BIST

Design BIST Hardware For our fourfunc Circuit

Mohammad Hashemi

University of Tehran
Tehran,Iran
md.hashemi@ut.ac.ir

***Abstract*—This paper is going to discuss about a hardware insertion called BIST. Our BIST hardware includes an input TPG method and an ORA output. Our goal will be testing our circuit fourfunc by adding BIST hardware architecture and check the hardware redundancy and timing of our test.**

BIST, ORA, TPG

I. Introduction

This template will explain: (1) deploy an input TPG method called LFSR, (2) an output ORA method called MISR, and (3) apply the input to our fourfunc circuit and check the ORA results.

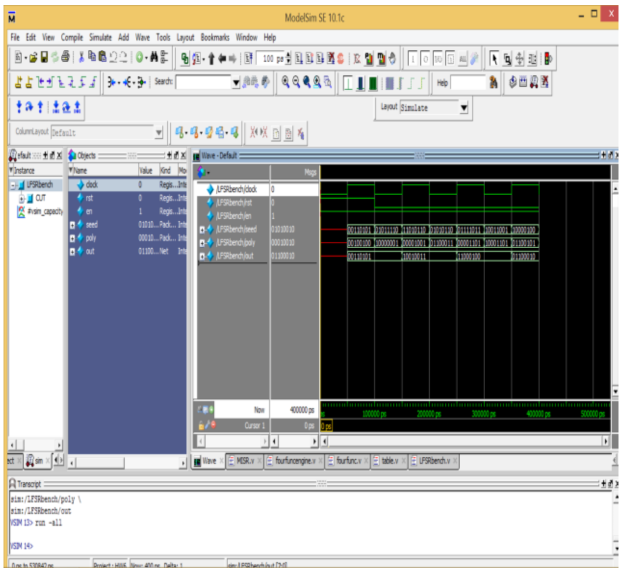
II. TPG Method LFSR

A. LFSR Opreation

LFSR is a test pattern generation method operates by left handing shift. We apply our LFSR to our circuit.

B. LFSR Outputs

After we develop our LFSR is time to check the outputs. For that we use Verilog virtual tester. The results are shown in Fig.1 (the bench is in codes folder named LFSR bench).



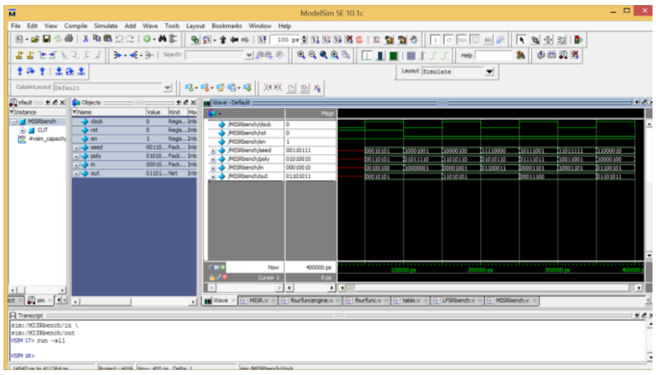
```
sim:/LFSRbench/poly \
sim:/LFSRbench/out
VSIM 13> run -all

VSIM 14>
```

Fig.1 LFSR output

III. ORA Method MISR

After deploy and apply our LFSR to our main circuit fourfunc we are going to develop our ORA called MISR. MISR is an output compaction method that operates with XOR gates. After that we use Verilog to test it. The results are shown in Fig.2 (the bench is in codes folder named MISR bench).



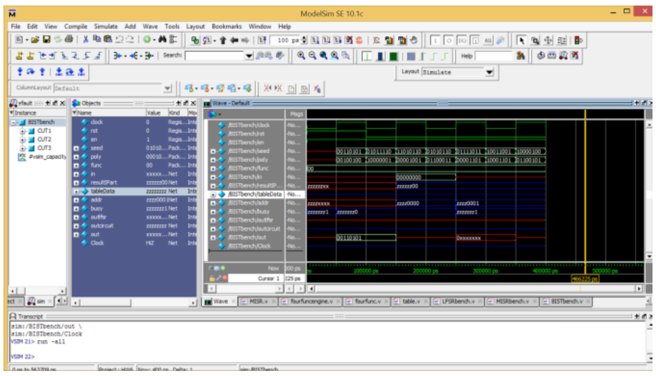
```
sim:/MISRbench/in \
sim:/MISRbench/out
VSIM 17> run -all

VSIM 18>
```

Fig.2 MISR output

IV. BIST Insertion

After apply our TPG and ORA we run a tester of our complete circuit tester and the results are in Fig.3.



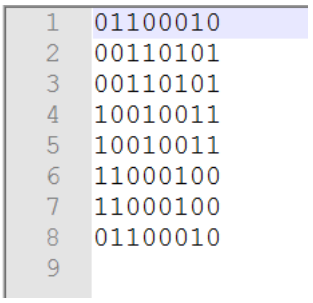
```
sim:/BISTbench/out \
sim:/BISTbench/Clock
VSIM 21> run -all

VSIM 22>
```

Fig.3 whole circuit output

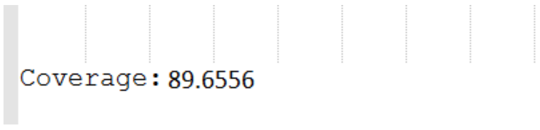
V. ADDER And Multiplier Testing

After apply our TPG and ORA we extract our adder and multiplier and inject fault on them then check them to obtain coverage the results of hole circuit coverage that outputs generate by LFSR is shown in Fig.4. For adder individually is shown in Fig.7 and for Multiplier is shown in Fig.8. As we see for adder with LFSR inputs we claim 89.6% Coverage that shown in Fig.8 and for Multiplier we claim 93.4% Coverage that shown in Fig.9. To inject and collapse our adder and multiplier we use code tester.MUT.adder and tester.MUT.Multiplier or we can rewrite a code for multiplier and adder (we use this method here). To obtain our netlist we use net_gen and obtain adder and multiplier (shown in Fig.5 and Fig.6).



```
01100010
00110101
00110101
10010011
10010011
11000100
11000100
01100010
```

Fig.4 LFSR Inputs Generated



```
Coverage: 89.6556
```

Fig.8 adder Coverage

REFERENCES

[1] Testable Array Multipliers for a Better Utilization of C-Testability and Bijectivity

[2] The Design of Easily Testable VLSI Array Multipliers

[3] Combined Pseudo-Exhaustive and Deterministic

[4] Testing of Array Multipliers

[5] Scalable and bijective cells for C-testable

iterative logic array architectures