# Robust FPGA Resynthesis Based on Fault-Tolerant Boolean Matching*

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

## ABSTRACT

We present FPGA logic synthesis algorithms for stochastic fault rate reduction in the presence of both permanent and transient defects. We develop an algorithm for *fault tolerant Boolean matching* (FTBM), which exploits the flexibility of the LUT configuration to maximize the stochastic yield rate for a logic function. Using FTBM, we propose a *robust* resynthesis algorithm (ROSE) which maximizes stochastic yield rate for an entire circuit. Finally, we show that existing PLB (programmable logic block) templates for area-aware Boolean matching and logic resynthesis are not effective for fault tolerance, and propose a new *robust template* with path re-convergence. Compared to the state-of-the-art academic technology mapper Berkeley ABC, ROSE using the proposed robust PLB template reduces the fault rate by 25% with 1% fewer LUTs, and increases MTBF (mean time between failures) by 31%, while preserving the optimal logic depth.

## 1. INTRODUCTION

Most of today's programmable logic device (PLD) synthesis flows target *nominal* designs. In this view, the low-level and uncertain physics of devices and transistors are abstracted into Boolean digital signals, 0 and 1, and a circuit is a *deterministic* function which maps input bits into output bits (and states). Logic synthesis optimizes the representation of these functions through Boolean reasoning. Deviations from the nominal behavior, e.g., through process variations and defects, are controlled at the testing level by discarding defective chips.

Unfortunately, as faults become more pronounced in emerging applications and technologies, such as permanent faults arising from circuit processing at nanometer scales or transient soft errors arising from high-energy particle hits, the deterministic view becomes limiting. For CMOS circuits vulnerable more to soft errors, these faults reduce the mean time between failures (MTBF). For future nano-circuits with more defective devices, they reduce the yield rate. This implies that logic design and synthesis flows must explicitly account for and tolerate faults.

Indeed, 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 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 configuration reduces 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.

We take an alternate route toward fault tolerant designs. We propose *stochastic synthesis* for fault tolerance, where the presence of random faults is reflected in the logic synthesis algorithm. We model faults in LUT configurations and the faults in intermediate wires as random variables, following the probabilistic nature of faults, and the probability that a LUT configuration bit or an intermediate wire is defective is given as an input to our synthesis algorithm. (Elsewhere, we discuss how these probabilities may be computed [11].) 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. Stochastic synthesis algorithms explicitly minimize fault rates, along with traditional metrics such as circuit area or delay.

As a particular example of stochastic synthesis, we propose ROSE, a RObust reSynthEsis algorithm, which minimizes the stochastic fault rate under random faults in FPGAs while incurring negligi-

*This research is partially supported by the NSF awards CCR-0306682, CNS-0702881 and CCF-0702743, and a UC MICRO grant sponsored by Actel.

ble area and performance overhead. ROSE exploits the flexibility in implementing a logic block by a programmable logic block (PLB) template (e.g., in selecting configuration bits), and rewrites logic blocks to minimize the stochastic fault rate. Unlike manufacturer-masking, ROSE does not need to locate faults by testing. Unlike chip-wise synthesis, ROSE uses the same design for different chips of an FPGA application for stochastic fault tolerance. It can be directly applied to tolerate faults in less critical tasks, such as internet routing switches and enterprise servers, to reduce MTBF. In addition, ROSE is orthogonal to the existing redundancy based fault-tolerant approaches [2-10], and therefore it also can be used to further increase the robustness while reducing the overhead of existing techniques such as TMR and column based redundancy for FPGAs.

The core algorithmic idea in ROSE is that of *fault-tolerant Boolean matching* (FTBM). FTBM generalizes the Boolean matching problem [12, 13] to the setting with stochastic faults. It takes as input a PLB $\mathcal{H}$, a Boolean function $F$, and the fault rates for the inputs and the SRAM bits of the PLB, and outputs either that $F$ cannot be implemented by PLB $\mathcal{H}$, or the configuration of $\mathcal{H}$ which minimizes the probability that the faults are observable in the output of the PLB under all input vectors. We describe an algorithm for FTBM by a reduction to *stochastic satisfiability* (SSAT) [14]. Since SSAT has a high computational cost (and state-of-the-art SSAT solvers are less developed than SAT solvers), we convert the SSAT problem into a sequence of deterministic Boolean satisfiability problems. While we apply FTBM in resynthesis, it can also be applied in various synthesis stages, such as technology independent optimization, technology mapping or post-mapping resynthesis.

ROSE applies FTBM on circuit logic blocks against a set of predefined *templates*, substituting the block with the minimum fault rate configuration of the template found by FTBM (provided this does not increase local logic depth or area). Orthogonally to FTBM, ROSE enhances robustness by a suitable choice of PLB templates. We evaluate some PLB templates for fault tolerance. First, we show that existing templates for area optimization are not effective for fault tolerance. Second, using the observation that reconvergence is a prime reason for don't cares, and that don't cares can occlude errors, we propose a new robust template with path reconvergence. Our template can be obtained by either configuring the existing full connection between LUTs within a cluster, or hardwiring selected connections between LUTs to reduce area.

On QUIP [15] benchmarks, ROSE (using the robust template) reduces the fault rate by 25% with 1% fewer LUTs, and increases MTBF (mean time between failures) by 31%, while preserving the optimal logic depth, when compared to the state-of-the-art FPGA technology mapping, Berkeley ABC mapper [16].

The remainder of this paper is organized as follows. Section 2 provides preliminary definitions. Sections 3 and 4 present the overall approach and FTBM, respectively. The experimental results are given in Section 5 and the paper is concluded with future research directions in Section 6. To the best of our knowledge, this work is the first in-depth study of stochastic synthesis for FPGA fault tolerance.

