Testability of Combinational and Sequential Circuits by Fault Injection Technique

S.Ravi¹, G.Balu², M.Kannan³, S. Ashik Ahamed⁴, K. Kavin Kumar⁵
¹Assistant professor, Dept of ECE, The Kavery Engineering College
²,³,⁴,⁵UG Student, Dept of ECE, The Kavery Engineering College, Salem, India

Abstract: Testing combinational and sequential blocks can be actually done easily. However, testing a complex circuit requires set of known test patterns and their corresponding responses to be stored in memory. Also, some of the internal nodes of the circuits cannot be accessed easily to apply desired input values at that node. This paper presents a fault injection technique to inject transient and permanent fault at the Verilog code description of both combinational and sequential digital circuits. The proposed method can verify the testability of the circuits using both offline and online testing. The proposed method is tested with two combinational circuits (8-Bit Adder and a benchmark circuit called, C17) and two sequential circuits (8 – Bit Counter and S27 Benchmark Circuit) for Injection of permanent fault and transient faults and their testabilities are evaluated.

Keywords-Fault injection technique, Online and offline testing, Permanent fault, Transient fault.

I. INTRODUCTION
Before it is delivered to the customers the system should be tested and evaluated for its functionality. The deviation from the intended working is termed as fault which in turn produces error [1]. It can occur in hardware as well as software components. Basically faults are classified into permanent faults and temporary faults based on time duration. Permanent fault exists permanently until the fault component is replaced. Transient fault occurs temporarily and then disappears. permanent faults are tested by offline testing whereas transient and intermittent fault are tested using online testing that is when the system is working [2]. In the case of complex fault tolerant systems, fault injection has been identified to be particularly attractive and valuable method, where faults are inserted into a system and examined its function in response to a fault. This paper evaluates combinational and sequential circuits using a method where fault injection can be used within Verilog code description.

J. Karlsson et al explains about the fault injection technique using heavy ion radiation [2]. J. Arlat et al explains about MESSALINE fault injection technique[3]. Both of the above mentioned techniques come under hardware fault injection technique. As technology advances and more amounts of transistors are mounted on a single chip, usage of these techniques become less efficient as they could not limit the impact of real faults. Software fault injection techniques like FERRARI proposed by Kanawati G. et al uses traps to inject faults [4]. Other techniques use interrupts and masks. It is found that these techniques sometimes can be corrupt the target application when used in memories[5]-[7]. Simulation fault injection techniques such as VERIFY uses verilog coding to inject faults in the coding level to find the circuit testability by modifying the original coding and is written with fault injection possibilities[8]. Other Simulation techniques given by Kammler, D. et al and Haissam Ziade et al use modification inside coding[9],[10].The injection technique which is present in this paper is done in Verilog programming using package concept[11]. It is different from the previous techniques by reducing the complexity of the coding of digital circuits and just a call is needed to invoke the fault injection wherever the fault has to be injected. Here fault coverage is estimated for the digital circuits that are chosen to inject fault.
II. EXISTING SYSTEM

To determine the time interval to inject the faults Linear Feedback Shift Registers are used. A 6 bit LFSR is taken. LFSRs include flip flops and the feedback is given by XOR gates. LFSR will remain in the same state because all zeros pattern should not be given as inputs, so it cannot produce different patterns. XNOR gates can also be used. In that case all 1’s bit pattern is eliminated. LFSR maximal length polynomial is also mentioned as $2^n - 1$ bit patterns. The LFSRs are constructed using primitive polynomial that is, the powers are relatively prime. The primitive polynomial is used here is $x^6 + x^5 + 1$. Hence it can produce up to $2^6 - 1$ patterns that is 63 different input patterns. The input bit pattern used to initiate the LFSR is called “seed”. The seeds for the two LFSRs are different. Polynomial is taken as “000001” and the other as “100000”. Control bits used for transient are from “001” to “110”. The control code determines the rate of injection.

![Fig 1. 6 Bit LFSR](image)

III. PROPOSED SYSTEM

This System requires basic software requirements which covers low cost, less power consumption and low size.

Software Requirement
- Modelsim
- XILINX ISE

A. MODELSIM

Modelsim is excellent digital simulator for both Verilog and VHDL hardware description languages. It has used TCL scripting since before TCL was popular. While the Personal Edition (PE) only supports TCL, the SE and EE versions actually permit elements to be used. ModelSim is an easy-to-use yet versatile VHDL/(System)Verilog/SystemC simulator by Mentor Graphics.

It supports behavioral, register transfer level, and gate-level modeling. ModelSim suitable for all platforms used here at the Institute of Digital and Computer Systems (i.e. Linux, Solaris and Windows) and many others too. ModelSim can be preinstalled on Institute's computers. Windows users, however, must install it by themself.

B. Using Modelsim Only For Simulation and Verification

Unlike Xilinx ISE, the Modelsim cannot synthesize/implement the design into real hardware, but it can compile and simulate HDL-based design, and display graphical and text information to facilitate debugging. The main advantages of using Modelsim standalone are convenience and speed. That is, instead of editing your code in the Project Navigator editor and re-invoking Modelsim every time the small change is made (like we have been doing so far), by using Modelsim standalone you can edit the code, re-compile it and re-simulate it -- all without closing the applications. The procedure is used to simulate a design in Modelsim is simple.

