Online Testing of Interconnect Faults in SRAM based FPGA Systems

L. Kalyan Kumar, Amol Mupid, Aditya S. Ramani, V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai- 600 036, Tamil Nadu, India.
Email: kama@iitm.ernet.in

Abstract
This paper describes a novel method for online detection and location of interconnects faults in SRAM-based FPGA systems. In safety critical systems like space probes, online checkers are used to report misbehavior of any of the subcircuit within the system. When one such subcircuit is reported to misbehave, the algorithm proposed in this paper attempts to detect and locate the interconnect faults, if any, within the faulty subcircuit without shutting down the other subcircuits. The proposed algorithm is in-place, i.e. it does not alter the routing structure of the application, which is vital for the other non-faulty subcircuits to function properly. The proposed algorithm takes advantage of the partial reconfiguration capability of FPGAs to locate the faulty interconnects by selective programming of LUTs in the LUT-network realizing the faulty subcircuit.

Keywords: FPGA, Interconnect faults, Reconfigurable systems, Online fault detection and location.

1. Introduction
Current day safety critical systems need hardware with complex logic and demand high degree of fault tolerance. With the increased degree of circuit integration and rapid development in packaging technology, system/chip level testing has become complicated. As stated in [10], it has become difficult to execute safety critical applications on low cost commercial technology providing the required fault tolerance. Field Programmable Gate Arrays (FPGAs) based adaptive/reconfigurable systems are better suited for configuring and executing such safety critical applications. These systems are not only cost effective but can provide safety features that in turn make the application configured on them highly dependable. The safety features include the ability of such systems to detect runtime errors, precisely locate faults causing them and rapidly reconfigure themselves to avoid the located faults [10]. Online checkers are used to detect the runtime errors in specific parts say \( p \), of the system. When such a fault is detected, the fault has to be located without bringing down the entire system. More precisely, it is desirable that some critical parts of the system still function even as the fault location is being carried out in another part of the system. The part \( p \) mentioned above will be a LUT-network.

Having identified a faulty LUT-Network \( p \), the algorithm proposed in this paper attempts to detect and locate the interconnection faults if any, in \( p \), without shutting down the other subcircuits/parts of the system. The proposed algorithm is in-place i.e. it does not alter the routing structure of the application which is vital for the other non-faulty subcircuits to function properly.

2. FPGA Architecture
FPGAs are widely used in the design of Reconfigurable systems. An FPGA consists of an array of Configurable Logic Blocks (CLBs) along with programmable interconnection channels and I/O blocks. An FPGA can be programmed to implement combinational and sequential functions. Each CLB consists of a set of LUTs, which are used to store the truth table architecture of the FPGA is shown in Figure 1.
The FPGA routing architecture in Xilinx [17] has the following features. Every row or column has two pairs of I/O pads associated with it one each on two of their boundaries as shown in Figure 1. All the routing channels have the same width (number of wires). Each circuit is mapped into the smallest square FPGA that can accommodate it. For example, a circuit containing 14 CLBs and 10 I/O pads would be mapped into an FPGA consisting of a 4x4 array of CLBs. Each CLB in the Xilinx architecture [17] consists of a 4-input look-up table (LUT), and a flip flop, as shown in Figure 2. There is only one output, which can be either the registered or the unregistered LUT output. Each CLB has four inputs for the LUT and an additional clock input. Since the clock is normally routed via a special-purpose dedicated routing network in commercial FPGAs [17], detecting interconnection faults in nets carrying the clock signal is not considered in this paper.

Figure 2. The CLB Structure

The locations of the CLB pins are shown in Figure 3. Each input pin is accessible from one side of the CLB, while the output pins can connect to routing wires in both channels, one to the right of the CLB and another below the CLB.

Figure 3. CLB pin locations

Each input pin of a CLB can connect to any one of the wiring segments in the channel adjacent to it. Each output pin of a CLB can connect to any of the wiring segments in the channels adjacent to it. Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it as shown in Figure 1.

Figure 4. Logic Block Pin to Routing Channel Interconnect

The FPGA routing is unsegmented i.e. each wiring segment spans only one CLB before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths are constructed.

Figure 5. Unsegmented FPGA Routing

At every point of intersection of a vertical and a horizontal channel there is a switch box. In the Xilinx architecture [17], when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The switches used in this architecture have either the planar or domain-based switch box topology. In this switch box topology, a wire in the track
number 1 of a channel segment connects only to wires in track number 1 of the adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The Figure 6 below illustrates the connections in a switch box [17].

