IPR: In-Place Reconfiguration for FPGA Fault Tolerance*

Zhe Feng[1], Yu Hu[1], Lei He[1] and Rupak Majumdar[2]
1. Electrical Engineering Department
2. Computer Science Department
University of California, Los Angeles

ABSTRACT

We describe *In-Place Reconfiguration (IPR)* for LUT-based FPGAs, an algorithm that maximizes identical configuration bits for complementary inputs of a LUT thereby reducing the propagation of faults seen at a pair of complementary inputs. Based on IPR, we develop a fault-tolerant logic resynthesis algorithm which decreases the circuit fault rate while preserving functionality and topology of the LUT-based logic network. Since the topology is preserved, the resynthesis algorithm can be applied post-layout and without changes in physical design. Compared to the state-of-the-art academic technology mapper Berkeley ABC, IPR reduces the relative fault rate by 48% and increases MTTF by 1.94× with the same area and performance, and IPR combined with a previous fault-tolerant logic resynthesis algorithm (ROSE) reduces the relative fault rate by 49% and increases MTTF by 2.40× with 19% less area but same performance. The above improvement assumes a stochastic single fault and more improvement is expected for multi-fault models.

1. INTRODUCTION

Compared to ASIC, FPGA is more vulnerable to soft errors due to a large number of SRAM cells used for configurability. This is not a critical concern when FPGA is used for prototypes, but it must be addressed when FPGA is used in system implementations such as internet routers. While robustness needs to be researched for different design stages of FPGA-based systems, this paper studies logic design and synthesis that explicitly accounts for and tolerates faults including soft errors.

For FPGAs, a *robust logic resynthesis* technique called ROSE has recently been proposed [1] as an effective design optimization for improving fault tolerance. This technique rewrites an LUT-based Boolean network and inserts logic masking to prevent the propagation of stochastic faults. It obtains 2× MTTF improvement with no area and performance overhead compared to the state-of-the-art FPGA logic synthesis tool ABC [2]. ROSE is orthogonal to the existing fault tolerance techniques such as [3, 4, 5, 6, 7], but with advantages of negligible overhead on area, performance, and testing. However, ROSE can change the topology of the LUT-based logic network, and as we explain next, this can limit its applicability in the overall design flow.

In the overall CAD flow for FPGAs (see Figure 1), logic resynthesis is usually performed after logic synthesis and before physical design. Because of dominant or non-negligible interconnect effects, the interaction of logic and physical syntheses must be considered and multiple iterations of logic and physical syntheses may be needed. For example, the modification of the LUT-based logic network due to resynthesis requires a new round of physical design, which is not only time-consuming, but also delays convergence between logic and physical syntheses. In addition, using more accurate fault information from physical design[1], logic resynthesis can selectively strengthen logic masking for more critical parts of a circuit. Unfortunately, a resynthesis that changes the topology of the LUT-based logic network is problematic for design closure as the criticality might change drastically in the next iteration of placement and routing. However, ROSE as well as the majority of existing fault tolerance techniques do not preserve the topology and therefore physical synthesis of the LUT-based logic network.

In this paper, we propose an *in-place* logic resynthesis algorithm, which performs logic transformation while preserving the function and the topology of the LUT-based logic network. Therefore, it does not require a redoing of the physical design and leads to a faster design closure. Our core algorithm *In-Place Reconfiguration* (IPR) maximizes identical configuration bits corresponding to complementary inputs of an LUT such that the faults seen at a pair of complementary inputs has less possibility of propagation and the overall reliability is optimized. IPR is iteratively carried out by simultaneously reconfiguring multiple adjacent LUTs without changing the function and topology of the LUT-based logic network. IPR applies to both combinational and sequential circuits. For sequential circuits, IPR applies to each combinational logic block.

Compared with ROSE, IPR scales to larger networks. IPR solves the LUT reconfiguration by a one-time Boolean satisfiability (SAT). In contrast, ROSE needs to solve a sequence of SAT problems and the length of the sequence is exponential to the number of configuration bits in an LUT. We observe over 50X speedup by IPR compared with ROSE in our experiments.