1. Create a new Modelsim project.
2. Add existing source files to the project or create new Verilog source files.
3. Compile the all source files.
4. Start the simulation.
5. Run the simulation to the desired length of time.
If you find some errors, whether they are compilation errors (syntax errors that are reported during compilation), or functional errors (you notice after simulating and observing waveforms), you can edit the code and repeat steps 3-5 to re-simulate the design. More details on each of the five steps are given in the Procedure section. Steps 3 to 5 can be automated using a DO file (with do extension).

C. XILINX ISE
For two-and-a-half decades, Xilinx has been on the forefront of the programmable logic revolution, with the invention and continued migration of FPGA platform technology. During that time, the role of the FPGA has evolved from a vehicle for prototyping and glue-logic in a highly flexible alternative to ASICs and ASSPs for a host of applications and markets. Today, Xilinx® FPGAs have become strategically essential to world-class system companies can be hoping to survive and compete in these times of extreme global economic instability, turning what was once the programmable revolution into the “programmable imperative” for both Xilinx and our customers.

IV. PROGRAMMABLE IMPERATIVE
When viewed from the customer's perspective, the programmable imperative is necessity to do more with less, to remove risk wherever possible, and to differentiate in order to survive. In essence, it is quest to simultaneously satisfy the conflicting demands created by ever-evolving product requirements (i.e., cost, power, performance, and density) and mounting business challenges (i.e., shrinking market windows, fickle market demands, capped engineering budgets, then escalating ASIC and ASSP non-recurring engineering costs, spiraling complexity, and increased risk). To Xilinx the programmable imperative represents a two-fold commitment. The first is used to continue developing programmable silicon innovations at every process node that deliver industry-leading value for every key figure of merit against, which FPGAs are measured the price, power, performance, density, features, and programmability. The second commitment is to provide customers with simpler, smarter, and more strategically viable design platforms for creation of world-class FPGA-based solutions in a wide variety of industries—what Xilinx calls targeted design platforms.

V. DESIGN METHODOLOGY
Injection of permanent fault and transient fault using a block diagram. Here two 8 bit LFSRs are used along with one 8 bit shift register block, one 4 bit control block and fault injection block.

![Fig 2. Proposed System Block Diagram](image-url)
8 bit data word is chosen as input data. Any data length can be chosen to inject faults. Here injection of faults indicates the bit flips.

A. FAULT INJECTION LOGIC
Fault injection is best techniques when used to measure the fault robustness or tolerance of a system. The fault injection is still in developing stage but the mechanisms are thoroughly proved. For a system designer attempting to measure the degree to which his design is immune to faults, fault injection can be a proper technique for quantifying this aspect of design. If a system is designed to endure a certain types of faults or exhibit certain behavior in the presence of certain faults and they are being injected into the system to examine their effects. The difficulty to test the system for robustness is in creating unusual conditions in its working and fault injection has been proposed as a method to address this problem.

The following points talk about inferences that are obtained from fault injection:
- Availability: Is the system ready to use.
- Reliability: Can the system run continuously without failure.
- Safety of the circuit.
- How easily the failed system can be repaired.

B. PERMANENT FAULT INJECTION
Fault injection block injects permanent faults with use the user’s specification and this is controlled by control block. Figure 2 gives us the working of permanent fault injection. Inputs are Stuck_at_bit, stuck_at_value and input data would. Fault injection is done by the fault injection block. Stuck_at_bit and stuck_at_value are given by the user. The control chosen is 4 bit word starting from “0000” to “1111”. The bits “0000” and “1111” are used for permanent.

C. TRANSIENT FAULT INJECTION
Transient is one of the types of temporary fault types. As technology improved year by year, more number of circuits can be implemented on the chip and circuits are scaled down to accommodate such large number of devices. As a result of this, transient faults occur in present frequently for manufactured devices. This technique provides an efficient way to test transient faults. When the bits in both the LFSRs matched fault insertion block performs, then fault injection in the input word in the place which is indicated by shift register. shift register is consists of arrangement of flip flop and are important in application involving the storage and transfer data in a digital system, it is a type of sequential logic circuit, mainly for storage of data.

D. 8 BIT LFSR
A linear feedback shift register (LFSR) is shift register whose input bit is a linear function of its previous state. the most commonly used linear function of an single bits is XOR. The initial value of the LFSR is called the send, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state.
The linear feedback shift register most commonly used in random number generator, error detection and correction, jamming, warfare.

Linear Feedback Shift Registers are used to calculate the time interval to inject the faults. A 8 bit LFSR is taken. LFSRs include flip flops and the feedback is given by XOR gates. Hence all zeros pattern should not be given as inputs due to LFSR will remain in the same state and it cannot produce different patterns. XNOR gates can also be used.

