1 (clk,enable,reset,yc1, ys1); // golden
sir s2 (clk,enable,reset,yc2, ys2); // faulty

// comparator
comp c11 (ys1, ys2, enable, e);

//$display("%b %b %b %b %b",a,b,c,d,y);
endmodule
```

Testbench used

To test the codes, run the alu_t.v file and the outputs can be observed through gtkwave.

```
// Test bench for the combinational logic circuit
`include "alu.v"
module test_bench();

// Inputs
reg [2:0] in;
reg [11:0] check;
reg clk,reset,enable,value;

// Variables
integer i,j;

// Outputs
wire y,e;

// Instantiate the Unit Under Test (UUT)
alu uut (in,clk,reset,enable,check,value,y,e);

always #5  clk = ~clk ;//setup clk
initial begin
```

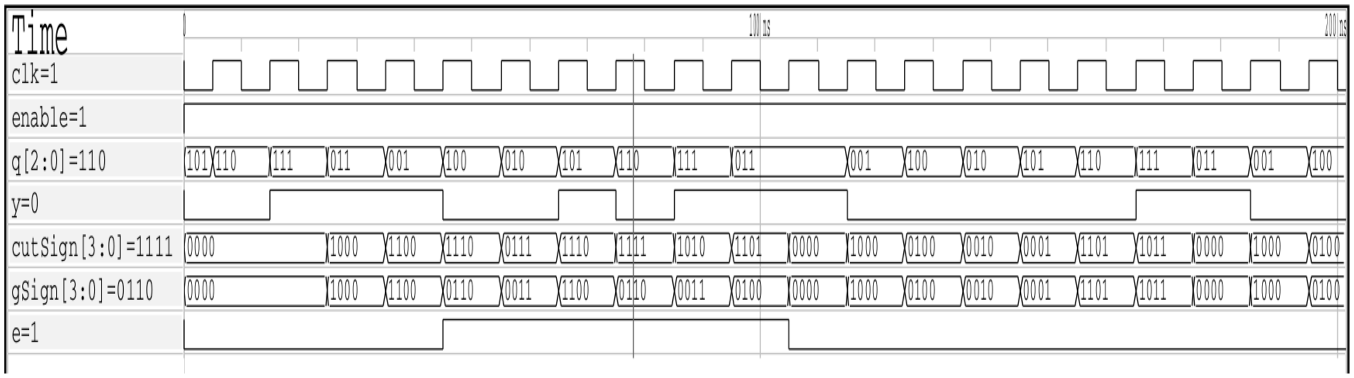
```
      // Initialize Inputs
      $dumpfile("test_bench.vcd");
      $dumpvars;

      // initialise all the wires
      reset = 1'b0;
      enable = 1'b1;
      value = 1'b0;
      check = 12'b0;
      clk = 0;
      in = 3'b110;

      #5 // se3tting wire B sa1
      check = 12'b000000000010;
//    enable = 1'b1;
      value = 1;
      #100

      // reset the wires
      reset = ~reset;
      #5
      reset = ~reset;

      #5 // setting ckt fault free
      check = 12'b0;
      #100
      $finish;
/*
      // reset the wires
      reset = ~reset;
      #5
      reset = ~reset;

      #5 // setting wire w9  to sa1
      check = 9'b100000000;
      value = 0;

      #100
*/
end
endmodule
```

Results

To check the working of the BIST ckt that is implemented, we have tried out testing with two different faulty circuits and a fault-free circuit:

(I) Faulty circuit with output wire having one s-a-f

(II) Fault-free circuit



- Clk=> global Clock
- q[2:0] => Pattern generator output
- y=> output of CUT
- CutSign => Signature response for CUT output
- GSign => Signature response for fault-free circuit (Golden reference)
- e => Comparator output

Case I:

Faulty circuit with input B wire having one s-a-f

We can observe that in the first 100ns, in the testbench file, we are inputting a faulty circuit. Accordingly, we obtain the combinational logic outputs and signature responses which match wit the theoretical values obtained above thus validating this test case.

Case II:

Fault-free circuit

We can observe that in the second 100ns (t>100ns), in the testbench file, we are inputting a fault-free circuit. Accordingly, we obtain the combinational logic outputs and signature responses which match with the theoretical values obtained above thus validating this test case.

References

Design and implementation of BIST - ieee