# 2. PRELIMINARIES

## 2.1 Boolean Network

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

A 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 $K\text{-}feasible$ 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, single-output (MISO) logic blocks, but the proposed algorithm can be applied to multi-output, multi-output (MIMO) logic blocks [44], as well.

## 2.2 Boolean Matching

Given a PLB $\mathcal{H}$ and a Boolean function $F$, the *Boolean matching problem* (BM) either maps function $F$ to PLB $\mathcal{H}$ by describing an appropriate setting of the LUT configuration bits, or concludes that PLB $\mathcal{H}$ cannot implement function $F$. *Boolean matching* [17, 18] 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 [12, 19, 20]. Consider a PLB template $\mathcal{H}$ with inputs $x'_1, \cdots, x'_k$, output $G$, intermediate wires $z_1, \cdots, z_m$, and LUT configuration $c_1, \cdots, c_n$ as shown in Figure 1. Let $F$ be a Boolean function of $k$ inputs, given as a truth table.

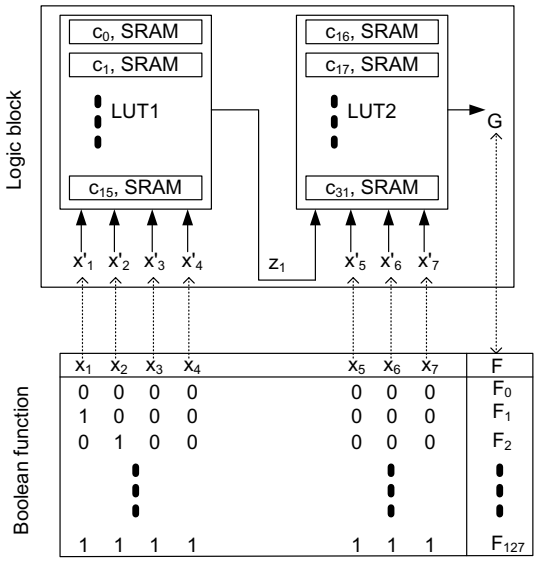



**Figure 1: Illustration of FPGA Boolean matching**

We can write a set of Boolean constraints that define each internal and output wire of $\mathcal{H}$ in terms of its inputs (see, e.g., [12, 20]). For

example, the internal wire $z_1$ in Figure 1 can be defined as

$$(\overline{x_1'} \wedge \overline{x_2'} \wedge \overline{x_3'} \wedge \overline{x_4'} \rightarrow (z_1 \leftrightarrow c_0)) \wedge \cdots \wedge (x_1' \wedge x_2' \wedge x_3' \wedge x_4' \rightarrow (z_1 \leftrightarrow c_{15}))$$

Let $\Psi(\mathcal{H})$ be the conjunction of constraints defining each wire of $\mathcal{H}$.

Similarly, the truth table for function $F$ can be expressed as a set of constraints between the input variables $x_1, \cdots, x_k$ and the output $F$:

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

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

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

$$\exists c_1 \cdots c_n \forall x_1 \cdots x_k \exists z_1 \cdots z_m \; . \; \Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G \leftrightarrow F) \quad (2)$$

By replicating the formula for each possible valuation to the bits $x_1 \cdots x_k$, 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 $F$. SAT-based Boolean matching offers high flexibility and, with recent optimizations [21, 19, 20], reasonable performance.

## 2.3 Fault Model and FTBM

In the presence of faults in the LUT configurations or intermediate wires between PLBs, 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 a 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. Based on the above fault modeling, we formulate *fault-tolerant Boolean matching* (FTBM) as follows.

DEFINITION 1. ***FTBM*** *takes as input a PLB $\mathcal{H}$, a Boolean function $F$, and the fault rates for the input pins (representing intermediate wires between PLBs) and the configuration bits of the PLB, and outputs either that $F$ cannot be implemented by PLB $\mathcal{H}$, or the configuration of $\mathcal{H}$ which minimizes the probability, over all input vectors, that the faults are observable in the output of the PLB.*

Note that faults at PLB input pins result from faults of the upstream logic and wires between PLBs. While we assume single fault in our experiments, our algorithm for FTBM (described in Section 4) allows multiple faults to occur simultaneously.

## 2.4 Resynthesis for LUT-based Network

*Resynthesis* [22, 23, 24, 25] is a technique that rewrites circuit structures to reduce area while maintaining the functionalities of transition and output functions. It can be performed simultaneously with technology mapping or as a post-mapping optimization. The simultaneous approaches perform logic resynthesis, such as Boolean decomposition of logic functions [26, 27], during the mapping process. Since they explore a large solution space, the simultaneous approaches tend to be time-consuming, limiting their use to small designs. To handle large designs, resynthesis is usually performed after technology mapping for area recovery. Recently, [12, 19, 28] proposed combinational resynthesis based on Boolean matching [18] for FPGA area reduction. The resynthesis is performed by mapping logic blocks extracted from a circuit against a library set of logic blocks (e.g., hard-wired LUTs) and replacing them with a functionally equivalent logic block if area can be reduced.

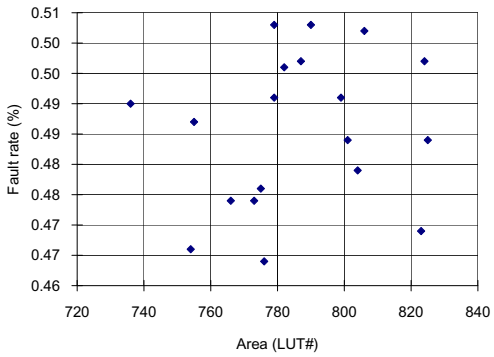



**Figure 2: Area (LUT#) vs. fault rate for MCNC benchmark "i10" (one LUT bit fault rate is 0.1%)**

While resynthesis guarantees a functionally equivalent circuit, the fault rate of the resynthesized circuit can be significantly different from the original one. We demonstrate this by example. Figure 2 shows the area (number of LUTs) vs fault rate for 18 resynthesis solutions obtained using different options to ABC [16]. Notice that the same application may be implemented using multiple distinct configuration settings with different logic masking, resulting in significantly different fault tolerance but with similar area. Hence, we propose a *robust* resynthesis algorithm to simultaneously reduce circuit area and fault rate.