*This research is funded in part by the NSF award CCR-0306682. Please address comments to lhe@ee.ucla.edu.

ICCAD '09, November 1-4, 2009, San Jose, California, USA.

[1]While design closure for timing has been well understood, an example of reliability convergence could be caused by bridging faults, where the possibility of a bridging fault between two gates is high if their input wires use adjacent routing tracks but such information is unknown until physical synthesis.

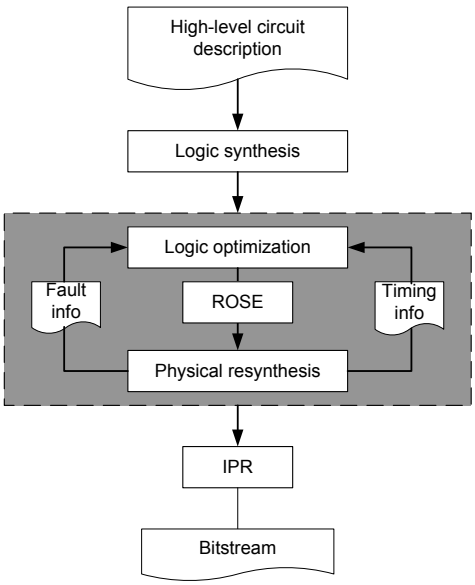

Figure 1: A simplified synthesis flow for FPGAs

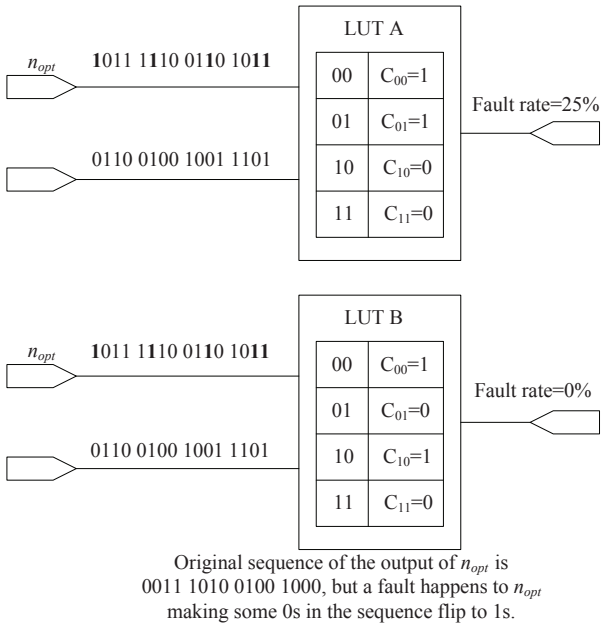

Figure 2: Motivation of IPR

Compared to the state-of-the-art academic technology mapper Berkeley ABC without considering faults explicitly, IPR reduces relative fault rate by 48% and increases MTTF by 1.94×. IPR is complementary to ROSE since we can first perform ROSE to increase the robustness before physical design and then carry out IPR as a post-physical design optimization for further reliability enhancement. Combining ROSE and IPR reduces the relative fault rate by 49% and increases MTTF by 2.40X compared with ABC. All the above improvement assumes a single fault, and more improvement can be expected for multi-fault models.

The remainder of this paper is organized as follows: Section 2 provide the in-place LUT reconfiguration algorithm. The experimental results are given in Section 3 and the paper is concluded with future research directions in Section 4.

2. ALGORITHM

2.1 Informal Overview

In-place resynthesis is a technique that optimizes a circuit after the placement and routing while preserving the results of physical design. The inherent flexibility of FPGAs makes in-place resynthesis particularly useful for various optimizations of FPGAs [8, 9]. Our algorithm introduces logic masking in the circuit using in-place LUT reconfiguration that reconfigures multiple LUTs simultaneously.

