On Error Tolerance and Engineering Change with Partially Programmable Circuits

Hratch Mangassarian[1], Hiroaki Yoshida[2], Andreas Veneris[1], Shigeru Yamashita[3], Masahiro Fujita[2]

***Abstract*— The growing size, density and complexity of modern VLSI chips are contributing to an increase in hardware faults and design errors in the silicon, decreasing manufacturing yield and increasing the design cycle. The use of *Partially Programmable Circuits* (PPCs) has been recently proposed for yield enhancement with very small overhead. This new circuit structure is obtained from conventional logic by replacing some subcircuits with programmable LUTs. The present paper lays the theoretical groundwork for evaluating PPCs with Quantified Boolean Formula (QBF) satisfiability. First, QBF models are constructed to calculate the *fault tolerance* and *design error tolerance* of a PPC, namely the percentages of faults and design errors that can be masked using LUT reconfigurations. Next, zero-cost Engineering Change Order (ECO) in PPCs is investigated. QBF formulations are given for performing ECOs, and for quantifying the *ECO coverage* of a PPC architecture. Experimental results are presented evaluating PPCs from [1], demonstrating the applicability and accuracy of the proposed formulations.**

I. Introduction

Larger, denser and more complex digital circuits are leading to an increase in hardware faults and design errors that slip into production silicon. In fact, manufacturing defect levels are expected to increase sharply in future technologies [2], further decreasing yield. In order to combat these trends, adding space redundancy and using reconfigurability have been proposed in different contexts to reduce the number of silicon respins [3], [4]. Double and Triple Modular Redundancy (DMR and TMR) are examples of design techniques that replicate parts of a design with the aim of yield enhancement as well as chip reliability improvement. Embedded FPGAs have also been used for yield improvement [5], [6]. However, these methods are costly because they incur significant area or performance overhead.

Partially Programmable Circuits (PPCs), recently introduced in [1], achieve a flexible balance between yield improvement versus the associated costs. PPCs are obtained from conventional combinational logic circuits by replacing some subcircuits with reconfigurable elements such as Look-Up Tables (LUTs) and configurable multiplexers (MUXs). The authors of [1] first use simple heuristics to pick which subcircuits to replace by LUTs. Next, they employ Sets of Pairs of Functions to be Distinguished (SPFDs) [7] to add redundant connections to these LUTs and configurable MUXs, such that a large number of faults can be "bypassed" by simply reprogramming the PPC post-silicon. Further, reconfiguring the PPC can also be used to mask some localized design errors that escape verification and propagate into the silicon.

Evidently, these reconfigurable structures can have other potential applications as well. In this work, we also investigate their use for implementing Engineering Change Orders (ECOs), namely small changes in the specification at later stages of the design cycle. It is well-known that even minor ECOs can lead to vastly different synthesized implementations [8] if a new iteration of the automated flow is used. This is usually unwanted because of the effort already invested in optimizing the original design [8], [9]. As such, synthesis for ECOs strives to make the smallest number of changes to the implementation [8]–[10] so that the design complies to its new specification. In a PPC, LUT/MUX reconfigurations can be used to implement such changes at virtually zero cost, avoiding time-consuming design iterations.

Along these observations, the contribution of this paper is multi-fold. First, the *fault tolerance* of a PPC is defined as the percentage of stuck-at-faults that can be made unobservable using post-silicon reconfigurations. Next, *design error tolerance* is defined in a similar fashion. We show how to compute both of these metrics using formal techniques. Following these contributions, we present a new method for performing ECOs in PPCs using reconfiguration. Finally, we define a measure for quantifying the effectiveness of a PPC in implementing ECOs, given an initial specification. We refer to this as the *ECO coverage* of a PPC architecture and we develop a methodology to compute it. To achieve our goals, we use Quantified Boolean Formulas (QBFs) [11] as the underlying computational platform. Our formulations demonstrate the theoretical appropriateness of QBFs for dealing with reconfigurability and we capitalize on the considerable advances in QBF solvers in recent years.

It should be noted that this work does not attempt to construct PPCs that maximize ECO coverage or fault tolerance. Instead, it lays the theoretical groundwork for calculating these quantities, as well as for performing ECOs. As such, the work here remains orthogonal and complementary to that in [1] which is strictly focused on constructing PPCs. Experimental results are presented evaluating PPCs from [1], demonstrating the applicability and accuracy of the proposed QBF formulations.

The paper is organized as follows. Section II contains preliminaries on PPCs and QBFs. Section III presents our formulations for calculating fault and design error tolerance. Section IV gives QBF encodings for performing ECOs and quantifying ECO coverage. Section V shows experimental results and Section VI concludes the paper.

II. Preliminaries

The following notation is used throughout the paper. We use the symbol $\mathcal{C}$ to denote a conventional combinational circuit, and $\hat{\mathcal{C}}$ to denote the corresponding PPC. The sets $\mathbf{x} = \{x_1, x_2, \ldots, x_{|\mathbf{x}|}\}$, $\mathbf{y} = \{y_1, y_2, \ldots, y_{|\mathbf{y}|}\}$ and $\mathbf{g} = \{g_1, g_2, \ldots, g_{|\mathbf{g}|}\}$ respectively refer to the sets of primary inputs, primary outputs and gates of $\mathcal{C}$. A *node* v can refer to a gate or a primary input. The sets $fanout(v)$ and $fanin(v)$ denote the fan-out and fan-in nodes of v, respectively. The set $\mathbf{l} = \{(u, v) \mid u, v \in \mathbf{x} \cup \mathbf{g}$ and $v \in fanout(u)\}$ contains all *lines* (also referred to as *connections* or *branches*) in $\mathcal{C}$. For each $\mathbf{z} \in \{\mathbf{x}, \mathbf{y}, \mathbf{g}, \mathbf{l}\}$, $\hat{\mathbf{z}} = \{\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_{|\hat{\mathbf{z}}|}\}$ denotes the corresponding set in $\hat{\mathcal{C}}$. Throughout the paper, bold ($\mathbf{z}$) versus regular (z) symbols differentiate sets from single variables, and a hat ($\hat{z}$ versus z) differentiates between variables in $\hat{\mathcal{C}}$ and $\mathcal{C}$, respectively.

A. Partially Programmable Circuits

The type of a node v is given by $type(v) \in \{\texttt{IN}, \texttt{AND}, \texttt{OR}, \ldots, \texttt{LUT}, \texttt{MUX}\}$. A PPC $\hat{\mathcal{C}}$ is a Boolean network with three types of nodes [1]:

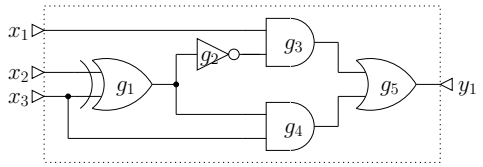
- Conventional logic gates, such as AND, OR, NOT and XOR.
- LUTs, whose internal functionality can be reconfigured.
- MUXs, whose select lines are controlled by programmable memory cells.