# 3. ROBUST RESYNTHESIS

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 PLB template which enables better fault tolerance with ROSE.

## 3.1 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 PLB templates; if a mapping with the minimal fault rate is found by FTBM, the logic block can be substituted by the PLB 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 (see Figure 3).

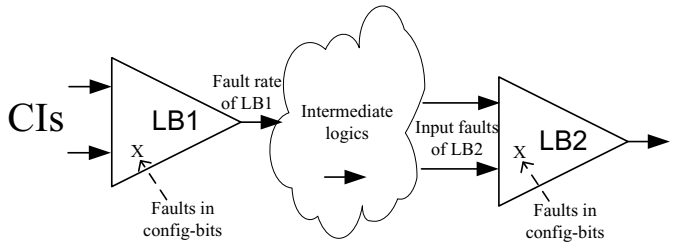



**Figure 3: Propagation of faults in ROSE, where input faults of LB2 are resulted from the output fault in LB1.**

## 3.2 Robustness of PLB Templates

Besides an effective robust resynthesis algorithm, it is also important to find an effective PLB 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 PLB 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)). The inherent disadvantage of these area efficient PLB 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 *re-convergence is a prime reason for don't-cares*, we propose a new PLB template, R-PLB, as shown in Figure 4 (d), which requires four 4-LUTs and forms re-convergent paths from input to output.

The logic don't-cares include *satisfiability don't-cares* (SDCs) due to the fact that some combinations are not produced as the inputs of the node, and *observability don't-cares* (ODCs) due to the fact that under some conditions the output value of the node does not matter (i.e., is controlled by certain input combinations) [29]. Figure 5 shows examples of don't-cares in R-PLB. In Section 5, we will experimentally show the effectiveness of R-PLB for fault tolerance.

# 4. FTBM ALGORITHMS

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 2.2, after solving (2), a set of LUT configurations $c_1, \cdots, c_n$ will be returned by the SAT solver if $F$ can be implemented by $\mathcal{H}$. There might exist multiple distinct implementations (i.e., different configurations) for $\mathcal{H}$ 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, $\mathcal{C} = (c'_1, \cdots, c'_n)$, we evaluate the fault rate at the output of this logic block under this configuration setting. The configuration, $\mathcal{C}^*$, which results in the minimal fault rate is chosen as the candidate for mapping or resynthesis.

## 4.1 Fault Rate Calculation

The main step of FTBM is an algorithm for fault rate calculation. We now show how fault rate calculation can be reduced to *stochastic satisfiability* (SSAT) [14], a generalization of Boolean satisfiability where some variables may be "randomly quantified" in addition to variables that are existentially or universally quantified. For example, the formula

$$\exists x_1, \Re x_2, \forall x_3, \ldots, \exists x_{n-1}, \Re x_n. (E\varphi(x_1, \ldots, x_n) \geq \beta)$$

asks whether there exists a value for $x_1$ such that *for random* values of $x_2$ (chosen from a given distribution), for all values of $x_3$, ... there exists a value of $x_{n-1}$ such that for random values of $x_n$ the *expected* value $E\varphi$ that the Boolean formula $\varphi$ is satisfied under the variable assignment is at least $\beta$.

For a logic block, we assume two (independent) sources of faults: defective input bits of the logic block (resulting from the faults of the upstream logic and wires between PLBs) and defective LUT configurations inside the block. Assume the fault rate of input bit $x'_i$ is $P_i$, i.e., that with probability $P_i$, the $i$th input in the mapped circuit is the opposite of the $i$th input of the function $F$. For each $i$, we introduce a Boolean variable $p_i$, where $p_i = 1$ with probability $P_i$ and $p_i = 0$ with probability $1 - P_i$ and the constraint

$$p_i \leftrightarrow (x'_i \neq x_i) \tag{3}$$

to indicate that $p_i$ is 1 iff the input bit $x'_i$ is not equal to $x_i$. Similarly, assume that the fault rate of the LUT configuration $c_i$ is $D_i$, i.e., with probability $D_i$, the $i$th LUT configuration bit in the logic block has a value opposite to the nominal LUT configuration bit. We introduce a Boolean variable $d_i$ that is 1 with probability $D_i$, and add the constraint

$$d_i \leftrightarrow (c'_i \neq c_i) \tag{4}$$

to indicate that the $i$th LUT configuration bit $c'_i$ is different from the correct value $c_i$ iff $d_i$ is 1.