We represent a Boolean circuit after placement and routing as a DAG with LUTs as nodes and interconnects as edges [1]. Our resynthesis starts with a full-chip simulation to compute the logic signature and observability don't-care (ODC) mask using the techniques presented in [6]. The ODC mask is used to compute the *criticality* of each LUT, which is defined as the percentage of ones in the ODC mask. It is used as a measure of the contribution of the LUT to the circuit fault rate. The nodes are ordered in descending order of criticality. The algorithm iteratively selects the next node in the ordering and tries to reduce its criticality as follows. For each selected node n_{opt}, a cone (i.e., a logic block that includes multiple LUTs) containing n_{opt} is formed, and the LUTs inside the cone are reconfigured using an *in-place Boolean matching* that preserves both the logic function and the topology of the cone. The objective of the reconfiguration is maximize the logic masking to prevent the propagation of faults. After each iteration, the logic signature and ODC mask are updated incrementally. The details of the algorithm are described in the following subsections.

2.2 In-Place LUT Reconfiguration

In-place LUT reconfiguration is the key design freedom that we propose for our fault-tolerant resynthesis. As mentioned above, we try to maximize the "logic masking" in order to increase the robustness of a circuit w.r.t. faults. Specifically, we maximize the number of identical LUT configuration bits so that we can logically mask the faults originated upstream.

The logic output for an input vector of a LUT is specified by the configuration bit corresponding to the input, e.g., for 4-LUT, the input vector 0011 generates logic output 0 if the configuration bit c_{0011} is 0. Therefore, the input vector and the configuration bit have a one-to-one relationship. For an input pin i of a K-LUT, there are 2^{K-1} pairs of configuration bits associated with it, e.g., for a 2-LUT, the pairs (c_{00}, c_{10}) and (c_{01}, c_{11}) are pairs of configuration bits associated with input pin 1.

Figure 2 shows an example for the motivation of IPR. Consider the 2-LUTs A and B, and input sequences to A and B from a LUT n_{opt}. When a fault happens to n_{opt} making some 0s in n_{opt}'s output sequence flip to 1s, LUT A's output changes, while LUT B's does not. This is because LUT B's configuration pairs (c_{00}, c_{10}) and (c_{01}, c_{11}) each have the same logic outputs (1 for both c_{00}, c_{10} and 0 for both c_{01}, c_{11}) while LUT A's do not. Therefore, to reduce the propagation of faults from n_{opt}, intuitively, we want to have more identical pairs of configuration bits in a fanout LUT of n_{opt}. Naively, such reconfiguration most likely changes the function of an LUT (indeed, LUTs A and B implement different functions). Yet, it may be possible to reconfigure multiple LUTs simultaneously to maximize the number of identical pairs and at the same time, preserve the functionality and topology of the LUT-based logic network.

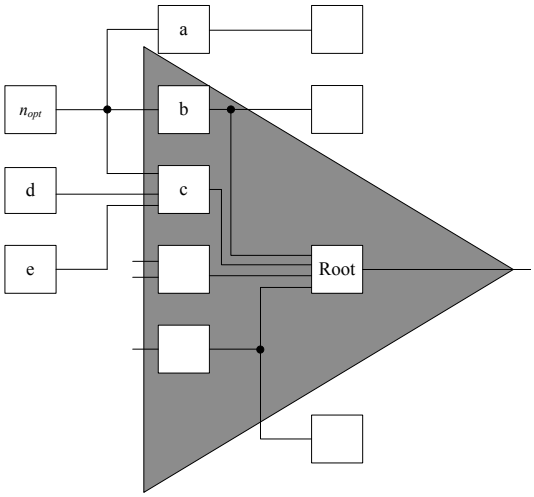

Figure 3: Illustration of cone construction

2.3 Criticality for Configuration Bit

Given the complementary inputs of an LUT, sometimes, we cannot set all the pairs of configuration bits identical. That setting which pairs of configuration bits identical is most beneficial becomes a problem. Therefore, we have to define the criticality of configuration bit. High priority should be given to the configuration bits which can mask more faults after setting as identical. Suppose $N_{sequence}$ denotes the length of the sequence of input vectors used for full-chip functional simulation, N_{vector} is the number of input vectors associated with the configuration bit c in the sequence, and $R_{tolerate}$ is the fraction of input vectors among the N_{vector} input vectors for which the fault is not propagated to the primary output when the input of the LUT is defected. $R_{tolerate}$ can be derived from ODC mask. 1-$R_{tolerate}$ represents the fault rate gap that the immediate fanout can fill. The criticality of configuration bit c of a LUT n can be formulated as follows:

$$Criticality\_bit(c) = \frac{N_{vector}}{N_{sequence}}(1 - R_{tolerate}) \qquad (1)$$