To simplify the presentation, we assume that the original circuit $\mathcal{C}$ does not contain LUTs/MUXs. Note that a LUT can itself be represented as a multiplexer with configurable data inputs. As such, the *configuration bits* of a LUT $\hat{g}_i$ are the set of Boolean variables:

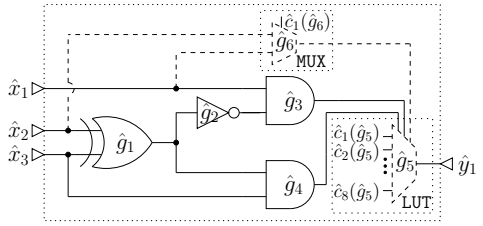
[1]ECE Department, University of Toronto, Toronto, ON ({hratch, veneris}@eecg.toronto.edu).

[2]VLSI Design and Education Center, University of Tokyo, Japan ({hiroaki@cad, fujita@ee}.t.u-tokyo.ac.jp).

[3]College of Information Science and Engineering, Ritsumeikan University, Japan (ger@cs.ritsumei.ac.jp).



(a) $\mathcal{C}$



(b) $\hat{\mathcal{C}}$

Fig. 1. A circuit and its corresponding PPC

$$\hat{\mathbf{c}}(\hat{g}_i) = \{\hat{c}_j(\hat{g}_i) \mid j = 1, \ldots, 2^n\},$$

where n denotes the number of input select lines of the LUT. On the other hand, the configuration bits of a configurable MUX $\hat{g}_i$ in a PPC are its select lines. They are given by:

$$\hat{\mathbf{c}}(\hat{g}_i) = \{\hat{c}_j(\hat{g}_i) \mid j = 1, \ldots, \lceil \log_2 n \rceil\},$$

where n denotes the number of data inputs of the configurable MUX.

Figures 1(a) and 1(b) show a combinational circuit $\mathcal{C}$ and a corresponding PPC $\hat{\mathcal{C}}$. Note that y_1 (respectively, $\hat{y}_1$) is the primary output label for g_5 (respectively, $\hat{g}_5$) and does not represent a separate node. In Figure 1(b), variables $\hat{c}_1(\hat{g}_6)$ and $\hat{c}_j(\hat{g}_5)$ $(j = 1, \ldots, 8)$ are the configuration bits of $\hat{g}_6$ and $\hat{g}_5$, respectively.

In [1], PPCs are constructed as follows. First, given an original circuit $\mathcal{C}$, an initial PPC is generated by replacing certain subcircuits of $\mathcal{C}$ with LUTs using simple heuristics. Next, redundant lines are added from selected nodes to some of these LUTs in an effort to increase the number of so-called *robust* connections in the PPC. A robust connection is a line where a stuck-at-0 and a stuck-at-1 can be made unobservable by reprogramming the PPC post-silicon. These added redundant lines are selected as follows. For each line (u, v), a set of new connections are added to the LUT inputs such that the *functional flexibilities* of the LUTs, represented by their Sets of Pairs of Functions to be Distinguished (SPFDs), allow them to be reconfigured to bypass stuck-at-faults at (u, v). Of course, this is not always possible given limited resources, so not all lines can be made robust. If more than one redundant line needs to be added to a certain LUT, a configurable MUX is placed in front of the LUT, which selects between these redundant lines. This paper is not concerned with constructing PPCs, hence the details of the algorithm given in [1] are not relevant. Our described techniques for evaluating PPCs can be applied to any PPC.

In the PPC shown in Figure 1(b), gate g_5 is replaced by a LUT $\hat{g}_5$. Of course, $\hat{g}_5$ can be easily programmed to implement OR$(\hat{g}_3, \hat{g}_4)$. Next, we have added redundant connections (shown using dashed lines) from $\hat{x}_1$ and $\hat{x}_2$ to a MUX $\hat{g}_6$, which is input to the LUT $\hat{g}_5$. In the coming sections, we present QBF formulations that can show that this PPC structure has 100% single stuck-at-fault tolerance (disregarding stack-at-faults at the primary output), 100% design error tolerance (assuming single gate arbitrary errors) and 100% ECO coverage (using $\mathcal{C}$ as the initial specification and our ECO coverage definition).

B. Quantified Boolean Formulas

A propositional logic formula Φ over a set of Boolean variables $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}$ is said to be satisfiable if it has a *satisfying assignment*: a truth assignment to $\mathbf{b}$ that makes Φ true (1). Otherwise, Φ is always false (0) and it is said to be unsatisfiable. This is known as the SAT problem. Φ is usually given in *Conjunctive Normal Form* (CNF) as a conjunction of *clauses*, where each clause is a disjunction of *literals*. A literal is an occurrence of a variable b_i or its negation $\neg b_i$. For example, $\Phi = (b_2 \vee b_3) \wedge (b_1 \vee \neg b_2 \vee \neg b_3) \wedge (\neg b_1)$ is in CNF. Given a combinational logic circuit, a CNF formula expressing the circuit constraints can be constructed in linear-time [12]. As such, a circuit and its corresponding CNF formula are referred to interchangeably in this work.

While SAT is NP-complete, QBF is a PSPACE-complete generalization of SAT that allows for the universal quantification of some variables. A QBF in *prenex normal form* is written as $Q.\Phi$, where Q is called the *prefix* and Φ is called the *matrix*. The matrix is a propositional logic formula over $\mathbf{b}$ in CNF. The prefix $Q = q_1\mathbf{v_1}\ q_2\mathbf{v_2}\ \cdots\ q_r\mathbf{v_r}$ consists of *quantifiers* $q_i \in \{\exists, \forall\}$, such that $q_i \neq q_{i+1}$, and mutually disjoint variable sets $\mathbf{v_i}$, called *scopes*, which partition $\mathbf{b}$. In other terms, $\bigcup_{i=1}^{r} \mathbf{v_i} = \mathbf{b}$ and $\bigcap_{i=1}^{r} \mathbf{v_i} = \emptyset$. A variable $b \in \mathbf{v_i}$ is labeled as an *existential* (respectively, *universal*) variable if $q_i = \exists$ (respectively, $q_i = \forall$). A scope $\mathbf{v_i}$ or variable $b \in \mathbf{v_i}$ is said to be *wider* (respectively, *narrower*) than a scope $\mathbf{v_j}$ or variable $b' \in \mathbf{v_j}$ if $i < j$ (respectively, $i > j$).