Then, given a threshold $\beta$, the Boolean matching problem under random faults in the input bits and the LUT configuration of the logic block is reduced to the stochastic satisfiability instance:

$$\begin{aligned}
&\exists c_1, \cdots, \exists c_n, \Re p_1, \cdots, \Re p_k, \Re d_1, \cdots, \Re d_n, \exists c'_1, \cdots, \exists c'_n, \\
&\forall x_1, \cdots, \forall x_k, \exists x'_1, \cdots, \exists x'_k, \exists z_1 \cdots \exists z_m, \exists G, \exists F \\
&E\{\Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G \leftrightarrow F) \wedge \\
&\quad \bigwedge_{i=1,\cdots,n} d_i \leftrightarrow (c'_i \neq c_i) \wedge \bigwedge_{i=1,\cdots,k} p_i \leftrightarrow (x'_i \neq x_i)\} \geq \beta
\end{aligned} \tag{5}$$

where $\Psi(\mathcal{H})$ and $\Psi(F)$ are the constraints that define the internal and output signals of $\mathcal{H}$ and the truth table $F$, respectively. The probabilities $P(d_i = 1) = D_i$ are given as inputs to the problem, and $P(p_i = 1) = P_i$ are obtained from the upstream logic block. If the above SSAT problem is satisfiable, then the choice of the LUT configuration $c_1, \ldots, c_n$ ensures that the probability that the circuit implements $F$ even when the LUT configuration and inputs are flipped is at least $\beta$. By a binary search on $\beta$, we can find the maximal probability that the defective circuit implements the function $F$ for this choice of the LUT configuration.

The above formulation (5) requires *all* min-terms of $G$ and $F$ are identical under all input vectors $x_1, \cdots, x_k$, which is too restrictive in practice. Therefore, we relax the definition of fault rate as the percentage of min-terms produced by $G$ that are not equal to the corresponding min-term in function $F$. Formally, suppose logic block $\mathcal{H}$ with output $G$ and the LUT configuration set to $c_1, \ldots, c_n$ implements a Boolean function $F(x_1, \cdots, x_k)$. The fault rate of logic block $\mathcal{H}$ under the input and the LUT configuration faults is defined as

$$DF(\mathcal{H}) = \frac{1}{2^k} \sum_{x_1,\cdots,x_k=0,\cdots,0}^{1,\cdots,1} DF_{\text{min-term}(x_1,\cdots,x_k)}$$

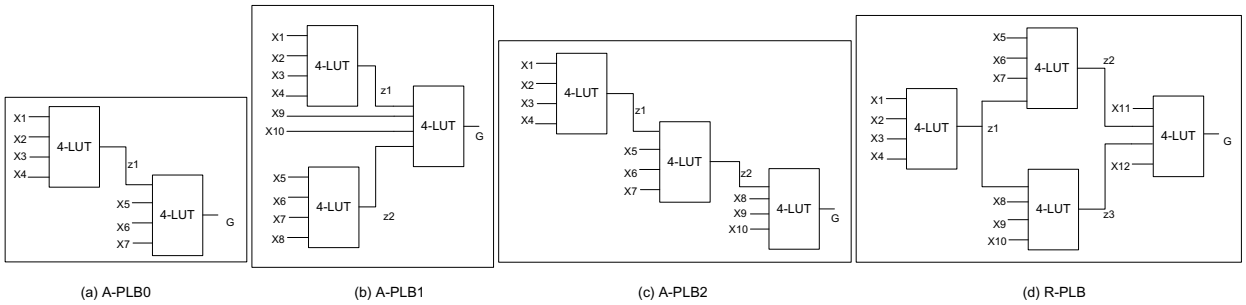



**Figure 4: Area efficient PLB templates (a),(b) and (c), and the proposed robust template (d)**

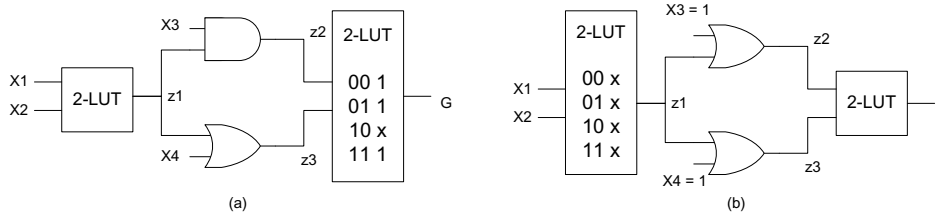



**Figure 5: Examples of logic masking in R-PLB. For simplicity, we use 2-LUT to replace 4-LUT. (a) LUTs $z_2$ and $z_3$ implement 2-AND and 2-OR, respectively. Gate $z_2$ and $z_3$ form a SDC for LUT $G$ because input vector $z_2 = 1 \wedge z_3 = 0$ will never happen due to the path reconvergence, therefore the faults in configuration bit 10 in LUT $G$ will not affect the output; (b) LUTs $z_2$ and $z_3$ both implement 2-OR. $x_3 = 1 \wedge x_4 = 1$ is the control signal of gate $z_2$ and $z_3$, which masks the output of LUT $z_1$, i.e., makes $z_1$ an ODC for the output, and therefore any faults in configuration bits in LUT $z_1$ will not affect the output.**

where $DF_{\text{min-term}(x_1,\cdots,x_k)}$ is the probability that $G$ and $F$ have different values when the input is fixed at $(x_1, \ldots, x_k)$ and faults occur randomly according to $P_i$ and $D_i$. This probability can be computed by maximizing $\beta$ in the SSAT formula:

$$\begin{aligned}
&\Re p_1, \cdots, \Re p_k, \Re d_1, \cdots, \Re d_n, \\
&\exists c'_1, \cdots, c'_n, \exists x'_1, \cdots, \exists x'_k, \exists z_1 \cdots \exists z_m, \exists G, \exists F \\
&E\{\Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G(x_1, \cdots, x_k) \leftrightarrow F(x_1, \cdots, x_k)) \wedge \\
&\bigwedge_{i=1,\cdots,n} d_i \leftrightarrow (c'_i \neq c_i) \wedge \bigwedge_{i=1,\cdots,k} p_i \leftrightarrow (x'_i \neq x_i)\} \geq \beta \quad (6)
\end{aligned}$$

Note that the bits $c_1, \ldots, c_n$ and $x_1, \ldots, x_k$ are assumed to be fixed in the above formula. Finally, we have

$$DF_{\text{min-term}(x_1,\cdots,x_k)} = 1 - \hat{\beta},$$

where $\hat{\beta}$ is the maximal probability of satisfying formula (6).