The general switch matrix will contains programmable interconnect points (PIP), which will be either a single pass transistor or a series of transistors. Each PIP in a switch matrix is directly connectable to any of the other PIPs, by suitably programming the switch matrix. On locating an erroneous PIP, our algorithm tries to reroute using the other available PIPs in the switch matrix, avoiding the faulty one. For most of the applications mapped on FPGAs, all the PIPs in a switch matrix are not used, providing more chances for successful rerouting. The switch matrix is shown in Figure 7.

3. Previous work
FPGA testing is divided in to five regions. Testing of logic blocks, interconnects, embedded memory, I/O block and miscellaneous testing which involves the clock and powerlines. Algorithms for testing the logic blocks in SRAM based FPGAs are presented in [1-3]. Efficient methods for testing interconnect faults are presented in [4-5]. Algorithms for location of interconnect faults are described in [6-7]. The technique presented in [8] for detecting and locating both LUT and interconnect faults is essentially a BIST technique, which uses some of the LUTs - both as test pattern generators (TPGs) and output response analyzers (ORAs) - to test the other LUTs. This technique needs more number of reconfiguration sessions and the reconfiguration data should be provided in advance. Using the result in [8], the errors cannot be detected and located online. An algorithm for location of faulty LUTs is described in [9], which is an extension of [8] and exploits the partial reconfigurability of the system. This method is efficient, because the time to test the LUT network depends only on the number of faulty LUTs and is independent of the number of LUTs. Application-dependent test methods (test methods that depend on the Application configured on the FPGA) for location of a single faulty LUT, based on partial reconfiguration technique are described in [10]. In this an efficient method is designed for testing the single faults in LUTs in a SRAM based FPGA systems, which have single output and without fanout. Methods for interconnect testing are presented in [11-15]. But these are not online and consume more number of reconfigurations. In [19], a method is described in which two CLBs are loaded with same logic and same input is given to both. The outputs of these two CLBs are compared and the faulty interconnect is identified. This is a BIST based technique in which uses some parts of the system as test pattern generators and output response analyzers. The method employed in [20] divides the system in to clusters and does intracluster and extracluster testing separately.

In this paper we present an online testing algorithm that detects interconnect faults, locate single interconnect faults, and, performs rerouting to avoid the faulty interconnection that is located. To the best of our knowledge, this is the first known fault-tolerant online algorithm for interconnection fault detection, location and rerouting on SRAM-based FPGAs realizing multi-output combinational functions. The
number of reconfigurations of the LUTs in the LUT-network, which is a very important efficiency measure of the algorithm, is linear in the number of interconnections (less than or equal to 3 * number of interconnections) in the LUT-network.

4. Test methodology
As mentioned earlier, it is assumed that a network of LUTs realizing a subcircuit \( p \) of the given system is reported to be faulty by online checkers. We take advantage of the partial reconfiguration of FPGAs, by selective programming of the LUTs in the LUT-network. The algorithm presented in this paper programs the LUTs in one of the following two modes to detect and locate the interconnection fault.

Mode 1: Normal pass mode:
In this mode the LUT is loaded with a function, which passes one of its specified inputs to the output. The input to be passed is decided based on the testing requirements. The truth table for the normal test mode, which passes its first input to the output, is described below.

<table>
<thead>
<tr>
<th>Input</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1. Normal pass mode

Mode 2: Inverse pass mode:
In this mode the LUT is loaded to output the complement of one of its inputs. The truth table for the Inverse pass mode, which passes the inverse of its second input to the output, is shown below.

<table>
<thead>
<tr>
<th>Input</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. Inverse pass mode
As stated in [19], one can assume all the LUTs are tested and working correctly.

5. Algorithms
Algorithm 0 performs normal functional testing of the LUT network. Once a faulty output is detected, it locates any one of the output lines giving a faulty output, then, generates the corresponding cone [16], and passes it to Algorithm 1.

Algorithm 0
*****************************************************
** Input: An n-input k-output LUT network to be tested**
** Output: A faulty interconnect (if any)**

Online checkers generate input vectors and verifies the output vectors.

If (online checker returns error ) do
begin
  • Let the \( r^{th} \) output bit be faulty.
  • Generate the cone \( C \) corresponding to this \( r^{th} \) output line.
  • Test the cone \( C \) using Algorithm 1 and locate the faulty interconnect
end
*****************************************************

Given a cone, Algorithm 1 identifies a path \( P \) in the cone having the faulty interconnect along with the type of fault - stuck-at-1 (s-a-1) or stuck-at-0 (s-a-0) and passes \( P \) to Algorithm 2 along with the type of fault. The path \( P \) is a sequence of LUTs \( (L(1),L(2),...,L(q)) \), such that the output of \( L(i) \) drives the \( t_{i+1}^{th} \) input bit of \( L(i+1) \), \( 0\leq i < q+1 \). The \( t_{i+1}^{th} \) input bit of \( L(i+1) \) is called the appropriate bit of \( L(i+1) \) in the path \( P \). Note that \( t_1 \) is a primary input and \( t_q \) is the primary output of the LUT-network.

