An Optimum ORA BIST for Multiple Fault FPGA Look-Up Table Testing

Armin Alaghi, Mahnaz Sadoughi Yarandi, Zainalabedin Navabi
Department of Electrical and Computer Engineering
University of Tehran, Iran
a.alaghi@ece.ut.ac.ir, m.sadoughi@ece.ut.ac.ir, navabi@ece.neu.edu

Abstract

This paper presents a BIST architecture for FPGA Look-Up Table testing using a minimum number of logic elements for its ORA. The propagation of faults in the TPGs and CUTs is formulated so that the ORA can detect multiple faults by monitoring a single signal. At the cost of using more cells for the ORA, the granularity of error detection can be reduced to as low as one fault per five LUTs. The increase in the ORA overhead, and thus the untested FPGA areas, can be compensated by more configurations. We will show that 100% test coverage and a maximum granularity can be achieved simultaneously by a reasonable number of FPGA configurations.

1. Introduction

Field Programmable Gate Arrays (FPGAs) have been widely used for rapid prototyping and manufacturing of complex digital systems, such as microprocessors and high speed telecommunication chips [1]. FPGAs are suitable for prototypes of systems [2] whose correct operation is necessary for the evaluation of new architectures. This requires changing the architecture during the design cycle with many reconfigurations of the same FPGA. The frequent reconfiguration of an FPGA makes it more fault-prone [3].

There are many components of an FPGA to test for ensuring reliable usage of this device. Algorithms for interconnect testing are discussed in References [4, 5]. In References [6, 7, 8] general test algorithms for Arrays of RAMs/LUTs has been proposed.

In this paper, we only consider test of LEs and focus on LUTs within LEs. There are different methods for LE testing. One may use I/O pins for applying test vectors to LEs and collecting test results [8, 9]. But, usage of I/O pins for test decreases the number of I/O pins available for normal operation. If detailed information for JTAG implementation was available, usage of JTAG pins as an interface to apply test vectors and retrieve LEs' results would be suitable [10].

A *Built-In-Self-Test* (BIST) architecture has been proposed for LEs testing [11], which eliminates the usage of I/O and JTAG pins. In this paper we address this approach for LUT testing of LEs. Our objective is to propose a BIST architecture with a good balance between various costs. Test time, test area and granularity are such trade-offs.

In this scheme, LUTs of the FPGA are partitioned into *Test Chains* in which some LUTs are configured as *Test Pattern Generators* (TPGs) and some as *Circuit Under Tests (*CUTs). The arrangement is introduced in Section 2. However, in our method while TPG is generating test vectors for CUT, LUTs configured as TPG are also tested. In Section 3 our test strategy is presented. Section 4 provides a mathematical justification for the design of the ORA. Section 5 shows its design and implementation results. In Section 6 alternative structures, such as ORA multiplexing and algorithms with more reconfiguration time, are proposed to evaluate the trade-offs between test area and test time. Finally, Section 7 presents the experimental results that show the optimized BIST structure as well as costs of alternative approaches that we have proposed in the preceding section. Conclusions are presented in Section 8.

2. BIST Architecture

Our proposed BIST architecture chains a group of LUTs of FPGA logic elements to form test pattern generators and circuits to test and their corresponding output analyzer. This BIST structure is shown in Figure 1. As shown, some of the LUTs are configured as the *Test Chain* (TC) and the others as the ORA. Note that ORA faults are not detected. A subdivision of the entire BIST logic, containing a single TC and its ORA, is referred as a *Partial BIST*. The overall structure covers all LEs of an FPGA.

15th Asian Test Symposium (ATS'06)
0-7695-2628-4/06 $20.00 © 2006 IEEE

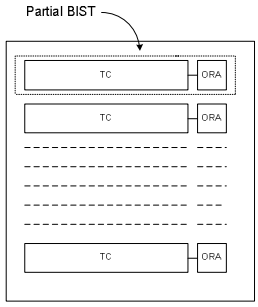

Figure 1. BIST architecture inside the FPGA

There are many alternative arrangements for TCs and ORAs in an FPGA [14]. Some of the parameters affected by this partitioning are test time, ORA size, test area, and the number of detected faults.