Equation (1) shows that the more there is room for the immediate fanout to optimize, the higher its criticality is.

2.4 Cone Construction

During the resynthesis algorithm, when a node n_{opt} is selected, a cone surrounding this node needs to be chosen for reconfiguration. Given $\mathcal{S}$, a subset of fanouts of n_{opt}, the cone construction routine builds a cone containing all LUTs in $\mathcal{S}$. In addition, the cone should not include all the first-order fanins for LUTs in $\mathcal{S}$, which guarantees that the criticality of configuration bits in $\mathcal{S}$ do not change after LUT reconfiguration. We ensure this since LUT reconfiguration may have reduced benefits if the criticality of these configuration bits is changed. Figure 3 shows an example of cone construction. LUTs a, b, and c are fanouts of n_{opt}. Suppose $\mathcal{S}$ contains only one fanout c. LUTs n_{opt}, d and e are first-degree fanins of LUT c. We choose one of fanouts of c as Root. From the Root, we construct the cone represented by the shaded triangle, and it contains the fanout c, and all the first-order fanins of LUT c, i.e., n_{opt}, d, and e, are not included in the cone.

2.5 In-place Boolean Matching

Given a cone C_F constructed based on the method presented in Subsection 2.4, we perform *in-place Boolean matching (IP-BM)* to check if we can reconfigure LUTs within the cone and while greedily maximizing identical pairs of configuration bits. If such a reconfiguration exists, IP-BM will return a set of feasible reconfigurations for all LUTs within the cone. Similar to the conventional Boolean matching, IP-BM preserves the logic function of the cone. In addition, IP-BM also preserves the topology of the cone, i.e., the interconnects among the LUTs within the cone do not change after IP-BM.

IP-BM is based on the SAT-based Boolean matching proposed in [10, 11, 12]. Suppose the cone C_F has m inputs $x_1, \cdots, x_m$, one output F, p LUTs: $L_0, \cdots, L_{p-1}$, and intermediate wires $z_1 \cdots z_p$. From the cone C_F, we can define a Boolean formula $\Psi(C_F)$ with free variables ranging over the configuration bits of the p LUTs such that a satisfying assignment to the formula (setting values to the configuration bits) ensures the topological structure and the functionality of the cone is preserved [10, 12]. To make a pair of configuration bits (c_i, c_j) in LUT L identical, we conjoin with $\Psi(C_F)$ the extra constraint $c_i \leftrightarrow c_j$ ensuring (c_i, c_j) is identical.

Our algorithm for IP-BM iteratively checks if

$$\Psi(C_F) \wedge \bigwedge_{(c_i,c_j)\in\mathcal{S}} c_i \leftrightarrow c_j \qquad (2)$$

is satisfiable for sets of pairs of configuration bits $\mathcal{S}_\mathcal{P}$, which is initialized as all the pairs of configuration bits of all the LUTs in $\mathcal{S}$, a subset of fanouts of n_{opt}. If (2) is satisfiable, there is a feasible reconfiguration of the cone such that all pairs of configuration bits in set $\mathcal{S}_\mathcal{P}$ can be set to identical values, and the configuration bits of LUTs can be obtained based on the satisfying assignment. Since the topology of the cone is constrained by the characteristic function, it is not changed after reconfiguration. If (2) is unsatisfiable, we reduce the size of set $\mathcal{S}_\mathcal{P}$ for the configuration bits that we want to make identical, and solve the IP-BM with fewer constraints until either a solution is found or all combinations of LUT configurations have been tried (i.e., $\mathcal{S}_\mathcal{P} = \emptyset$).

