

# Power-Efficient and Fault-Tolerant Circuits and Systems

Lei He \* and Yu Hu

**Abstract** — *As devices become smaller, circuits and systems are more vulnerable to soft errors caused by radiation and other environmental upsets. Fault tolerance measured by mean time to failure (MTTF) is desired, especially if no extra area, power and delay and little change of the existing design flow are introduced. Using FPGA as a testbed, this paper first presents fault tolerance techniques applying (1) logic don't care and path re-convergence (ROSE) and (2) in-place logic re-writing (IPR). Both increase MTTF by 2X with little or no overhead. Particularly, IPR does not change circuit placement and routing, and can be readily used with the existing industrial design flow. It also leads to a self evolution method to enhance fault tolerance for FPGA based circuits and systems. The ideas presented in the paper can be extend to handle regular logic fabrics, which are natural to nano-technologies and are also preferred by design for manufacturability (DFM) in scaled CMOS technologies<sup>1</sup>.*

**Index Terms** — Field Programmable Gate Array, Fault Tolerance, Logic Synthesis

## I. INTRODUCTION

As devices scale to nanometer regime, circuits and systems are more vulnerable to soft errors caused by radiation and other environmental upsets. Fault tolerance measured by mean time to failure (MTTF) is desired, especially if no extra area, power and delay and minimal change of the existing design flow are introduced. Unfortunately, for those mission-critical applications, e.g., aerospace and military applications, rigorous requirement for robustness is often coupled with excessive redundancy, resulting in area, power and delay overhead. On the other hand, mission-non-critical applications (e.g., enterprise servers and internet routers) can tolerate certain level of soft errors using software and protocol redundancy (e.g., resend a packet if data are corrupted in an internet router) or system reloading. However, "critical" faults may cause frequent system reloading or even system breakdown, which result in a reduced MTTF.

In this paper, we study the fault tolerance for mission-non-critical applications, and use SRAM-based field programmable gate array (FPGAs) as the testbed. For fault-tolerant FPGA designs, both error detection and correction are important. Various fault-tolerant encoding schemes (e.g., error-correction-code (ECC)) have been applied to commercial SARM-based FPGAs, e.g., Xilinx Virtex-5, to detect or correct soft error-induced bit flips or data corruption. However, they cannot be applied to in real-time due to the

severe latency caused by the slow scan-based readback (the longest readback CRC scan time could be up to 100ms [1]). In this paper, we focus on the CAD approaches applicable to any existing FPGAs to optimize the robustness of an FPGA-based circuit and system with minimal overhead for area, delay and power, and therefore they can be applied to real-time applications for an increased MTTF.

Specifically, two fault-tolerant logic synthesis algorithms are presented, namely robust resynthesis (ROSE) and in-place reconfiguration (IPR). Both algorithms leverage logic masking to reduce the probability that a fault is propagated to the primary outputs. In addition to logic masking, ROSE also uses a robust logic template (with path re-convergence) that can be realized by any LUT-based FPGAs to further strengthen the robustness. ROSE and IPR have complementary features. ROSE reduces logic area, and therefore effectively produces designs with lower power. IPR preserves the topology (or layout) of the netlist, and therefore requires no re-placement and re-routing, resulting in minimal change of the existing design flow and a faster design closure. It also leads to a self evolution method to enhance fault tolerance for FPGA-based circuits and systems. Both ROSE and IPR have about 2X MTTF increase on the QUIP benchmark set [12].

We show that the proposed approaches can be extended to consider other optimization objectives (e.g., leakage power minimization), and handle regular logic fabrics, which are natural to nano-technologies and are also preferred by design for manufacturability (DFM) in scaled CMOS technologies. Particularly, IPR can be applied in post-silicon debugging (e.g. [24]) and engineering change order (ECO) for a quicker timing closure.

The remaining of this paper is organized as follows. Section II and Section III briefly reviews the previous fault-tolerant techniques for PLDs, and provides the preliminaries, respectively. Section IV and Section V describe the proposed fault-tolerant synthesis algorithms, ROSE and IPR, respectively. A hardware-based emulation for optimization and validation is presented Section VI. Some experimental results are highlighted in Section VII, and the paper is concluded in Section VIII.