## 4.2 Reduction to Deterministic SAT

While there are tools to solve SSAT directly [30], we found that their runtimes are too long to be applied directly to our problem. For example, it takes over four hours on a Xeon 1.9GHz workstation to solve a SSAT instance with 16 random variables, 14 universal variables and 2K clauses using the implementation from [30]. Instead, we solve Equation (6) iteratively using a deterministic SAT solver (miniSAT [31] in our experiments), by enumerating satisfying assignments and evaluating the expectations according to the probability distributions of the randomly quantified variables.

We first express (6) as a deterministic SAT problem by changing the random quantifiers before $p_i$'s and $d_i$'s to existential quantifiers:

$$\begin{aligned}
&\exists p_1, \cdots, \exists p_k, \exists d_1, \cdots, \exists d_n, \\
&\exists c'_1, \cdots, \exists c'_n, \exists x'_1, \cdots, \exists x'_k, \exists z_1 \cdots \exists z_m, \exists G, \exists F \\
&\{\Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G(x_1, \cdots, x_k) \leftrightarrow F(x_1, \cdots, x_k)) \wedge \\
&\bigwedge_{i=1,\cdots,n} d_i \leftrightarrow (c'_i \neq c_i) \wedge \bigwedge_{i=1,\cdots,k} p_i \leftrightarrow (x'_i \neq x_i)\} \quad (7)
\end{aligned}$$

Then, we solve an ALL-SAT problem for (7), obtaining all satisfiable assignments for $p_i$'s and $d_i$'s:

$$(p_1^1, \cdots, p_k^1, d_1^1, \cdots, d_n^1), \cdots, (p_1^m, \cdots, p_k^m, d_1^m, \cdots, d_n^m)$$

The maximal probability of satisfying (6) can be calculated by using the following formula:

$$\hat{\beta} = \frac{1}{2} \sum_{i=1}^{m} \left( \prod_{j=1}^{k} Pr(p_j = p_j^i) \cdot \prod_{j=1}^{n} Pr(d_j = d_j^i) \right)$$

assuming the independence of each fault [1] and the probabilities $P_i$ and $D_i$. Note that $P_i$ denotes the probability that either $(x_i = 0 \wedge x'_i = 1)$ or $(x_i = 1 \wedge x'_i = 0)$. If we distinguish between these two cases assuming that they are equally probable, we have that $(x_i = 0 \wedge x'_i = 1)$ with the probability $(P_i)/2$ and $(x_i = 1 \wedge x'_i = 0)$ also with the probability $(P_i)/2$. Therefore we have to distinguish how the corresponding variables are assigned and to use $P_i/2(D_i/2)$ in our calculation.

An ALL-SAT problem (e.g., 7) can be solved by iteratively adding the negation of the previously satisfiable solutions into the CNFs, which is computationally expensive, and we exploit the structure

[1] Although independence between faults is assumed here for the reduction from SSAT to deterministic SAT, such reduction can be done for multiple faults with given correlation. Therefore, the SSAT-based formulation and algorithm can be extended to handle multiple faults.

of the problem to speed up the fault rate calculation. We note that the probabilities $P_i$ and $D_i$ are usually very small. So we rule out certain combinations of $p_i$'s and $d_i$'s that have very low probability of occurrence. For example, we can restrict the search to assignments in which at most a small number of $p_i$'s and $d_i$'s are one simultaneously. When the fault rate of single bit is low, this is a nice approximation that may slightly reduce the total fault rate but with significant speedup of the reasoning runtime. In our experiments, we shall assume there is a single faulty bit.

# 5. EXPERIMENTAL RESULTS

## 5.1 Evaluation Based on Boolean Functions

We have implemented ROSE in C++ in the OAGear package [32]. We use miniSAT2.0 [31] as the SAT engine in FTBM. All experimental results are collected on a Ubuntu workstation with 2.6GHZ Xeon CPU and 2GB memory. We test our algorithms on QUIP benchmarks [15], where each benchmark is mapped by Berkeley ABC mapper [16] with 4-LUTs. We assume that one and only one bit of the LUT configuration in the mapped FPGAs is defective. The fault rate of the chip is the percentage of the input vectors that produce the defective outputs, and it is calculated by Monte Carlo simulation with 20K iterations where one bit fault is randomly injected in each iteration.

We evaluate the effectiveness of the three templates for fault tolerance. We first extract 3000 9-input Boolean functions from QUIP benchmarks by enumerating 9-feasible cuts. Without considering input faults, we perform FTBM to map these 9-input Boolean functions against A-PLB1, A-PLB2, and R-PLB, respectively. The configurations with minimal or maximal fault rate can be obtained by using two versions of FTBM algorithms, i.e., FTBM$^+$ and FTBM$^-$, where FTBM$^-$ returns the configurations with the maximal fault rate while FTBM$^+$ returns the ones with the minimal fault rate. We compare R-PLB with A-PLB1 and A-PLB2 in terms of (a) the minimal fault rate that can be achieved by these templates, and (b) the gap of the fault rates (i.e., the difference between the minimal and maximal fault rates) under all feasible configurations. Due to the space limit, we only show the comparison results between R-PLB and A-PLB2 in Figure 6. In these plots, only those Boolean functions that can be implemented by both templates are taken. As shown in the plots, the minimal fault rates achievable by R-PLB are generally less than those achievable by A-PLB2. In addition, the solution space, in terms of the gap of the fault rates that can be produced by all feasible configurations of template R-PLB, is generally wider than that of template A-PLB2, which indicates that there exists more flexibility to place don't-cares in R-PLB. Similar observations are obtained by comparing R-PLB and A-PLB1. With these observations, we consider R-PLB as a more effective template for fault mitigation. Nevertheless, all other templates, A-PLB0, A-PLB1, and A-PLB2, can be tested by FTBM during technology mapping or resynthesis to trade-off area and fault rate.