A QBF is true or QSAT if it has a so-called *Q-model*, otherwise it is false or UNQSAT. A *Q-model* is a tree of truth assignments satisfying the QBF semantics, where each existential variable is a function of wider universal variables, such that the matrix is satisfied for all universal variable assignments. For example, the QBF problem:

$$\exists b_1\ \forall b_2\ \exists b_3\ .\ (b_2 \vee b_3) \wedge (b_1 \vee \neg b_2 \vee \neg b_3) \wedge (\neg b_1)$$

is QSAT because when $b_1 = 0$, for all values of b_2, there exists an assignment to b_3 ($b_3 = 1$ when $b_2 = 0$ and $b_3 = 0$ when $b_2 = 1$) that satisfies the matrix. This tree of satisfying truth assignments is a Q-model. Some QBF solvers can return the satisfying assignments to the widest existential scope (here b_1) in a Q-model [11].

III. Fault and Design Error Tolerance

In this section, we first construct a QBF formulation for calculating the stuck-at-fault tolerance of a PPC. We use stuck-at-faults because this type of fault can model many defects [13]. Then, we extend this formulation to calculate the gate design error tolerance of a PPC. Finally, for single stuck-at-fault tolerance and single gate design error tolerance, we show how to partition our formulations into smaller parallelizable problems in order to achieve faster QBF solving times by taking advantage of modern multi-core architectures.

A. Fault Tolerance

Given a specification $\mathcal{C}$, and a corresponding implementation in the form of a PPC $\hat{\mathcal{C}}$ *with a fixed configuration*, in this paper we say that a stuck-at-fault (or a design error) in $\hat{\mathcal{C}}$ is *unobservable* if there does not exist any primary input vector for which $\hat{\mathcal{C}}$ and $\mathcal{C}$ produce different primary outputs. This can be extended to N stuck-at-faults, where N denotes the cardinality of simultaneous stuck-at-faults. In what follows, we use the term N*-faults* to denote N simultaneous stuck-at-faults.

Definition 1 *Given a specification $\mathcal{C}$, a PPC $\hat{\mathcal{C}}$ and a stuck-at-fault cardinality N, the* fault tolerance *of $\hat{\mathcal{C}}$ is the percentage of N-faults that can be made unobservable using reconfigurations.*

We emphasize that (assuming $N = 1$ for illustration purposes), different single stuck-at-faults are allowed to be made unobservable by *different* PPC reconfigurations. The goal is that in silicon, if a stuck-at-fault is detected during testing, we would like to be able to reprogram the PPC to "mask" it. In general, if for a given N-fault there exists a PPC reconfiguration making it unobservable, this N-fault counts towards the fault tolerance of the PPC. Again, reconfigurations can vary for different N-faults. Clearly, a high fault tolerance increases manufacturing yield because faults that otherwise would make the circuit unusable can now be made unobservable by reconfiguring the PPC LUTs/MUXs.

The key idea is to build a QBF instance whose "solutions" are in a one-to-one correspondence with all N-faults that cannot be made unobservable by any reconfiguration of $\hat{\mathcal{C}}$. In what follows, we explain how to create the matrix of our QBF formulation using an appropriate circuit construction. In order to assist the reader in visualizing our descriptions, Figure 2 illustrates this construction (which is described shortly) for $\mathcal{C}$ and $\hat{\mathcal{C}}$ given in Figures 1(a) and 1(b).

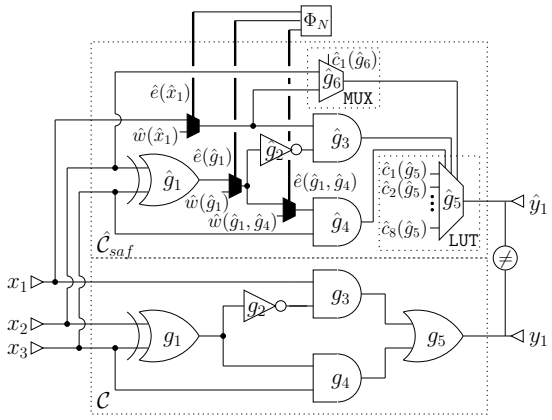


Fig. 2. Stuck-at-fault tolerance matrix

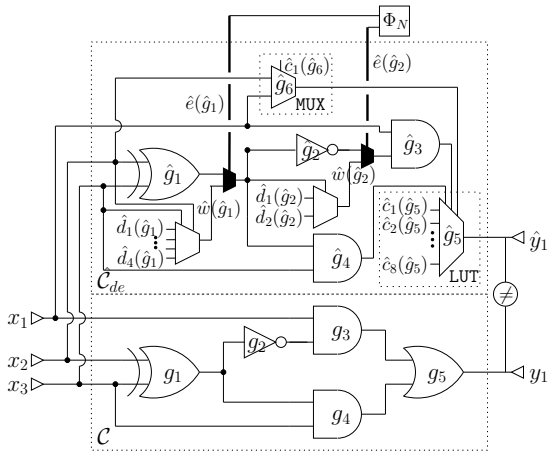


Fig. 3. Gate design error tolerance matrix

We first create an enhanced version of $\hat{\mathcal{C}}$, which we call $\hat{\mathcal{C}}_{saf}$. To prevent any confusion, we stress that any enhancements to $\hat{\mathcal{C}}_{saf}$ are only added to construct our QBF formulation. We do not modify the actual PPC $\hat{\mathcal{C}}$ in any way. We start by adding a special multiplexer in front of each gate, each line and each primary input, which determines whether or not a stuck-at-fault is excited at that gate, line or primary input. Note that gate and line stuck-at-faults in this context correspond to *stem* and *branch* stuck-at-faults [13], respectively. Of course, if a gate has only one fan-out, we do not double-count by adding two multiplexers at its output. The shaded multiplexers in Figure 2 illustrate this process for gate $\hat{g}_1$, line $(\hat{g}_1, \hat{g}_4)$ and primary input $\hat{x}_1$. We do not show the multiplexers for the remaining gates, lines and primary inputs to avoid overcrowding that figure. The select-line of each of these multiplexers is called an *excitation* variable, denoted by the letter $\hat{e}$.

In more detail, at each gate $\hat{g}_i$ (respectively, each line $(\hat{u}, \hat{v})$ and each primary input $\hat{x}_j$), setting $\hat{e}(\hat{g}_i) = 1$ (respectively, $\hat{e}(\hat{u}, \hat{v}) = 1$ and $\hat{e}(\hat{x}_j) = 1$) "excites" the stuck-at-fault, by disconnecting $\hat{g}_i$ (respectively, $(\hat{u}, \hat{v})$ and $\hat{x}_j$) from its fan-ins, and instead connecting it to a newly created variable $\hat{w}(\hat{g}_i)$ (respectively, $\hat{w}(\hat{u}, \hat{v})$ and $\hat{w}(\hat{x}_j)$), which we call a *replacement* variable. As will be seen later, these $\hat{w}$'s will denote the polarities of the stuck-at-faults. On the other hand, setting $\hat{e} = 0$ keeps the gate/line/primary input unchanged, as can be seen in Figure 2.