Algorithm 1
*****************************************************
** Input: Cone \( C \) with faulty interconnect**
** Output: A Faulty Interconnect**
Mark all interconnects as untested.
Perform the preprocessing step of method0
While (there exists untested interconnects) do
  • Choose an untested interconnect \( k \) and generate a path \( P \) from a primary input to the output of \( C \) that passes through \( k \) as suggested by Query mode of Method 0 that follows.
  • For all LUTs \( L \) in \( P \),
    o if (\( L \) is not already configured in normal pass mode or its appropriate bit in \( P \) is different from the appropriate bit in the path of the previous iteration) then configure \( L \) in normal pass mode to pass its appropriate bit in \( P \).
  • Apply a \( 0 \) to the primary input in \( P \).
  • If (primary output in \( P \) is 1) path \( P \) has an s-a-1 fault. Input the path \( P \) with s-a-
1 to Algorithm 2 and return the faulty interconnect.

- Apply a 1 to the primary input in P.
- If (primary output in P is 0) path P has an s-a-0 fault. Input the path P with s-a-0 to Algorithm 2 and return the faulty interconnect.
- Mark all untested interconnects on P as tested.

endwhile

Method 0

Input: An LUT Network in the form of a cone.
Output: Returning an untested interconnect k with a path P from primary input to primary output passing through k.

Preprocessing:
1. The LUT-network L, realizing a combinational function is a Directed Acyclic Graph (DAG), with every edge (interconnect) of it, oriented from its input end to its output end. Inverting the orientation of the edges of L also result in a DAG which we denote by L’. Do a Depth First Search (DFS) [21] of L’ such that at every LUT, which is a vertex of the LUT-network, the non-forward edges are visited before the forward-edges. Since L’ is a DAG, the DFS structure would not have any back-edges. Construct a sequence S of edges consisting of the primary inputs and the forward-edges in the order in which they were visited in the DFS.

Query Mode
1. Consider the next edge k in S. If k is a primary input, then P is the path from k to the primary output in the DFS tree. If k is a forward-edge (s,t), then, P is the path which is the concatenation of (the last reported path using t, (t,s), path from s to the primary output in the DFS tree).

In Algorithm 2 we use a binary search like method to rapidly identify the faulty interconnects in the path identified by Algorithm 1.

Algorithm 2

Input: Path P with a faulty interconnect and type of fault (s-a-0 or s-a-1)
Output: Faulty Interconnect
Let \( p' = P \)
While (true)

- If \(|p'|=1\) then \( p' \) is the faulty interconnect
  break;
- Let x be the LUT in the middle of path \( p' \) (refer Figure8)
- Configure x in inverse pass mode
- If (the fault inputted is s-a-0)
  o Pass 0 at the primary input
  o If (primary output=1) then \( p' \) is the path from primary input to x
  o If (primary output=0) then \( p' \) is the path from x to primary output
- If (the fault inputted is s-a-1)
  o Pass 1 at the primary input
  o If (primary output=1) then \( p' \) is the path from x to primary output
  o If (primary output=0) then \( p' \) is the path from primary input to x

endwhile

Once the fault is located, then we need to reroute the path through another switching element, which is fault free. We try to repair it if possible otherwise we disconnect that particular switch from the switch matrix. With this we can ensure that the faulty switch would not affect the functioning of the other subsystems, thereby making the system fault-tolerant.

Theorem: The maximum (worst-case) number of reconfigurations is upper bound by \( (3* \text{number of interconnects}) \).

Proof: The following properties follow from the Method 0, Algorithm 1, definition of the DFS and the fact that L’ in Method 0 is a DAG:

Property 1: All interconnects that are tree-edges of the DFS are automatically tested while testing the primary inputs.

Property 2: The LUT configured with the normal pass mode while passing an input p in Algorithm 1, is reconfigured again in normal pass mode to pass another input \( p' \) only after all the interconnects \( q \) that needs \( p \) to realize a path from \( q \) to the primary output are tested.

Property 3: Query mode of Method 0 ensures that while considering a forward-edge \( t \) there would already exist a path from the LUT at input-end of the edge \( t \) to a primary input such
that the LUTs in this path are already configured in normal pass mode and can be reused. This along with the fact that all the forward-edges of a LUT (which is due to fanouts) are considered after the non-fanout edges, ensures that every LUT is reconfigured in Algorithm 1 once for each of its inputs. Having identified a faulty path, Algorithm 2, reconfigures every LUT in the path to the inverse pass mode at most once and then, back to normal mode. Hence every LUT may be reconfigured at most three times. Hence the proof.