![Fig 3. 8 bit LFSR](image)

In that case all 1’s bit pattern are eliminated. LFSR is also mentioned as maximal length polynomial as it can produce $2^n-1$ bit patterns. Here it can produced up to the $2^8-1$ patterns that is 125 different input patterns.

The input bit pattern are used to initiate the LFSR is called “seed”. The seeds for the two LFSRs are different. The seed for one polynomial is taken as “00000001” and the other as “10000000”. Control bits are used for transient are from “000001” to “110000”. The control code determines the rate of injection. The control code which is processed by the Control Logic block explains how many bits are need to be matched in the two LFSR’s to control the percentage at which faults are injected. Certain bit matches for each control code. The rate of injection should decrease from “000001” to “110000”.

When there is no bit matches in both the LFSR then there is fault inserted at that point. Fault injection occurs in accordance with control code and bit matches in LFSRs. The rate of injection starts within 50% and ends within least value depending upon the control word.

D. FAULT INJECTION BLOCK

One hot encoded shift register used here has data length of 8 bit. The data length of shift register and input data word should be same. Since the shift register tells us where to insert the faults. Shift Register contains only one ‘1’ value rest all other is ‘0’s. The input taken is “10000000”. ‘1’ shifts continuously for every clock input. When the fault injection block understands that a fault should be injected, it check the bit position of ‘1’ in shift register. In that bit position value fault is injected in input data word. Transient fault injection can be tested only in online testing that is when the system is in working state. To indicate online testing an online self checking signal is included.

VI. SIMULATION RESULT FOR COMBINATIONAL CIRCUITS

The fault injection is done in both combinational and sequential circuits. Simulation is done using Xilinx software. 4 bit adder and C17 benchmark circuit are taken as examples for combinational to do fault insertion. Structural level modelling is used for combinational circuit description coding.
A. **8 BIT ADDER**

Figure 1 and 2 shows fault free condition and fault condition in 8 bit adder. Here fault is injected at the third bit of input a. The output sum varies with respect to fault free condition.

![Fig 1. Fault free condition](image1)

![Fig 2. Fault condition](image2)

B. **C17 BENCHMARK CIRCUIT**

Table I shows the outcome for various input patterns for both permanent and transient fault for C17 benchmark circuits.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Type of Fault Injection</th>
<th>Input Patterns</th>
<th>Expected Results</th>
<th>Obtained Results</th>
<th>Fault Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>C17 BENCHMARK CIRCUIT</td>
<td>Permanent</td>
<td>1,1,1,1 s-a-1 is used.</td>
<td>Output1-1</td>
<td>Output1-0</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0,0,1 s-a-1 is used.</td>
<td>Output1-0</td>
<td>Output1-0</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0,1,0 s-a-0 is used.</td>
<td>Output1-1</td>
<td>Output1-1</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>Transient</td>
<td>1,0,0,1</td>
<td>Output1-1</td>
<td>Output1-0</td>
<td>YES</td>
</tr>
</tbody>
</table>

VII. **SIMULATION RESULT FOR SEQUENTIAL CIRCUIT**

Most of the sequential circuits are simulated using behavioral level of modelling. It is possible to use this injection block in behavioral level also. 4 bit counter and S27 benchmark circuit are used as an example.
A. 8 BIT COUNTER

Figure 1 shows the result of fault free condition in 8 bit counter. The fault injection in 8 bit counter but produces same output as fault free. Figure 2 shows the result of fault condition of 8 bit counter.

![Fig 1. Fault free condition](image1)

![Fig 2. Fault condition](image2)

B. S27 BENCHMARK CIRCUIT

Table II shows the outcome for various input patterns for both permanent and transient fault table gives a pictorial view of fault coverage. Fault coverage specifies the amount of faults that can be detected and it is calculated in terms of percentage for all the above mentioned circuits in this paper. Permanent Fault Injection coverage is found to be reduced by 4% to 5% from 100% in all the four types of circuits. It is because fault injected inputs may produce the same fault free output when they are processed since permanent fault is injected by off line testing and there is no online checking signal to indicate the presence of fault which is contrast to transient fault injection which is tested using online testing and it has online self-checking signal which provides 100% fault coverage.

![Fig. 8. Bar Graph for Fault Coverage.](image3)
VIII. CONCLUSION

Thus an efficient Fault injection technique at the Verilog level using package is constructed. This injects permanent and transient fault at any signal present inside the circuit which is written in Verilog coding. It allows the injection of transient faults randomly across a data word and allows the insertion of a permanent fault at any chosen point in a data word. Many illustrations are provided to illustrate the use of the proposed fault injection system in on-line and off-line testing environments for both combinational and sequential circuits. A major advantage is that the fault insertion process is significantly simpler and easier than other fault insertion approaches. Since the fault insertion block is included in a package, only a simple call at the beginning of the Verilog description and a component instantiation is needed to activate the insertion mechanism. Fault coverage for permanent fault injection in 8 bit adder and 8 bit counter is estimated to be 95%. For benchmark circuits it is found as 96%. For transient fault injection, coverage is calculated and estimated as 100% for all the four circuits.

REFERENCES