## II. PREVIOUS WORK

Fault tolerance techniques have been studied extensively for PLDs [1]. Without considering dynamic re-configuration during runtime, the following techniques have been developed to tolerate faults for PLDs: (a) Locating and masking faults by circuit redundancy. For example, column-based redundancy, proposed in [2, 3], has been used in Altera's Stratix II FPGA [4]. If one logic block in a column of logic blocks is found defective during testing of the device, the entire column is

<sup>1</sup> Both Lei He and Yu Hu are with the Electrical Engineering Department, University of California Los Angeles (UCLA), Los Angeles, CA 90066, USA. (e-mail: {lhe,hu}@ee.ucla.edu).

bypassed and its function is implemented by the redundant column. Besides redundant column and rows, some fine-grained redundancy architectures were also proposed, e.g., in [5, 6], where redundant routing resources are evenly distributed in the FPGA to tolerate faults. The aforementioned tolerance is transparent to FPGA users, and the same synthesis can be used for all chips of the same FPGA application. This manufacturer-masking approach lowers synthesis cost for massive production, but suffers from low fault coverage, large area overhead, and extra delay due to the bypass circuit. For example, only defective logic blocks within the same column are tolerated with one extra column as in Stratix II. (b) Chip-wise synthesis, which has been applied to circuits with high fault rates, especially for nano-technologies [7, 8, 9]. Here, each fault is located, and then placement and routing is customized for each chip in order to work around faults. Chip-wise synthesis is not suitable for massive production of one FPGA application, and testing costs could be intolerably high for a large number of faults, although there is active research in reducing the testing cost. (c) Triple-modular redundancy (TMR) [10]. Compared to the previous two approaches ((a) and (b)), TMR does not require to locate faults during synthesis and it can tolerate transient soft errors. However it has the practically highest overhead on area, power and performance. (d) Multiple configurations. EasyPath by Xilinx, pre-develops multiple synthesis solutions for an FPGA application. During testing, each chip chooses a synthesis that can tolerate manufacturing defects for the particular application. Compared to chip-wise synthesis, multiple configurations reduce testing and synthesis costs. Compared to TMR, multi-configuration has reduced circuit overhead but cannot tolerate transient soft errors. Thus, existing techniques suffer from either expensive testing overhead, excessive overhead on performance, power and area, long design time, or a low fault coverage rate.

### III. PRELIMINARIES

#### A. Boolean Network

A logic template  $H$  consists of a network of interconnected logic devices with a set of input pins and an output pin. A  $K$ -LUT is an LUT with  $K$  inputs, one output, and  $2^K$  LUT configuration bits.

An LUT-based Boolean network is represented using a directed acyclic graph (DAG) whose nodes correspond to LUTs and directed edges correspond to wires connecting the LUTs. The nodes in the lowest level of the DAG are called circuit inputs (CIs), which include the primary inputs (PIs) and the outputs of registers. The nodes in the highest level are called circuit outputs (COs), which include primary outputs (POs) and the inputs to registers.

A fanin (resp. fanout) cone of node  $n$  is a sub-network whose nodes can reach the fanin edges of  $n$  (resp. can be reached from the fanout edges of  $n$ ). A maximum fanout free cone (MFFC) of node  $n$  is a subset of the fanin cone such that every path from a node in the subset to the CO passes through  $n$ . Informally, the MFFC of a node contains all the logic used

exclusively by the node. When a node is removed or substituted, its MFFC can be removed.

A cut  $C$  of node  $n$  is a set of nodes of the network such that each path from a CI to  $n$  passes through at least one node in  $C$ ; node  $n$  is called the root of cut  $C$ . A cut is  $K$ -feasible if the number of nodes in it does not exceed  $K$ . A logic block is a sub-network which covers all nodes found on the path from the outputs (called root nodes of the logic block) to the cut, including the roots and excluding the cut. In this paper, we consider multi-input, singleoutput (MISO) logic blocks, but the proposed algorithm can be applied to multi-output, multi-output (MIMO) logic blocks [13], as well.

#### B. Boolean Matching

Given a logic template  $H$  and a Boolean function  $F$ , the Boolean matching problem (BM) either maps function  $F$  to logic template  $H$  by describing an appropriate setting of the LUT configuration bits, or concludes that logic template  $H$  cannot implement function  $F$ . Boolean matching [14] is one of the most important sub-problems in logic synthesis and technology mapping for FPGAs.