## 5.2 Evaluation Based on Benchmark Circuits

Under the same setting in Section 5.1, we now evaluate the effectiveness of ROSE for fault tolerance. All ABC-mapped QUIP benchmarks are resynthesized by ROSE using area-efficient PLB templates (Figure 4 (a)-(c)) and the robust PLB template (Figure 4 (d)). In resynthesis using area-efficient templates, each of the three templates is examined and the configuration with the maximal fault rate is returned. The logic depths are preserved in all resynthesis algorithms. The results are summarized in Table 1.

We first show the gap between the minimal fault rate and the maximal fault rate that can be achieved by ROSE. The FTBM$^+$ and FTBM$^-$ mentioned in the previous sub-section are used in ROSE to obtain the minimal and maximal fault rates, respectively. As shown in column "gap" in Table 1, ROSE using FTBM$^+$ consistently returns results with smaller fault rates compared to ROSE using FTBM$^-$, and the gap is up to 2.42%. Note that ROSE only maximizes the local logic masking, i.e., the best logic masking introduced by ROSE may not always be the most effective to the primary outputs. The above observation indicates our approach in ROSE is a high quality heuristic in terms of global optimization. In the following experiments, we always use FTBM$^+$ in ROSE. In addition, ROSE/R using the robust template consistently obtains a bigger gap than ROSE/A using area-efficient templates. This demonstrates that the robust template does offer a higher potential for fault rate reduction.

Moreover, Table 1 compares the fault rates resulted from ABC (sub-column "ABC"), ROSE using area-efficient templates (sub-column "ROSE/A") and ROSE using robust template (sub-column "ROSE/R"), respectively. Compared to the mapping by ABC, our ROSE using the robust template reduces fault rate by 25% (1.06% vs. 0.80%) on average. Compared to ROSE using area-efficient templates, ROSE using the robust template reduces fault rate by 14% (0.80% vs. 0.93%) on average.

As shown in column "area" of Table 1, our ROSE using area-efficient templates and the robust template reduces area by 10% and 1%, respectively, compared to ABC mapping results. Since R-PLB has larger area compared to area-efficient templates, ROSE/R results in 11% larger area than ROSE/A.

In addition, Table 1 compares the runtime of the deterministic SAT-based resynthesis using area efficient templates [28]. The proposed ROSE using two different template sets. It shows that ROSE using area efficient templates and ROSE using the robust template has 3X and 10X runtime overhead, respectively, compared to the deterministic resynthesis. The ROSE using the robust template is slower than the one using area efficient templates because the robust template contains more LUTs and FTBM requires more runtime to solve the SAT problem.

## 5.3 Estimation of MTBF

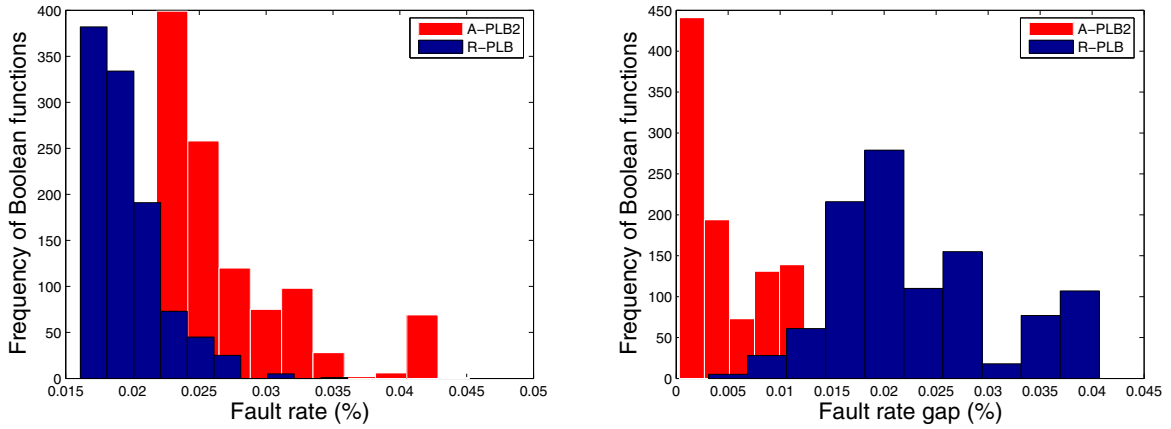
We now estimate the mean time between failure (MTBF) obtained by different resynthesis algorithms for industrial FPGAs. As suggested by [33], the MTBF can be estimated by the following formula:

$$\begin{aligned} \text{MTBF} &= 10^9/(24 \cdot 365)\ \text{FIT}_{\text{total}} \\ \text{FIT}_{\text{total}} &= 100 \cdot R_{\text{vulnerability}} \cdot R_{\text{intrinsic error}} \\ R_{\text{intrinsic error}} &= \text{Area} \cdot R_{\text{FIT}} \end{aligned} \quad (8)$$

where $R_{\text{vulnerability}}$ is the vulnerability factor, which is the fraction of faults that become errors, and can be estimated by the mean of the fault rates in Table 1, and $R_{\text{intrinsic error}}$ is the intrinsic error rate, which is proportional to the area (SRAM bits number) and the raw FIT rate[2] for a single bit, $R_{\text{FIT}}$. Typically, $R_{\text{FIT}}$ is 0.001-0.01 FIT/bit and we use 0.01 FIT/bit in our estimation.