Test chains consist of *Partial Chains* (Figure 2) within which we have a *Test Pattern Generator* (TPG) and the *Circuit Under Test* (CUT).

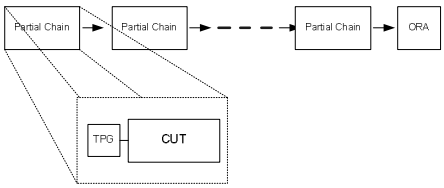

Figure 2. Chain Architecture

The TPG is an address generator that produces addresses 0 to 15. For LUT testing, the input address should cover the complete address space of the 4-input LUT. Output of the TPG directly drives LE0 and is treated as the address for its LUT. The output of the LE0 along with the three address bits of the TPG produces the address line of the next LE. Depending on the configuration, the arrangement of the address line varies. Address lines of the proceeding LUTs have the same arrangement relative to the preceding LEs. The *Partial Chain* structure is shown in Figure 3.

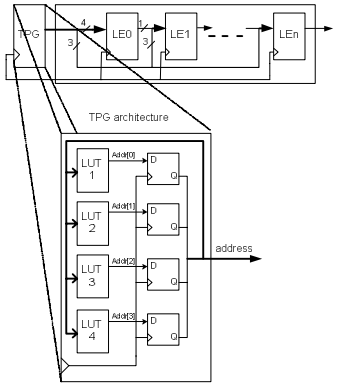

Figure 3. Partial Chain Architecture

3. Test Methodology

Algorithms for LUT testing need to write different test patterns into LUTs, read contents of LUTs and check the correctness of what has been read. Arrays of LUTs with n inputs can be tested with at least by $2n$ configurations [6].

These configurations for FPGA with 4-input LUTs are shown in Table 1 [12]. As shown, there are two groups of complementary patterns.

The table shows standard patterns for memory testing treating a 16-bit Look-Up Table as a 16-bit word to test. For reading this memory contents we address the LUT such that data read are from alternating 0 and 1 locations. The data read from an LUT is a periodic signal that is used for the clock input of the next Partial Chain. Being able to use arbitrary clocks for different LEs facilitates this procedure. Irregularities in the periodic signal indicate PC errors.

We use these configurations to detect single stuck-at, wrong cell read (write), no cell read (write) and additional cell read (write) faults in LUTs [12].

Table 1. Test Patterns

G1				*G2*			
C1	**C3**	**C5**	**C7**	**C2**	**C4**	**C6**	**C8**
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

With these configurations output signals toggle regularly, therefore it can be used as clock input of its succeeding *Partial Chain*. Hence, the clock pulse width will be multiplied by two after passing through each *Partial Chain*. Figure 4 shows fault free signals for clocks.

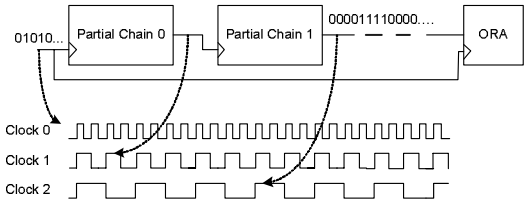

Figure 4. Fault Free Signal

4. Fault Propagation

Propagation of faults follows rules that can be derived mathematically. An overview of two propagated faults can be seen in the simulation result shown in Figure 5.

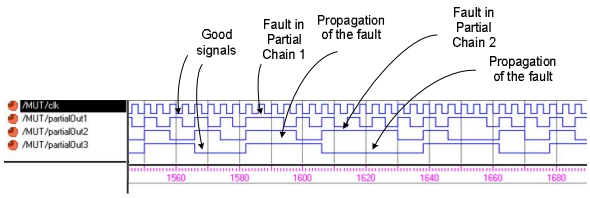

Figure 5. Simulation Result

As mentioned earlier, clock pulse width is multiplied by two after passing through each *Partial Chain*. Referring to "width" as the time between an edge to the next similar edge of a signal:

$$(w_r)_i = 2(w_r)_{i-1} = \ldots = 2^i w_0$$

In this equation i refers to the present *Partial Chain*, and i-1 to the previous *Partial Chain*. w_r stands for the reference output pulse width of each *Partial Chain*. We assume w_0 is the pulse width of the original clock (c_0) driving the first TPG.