The Boolean matching problem can be formulated as a (quantified) Boolean satisfiability problem in the following way [15]. Consider a logic template  $H$  with inputs  $x'_1, \dots, x'_k$ , output  $G$ , intermediate wires  $z_1, \dots, z_m$ , and LUT configuration  $c_1, \dots, c_n$ . Let  $F$  be a Boolean function of  $k$  inputs, given as a truth table.

We can write a set of Boolean constraints that define each internal and output wire of  $H$  in terms of its inputs (see, e.g., [15]). For example, the internal wire  $z_l$  for a 4-LUT can be defined as

$$\begin{aligned} &(\overline{x'_1} \wedge \overline{x'_2} \wedge \overline{x'_3} \wedge \overline{x'_4} \rightarrow (z_l \leftrightarrow c_0)) \wedge \dots \wedge \\ &(x'_1 \wedge x'_2 \wedge x'_3 \wedge x'_4 \rightarrow (z_l \leftrightarrow c_{15})) \end{aligned}$$

Let  $\Psi(H)$  be the conjunction of constraints defining each wire of  $H$ . Similarly, the truth table for function  $F$  can be expressed as a set of constraints between the input variables  $x_1, \dots, x_k$  and the output  $F$ :

$$\begin{aligned} \Psi(F) = &(\overline{x_1} \wedge \overline{x_2} \wedge \dots \wedge \overline{x_k} \rightarrow F_0) \wedge \\ &(x_1 \wedge x_2 \wedge \dots \wedge x_k \rightarrow F_1) \wedge \dots \wedge \\ &(x_1 \wedge x_2 \wedge \dots \wedge x_k \rightarrow F_{2^k-1}), \quad (1) \end{aligned}$$

where  $F_i = F$  if  $F(i) = 1$ , otherwise,  $F_i = \overline{F}$ .

The Boolean matching problem for  $(H, F)$  can then be expressed as the quantified Boolean formula problem that asks, does there exist some setting of the LUT configuration  $c_1, \dots, c_n$  such that for all inputs  $x_1, \dots, x_k$ , the output  $G$  of  $H$  is equivalent to  $F$ ? Formally, we ask:

$$\exists c_1 \dots c_n \forall x_1 \dots x_k \exists z_1 \dots z_m \Psi(H) \wedge \Psi(F) \wedge (G \leftrightarrow F) \quad (2)$$

By replicating formula (2) for each possible valuation to the bits  $x_1, \dots, x_k$ , we reduce the quantified formula to an (existential) satisfiability problem. Each satisfying assignment gives an instantiation of the LUT configuration bits that implement the same function  $F$ .

#### C. Fault Model

In the presence of faults in the LUT configurations or intermediate wires between LUTs, we extend the Boolean matching algorithm in the following way. We model faults in

LUT configurations and the faults in intermediate wires as random variables, and assume that the probability that an LUT configuration bit or an intermediate wire is defective is known. Under these fault sources, the fault rate of a circuit is the percentage of primary input vectors under which the circuit does not produce the desired logic output values. While we assume single fault in our experiments, our algorithms allows multiple faults to occur simultaneously.

#### D. Logic Don't-cares

Logic don't-cares arise from both satisfiability don't-cares (SDCs) due to some combinations not being produced as input vectors of a node, and observability don't-cares (ODCs) because under some conditions, the output value of a node does not propagate to the COs (i.e., the output value is controlled by certain input vectors) [16].

In this paper, we use the ODC mask [17] to represent the ODCs of the maximum fanout cone of a node. The ODC mask of node  $n$  is defined as follows.

**Definition 1 (ODC Mask)** Let  $\langle X_1, \dots, X_K \rangle$  be a sequence of  $K$  input vectors for node  $n$ . The ODC mask of  $n$ , written  $ODCmask(n)$ , is a  $K$ -bit sequence where  $i$ th bit is 0 if the input vector  $X_i$  is in the don't-care set of  $n$ ; otherwise, the  $i$ th bit is 1. Formally,  $ODCmask(n) \in \{0, 1\}^K$  such that  $ODCmask(n)_i \equiv X_i \notin ODC(n)$ , where  $ODC(n)$  is the don't-care set of  $n$ .

The ODC mask quantifies the impact of the node on the primary output. Given the definition of the ODC mask, we can define the *criticality* of node  $n$  as

$$Criticality\_Node(n) = \frac{\sum_{i=1}^K X_i \in ODC(n)}{K}$$

where  $K$  is the number of input vectors.