Next, we apply common primary inputs ($\mathbf{x}$) to both $\mathcal{C}$ and $\hat{\mathcal{C}}_{saf}$, as shown in Figure 2. Furthermore, at least one primary output is forced to be different. Finally, a *cardinality constraint* Φ_N is added to force the number of simultaneously active (*i.e.*, assigned to 1) excitation variables to a pre-specified constant N. This can be done using a bitonic sorter [14]. This completes the matrix of our QBF formulation.

In order to abbreviate the prefix of our QBF (as well as the remaining QBFs in this paper), we use the following notation:

$$\begin{aligned}
\hat{\mathbf{e}}(\hat{\mathbf{g}}) &= \{\hat{e}(\hat{g}_i) \mid \forall \hat{g}_i \in \hat{\mathbf{g}}\} & \hat{\mathbf{w}}(\hat{\mathbf{g}}) &= \{\hat{w}(\hat{g}_i) \mid \forall \hat{g}_i \in \hat{\mathbf{g}}\} \\
\hat{\mathbf{e}}(\hat{\mathbf{l}}) &= \{\hat{e}(\hat{u}, \hat{v}) \mid \forall (\hat{u}, \hat{v}) \in \hat{\mathbf{l}}\} & \hat{\mathbf{w}}(\hat{\mathbf{l}}) &= \{\hat{w}(\hat{u}, \hat{v}) \mid \forall (\hat{u}, \hat{v}) \in \hat{\mathbf{l}}\} \\
\hat{\mathbf{e}}(\hat{\mathbf{x}}) &= \{\hat{e}(\hat{x}_i) \mid \forall \hat{x}_i \in \hat{\mathbf{x}}\} & \hat{\mathbf{w}}(\hat{\mathbf{x}}) &= \{\hat{w}(\hat{x}_i) \mid \forall \hat{x}_i \in \hat{\mathbf{x}}\} \quad (1)
\end{aligned}$$

And the sets of all excitation and replacement variables are respectively given by:

$$\hat{\mathbf{e}} = \hat{\mathbf{e}}(\hat{\mathbf{g}}) \cup \hat{\mathbf{e}}(\hat{\mathbf{l}}) \cup \hat{\mathbf{e}}(\hat{\mathbf{x}}) \qquad \hat{\mathbf{w}} = \hat{\mathbf{w}}(\hat{\mathbf{g}}) \cup \hat{\mathbf{w}}(\hat{\mathbf{l}}) \cup \hat{\mathbf{w}}(\hat{\mathbf{x}}) \quad (2)$$

When the context of the type of excitation/replacement variable is clear, we just use the symbols $\hat{e} \in \hat{\mathbf{e}}$ and $\hat{w} \in \hat{\mathbf{w}}$ for brevity.

Recall that the set $\hat{\mathbf{c}}(\hat{g}_i)$ refers to the configuration bits of the LUT/MUX $\hat{g}_i$. Let:

$$\hat{\mathbf{c}} = \bigcup_{\substack{\hat{g}_i \in \hat{\mathbf{g}}, \\ type(\hat{g}_i) \in \{\texttt{LUT},\texttt{MUX}\}}} \hat{\mathbf{c}}(\hat{g}_i)$$

denote the set of all configuration bits in $\hat{\mathcal{C}}$.

Informally, the QBF problem can be stated as follows:

> *Is it possible to assign exactly N excitation variables in $\hat{\mathbf{e}}$ to 1, and set what each corresponding gate/line/primary input is "stuck-at" (by assigning $\hat{\mathbf{w}}$), such that for all configurations of the PPC (assignments to $\hat{\mathbf{c}}$), there exists a primary input vector satisfying the constraints in Figure 2?*

This question can be formalized as:

$$\exists \hat{\mathbf{e}}, \hat{\mathbf{w}}\ \forall \hat{\mathbf{c}}\ \exists \mathbf{x}, \mathbf{g}, \hat{\mathbf{g}}\ .$$
$$\mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{g}) \wedge \hat{\mathcal{C}}_{saf}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}, \hat{\mathbf{e}}, \hat{\mathbf{w}}) \wedge (\mathbf{y} \neq \hat{\mathbf{y}}) \wedge \Phi_N(\hat{\mathbf{e}}) \quad (3)$$

Notice that the placement of $\hat{\mathbf{w}}$ in the widest existential scope forces their assignment before the assignment of primary inputs, producing the semantics of stuck-at-faults. Adding constraints on these $\hat{w}$'s or moving them in the prefix can result in different error models, as will be seen shortly. If (3) is false or UNQSAT, then every N-fault can be made unobservable (*i.e.*, is "maskable") by a reconfiguration of the PPC.

In order to count the number of maskable (or unmaskable) stuck-at-faults using (3), we need to add another term to the matrix in (3). In fact, notice that if a certain excitation variable $\hat{e}$ is not active, its corresponding $\hat{w}$ can simply be "grounded" to 0, since its value does not propagate through the multiplexer. As such, we add the following constraints to (3):

$$\bigwedge_{\hat{e} \in \hat{\mathbf{e}}} (\neg \hat{e} \rightarrow \neg \hat{w}) \quad (4)$$

Adding (4) prunes the search-space of the QBF solver, such that in any Q-model of (3), the $\hat{w}$'s corresponding to the inactive $\hat{e}$'s are assigned to 0. As such, two Q-models of this QBF that differ in their truth assignments to the widest existential scope $(\hat{\mathbf{e}}, \hat{\mathbf{w}})$ will correspond to two different N-faults that cannot be fixed by the PPC. Therefore, finding all distinct truth assignments to $\hat{\mathbf{e}}, \hat{\mathbf{w}}$ that satisfy (3) (*i.e.*, that can be extended to Q-models of (3)) is equivalent to finding all unmaskable N-faults. This can be done using a QBF solver, by blocking the assignment to $\hat{\mathbf{e}}, \hat{\mathbf{w}}$ in the returned Q-model using a blocking clause and re-solving (3) iteratively until the problem becomes UNQSAT. Subtracting the number of such solutions from the total number of N-fault combinations, and dividing the result by this number gives the stuck-at-fault tolerance of the PPC for cardinality N.

B. Design Error Tolerance

In this subsection, we propose a QBF formulation to quantify the effectiveness of a PPC in masking localized design errors that escape verification and slip into the silicon. Our design error model consists of *any* functional modification in the function of a gate. We use the term *N-gates* to denote a set of N gates.