4.1. Single Fault Rules

A single fault in any *Partial Chain* causes elimination of a pulse in the output of that *Partial Chain*. In effect, this extends width of a section of this output signal by w_c (corrupt width). Figure 6 shows the effect of w_c on the output of the *Partial Chain*.

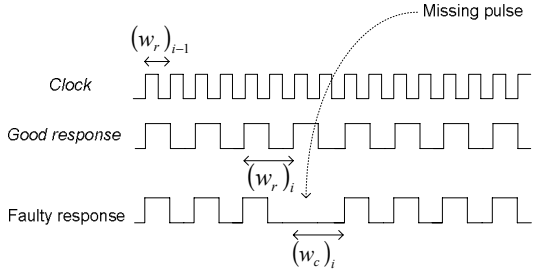

Figure 6. Fault Effect on the Response Signal

Provided that this *Partial Chain* (i^{th}) is the only faulty *Partial Chain* in the entire chain, the extra amount of w_c will be carried through all *Partial Chains* to the final output of the chain by the following expression:

$$(w_c)_i = 2(w_r)_{i-1} = 2^i w_0 \qquad (1)$$

Thus, the corrupt pulse width (w_c) depends on the *Partial Chain* in which the error occurs.

4.2. Multiple Fault Rules

For analysis of error and location of a fault in a *Partial Chain*, let us assume we have a chain of n *Partial Chains*, and an input frequency of f_0. In a good circuit, the pulse width of the input signal is $w_0 = 1/f_0$ and that of the last expected output is $w_n = 2^n.w_0$.

Multiple faults in different *Partial Chains* can be detected by examining the pulse width of this output signal. This output pulse width on the output resulting from multiple *Partial Chain* faults is an accumulation of all corrupt widths in *Partial Chains* leading to that output. Let w_f be this accumulated width. This parameter is calculated by:

$$(w_f)_n = \sum_{m=1}^{n} c_m.2^m.w_0 \qquad (2)$$

Where c_m = 1 if there is a fault in the mth *Partial Chain* and 0 otherwise.

The above expression can be explained by the discussion that follows. Locations of *Partial Chain* faults can be found by examining the final response of the chain, whose width becomes:

$$w_n = (w_r)_n + (w_f)_n$$

Where n is the number of *Partial Chains*. $(w_r)_n$ is the pulse width as expected at point n which is equal to $2^n w_0$, and $(w_f)_n$ is the width of accumulated errors at that point.

Propagation and accumulation of multiple faults is described by:

$$(w_f)_i = (w_f)_{i-1} + (w_c)_i \qquad (3)$$

which results in equation 2 for an entire chain.

4.3. ORA Requirements

Our proposed ORA is based on Equation 2. In this expression, c_m value of 1 identifies the location of a fault in the mth *Partial Chain*. c_m values form a binary number which is embedded in the pulse width of the test chain output and is extracted by our ORA to identify faulty *Partial Chains* and their locations. Figure 7 shows a typical chain with four *Partial Chains* and the ORA production.

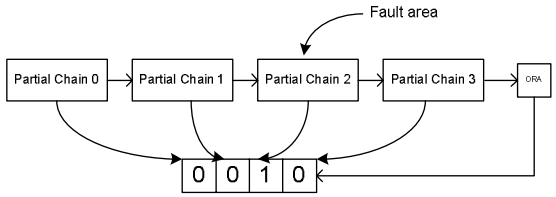

Figure 7. Fault Detection Areas

5. ORA Hardware

As mentioned in the previous section and shown in Figure 7, the ORA is placed after the last *Partial Chain*. The ORA's input signal is the *Response* signal

that carries the propagated faults in form of a widened pulse. The *Response* signal is expected to be a periodic signal and the faults appear as irregularities in this signal.