## IV. ROSE ALGORITHM

The fault rate of a circuit is impacted by both the synthesis algorithm and the topological structure of the implementation. In this section, we first describe the overall flow of our proposed robust resynthesis algorithm ROSE, and then present a robust logic template which enables better fault tolerance with ROSE.

#### A. Overall Algorithm of ROSE

Our procedure takes an application mapped to  $K$ -LUTs and scans the combinational portion of the circuit in topological order from primary inputs to primary outputs. In the course of scanning, new logic blocks are generated by combining the logic blocks at the input LUTs. Each logic block is mapped against one or more pre-defined logic templates; if a mapping with the minimal fault rate is found by FTBM (fault-tolerant Boolean matching), the logic block can be substituted by the logic template. However, any substitution that increases the local logic depth or area is discarded. This ensures that the logic depth and area does not increase. In our implementation, only MFFCs are considered as candidates for mapping.

As the resynthesis of a logic block will change the fault rate of its output and therefore change the fault rates observable by the inputs of the downstream network, ROSE processes all MFFCs in a topological order (from CIs to COs) to guarantee

that the input fault rates of a logic block have been correctly updated before the block is resynthesized. To calculate the fault rate for a logic block, both faults in LUT configurations and the inputs of the block need to be considered. After resynthesis, we can obtain the fault rate of the block output and need to update the fault rates for all downstream intermediate pins under the fanout cone of the block output.

#### B. Robustness of Logic Templates

Besides an effective robust resynthesis algorithm, it is also important to find an effective logic template for fault tolerance, because different templates may have significantly different capability of carrying fault tolerance and therefore they can pre-determine the potential of the effectiveness of FTBM.

We consider Boolean functions with up to 10 inputs. According to [12], there are three possible logic templates with no-more-than three 4-LUTs to implement a Boolean function with up to 10 inputs (see Figure 4 (a), (b) and (c) in [18]). The inherent disadvantage of these area efficient logic templates is the lack of opportunities to place don't-cares, which are the major source of logic masking and fault mitigation. Inspired by a well-known observation that reconvergence is a prime reason for don't-cares, we propose a new logic template, R-PLB, as shown in Figure 1, which requires four 4-LUTs and forms re-convergent paths from input to output. It has been shown that R-PLB can carry both SDCs and ODCs. Note that R-PLB can be realized by any LUT-based FPGAs.

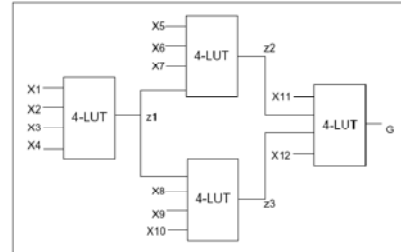


Fig. 1. R-PLB: a robust logic template for more logic don't-cares.

#### C. Fault-Tolerant Boolean Matching (FTBM)

We now describe an algorithm for FTBM, which is the core of ROSE, and discuss implementation issues. Recall the CNF encoding procedure described in Section III.B, after solving (2), a set of LUT configurations  $c_1, \dots, c_n$  will be returned by the SAT solver if  $F$  can be implemented by  $H$ . There might exist multiple distinct implementations (i.e., different configurations) for  $H$  all of which implement  $F$ . In fact, we can obtain partial or even all feasible configurations by iteratively adding the negation of previously obtained configurations into the CNFs and solving an augmented SAT problem. For each of these feasible configurations,  $C = (c_1, \dots, c_n)$ , we evaluate the fault rate at the output of this logic block under this configuration setting. The configuration,  $C^*$ , which results in the minimal fault rate, is chosen as the candidate for mapping or resynthesis. The fault rate calculation in FTBM can be solved by functional simulation for single fault or stochastic SAT [18] for multiple faults. Interested readers are

referred to [18] for further details.

## V. IPR ALGORITHM

### A. Motivation of IPR

Before explaining the motivation of IPR algorithm, the follows are some notions which are frequently used in the rest of the section. An input vector of an LUT has a logic output specified by a configuration bit, e.g., for 4-LUT, input vector 0011 generates logic output 0 if the configuration bit  $c_{0011}$  is 0. For an input pin  $i$ , a  $K$ -LUT has  $2^K-1$  pairs of configuration bits associated with it. E.g., for a 2-LUT, both pair  $(c_{00}, c_{10})$ , and pair  $(c_{01}, c_{11})$  are pairs of configuration bits associated with input pin 1. In the rest of the section, without specific declaration, let the node under optimization be  $n_{opt}$ , and pairs of configuration bits refer to the ones in the fanout LUT driven by  $n_{opt}$ .

