Toward Automated ECOs in FPGAs

Andrew C. Ling, *Member, IEEE,* Stephen D. Brown, *Member, IEEE,* Sean Safarpour, *Member, IEEE,* and Jianwen Zhu, *Member, IEEE*

***Abstract*—Engineering change orders (ECOs), which are used to apply late-stage specification changes and bug fixes, have become an important part of the field-programmable gate array design flow. ECOs are beneficial since they are applied directly to a placed-and-routed netlist which preserves most of the engineering effort invested previously. Unfortunately, designers often apply ECOs in a manual fashion which may have an unpredictable impact on the design's final correctness and end costs. As a solution, this paper introduces an automated method to tackle the ECO problem. This paper uses a novel resynthesis technique which can automatically update the functionality of a circuit by leveraging the existing logic within the design, thereby removing the inefficient manual effort required by a designer. The technique presented in this paper is robust enough to handle a wide range of changes. Furthermore, the technique can successfully make late-stage functional changes while minimally perturbing the placed-and-routed netlist: something that is necessary for ECOs. Also, this technique does this with a minimal impact on the circuit performance where on average over 90% of the placement and routing wires remain unchanged.**

***Index Terms*—Boolean satisfiability, field-programmable gate array (FPGA), logic design—automatic synthesis, verification.**

I. Introduction

As FIELD-PROGRAMMABLE gate array (FPGA) design complexity increases, achieving tight timing and area constraints is becoming very challenging and often requires several design iterations as indicated at the top of Fig. 1. Here, a design described in a hardware description language (HDL) is passed to an FPGA CAD flow such as Altera's Quartus II CAD system. If design constraints are not met after the CAD flow finishes, the designer must modify their existing HDL code and rerun the CAD flow. This initial design iteration can often take several months to complete. To complicate things, even if performance constraints are met, bug fixes or feature changes may still be required. Applying these changes directly to the HDL code and using a "from scratch" FPGA recompile (i.e., full recompile) is often not possible since this does not guarantee that circuit performance will be maintained [1]. To avoid this problem, designers typically handle late-stage changes through a process known as engineering change orders (ECOs). ECOs are small functional changes applied directly to a placed-and-routed netlist; common operations include rewiring of lookup tables (LUTs) or changing LUT masks. Since the changes are small, ECOs have a very predictable impact on the final performance of the circuit and preserve much of the engineering effort previously invested in the design.

In a design flow where almost all tasks are automated, ECOs remain a primarily manual process. As a result, applying ECOs is very error-prone and can require several iterations to correctly modify a design. This is shown in Fig. 1 through the path labeled as *Original Flow*. The feedback process shown in *Original Flow* can often tie up a designer for several months [2]–[5] leading to missed project deadlines. With very short design cycles demanded today, this is very detrimental to a product's success.

To avoid the expensive and error-prone nature of ECOs, we introduce a resynthesis approach that is able to automatically update the behavior of a circuit by using existing logic within the design. This is beneficial because it removes the need for manual work by designers and can potentially reduce the time spent on ECOs from weeks to a few hours.

The proposed flow is shown at the bottom of Fig. 1 labeled as *New Flow*. Unlike the *Original Flow*, the *New Flow* does not require an iterative approach. During *New Flow*, the circuit behavior is updated using two steps. First, the proposed algorithm applies a localization step that isolates nodes within the netlist which need to be changed. For example, during bug-fixes, the localization step will isolate nodes which are the source of the erroneous behavior of the circuit. Next, the proposed algorithm applies a resynthesis step to alter the circuit. This step first defines a function that will replace the existing nodes found in the localization step. Then, the proposed algorithm uses a SAT-based decomposition to decompose the necessary changes into a set of subfunctions. During decomposition, subfunctions must exist within the placed-and-routed netlist. This ensures that the proposed algorithm has a predictable impact on the final performance of the circuit. By using a SAT-based approach, the technique can handle multiple changes, and does not require a formal verification step to ensure that the modifications are correct.

The ultimate goal of this paper is to create an ECO flow that can automatically update the behavior of a circuit in a manner

Manuscript received February 14, 2010; revised May 23, 2010; accepted July 21, 2010. Date of current version December 17, 2010. This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Altera Corporation. This paper was presented in part at FPGA'09. This paper was recommended by Associate Editor K. Bazargan.