Definition 2 *Given a specification $\mathcal{C}$, a PPC $\hat{\mathcal{C}}$ and a gate design error cardinality N, the* design error tolerance *of $\hat{\mathcal{C}}$ is the percentage of N-gates where* any *simultaneous modifications can be made unobservable using reconfigurations.*

For instance, if $N = 1$, the design error tolerance is equal to the percentage of gates where any design error can be masked by a reconfiguration. In other terms, gates where at least one type of design error cannot be masked by any reconfiguration do not contribute to the design error tolerance. In the event a design error is identified post-silicon, a PPC with high design error tolerance is likely to offer a configuration fix, allowing the circuit to operate correctly without the need for a costly respin.

In this subsection, we modify the QBF in (3) to deal with gate design errors. We model design errors by again enhancing $\hat{\mathcal{C}}$. Here, $\hat{\mathcal{C}}_{de}$ adds similar multiplexers as in Figure 2 but now only at the outputs of gates. Furthermore, the $\hat{w}(\hat{g}_i)$'s are no longer unconstrained and instead are the outputs of newly added LUTs whose select lines are $\hat{g}_i$'s inputs. This allows each $\hat{w}(\hat{g}_i)$ to be any function of the inputs of $\hat{g}_i$, thus implementing any gate design error. This construction is illustrated in Figure 3, where shaded multiplexers are added for gates $\hat{g}_1$ and $\hat{g}_2$. For each gate, $\hat{g}_i$, the set:

$$\hat{\mathbf{d}}(\hat{g}_i) = \{\hat{d}_j(\hat{g}_i) \mid j = 1, \ldots, 2^{|fanin(\hat{g}_i)|}\}$$

refers to the configuration bits of the *replacement* LUT $\hat{w}(\hat{g}_i)$.

Again, applying common primary inputs, forcing different primary outputs and adding cardinality constraints yields the matrix in Figure 3. Using this, our QBF formulation is given as follows:

$$\exists \hat{\mathbf{e}}(\hat{\mathbf{g}}), \hat{\mathbf{d}}(\hat{\mathbf{g}})\ \forall \hat{\mathbf{c}}\ \exists \mathbf{x}, \mathbf{g}, \hat{\mathbf{g}}, \hat{\mathbf{w}}(\hat{\mathbf{g}})\ .$$
$$\mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{g}) \wedge \hat{\mathcal{C}}_{de}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}, \hat{\mathbf{e}}(\hat{\mathbf{g}}), \hat{\mathbf{w}}(\hat{\mathbf{g}}), \hat{\mathbf{d}}(\hat{\mathbf{g}})) \wedge (\mathbf{y} \neq \hat{\mathbf{y}}) \wedge \Phi_N(\hat{\mathbf{e}}) \tag{5}$$

which asks whether there exist N gates that can be arbitrarily modified such that for all PPC configurations ($\hat{\mathbf{c}}$), there is always an input vector exhibiting the error at a primary output. Similarly to (4), we add the following constraints that ground the configuration bits $\hat{\mathbf{d}}(\hat{g}_i)$ of $\hat{w}(\hat{g}_i)$ for gates whose excitation variables are inactive:

$$\bigwedge_{\hat{e}(\hat{g}_i) \in \hat{\mathbf{e}}(\hat{\mathbf{g}})} \left(\neg \hat{e}(\hat{g}_i) \rightarrow \left(\bigwedge_{\hat{d}_j(\hat{g}_i) \in \hat{\mathbf{d}}(\hat{g}_i)} \neg \hat{d}_j(\hat{g}_i) \right) \right) \tag{6}$$

This is done in order to create a one-to-one correspondance between different satisfying truth assignments to $\hat{\mathbf{e}}(\hat{\mathbf{g}}), \hat{\mathbf{d}}(\hat{\mathbf{g}})$ and different N-gate design errors that cannot be masked by the PPC. Finding all these satisfying assignments using blocking clauses enables us to calculate the gate design error tolerance of the PPC for cardinality N.

C. Problem Partitioning for $N = 1$

Most often, we are interested in calculating single stuck-at-fault tolerance and single gate design error tolerance. It is usually very difficult to mask multiple simultaneous faults or errors, especially with limited redundancy as in PPCs. Here, we show that when $N = 1$, we can partition the QBF problem (for both (3) and (5)) into a linear number of independently solvable and much easier subproblems, in order to take advantage of the modern multi-core architectures in solving these QBF instances.

For single stuck-at-fault tolerance, the partitioning is done by enumerating each $\hat{e} \in \hat{\mathbf{e}}$ and the corresponding two polarities of $\hat{w}$. For each gate/line/primary input with excitation variable $\hat{e}^*$ and replacement variable $\hat{w}^*$, and each stuck-at value $b \in \{0, 1\}$, we let:

$$\hat{\mathcal{C}}_{saf}|_{\hat{e}^*, \hat{w}^* = b} \triangleq \hat{\mathcal{C}}_{saf} \wedge \hat{e}^* \wedge (\hat{w}^* = b) \wedge \bigwedge_{\hat{e} \in \hat{\mathbf{e}} - \{\hat{e}^*\}} (\neg \hat{e}) \tag{7}$$

denote the PPC with only that gate/line/primary input stuck-at-*b*. We now ask whether there exists a PPC configuration, such that for all primary inputs, this faulty circuit produces the same outputs as $\mathcal{C}$. Formally, this is stated as:

$$\exists \hat{\mathbf{c}}\ \forall \mathbf{x}\ \exists \mathbf{g}, \hat{\mathbf{g}}\ .\ \mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{g}) \wedge \hat{\mathcal{C}}_{saf}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}, \hat{\mathbf{e}}, \hat{\mathbf{w}})|_{\hat{e}^*, \hat{w}^* = b} \wedge (\mathbf{y} = \hat{\mathbf{y}}) \tag{8}$$

Note that the cardinality constraints are no longer necessary because $\hat{\mathbf{e}}$ is already assigned a-priori, and all the inactive shaded multiplexers in Figure 2 can be discarded due to (7). Now although (8) must be solved for every single stuck-at-fault, each of these QBF instances is completely independent and much easier to solve than (3). As such, the number of maskable single stuck-at-faults can be computed by heavily parallelizing all the QBFs of the form (8) and simply counting the number of QSAT results.