Figure 2 is an example for the motivation of IPR. When a fault happens to  $n_{opt}$  making some 0's in  $n_{opt}$ 's output sequence flip to 1s, LUT A cannot tolerate any fault, while LUT B can tolerate all because its configuration pairs  $(c_{00}, c_{10})$  and  $(c_{01}, c_{11})$  are the same to each pair. We call a pair of configuration bits with a same configuration value as a *symmetric pair of configuration*. Therefore, to reduce propagation of fault from  $n_{opt}$ , intuitively, we want to have more symmetric pairs of configurations in the fanouts of  $n_{opt}$ . However, such reconfiguring most likely changes the function of an LUT. Yet, it may be possible to reconfigure multiple LUTs simultaneously to maximize the number of symmetric pairs and at the same time, preserve the functions and topology of the LUT-based logic network.

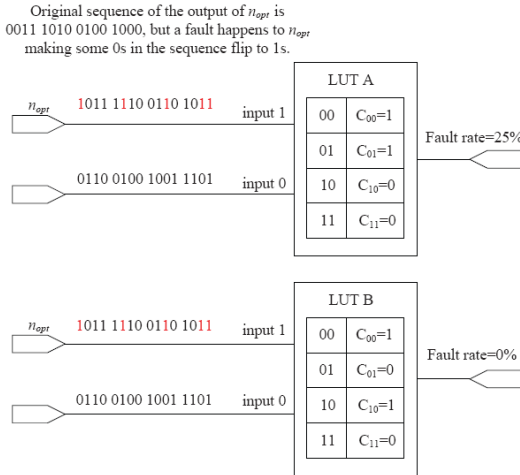


Fig. 2. Motivation example of IPR.

### B. Overall Algorithm of IPR

Algorithm 1 is the overview of the IPR algorithm. First, the criticalities for each pair of configuration bits for all the fanouts of  $n_{opt}$  are calculated. Let the set of all the fanouts of  $n_{opt}$  be  $S_{FO}$ , and  $Powerset(S_{FO})$  be the power set for  $S_{FO}$ , i.e., it includes all the sets consisting of all combinations for fanouts of  $n_{opt}$ . Also, let fanouts  $S_N$  be an element of  $Powerset(S_{FO})$ . We process each  $S_N$  according to the descendant order of its size. For each  $S_N$ ,  $S_P$  is initialized as all the pairs of

configuration bits of all fanouts in  $S_N$ . Then we iteratively search a feasible cone containing  $S_N$  by function *constructCone* ( $S_N$ ). We check whether there is an LUT reconfiguration for all LUTs in the cone without changing the function and topology of the LUT-based cone after making all the pairs in  $S_P$  symmetric by function Boolean Matching ( $S_P$ ,  $C_F$ ). If so, the new LUT configuration is applied and IPR for  $n_{opt}$  is terminated. Otherwise,  $P_{least}$ , the pair of configuration bits with the least criticality in  $S_P$  is deleted from  $S_P$ , and a new round is invoked with the new  $S_P$ . Finally, we terminate the IPR algorithm when the size of  $S_N$  is 1. Because when there is only one fanout of  $n_{opt}$  to be reconfigured, either the fanout can already tolerate the fault from  $n_{opt}$ , or there is no valid solution.

#### Algorithm 1 IPR( $n_{opt}$ )

```

1: Calculate the criticalities for each pair of configuration bits for
   all the fanouts of  $n_{opt}$ ;
2:  $Valid \leftarrow false$ ;
3: repeat
4:   For each  $S_N \in Powerset(S_{FO})$  according to the descendant
   order of  $|S_N|$  { //  $S_{FO}$  is the set of all the fanouts of  $n_{opt}$  }
5:    $S_P \leftarrow$  All the pairs of configuration bits of all fanouts in  $S_N$ ;
6:   while  $S_P \neq \Phi$  do
7:      $C_F \leftarrow constructCone(S_N)$ ;
8:      $Valid \leftarrow booleanMatching(S_P, C_F)$ ;
9:     if  $Valid$  then
10:      break;
11:    else
12:       $S_P = S_P - P_{least}$ ;
13: until  $|S_N| == 1$  ||  $Valid == true$ 

```

Fig. 3. Pseudo-code of IPR algorithm.