3. EXPERIMENTAL RESULTS

We have implemented IPR in C++ and used miniSAT2.0 [13] as the SAT solver. All experimental results are collected on a Ubuntu workstation with 2.6GHZ Xeon CPU and 2GB memory. We test our algorithms on QUIP benchmarks [14]. We assume that all configuration bits have an equal possibility to be defective during IPR optimization. For verification, the fault rate of the chip is the percentage of the primary input vectors that produce the defective outputs. We calculate the fault rate by Monte Carlo simulation with 20K iterations where one bit fault is randomly injected in each iteration for 1k input vectors.

As shown in Figure 4, we first map each benchmark by the Berkeley ABC mapper [2] for 4-LUTs, then perform and compare the following synthesis flows: (1) ABC followed by physical synthesis, VPR [15], without any defect-oriented logic resynthesis, (2) ABC followed by physical synthesis, and finally in-place optimization by IPR, and (3) ABC followed by ROSE and physical synthesis, and finally in-place optimization by IPR. In each synthesis flow, the logic depth produced by ABC is preserved. The number of configuration bits in the interconnects is extracted after the routing. Considering faults in configuration bits of both LUTs and interconnects, Monte Carlo simulation is performed to calculate the full-chip fault rate, with results summarized in Table 1.

In the Table, flow 1 is called *ABC*, flow 2 called *IPR* and flow 3 called *ROSE+IPR*. ROSE has two options, one using area-efficient templates to optimize both area and robustness, and the

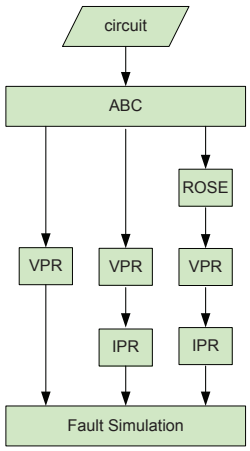

Figure 4: Experimental Flows

other using a path-reconvergent template for more robustness improvement with slightly more area. We use the area-efficient templates for flow 3 as it leads to better robustness and area for ROSE+IPR.

We use ABC as the baseline for comparison. One can see from the Table, IPR reduces the fault rate by 48% without area overhead. The best flow in terms of the robustness and area is ROSE+IPR, which reduces fault rate by 49% with 19% smaller area. Table 1 also reports MTTF. Because we assume that all the testing conditions are the same for ABC and IPR, MTTF is inversely proportional to the product of fault rate and area. Compared to ABC, IPR and ROSE+IPR increases MTTF by 1.94× and 2.40×, respectively.

In addition, compared to the case where ROSE is performed before placement and routing, the flows IPR and ROSE+IPR have better design closure between logic and physical synthesis as IPR is performed after placement and routing. This is important as configuration bits in FPGA interconnects dominate the total number of configuration bits, therefore pre-layout ROSE is less accurate or optimized.

QUIP Benchmark							
	Fault Rate			LUT#			Runtime
	ABC	IPR	ROSE +IPR	ABC	IPR	ROSE +IPR	IPR (s)
barrel64	2.02%	0.93%	0.94%	1931	1931	1484	19.46
fip_cordic_cla	1.92%	0.78%	0.79%	1042	1042	802	6.11
fip_cordic_rca	1.91%	0.83%	0.77%	981	981	751	5.71
mux8_128bit	5.37%	2.84%	2.74%	1923	1923	1796	1.52
oc_ata_ocidec1	2.98%	1.83%	1.80%	695	695	632	4.52
oc_ata_ocidec2	3.20%	1.69%	1.73%	840	840	742	4.89
oc_ata_v	2.15%	1.13%	1.11%	514	514	411	1.43
oc_dct_slow	1.55%	0.90%	0.85%	509	509	439	4.91
oc_des_area_opt	1.59%	0.98%	0.99%	1190	1190	1007	20.51
oc_des_des3area	1.68%	1.16%	1.23%	1782	1782	1479	39.64
oc_rtc	1.59%	0.72%	0.77%	879	879	591	3.90
oc_sdram	1.97%	0.89%	0.86%	729	729	553	1.89
os_sdram16	1.65%	0.79%	0.80%	947	947	719	5.28
GeoMean	2.12%	1.09%	1.09%	977.72	977.72	791.10	5.58
Ratio	1	0.52	0.51	1	1	0.81	
MTTF	1	1.94	2.40				