The ORA counts the width (an edge to a similar edge) of pulses on the *Response* signal by the original system clock (whose width is w_0). Depending on the number of *Partial Chains*, at least two counters are needed to detect *Partial Chain* faults. While the result of one of the counters is being registered, the other one counts the next *Response* width. The register accumulates the results of different widths by ORing them. This combines the previous results with the newer ones without losing any information. The ORA architecture is shown in Figure 8.

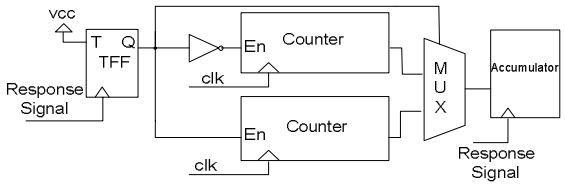

Figure 8. ORA Hardware Design

The number of *Partial Chains* affects the ORA hardware. Calculations and experimental results show that the optimum ORA can detect multiple faults in 4 partitions and since the initial TPG forms the first partition, the best number of *Partial Chains* becomes 3.

6. Alternative Structures

6.1. ORA Multiplexing

The general structure introduced earlier in Section 3 can be altered for special purposes. One idea is to change the partial BIST architecture by multiplexing an ORA between more than one test chains, as shown in Figure 9. By doing this, we can gain an expanded test area in the expense of increasing test time. Depending on the number of TCs detected by a single ORA (n), the test time increases n times.

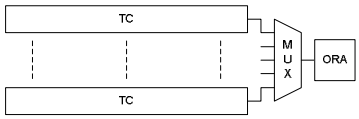

Figure 9. Alternative Partial BIST Structure

6.2. Reconfiguring the FPGA

Another option for improving testability of an FPGA is to use a large number of ORAs, and thus increase granularity of tested blocks. Recall that we can only detect one fault per *Partial Chain*. Making *Partial Chains* smaller gives us a better testability, but requires more ORAs.

Obviously the more ORAs cause a larger area of an FPGA to be left untested. This can be remedied by switching FPGA areas that are dedicated to TPG-CUT with those of the ORAs. Switching FPGA areas requires twice as many configurations, but results in 100% FPGA test coverage.

The drawback of this method is the long time it takes for FPGA reconfiguration. With our test method, typically, a configuration time is 10^6 times a test time.

7. Experimental Results

7.1. Implementation

Our proposed BIST has been synthesized and implemented on several Altera devices, such as FLEX10K, Cyclone, Stratix, ACEX1K and APEX20KE, using Altera Quatus II.

7.2. Optimum ORA

As mentioned before, the number of the *Partial Chains* affects the ORA hardware. Table 2 shows the results of synthesized ORAs for different number of *Partial Chains* on Altera FLEX10K device.

Table 2. Results of Synthesis on Altera FLEX10K

Number of Partial Chains	1	2	3	4	5	6	7
ORA Logic Cells	15	19	28	76	165	271	392

To find the optimum number of *Partial Chains* in a TC for the best ORA, we express the hardware overhead as:

$$Hardware\ Overhead = \frac{Number\ of\ LCs\ used\ for\ the\ ORA}{number\ of\ faults}$$

The "number of faults" which can be detected is same as the number of *Partial Chains*. Figure 10 shows the hardware overhead of the required ORA for different number of *Partial Chains*. This graph shows the best ORA is obtained when the number of *Partial Chains* is 3.

0-7695-2628-4/06 $20.00 © 2006 IEEE

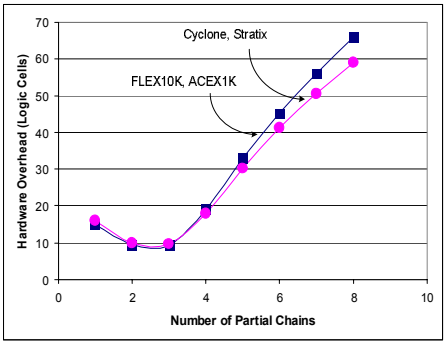

Figure 10. ORA Hardware Overhead for Different Number of Partial Chains

7.3. Test Area vs. Granularity

By changing the number of chains, we have a trade off between test area and granularity. We define the test area as the ratio of the tested area to the total FPGA area:

$$test\ area = \frac{TC\ Logic\ Cells}{TC\ Logic\ Cells + ORA\ Logic\ Cells} \times 100\%$$