The calculation of the criticality of the configuration bits can be carried out in the similar way presented in Section III.D based on ODC masking and the functional simulation. The Boolean matching in IPR again is based on the SAT-based BM presented in Section III.B with the addition of the following CNF constraints for each try:

$$c_i \leftrightarrow c_j, \quad (5)$$

which makes a pair of configuration bits  $(c_i, c_j)$  in an LUT symmetric. For a detailed description of IPR algorithm, interested readers are referred to [19].

## VI. EMULATION-BASED VALIDATION AND OPTIMIZATION

Simulation is used in both ROSE and IPR for the following tasks, i.e., (a) criticality calculation during optimization and (b) validation of the full-chip fault rate after optimization. The software-based simulation is timing consuming and not practical for large circuits. To improve the runtime efficiency of the simulation in the fault-tolerant synthesis, we use hardware emulation-based simulation using FPGAs. In the following, we present two types of emulators with complementary features.

### A. Virtual FPGA-based Emulator

The virtual FPGA-based emulator uses the existing block RAMs in an FPGA to implement an abstract of a circuit in logic-level for technology in-dependent optimization. Using *virtual LUTs* as the building block (see Figure 4), it stores the configuration bits for LUTs, FFs and other reconfigurable elements. One can dynamically reconfigure a specific LUT or FF by change the proper address in the block RAM. This approach enables dynamic reconfiguration without the

physical support from FPGA vendors. In addition, it allows one to locate and reconfigure a specific element (LUT or FF) in a logic netlist without the knowledge of the physical FPGA architecture. Figure 4 shows the schematic of the virtual FPGA-based emulator, which can be implemented in RTL and then synthesized to any FPGAs as long as the area fits.

Although it is flexible and easy to implement, the virtual FPGA-based emulator has the following drawbacks. (a) It

requires about 5x area overhead compared with the original circuit; (b) Interconnect faults cannot be simulated directly as it is only a logic level and technology independent abstraction of the circuit under optimization. Therefore, this approach is mainly used to calculate (or update) the criticality during the course of optimization.

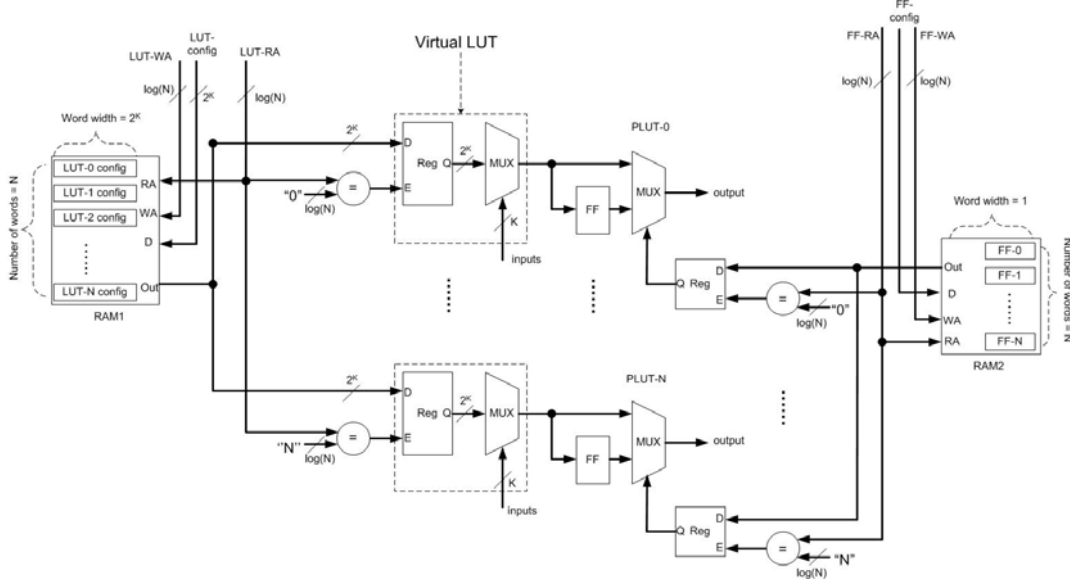


Fig. 4. Schematic for the virtual FPGA-based emulator.