A similar partitioning can be accomplished for the single gate design error tolerance formulation in (5). Here, for each gate $\hat{g}_i$, we let:

$$\hat{\mathcal{C}}_{de}|_{\hat{e}(\hat{g}_i)} \triangleq \hat{\mathcal{C}}_{de} \wedge \hat{e}(\hat{g}_i) \wedge \bigwedge_{\hat{e}(\hat{g}_j) \in \hat{\mathbf{e}}(\mathbf{g}) - \{\hat{e}(\hat{g}_i)\}} (\neg \hat{e}(\hat{g}_j)) \tag{9}$$

denote the PPC where only $\hat{g}_i$ can have a design error. We now ask whether for all possible design errors at $\hat{g}_i$, there exists a PPC configuration that masks the error. Formally,

$$\forall \hat{\mathbf{d}}(\hat{g}_i)\ \exists \hat{\mathbf{c}}\ \forall \mathbf{x}\ \exists \mathbf{g}, \hat{\mathbf{g}}, \hat{w}(\hat{g}_i)\ .$$
$$\mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{g}) \wedge \hat{\mathcal{C}}_{de}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}, \hat{\mathbf{e}}(\hat{\mathbf{g}}), \hat{\mathbf{w}}(\hat{\mathbf{g}}), \hat{\mathbf{d}}(\hat{\mathbf{g}}))|_{\hat{e}(\hat{g}_i)} \wedge (\mathbf{y} = \hat{\mathbf{y}}) \tag{10}$$

In each QBF of the form of (10), all $\hat{\mathbf{d}}(\hat{g}_j)$ and $\hat{w}(\hat{g}_j)$ with $j \neq i$ can be disregarded, since they cannot propagate through the shaded multiplexers in Figure 3. Again, for each gate, a QBF of the form of (10) must be solved to determine whether all possible errors at that gate can be masked by the PPC. All these QBFs can be solved in parallel. The single gate design error tolerance of the PPC is equal to the ratio of these QBFs that are QSAT.

IV. Engineering Change Order

In this section, we first construct a QBF for performing an ECO using a PPC. Then, we define the ECO coverage of a PPC and show how to compute it using a QBF.

A. Performing ECOs

ECOs are small changes in the specification at later stages of the design cycle. Synthesis for ECOs strives to make the smallest number of changes to the implementation [8]–[10]. PPCs can be used to implement ECOs pre- or post-silicon by simply reprogramming the MUXs/LUTs.

Given a modified specification $\mathcal{C}_{mod}$, if there exists a configuration of the PPC $\hat{\mathcal{C}}$, such that for all primary inputs, $\hat{\mathcal{C}}$ and $\mathcal{C}_{mod}$ behave identically, then the ECO can be implemented by reprogramming the PPC. This is easily expressed as the following QBF:

$$\exists \hat{\mathbf{c}}\ \forall \mathbf{x}\ \exists \mathbf{g}, \hat{\mathbf{g}}\ .\ \mathcal{C}_{mod}(\mathbf{x}, \mathbf{y}, \mathbf{g}) \wedge \hat{\mathcal{C}}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}) \wedge (\mathbf{y} = \hat{\mathbf{y}}) \tag{11}$$

Figure 4 illustrates the matrix of (11) given a specification $\mathcal{C}_{mod}$ where the NOT gate g_2 has been eliminated and $g_4 = \text{AND}(x_3, g_1)$ has been replaced by $g_4 = \text{NAND}(x_1, g_1)$. Using a QBF solver, it can be easily verified that the QBF (11) with the matrix shown in Figure 4 is QSAT. The satisfying assignment to the configuration bits $\hat{\mathbf{c}}$ returned by the solver can be used to reprogram the PPC to implement the modified specification at essentially zero-cost. Interestingly, the QBF in (11) is similar to a formualtion used for FPGA technology mapping given in [15].

B. ECO Coverage

Given a PPC $\hat{\mathcal{C}}$ and an original specification $\mathcal{C}$, we would like to measure the effectiveness of this PPC architecture in implementing small changes in $\mathcal{C}$. Given a change cardinality N, a simple way to model small changes in the specification netlist $\mathcal{C}$ is to allow N gates to be changed *arbitrarily*. As such, we define the *ECO coverage* of a PPC as follows:

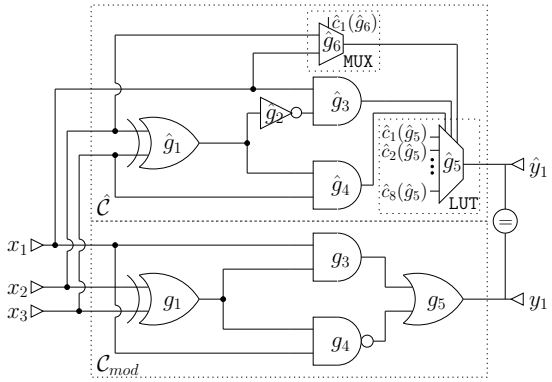


Fig. 4. Engineering change matrix

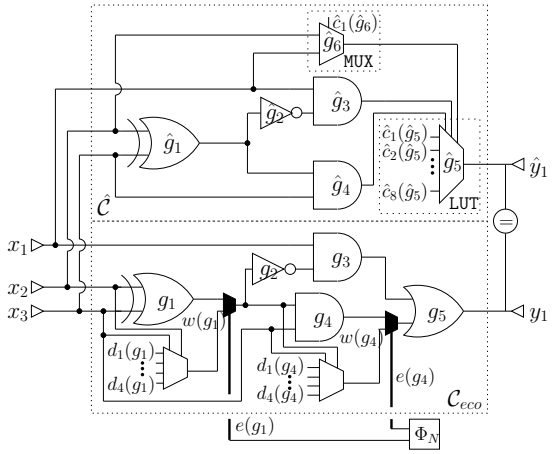


Fig. 5. ECO coverage matrix

Definition 3 *Given an original specification $\mathcal{C}$, a PPC $\hat{\mathcal{C}}$ and a change cardinality N, the* ECO coverage *of $\hat{\mathcal{C}}$ is the percentage of N-gates in $\mathcal{C}$, where* any *simultaneous modifications can be implemented using reconfigurations in $\hat{\mathcal{C}}$.*

Note that many ECOs involve changes at a higher abstraction level, for which different models should be considered. Furthermore, since this paper deal with combinational PPCs, sequential specification changes are not covered. Our formulation for ECO coverage is essentially the dual of the formulation for design error tolerance given in (5). Here, we must enhance the specification circuit $\mathcal{C}$ instead of $\hat{\mathcal{C}}$, since we are allowing the specification to change. A multiplexer is added at the output of each gate g_i in $\mathcal{C}$, with *excitation* select line $e(g_i)$. Furthermore, similarly to Figure 3, the $w(g_i)$'s are the outputs of newly added replacement LUTs, whose select lines are g_i's inputs. This allows each $w(g_i)$ to be any function of the inputs of g_i, thus modeling any gate change at g_i, when $e(g_i) = 1$. This construction is illustrated in Figure 5, where shaded multiplexers are added for gates g_1 and g_4 (we have skipped the remaining gates to avoid overcrowding the figure). As before, for each gate g_i, the set:

$$\mathbf{d}(g_i) = \{d_j(g_i) \mid j = 1, \ldots, 2^{|fanin(g_i)|}\}$$

refers to the configuration bits of the replacement LUT $w(g_i)$.