A. C. Ling is with the CAD Team, Toronto, ON M5S 3G4, Canada (e-mail: aling@eecg.toronto.edu).

S. D. Brown is with the University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: brown@eecg.toronto.edu).

S. Safarpour is with Vennsa Technologies, Toronto, ON M5V 3B1, Canada (e-mail: sean@eecg.toronto.edu).

J. Zhu is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: jzhu@eecg.toronto.edu).

Digital Object Identifier 10.1109/TCAD.2010.2067833

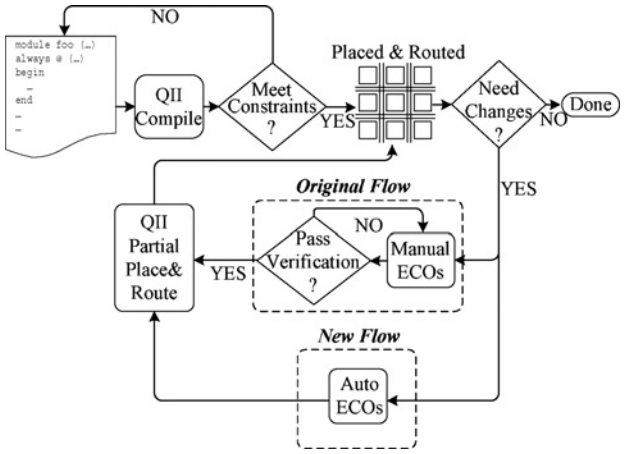


Fig. 1. Generalized flow using ECOs.

which preserves as much as possible the existing placement, routing, and timing. This can be achieved by satisfying the following three criteria: 1) be minimally disruptive to an existing placed-and-routed circuit; 2) have little or no impact on the performance of a circuit; and 3) be robust enough to handle a wide range of changes. In later sections, we will show that our approach satisfies these criteria. In particular, when applied to Altera's Stratix architecture using several benchmarks from the Altera QUIP [6] and the ITC benchmark suite we will show our technique on average leaves over 90% of a placed-and-routed circuit unchanged, and has a marginal impact on circuit performance where we reduce the frequency by less than 3% on average.

The rest of this paper is organized as follows. Section II discusses the background and related work, Section III describes the proposed algorithm in detail, Section IV gives experimental results, Section V discusses some limitations of the proposed algorithm, and Section VI concludes.

II. Background and Related Work

A. Terminology

The combinational portion of a LUT network can be represented as a directed acyclic graph. A node in the graph represents a LUT, primary input (PI), or primary output (PO). An *edge* in the graph with head v, and tail u, represents a signal in the logic circuit that is an output of node u and an input of node v. PIs are nodes with no inputs, and primary outputs (POs) are nodes with no outputs.

A node u is a *transitive fanout* (*fanin*) of v if there exists a path between an input (output) of u and an output (input) of v. In this paper, we will often use the term transitive fanout or transitive fanin of a node v to refer to all nodes which are a transitive fanout or transitive fanin to node v. A *dominator*, x, of node v is a node where all transitive fanouts of v that are not transitive fanouts of x, are transitive fanins of x (i.e., x dominates v if all paths that leave v will contain node x).

When defining a function at a node v, it is termed a *global function* if the function support set consists of only PIs. Conversely, a *local function* is a function whose support set consists of variables which may not be PIs. A *support set* of a function is the set of variables that directly determine the output value of a function and the size of the support set is the number of variables in the set.

B. Related Work

ECOs cover a wide range of work which is either used to incrementally improve the performance of a design [7] or help modify the behavior of a design such that circuit performance is maintained [8]–[14]. Our work falls in the latter category where we focus on late-stage ECOs which are applied directly to a placed-and-routed netlist. Late-stage functional changes often occur due to last minute feature changes or due to bugs which have been missed in previous verification phases. The most recent steps toward the automation of the ECO experience include [9], [10], and [14], where ours is most similar to [10]. Here, using formal methods and random simulation, the authors in [9] and [10] show how netlist modifications can be automated. To apply modifications, the authors use random simulation vectors to stimulate the circuit. Using the resulting vectors at each circuit node, they are able to find suggested alterations to their design to match a specified behavior. Following their modifications, they require a formal verification step to ensure that their modification is correct. The results of their work are promising where they can automatically apply ECOs in more than 70% of the cases they present.