### B. Partial Reconfiguration-based Emulator

Partial reconfiguration-based emulator uses the (partially) dynamic reconfiguration feature provided by FPGA vendors. Particularly, Xilinx XAPP864 [11] allows one to set an address of a configuration bit, and flip that bit during the runtime. In such a way, one can inject faults over the full-chip without rerun the CAD flow. However, due to the IP issues, FPGA vendors do not provide APIs to locate and inject faults in a specific LUT, i.e., a link between logic-level netlist and the physical layout in the bitstream-level is missing. Therefore, the partial reconfiguration-based emulator cannot be applied to compute the criticality for a specific LUT. Instead, it is mostly suited to be used to compute the full-chip fault rate in the post-optimization stage, and it can easily take into account faults in interconnect and other heterogeneous components (e.g., DSP and RAM) for a more accurate estimation of the full-chip fault rate.

Combining the emulator-based simulation and the proposed fault-tolerant synthesis, we can build a new robust synthesis paradigm, called *self-evolution system*, which dynamically changes implementation for a better fault tolerance. Under such a paradigm, we can study the system-level vulnerability [23], including the predication and mitigation of errors, by linking the system-level vulnerability to the criticality of a configuration bit in the circuit level.

## VII. EXPERIMENTAL RESULTS

We have implemented both ROSE and IPR in C++ and used

miniSAT2.0 [20] 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 [12]. We assume that all configuration bits have an equal possibility to be defective, and only a single fault occurs at the same time. 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.

Figure 5 shows the CAD flow used in our experiments. We first map each benchmark by the Berkeley ABC mapper [22] for 4-LUTs, then perform and compare the following synthesis flows: (1) ABC followed by physical synthesis, VPR [21], 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.

The experimental results are summarized in Figure 6. In terms of fault rate, all these three flows give similar reduction of fault rate, i.e., about 50% less than the synthesized circuit resulted from ABC. While reducing fault rate, ROSE also reduces area (i.e., LUT number) by 20%, compared to ABC,

which further increases the MTTF. In terms of runtime, IPR is about 50x faster than ROSE. A combination of ROSE and IPR (flow (3)) gives a 2X MTTF improvement. Depending on the MTTF, area and runtime requirement for a specific design, one can use the proper CAD flow of ROSE and IPR.

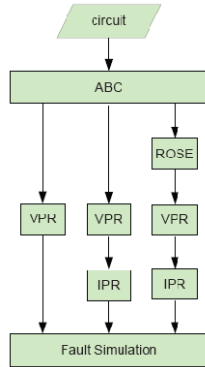


Fig. 5. CAD flow used in experiments.