The current biggest FPGAs are Stratix III (with up to 338,000 logic cells) [34] from Altera and Virtex-5 LXT (with up to 330,000 logic cells) from Xilinx [35]. We use 330,000 4-LUTs as the typical industrial FPGA size to estimate the MTBF obtained by our ROSE. As shown in Table 2, ROSE using the robust template increases MTBF by 31% (27.15 years vs. 20.66 years) compared to ABC. Note that the MTBFs in our experimental results match the typical server system reliability goals as reported in [33], which validates our experimental settings. Note that the purpose of this comparison

[2]One FIT is one failure in a billion hours.




(a) Achievable minimal fault rate

(b) Gap between minimal and maximal fault rates

**Figure 6: Comparison between template R-PLB and template A-PLB2**

| benchmarks | gap | | fault rate | | | area (LUT#) | | | runtime (min) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ROSE/R | ROSE/A | ABC | ROSE/A | ROSE/R | ABC | ROSE/A | ROSE/R | [28] | ROSE/A | ROSE/R |
| barrel64 | 2.42% | 0.64% | 2.09% | 1.82% | 1.63% | 1862 | 1414 | 1734 | 1.55 | 9.22 | 16.77 |
| fip_cordic_cla | 0.62% | 0.35% | 1.25% | 1.07% | 0.93% | 1044 | 823 | 1027 | 0.7 | 0.78 | 7.10 |
| fip_cordic_rca | 0.54% | 0.31% | 1.17% | 1.03% | 0.84% | 983 | 789 | 970 | 0.58 | 0.55 | 5.88 |
| mux8_128bit | 0.30% | 0.10% | 0.97% | 0.94% | 0.67% | 898 | 770 | 898 | 0.23 | 0.25 | 149.03 |
| nut_000 | 0.51% | 0.38% | 0.97% | 0.83% | 0.63% | 927 | 912 | 926 | 2.23 | 182.03 | 6.38 |
| nut_002 | 0.44% | 0.25% | 0.71% | 0.60% | 0.48% | 652 | 645 | 652 | 3.03 | 4.20 | 4.80 |
| nut_004 | 0.32% | 0.23% | 0.69% | 0.58% | 0.50% | 618 | 550 | 615 | 0.35 | 1.10 | 8.27 |
| oc_ata_ocidec1 | 0.52% | 0.14% | 0.72% | 0.65% | 0.52% | 693 | 660 | 692 | 1.5 | 1.60 | 4.48 |
| oc_ata_ocidec2 | 0.66% | 0.18% | 0.88% | 0.79% | 0.64% | 838 | 769 | 837 | 1.55 | 1.72 | 4.77 |
| oc_ata_v | 0.38% | 0.19% | 0.55% | 0.47% | 0.40% | 512 | 452 | 494 | 0.15 | 0.20 | 3.10 |
| oc_cordic_p2r | 2.06% | 0.64% | 3.35% | 3.09% | 3.09% | 3175 | 3131 | 3175 | 0.98 | 19.72 | 9.12 |
| oc_correlator | 0.13% | 0.20% | 0.67% | 0.64% | 0.59% | 611 | 585 | 610 | 0.13 | 0.22 | 9.35 |
| oc_dct_slow | 0.37% | 0.20% | 0.55% | 0.46% | 0.36% | 513 | 479 | 512 | 2.87 | 36.70 | 6.50 |
| oc_des_area_opt | 0.83% | 0.47% | 1.26% | 1.08% | 0.93% | 1192 | 1127 | 1191 | 6.63 | 31.15 | 6.52 |
| oc_des_des3area | 1.25% | 0.52% | 2.04% | 1.89% | 1.44% | 1784 | 1580 | 1783 | 3.77 | 29.32 | 5.35 |
| oc_i2c | 0.37% | 0.22% | 0.62% | 0.52% | 0.46% | 597 | 548 | 590 | 3.25 | 2.45 | 28.67 |
| oc_rtc | 0.60% | 0.30% | 0.95% | 0.79% | 0.71% | 887 | 708 | 881 | 0.43 | 0.65 | 22.13 |
| oc_sdram | 0.65% | 0.28% | 0.76% | 0.64% | 0.60% | 731 | 648 | 728 | 0.32 | 1.50 | 4.08 |
| os_sdram16 | 0.78% | 0.35% | 0.97% | 0.83% | 0.74% | 947 | 821 | 923 | 0.45 | 2.40 | 69.75 |
| geomean | 0.58% | 0.30% | 1.06% | 0.93% | 0.80% | 980 | 877 | 970 | 0.93 | 3.07 | 9.42 |
| normalized mean | | | 1.00 | 0.87 | 0.75 | 1.00 | 0.90 | 0.99 | 1.00 | 3.29 | 10.08 |

**Table 1: Comparison of ABC and ROSE using robust and area-efficient PLB templates**

is to provide a sanity check for the effectiveness of our approach; clearly there exist many differences between FPGAs and server systems, but we do expect "ball-park" parameter values for MTBF calculations to be similar.

| | MTBF (year) | |
|---|---|---|
| benchmark area | ABC | ROSE using R-PLB |
| 330,000 LUTs | 20.66 | 27.15 |
| ratio | 1 | 1.31 |

**Table 2: MTBF (mean time between failures)**

## 6. CONCLUSIONS AND FUTURE WORK

We are encouraged by the initial success of ROSE in minimizing the stochastic fault rate while preserving optimal logic depth and minimally impacting area. On QUIP benchmarks, ROSE reduced the fault rate by 25% with 1% fewer LUTs, and increased MTBF by 31%, while preserving the optimal logic depth, when compared to ABC [16]. We believe that our study of robust FPGA resynthesis provides a first step toward a general methodology for stochastic synthesis. In particular, there are several open directions.