Informally, the QBF problem can be stated as follows:

> *Do there exist N gates in the specifications ($\mathbf{e}(\mathbf{g})$), such that for* any *modification of these gates ($\mathbf{d}(\mathbf{g})$), there exists a PPC configuration ($\hat{\mathbf{c}}$), such that for all primary inputs, this PPC correctly implements the modified specification?*

Adding cardinality constraints $\Phi_N(\mathbf{e})$, applying common primary inputs and forcing the primary outputs to be equal, we get the matrix in Figure 5 and the following QBF formulation:

$$\exists \mathbf{e}(\mathbf{g})\ \forall \mathbf{d}(\mathbf{g})\ \exists \hat{\mathbf{c}}\ \forall \mathbf{x}\ \exists \mathbf{g}, \hat{\mathbf{g}}, \mathbf{w}(\mathbf{g})\ .$$
$$\mathcal{C}_{eco}(\mathbf{x}, \mathbf{y}, \mathbf{g}, \mathbf{e}(\mathbf{g}), \mathbf{w}(\mathbf{g}), \mathbf{d}(\mathbf{g})) \wedge \hat{\mathcal{C}}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}) \wedge (\mathbf{y} = \hat{\mathbf{y}}) \wedge \Phi_N(\mathbf{e}) \quad (12)$$

where $\mathbf{e}(\mathbf{g})$ and $\mathbf{w}(\mathbf{g})$ are defined similarly to (1).

In (12), only $\mathbf{e}(\mathbf{g})$ is in the widest scope, so counting all the satisfying assignments to $\mathbf{e}(\mathbf{g})$ using blocking clauses gives the number of N-gates where any change can be implemented by the PPC using reconfigurations.

C. Problem Partitioning for $N = 1$

In the case where exactly one gate is allowed to arbitrarily change in the specification (*i.e.,* $N = 1$), (12) can be partitioned into $|\mathbf{g}|$ smaller, independent QBFs by enumerating each $e(g_i) \in \mathbf{e}(\mathbf{g})$. For each gate g_i, we let:

$$\mathcal{C}_{eco}|_{e(g_i)} \triangleq \mathcal{C}_{eco} \wedge e(g_i) \wedge \bigwedge_{e(g_j) \in \mathbf{e}(\mathbf{g}) - \{e(g_i)\}} (\neg e(g_j)) \quad (13)$$

denote the specification where only g_i is allowed to change. We now ask whether for all possible changes at g_i, there exists a PPC configuration that can implement it. Formally,

$$\forall \mathbf{d}(g_i)\ \exists \hat{\mathbf{c}}\ \forall \mathbf{x}\ \exists \mathbf{g}, \hat{\mathbf{g}}, w(g_i)\ .$$
$$\mathcal{C}_{eco}(\mathbf{x}, \mathbf{y}, \mathbf{g}, \mathbf{e}(\mathbf{g}), \mathbf{w}(\mathbf{g}), \mathbf{d}(\mathbf{g}))|_{e(\hat{g}_i)} \wedge \hat{\mathcal{C}}(\mathbf{x}, \hat{\mathbf{y}}, \hat{\mathbf{g}}, \hat{\mathbf{c}}) \wedge (\mathbf{y} = \hat{\mathbf{y}}) \quad (14)$$

In each QBF of the form of (14), all $\mathbf{d}(g_j)$ and $w(g_j)$ with $j \neq i$ can be disregarded, since they cannot propagate through the shaded multiplexers in Figure 5. For each gate, a QBF of the form of (14) must be solved to determine whether all possible modifications at that gate in the specification can be implemented by the PPC. All these QBFs can be solved in parallel. The ECO coverage of the PPC is equal to the ratio of these QBFs that are QSAT.

V. Experimental Results

This section presents the experimental evaluation of 21 PPCs from [1] using our proposed QBF formulations. These PPCs are generated by [1] from some of the MCNC benchmark circuits [16]. Experiments are run on a quad-core Intel i5, 3.1 Ghz workstation with 16 GB of RAM. Since complex faults can be modeled using single stuck-at-faults [13], and given the limited number of LUTs in the PPCs of [1], we set $N = 1$ in our tolerance and coverage calculations. We use the proposed QBF partitioning schemes in Subsections III-C and IV-C to speed up the solving process. For each tolerance/coverage computation, the QBF subproblems are solved in parallel over the four cores. A timeout of 100 seconds is used for *each* QBF subproblem. The QBF solver `sKizzo-v0.11c` [11] is used to solve all QBF instances. Other QBF solvers, such as QuBE7 [17] give similar results.

Table I shows the results of our evaluations. The first eight columns under **PPC information** describe the PPCs [1]. The first five columns respectively show the PPC name, its number of gates $|\hat{\mathbf{g}}|$, lines $|\hat{\mathbf{l}}|$, added LUTs and added MUXs. Next, columns *added lines* and *% added lines* respectively show the number of redundant lines added by [1] to the LUTs/MUXs and the percentage of added lines to all lines in the PPC. Column *% LUTs+MUXs* gives the percentage of gates that are added LUTs/MUXs compared to all gates in $|\hat{\mathbf{g}}|$. The columns under **PPC evaluation** present the results of the tolerance and coverage metrics outlined in this work. The first two columns respectively show the *fault tolerance* of $\hat{\mathcal{C}}$ and the total time required for all the corresponding QBF subproblems to terminate. The next two columns give the *design error tolerance* of $\hat{\mathcal{C}}$ and the total time to compute it. And finally, the *ECO coverage* measure along with its computation run-time are given.

For the circuit *pair* shown in table I, the fault coverage and ECO coverage are, respectively, at least 40% and at least 52%, because a small number (roughly 5%) of the QBF subproblems for each of these calculations does not terminate by 100 seconds. Note that since