And the granularity can be defined by the number of detectable faults in 1000 LUTs.

$$granularity(faults/1000 LUTs) = \frac{1000}{LUTs\ per\ Partial\ Chain}$$

In what follows we present our case study and experimental results using a FLEX10K FPGA. Features utilized in this FPGA are found in all more advanced Altera devices. For the "Number of *Partial Chains*" and "ORA logic cells" we use 3 and 28, respectively. This calculates the test area and granularity as shown in Table 3-a and 3-b.

By increasing the number of chains, we will need more ORAs and the test area will be decreased consequently. On the other hand, increasing the number of chains will make them smaller which results in less LUTs per *Partial Chain* and higher granularity. In other words, one can gain a high granularity by having a low test area and vice versa.

Table 3-a. Granularity vs. Test Area

Number of Chains	1	10	20	50	80	87
Test Area (%)	99.3	92.5	85	62.6	40.2	34.9
LUTs per PC	1239	115	53	16	6	5
Granularity	0.81	8.66	18.8	63.9	160	200

Table 3-b. Granularity vs. Test Area

LUTs per Partial Chain	5	50	100	200	400
Test Area (%)	34.9	84	91.4	95.5	97.7
Granularity	200	21	10	5	2.5
Number of Chains	87	64	11	6	3

The highest test area can be achieved by having only one chain, which results in having 1239 LUTs per *Partial Chain* and a low granularity (0.81 faults in 1000 LUTs).

The best granularity can be achieved by having only 5 LUTs per *Partial Chain* (4 LUTs for the TGP and 1 for the CUT), which results in having 87 chains and 34.9% test area.

7.4. Timing Analysis

After configuring the FPGA and resetting the circuit, we will have to wait a number of clock periods for the circuit to become stable, for the faults to propagate, and for the ORA to detect the fault. Experimental observations show that for 8 LUTs per *Partial Chain,* the worst case takes 46 clock pulses for the circuit to become stable, and 86 clock pulses for the ORA to detect the faults. The synthesis results show that the circuit clock frequency can be up to 500 MHz and consequently the total test time becomes 264ns ($2ns * (46 + 68) = 264ns$).

A typical configuration time for an Altera FPGA is about 250ms, which causes the test time become negligible [13].

7.5. ORA Multiplexing

By assigning more test chains to a single ORA, we can have a gain in the test area. This is because we have eliminated some untested areas by replacing the ORAs with CUTs. Instead, the test time will increase. However since the test time is negligible compared to the configuration time and can be ignored. Table 4 shows the gain in the test area by multiplexing ORAs between different numbers of TCs.

Table 4. Test Area Gained by ORA Multiplexing

Chains per ORA	1	5	10	20	50	100	154
Test Area%	46	81	89	94	97	99.6	99.6

Another possibility is breaking and multiplexing test chains for a fixed number of ORAs (Table 5). In this case no change in the test area will occur but, granularity can increase by having a slight increase in the test time.

Table 5. Granularity Gained by ORA Multiplexing

Chains per ORA	1	2	5	8	10
Granularity	18.8	53	94	150.8	188

7.6. Reconfiguring the FPGA

The second alternative introduced in the previous section can be used to have a 100% test area with a desired granularity. The main cost is the number of

FPGA reconfigurations, which is relatively high, and significant. Table 6 shows test area and granularity for different number of reconfigurations. Note that we need to have 8 different configurations for each LUT in order to completely test it. If we partition the FPGA into two halves, one for the TCs and the other for the ORAs, then testing the ORA parts by replacing them doubles the number of reconfiguration (i.e. 2x8).