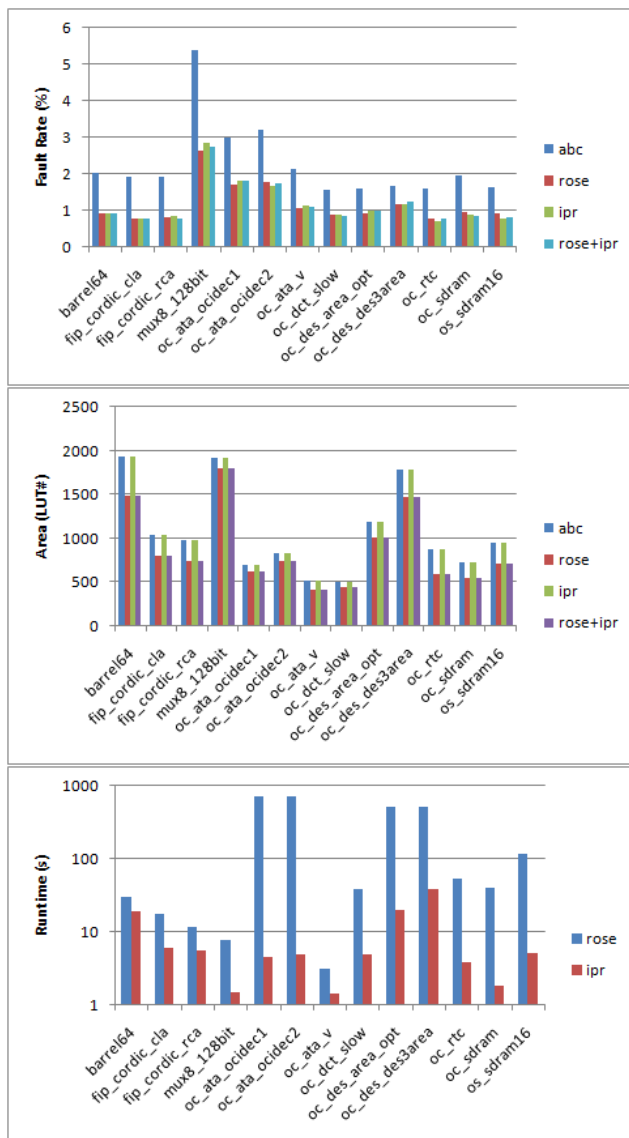


Fig. 6. Summary of experimental results

## VIII. CONCLUSIONS AND FUTURE WORK

Targeting FPGAs, we have presented two fault-tolerant logic synthesis algorithms, ROSE and IPR, which increase the MTTF by 2X with minimal overhead for area, power, performance and the existing CAD flow, and therefore eases the design closure. Hardware emulation is used for efficient criticality calculation and post-optimization validation for the fault-tolerant synthesis.

In the future, the proposed paradigm can be easily extended to cope with other objective functions, such as leakage power reduction and timing optimization, and they can also handle ASIC circuits for timing closure during the engineering change order (ECO) or post-silicon debugging.

## REFERENCES

- [1] A. Djupdal and P. C. Haddow, "Yield enhancing defect tolerance techniques for FPGAs," in MAPLD International Conference, 2006.
- [2] S. Durand and C. Piguet, "FPGA with self-repair capabilities," in FPGA, 1994.
- [3] N. J. Howard, A. M. Tyrrell, and N. M. Allinson, "The yield enhancement of field-programmable gate arrays," in TVLSI, 1994.
- [4] "Altera stratix II features," in <http://www.altera.com/products/devices/stratix2/>, 2006.
- [5] A. Doumar and H. Ito, "Design of switching blocks tolerating defects/faults in FPGA interconnection resources," in DFT, 2000.
- [6] A. J. Yu and G. G. Lemieux, "Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement," in FPL, 2005.
- [7] H. Naeimi, "A greedy algorithm for tolerating defective crosspoints in NanoPLA design," in Master Thesis, California Institute of Technology, 2005.
- [8] M. Joshi and W. Al-Assadi, "Development and Analysis of Defect Tolerant Bipartite Mapping Techniques for Programmable cross-points in Nanofabric Architecture," Springer Netherlands, 2007.
- [9] R. Bonam, Y.-B. Kim, and M. Choi, "Defect-tolerant gate macro mapping and placement in clock-free nanowire crossbar architecture," in DFT, 2007.
- [10] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," in IBM Journal of Research and Development, 1962.
- [11] Ken Chapman and Les Jones, SEU Strategies for Virtex-5 Devices, Application Note, XAPP864 (v1.0.1), March 5, 2009.
- [12] "Altera: QUIP for Quartus II V5.0," in <http://www.altera.com/education/univ/>.
- [13] Yu Hu, Victor Shih, Rupak Majumdar, and Lei He, FPGA Area Reduction by Multi-Output Function Based Sequential Resynthesis, DAC, 2008.
- [14] J. Cong and Y.-Y. Hwang, "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping," in TODAES, 2001.
- [15] Y. Hu, V. Shih, R. Majumdar, and L. He, "Exploiting symmetry in SAT-based boolean matching for heterogeneous FPGA technology mapping," in ICCAD, 2007.
- [16] A. Mishchenko and R. K. Brayton, "SAT-based complete don't-care computation for network optimization," in DATE, 2005.
- [17] S. Krishnaswamy, S. Plaza, I. Markov, and J. Hayes, "Signature-based SER analysis and design of logic circuits," TCAD, 2009.
- [18] Yu Hu, Zhe Feng, Lei He, Rupak Majumdar, "Robust FPGA resynthesis based on fault-tolerant Boolean matching," ICCAD 2008: 706-713
- [19] Zhe Feng, Yu Hu, Lei He and Rupak Majumdar, "IPR: In-Place Reconfiguration for FPGA Fault Tolerance", ICCAD, 2009
- [20] N. Een and N. Sorensson, minisat2.0, <http://minisat.se/>.
- [21] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," FPL, 1997.
- [22] "ABC: A system for sequential synthesis and verification," in <http://www.eecs.berkeley.edu/~alanmi/abc/>.
- [23] Vilas Sridharan and David Kaeli, "The Effect of Input Data on Program Vulnerability," SELSE 2009.
- [24] Kai-hui Chang, Igor L. Markov and Valeria Bertacco, "Reap What You Sow: Spare Cells for Post-Silicon Metal Fix," ISPD 2008