- Our experiments assume single fault, but the proposed algorithm can deal with multiple uncorrelated faults, which will be first explored in future work. In addition, we will extend the proposed algorithm to consider given correlations between faults.

- To cope with the runtime overhead of the SAT-based Boolean matching, we will study more efficient alternatives for Boolean matching, e.g., building a hybrid algorithm combining SAT and Boolean decomposition [36], which enables us to perform the resynthesis on logic blocks with wider inputs and leading to optimization for robustness from a more global point of view. In addition, potential advance of SSAT solvers will also provide more optimization power of our resynthesis.

- As the essence of the proposed stochastic synthesis is to take advantage of don't-cares in a logic network, we will explore the don't-cares explicitly for robust resynthesis. For example, one can first calculate the complete don't-cares [29] for a logic block, and then explore all possible mapping solutions considering these don't-cares and select the one which max-

imizes logic masking to prevent the fault propagation. More generally, one can take advantages of the existing flexibilities in the network, e.g., Boolean relations [37], Sets of Pairs of Functions to be Distinguished (SPFDs) [38], and sequential flexibilities [43]. These may lead to more efficient algorithms and in turn more globally optimized solutions.

- By introducing additional logic masking, defect-aware logic synthesis algorithms such as ROSE can make verification and silicon debugging tasks difficult. In the future, it is necessary to address the tradeoff between logic masking and testability or verifiability. One possible solution is to provide proofs of correctness of transformations [39].
- In the longer term, we shall investigate how other logic synthesis and optimization algorithms can be made fault-tolerant by exploiting redundancy or flexibility in the solution space. Finally, we shall investigate how fault tolerance at the logic synthesis level interacts with algorithm-level fault tolerance [40].
- Our algorithm applies to standard cell-based circuits as well. There is some existing work on fault-tolerant logic synthesis for standard cell designs. For example, [41] developed a critical-area driven technology mapping, and [42] applied logic redundancy and structural restructure to mask soft errors based on a fast simulation. Our work extends these ideas to an explicit formulation of stochastic synthesis.

# 7. 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] Y. Lin and L. He, "Device and architecture concurrent optimization for FPGA transient soft error rate," in *ICCAD*, 2007.

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

[13] A. Ling, D. Singh, and S. Brown, "FPGA logic synthesis using quantified boolean satisfiability," in *SAT*, 2005.

[14] C. Papadimitriou, "Games against nature," *Journal of Computer and Systems Sciences*, 1985.

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

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

[17] J. Cong and Y.-Y. Hwang, "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping," in *TODAES*, 2001.

[18] L. Benini and G. D. Micheli, "A survey of Boolean matching techniques for library binding," in *TODAES*, 1997.

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

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

[21] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient SAT-based boolean matching for FPGA technology mapping," in *DAC*, 2006.

[22] G. D. Micheli, "Synchronous logic synthesis: algorithms for cycle-time minimization," in *TCAD*, 1991.

[23] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting," in *DAC*, 2005.

[24] S. Malik, E. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," in *TCAD*, 1991.

[25] R. Brayton and A. Mishchenko, "Sequential rewriting," in *IWLS*, 2007.

[26] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," in *TCAD*, 1997.

[27] A. Mishchenko, X. Wang, and T. Kam, "A new enhanced constructive decomposition and mapping algorithm," in *DAC*, 2003.

[28] Y. Hu, V. Shih, R. Majumdar, and L. He, "FPGA area reduction by multi-output function based sequential resynthesis," in *DAC*, 2008.

[29] A. Mishchenko and R. K. Brayton, "SAT-based complete don't-care computation for network optimization," in *DATE*, 2005.

[30] M. Littman, "Initial experiments in stochastic satisfiability," in *AAAI/IAAI*, 1999.

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

[32] Z. Xiu, D. A. Papa, P. Chong, A. Kuehlmann, R. A. Rutenbar, and I. L. Markov, "Early research experience with OpenAccess Gear," in *ISPD*, 2005.

[33] S. Mukherjee, J. Emer, and S. K. Reinhardt, "Radiation-induced soft errors: An architectural perspective," in *HPCA*, 2005.

[34] "Altera stratix III features," in *http://www.altera.com*, 2007.

[35] A. Cosoroaba and F. Rivoallon, "Achieving higher system performance with the Virtex-5 family of FPGAs," in *http://www.xilinx.com/literature*.

[36] A. Mishchenko, R. K. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks," in *ICCAD*, 2008.

[37] R. K. Brayton and F. Somenzi, "Boolean relations and the incomplete specification of logic networks," in *VLSI*, 1989.

[38] S. Yamashita, H. Sawada, and A. Nagoya, "A new method to express functional permissibilities for LUT based FPGAs and its applications," in *ICCAD*, 1996.

[39] S. Kundu, S. Lerner, and R. Gupta, "Validating high level synthesis," in *CAV 08*, 2008.

[40] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies* (C. Shannon and J. McCarthy, eds.), Princeton Univ. Press, 1956.

[41] A. Nardi and A. Sangiovanni-Vincentelli, "Logic Synthesis for Manufacturability," in IEEE Des. Test, 2004.

[42] S. Krishnaswamy, S. Plaza, I. Markov, and J. Hayes, "Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation," in ICCAD, 2007.

[43] M. Case, V. Kravets, A. Mishchenko, and R. Brayton, "Merging nodes under sequential observability," in DAC, 2008.

[44] Y. Hu, V. Shih, R. Majumdar, and L. He, "FPGA Area Reduction by Multi-Output Function Based Sequential Resynthesis," in DAC, 2008.