Table 6. Granularity for Different Number of Reconfigurations

Number of FPGA Reconfigurations	1x8	2x8	3x8
Granularity (Faults/1000LUTs)	0.81	104.4	206.3

All the cases examined in Table 6 have a 100% test area. For the case of 3x8 configurations, the partitioning made the ORAs hardware twice as much as the TCs hardware (33% test area in 1x8 reconfigurations). By switching the location of the TCs and ORAs three times, a 100% test area can be achieved. The maximum possible granularity is 200 faults per 1000 LUTs since we can have the minimum of 5 LUTs per *Partial Chain*. Thus the maximum test area and granularity can be achieved simultaneously by having 3x8 number of reconfigurations.

8. Conclusion

This paper presented a BIST architecture for FPGAs Look-Up Table testing. The method is general and can be applied to most LUT based FPGAs. Our method is based on altering periodic signal pulse widths in the presence of Look-Up Table faults. Using this method, we have designed an on-chip ORA which can detect one fault for every group of LUTs that we refer to as *Partial Chains*. Although using FPGA for the ORA reduces the test area, a 100% test can be achieved.

There is a tradeoff between the ORA size and the number of faults that can be detected in an entire FPGA. We have found that a *Chain* size of three *Partial Chain*s yields the best ORA hardware. This BIST has been implemented in Altera FPGAs, and methods of programming the FPGA for the specific placement of the LUTs have been devised. Although the focus has been on LUTs, many logic faults between LUTs and LE Flip-flops are also detected. More work for FPGA RAM, and PLL testings is required to complete this methodology.

References:

[1] S. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, Boston, MA, 1992.

[2] M. Alderighi, E. Gummati, V. Piuri, and G. Sechi", A FPGA based Implementation of a Fault Tolerant Neural Architecture for Photon Identification," in Proc. of the International Symposium on Field-Programmable Gate Arrays, pp. 166 – 172, 1997.

[3] C. Metra et al., "Novel Technique for Testing FPGA", Design, Automation and Test in Europe, pp 89-94, 1998.

[4] M. Renovell, J. Figueras, Y. Zorian, "Test of RAM-based FPGA: Methodology and Application to the Interconnect", in Proc. 15th VLSI Test Symp., pp. 230-237, 1997.

[5] M. Renovell, "SRAM-Based FPGAs: A Structural Test Approach", IEEE XI Brazilian Symposium on Integrated Circuit Design SBCCI98, pp. 67-72, Oct. 1998.

[6] W.K. Huang, F.J. Meyer, N. Park, F. Lombardi. "Testing memory modules in SRAM-based configurable FPGAs," *mtdt*, p. 79, 1997 IEEE International Workshop on Memory Technology, Design and Testing (MTDT '97), 1997.

[7] X. Sun, J. Xu and P. Trouborst. "Testing Xilinx XC4000 Configurable Logic Blocks with Carry Logic Modules", in the Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, October 2001.

[8] M. Renovell, J.M. Portal, J. Figueras, Y. Zorian, "RAM-based FPGAs: a test approach for the configurable ", in proceedings of the Design, Automation and Test in Europe, 1998, pp.82-88.

[9] T. Inoue, et al., "Universal test complexity of field programmable gate arrays," in Proc. of Asian Test Symp., pp.259-265, 1995.

[10] C. Stroud, S. Wijesuriya, C. Hamilton, M. Abramovici, "Built-In Self-Test of FPGA Interconnect", in Proc. of the IEEE International Test Conference, pp. 404-411, 1998.

[11] C. Stroud, et al., "Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)," in Proc. of VLSI Test Symp., pp.387-392, 1996.

[12] E. Atoofian, Z. Navabi. "A BIST Architecture for FPGA Look-Up Table Testing Reduces Reconfigurations," *ats*, p. 84, 12th Asian Test Symposium (ATS'03), 2003.

[13] Altera Data Book, San Jose, USA, 2002.'

[14] M. Sadoughi Yarandi, A. Alaghi, Z. Navabi. "An Optimized BIST Architecture for FPGA Look-Up Table Testing" IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06), pp. 420-421, 2006.

[15] Quartus II Version 5.1, Handbook: Altera Corporation http://www.altera.com/literature/hb/qts