Table 1: Summary of experimental results

4. CONCLUSION

Our preliminary experiments with IPR are encouraging. IPR increases MTTF by 2× over ABC, the state-of-the-art academic technology mapper. More importantly, unlike its predecessors, IPR preserves the topology of the logic network for a faster design closure between physical and logic syntheses. In addition, IPR is complementary to existing fault-tolerant resynthesis algorithms. Combining IPR and ROSE, the best design flow for reliability and design closure is to perform pre-layout ROSE and post-layout IPR.

Although our experiments assume single fault, the proposed algorithm can deal with multiple uncorrelated faults, which will be explored in future work. We will also extend the proposed algorithm to consider given correlations between faults.

While our criticality calculation in this paper does not consider interconnect explicitly, IPR algorithm can be used without change if interconnects are taken into consideration for criticality. In addition, the current algorithm tolerates interconnect defects implicitly by modeling them as defects on outputs of an LUT. Our experiments in fact have taken into account both interconnect and LUT configuration bits for defects and have shown clearly improvement. In the future, we will extend IPR with criticality considering interconnects explicitly.

As our proposed IPR tends to increase logic masking to prevent the propagation of faults, it inevitably lowers the testability of a circuit. Particularly, IPR may produce some random pattern resistant (RPR) faults [16], which have very low detection probabilities and therefore are hard to detect in the random pattern-based testing. In this case, specifical treatments (e.g., BIST or logic re-synthesis [16]) can be performed for those RPR faults to enhance the testability.

5. REFERENCES

[1] Y. Hu, Z. .Feng, L. He, and R. Majumdar, "Robust FPGA resynthesis based on fault-tolerant boolean matching," in *ICCAD*, Nov 2008.

[2] "ABC: A system for sequential synthesis and verification," in *http://www.eecs.berkeley.edu/ alanmi/abc/*.

[3] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, 1962.

[4] Y. Lin and L. He, "Device and architecture cocurrent optimization for FPGA transient soft error rate," in *ICCAD*, 2007.

[5] L. Cheng, Y. Lin, L. He, and Y. Cao, "Trace-Based Framework for Concurrent Development of Process and FPGA Architecture Considering Process Variation and Reliability," in *FPGA*, 2008.

[6] S. Plaza, K.-H. Chang, I. Markov, and V. Bertacco, "Node mergers in the presence of don't cares," in *ASP-DAC*, pp. 414–419, 2007.

[7] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *ICCAD*, 2007.

[8] S. Gupta and J. Anderson, "Optimizing FPGA power with ISE design tools," *Xilinx.com*, 2007.

[9] J. Anderson and F. Najm, "Active leakage power optimization for FPGAs," *TCAD*, vol. 25, no. 3, 2006.

[10] A. Ling, D. Singh, and S. Brown, "FPGA technology mapping: a study of optimality," in *DAC*, 2005.

[11] J. Cong and K. Minkovich, "Improved SAT-based boolean matching using implicants for LUT-based FPGAs," in *FPGA*, 2007.

[12] Y. Hu, V. Shih, R. Majumdar, and L. He, "Exploiting symmetry in SAT-based boolean matching for heterogeneous FPGA technology mapping," in *ICCAD*, 2007.

[13] N. Een and N. Sorensso, *http://minisat.se/*.

[14] "Altera: QUIP for Quartus II V5.0," in *http://www.altera.com/education/univ/*.

[15] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *FPL*, 1997.

[16] N. A. Touba and E. J. McCluskey, "Automated logic synthesis of random pattern testable circuits," in *ITC*, 1994.