6. Example

Consider the FPGA as shown in Figure 9, having input lines I1 through I4 and output lines O1 through O4. Let us assume that interconnect between LUTs numbered 3 and 8 is stuck-at-1. This interconnect has been marked with a cross in the Figure 10. Online checkers generate input test vectors and check the output for errors. Let us assume that the online checker detects an error in the output line O1. We first construct the cone corresponding to this output line O1. The cone is shown in the Figure 10. Once we have extracted the cone, we apply Algorithm 1 to it. Initially, we mark all interconnects as untested and then construct various paths from input lines to output lines.
Let us assume Algorithm1 starts by choosing interconnect and generating the input-to-output path shown in figure11(a). All LUTs on this particular path should be in pass-mode to pass the relevant input to their output. Then the input line is fed with a 0 and the response is noted, if the response is 1, we can infer that in that path some interconnect is stuck-at-1.

A similar logic can be applied if we receive a 0 on sending a 1 down the line. From this stage we detect the path with a faulty interconnect and in addition, we also know the type of stuck fault. Algorithm 1 is executed till we detect a faulty path in the cone. Once we detect a faulty path we execute algorithm 2 on that path.

Algorithm 1

In the above example faulty path is shown in the figure11 (b). Now according to algorithm 2 we configure LUT number 19 in inverse pass mode and all other LUTs except 19 in normal pass mode. Since we know that the type of fault is stuck-at-1, we apply 1 to input line I3. If the response is 1, then lower part of the path is erroneous else upper part of the path is erroneous. We know that the faulty interconnect is in the upper part, so the response will be 0. Then, we configure LUT 8 in inverse pass mode and configure LUT 19 back to normal pass mode. Again feed the input line with 1, check the response. It will be 0 as the faulty interconnect will be in the upper part of this subpart. Again we configure LUT 8 back to normal mode and configure LUT 3 in inverse pass mode. The response will be 1, which means the fault is in the lower part of the subpath and hence we can infer that interconnect between LUT no 3 and LUT no 8 is erroneous, precisely stuck-at-1.

7. Results

In our simulation we generated LUT networks and, randomly inserted an interconnect fault and simulated our algorithm on the network. The results shown in table3 correspond to the stage after cone extraction. The table includes statistics till when the path was located. Once the path has been located, the faulty interconnect in the path is found quickly. Even if the length of the path is 50, we would need only log 50 which is less than 6 sessions to complete the final procedure.

The table shows the number of LUTs in the cone, number of interconnects in the cone, number of sessions held till when the faulty interconnect containing path was detected, and the total number of interconnects tested in those sessions. Note that we test only a fraction of total interconnects before locating the faulty interconnect.

8. Conclusion and Future work

The algorithm presented above deals with location of single stuck faults in a SRAM based FPGA systems. This method takes advantage of partial reconfiguration. After a fault is located, we reroute the path through another unused switching element. This is possible since a large number of interconnects will be unused when the application is loaded. We are currently working on extending these methods to sequential LUT networks with multiple interconnect faults.

9. References


---

### Table 3: Test results

<table>
<thead>
<tr>
<th># of LUTs</th>
<th># of Interconnects</th>
<th># of sessions</th>
<th># of reconfigurations</th>
<th># of tested interconnects</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>99</td>
<td>13</td>
<td>57</td>
<td>47</td>
</tr>
<tr>
<td>100</td>
<td>467</td>
<td>36</td>
<td>325</td>
<td>310</td>
</tr>
<tr>
<td>629</td>
<td>1813</td>
<td>116</td>
<td>1429</td>
<td>1362</td>
</tr>
<tr>
<td>427</td>
<td>812</td>
<td>51</td>
<td>623</td>
<td>573</td>
</tr>
<tr>
<td>124</td>
<td>331</td>
<td>19</td>
<td>145</td>
<td>123</td>
</tr>
<tr>
<td>879</td>
<td>3158</td>
<td>229</td>
<td>2948</td>
<td>2578</td>
</tr>
<tr>
<td>388</td>
<td>710</td>
<td>57</td>
<td>703</td>
<td>607</td>
</tr>
<tr>
<td>742</td>
<td>1209</td>
<td>89</td>
<td>1012</td>
<td>846</td>
</tr>
<tr>
<td>947</td>
<td>4347</td>
<td>219</td>
<td>2429</td>
<td>2297</td>
</tr>
</tbody>
</table>

---