TABLE I
PPC EVALUATION RESULTS

PPC information								PPC evaluation					
$\hat{C}$	$\|\hat{g}\|$	$\|\hat{l}\|$	LUTs	MUXs	added lines	% added lines	%LUTs +MUXs	fault tolerance	time (sec)	DE tolerance	time (sec)	ECO coverage	time (sec)
alu2	335	797	5	3	21	3%	2%	33%	35.5	10%	11.2	16%	12.1
alu4	627	1459	7	5	35	2%	2%	29%	1145.3	12%	216.7	16%	269.3
apex6	866	1805	98	29	143	8%	15%	54%	461.0	28%	140.1	65%	197.3
apex7	295	586	37	17	74	13%	18%	58%	39.4	22%	9.1	71%	13.6
b9	163	349	20	14	66	19%	21%	70%	8.0	23%	2.0	67%	3.0
c8	145	282	18	6	27	10%	17%	33%	2.5	6%	0.8	63%	1.2
cc	116	206	19	9	26	13%	24%	63%	1.8	23%	0.5	85%	0.7
comp	106	234	2	2	21	9%	4%	13%	96.2	7%	60.1	10%	37.8
example2	477	997	65	31	148	15%	20%	48%	102.7	17%	22.3	70%	39.2
f51m	109	250	8	2	17	7%	9%	60%	1.3	37%	0.5	60%	0.6
frg1	94	197	3	1	5	3%	4%	11%	7.1	5%	3.0	20%	4.7
lal	150	298	19	10	45	15%	19%	76%	4.7	36%	1.2	70%	1.5
mux	60	146	1	1	6	4%	3%	39%	6.9	12%	3.9	25%	4.6
pair	1364	3246	100	85	482	15%	14%	$\geq$ 40%	18431.8	12%	8736.4	$\geq$ 52%	10566.9
t481	824	2319	1	1	39	2%	0%	81%	7843.7	65%	1871.6	68%	2921.8
term1	186	458	10	7	57	12%	9%	90%	64.2	66%	15.9	56%	16.6
too_large	436	1029	3	3	38	4%	1%	37%	9128.1	21%	1633.7	22%	1946.7
vda	749	1928	39	20	192	10%	8%	95%	980.6	86%	257.4	88%	234.9
x1	359	783	35	20	109	14%	15%	48%	140.4	16%	31.9	50%	42.0
x3	912	1871	99	70	285	15%	19%	55%	823.4	18%	171.6	65%	222.7
x4	563	1279	71	47	287	22%	21%	66%	325.5	22%	54.9	62%	78.3

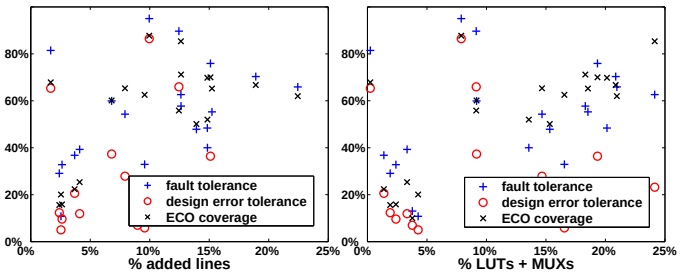
the QBF subproblems used in our computations are independent, it is easy to improve our run-times by simply parallelizing more heavily.

Figures 6(a) and 6(b) plot the calculated metrics against *% added lines* and *% LUTs+MUXs*, respectively. As expected, adding more redundant lines to the LUTs/MUXs, and replacing more gates with LUTs increases both fault tolerance and ECO coverage. On the other hand, the correlation of these two variables with design error tolerance is weaker, at least given the considered family of PPCs.

On average, only 10% of the lines in the PPCs are added as overhead, and only 12% of the gates are added LUTs or MUXs. In fact, LUTs *replace* other gates in the original circuit, so the overhead in the number of added gates is much less than 12%. We found that these PPCs have a 53% average single stuck-at-fault tolerance, a 26% average single gate design error tolerance, and a 52% average ECO coverage. From these results, we can conclude that the small hardware overhead is more than compensated by the fault/error tolerance and ECO coverage that these architectures demonstrate, confirming that PPCs are attractive architectures to increase silicon yield and reduce the cost of the design/manufacturing cycle. Furthermore, the existence of methods for computing these metrics encourages further research on improving PPCs.

VI. CONCLUSION

PPCs are circuits with limited reconfigurability. This paper lays the theoretical groundwork for evaluating PPCs with QBF satisfiability. QBF models are given to calculate the fault tolerance and design error tolerance of a PPC. Next, QBF formulations are proposed for performing ECOs, and for quantifying the ECO coverage of a PPC architecture. Experimental results are presented that evaluate existing PPCs, demonstrating the applicability of the proposed formulations and confirming the attractiveness of PPCs for increasing silicon yield and reducing the cost of the design/manufacturing cycle.



(a) vs % added lines (b) vs %LUTs+MUXs

Fig. 6. Fault tolerance, design error tolerance and ECO coverage

REFERENCES

[1] S. Yamashita, H. Yoshida, and M. Fujita, "Increasing yield using partially-programmable circuits," in *Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, 2010, pp. 237–242.

[2] *International technology roadmap for semiconductors*, 2007.

[3] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[4] S. Sarangi, S. Narayanasamy, B. Carneal, A. Tiwari, B. Calder, and J. Torrellas, "Patching processor design errors with programmable hardware," *IEEE Micro*, vol. 27, no. 1, pp. 12–25, 2007.

[5] M. Abramovici, C. Stroud, and M. Emmert, "Using embedded FPGAs for SoC yield improvement," in *Design Automation Conf.*, 2002, pp. 713–724.

[6] A. Doumar and H. Ito, "Detecting, diagnosing, and tolerating faults in sram-based field programmable gate arrays: a survey," *IEEE Trans. on VLSI Systems*, vol. 11, no. 3, pp. 386–405, 2003.

[7] S. Yamashita, H. Sawada, and A. Nagoya, "SPFD: A new method to express functional flexibility," *IEEE Trans. on CAD*, vol. 19, no. 8, pp. 840–849, 2000.

[8] G. Swamy, S. Rajamani, C. Lennard, and R. Brayton, "Minimal logic re-synthesis for engineering change," in *IEEE International Symposium on Circuits and Systems*, vol. 3, 1997, pp. 1596–1599.

[9] D. Brand, A. Drumm, S. Kundu, and P. Narain, "Incremental synthesis," in *Int'l Conf. on CAD*, 1994, pp. 14–18.

[10] Y. Kuo, Y. Chang, S. Chang, and M. Marek-Sadowska, "Engineering change using spare cells with constant insertion," in *Int'l Conf. on CAD*, 2007, pp. 544–547.

[11] M. Benedetti, "sKizzo: a suite to evaluate and certify QBFs," in *Int'l Conf. on Automated Deduction*, 2005, pp. 369–376.

[12] G. S. Tseitin, "On the complexity of derivations in the propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic*, 1968, pp. 115–125.

[13] N. Jha and S. Kundu, *Testing and reliable design of CMOS circuits*. Kluwer Academic Publishers Boston., 1990.

[14] K. Batcher, "Sorting networks and their applications," in *Proceedings of AFIPS, Spring Joint Computer Conference*, 1968, pp. 307–314.

[15] A. Ling, D. Singh, and S. Brown, "FPGA PLB architecture evaluation and area optimization techniques using boolean satisfiability," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 7, pp. 1196–1210, 2007.

[16] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," *MCNC*, 1991.

[17] E. Giunchiglia, M. Narizzano, and A. Tacchella, "QUBE: A system for deciding quantified boolean formulas satisfiability," *Automated Reasoning*, pp. 364–369, 2001.