The technique in [9] and [10] requires an explicit representation of any modification, which does not scale to large changes. In particular, their technique builds a truth-table based on the information returned by a set of random simulation vectors. This truth-table describes the desired logic function for the ECO and is derived exhaustively by enumerating the entire vector input space of the logic function. This is only possible for small ECO changes which use small logic functions. In [10], this is not a problem in ASICs since ECOs requiring large changes are not desired since they are difficult to implement; however in FPGAs, where we can reprogram individual logic cells, large changes can be implemented while maintaining circuit performance, thus a more scalable technique is required. Fortunately, our approach is much more scalable and we can handle large changes (i.e., ECO logic functions with more than six input variables) by using a SAT-based approach and avoid the need to explicitly derive the resynthesis functions which we will describe in the following sections.

C. Boolean Satisfiability (SAT)

Given a Boolean expression $F(x_1, x_2, ..., x_n)$, Boolean satisfiability (SAT) seeks an assignment to the variables, x_1, x_2, ..., x_n, such that F evaluates to true. If this is possible, F is said to be *satisfiable*, otherwise, it is *unsatisfiable*. SAT solvers are tools that serve to find if a Boolean formula is satisfiable or not. For practical reasons, modern day SAT solvers work on Boolean expressions in conjunctive normal form (CNF). An example of a Boolean expression is shown in equation F_{AND} in Fig. 2.

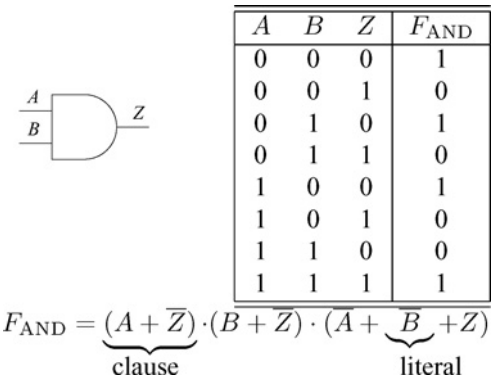
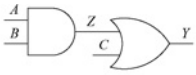


Fig. 2. Characteristic function derivation for 2-input AND gate.



$$F = (A+\overline{Z})\cdot(B+\overline{Z})\cdot(\overline{A}+\overline{B}+Z)\cdot (\overline{Z}+Y)\cdot(\overline{C}+Y)\cdot(Z+C+\overline{Y}) \quad (1)$$

Fig. 3. Cascaded gate characteristic function: top clauses from AND gate, bottom clauses from OR gate.

Recent advancement in SAT solvers [15] has improved their runtime performance by an order of magnitude, thus several problems in electronic design automation can be practically solved using SAT. In order to solve circuit problems with SAT, often the circuit needs to be converted into a Boolean expression which is then input into the SAT solver. This Boolean expression [16] is known as a *characteristic function* for the circuit. The characteristic function of a circuit evaluates to true if all variable assignments of the wires, inputs, and outputs of the circuit have a feasible assignment. For example, consider the AND gate shown in Fig. 2. The table to the left gives the truth table for the AND gate characteristic function which can be converted to CNF using any standard minimization technique. Fig. 3 shows how to derive the characteristic function of larger circuits by conjoining the characteristic functions of individual gates within the circuit.

After deriving the characteristic function of a circuit, we can use it in conjunction with SAT solvers to examine logical properties of the circuit for various CAD problems. For example, in Section III-A, we will show a SAT-based technique using the characteristic function to isolate errors within a design.

In this paper, we will often refer to a circuit as being *satisfiable* without explicitly showing its characteristic function. We define a circuit as being satisfiable if there is a satisfying assignment to the characteristic function of the circuit. The circuit is unsatisfiable if no such assignment exists. Furthermore, a *satisfying assignment* to a circuit is an assignment which is consistent with the circuit's characteristic function. For example, one possible satisfying assignment to the circuit in Fig. 3 is $ABCZY = 01101$.

III. Automated ECOs: General Technique

The goal of our work is to incrementally modify an existing placed-and-routed circuit in a minimally disruptive manner such that its new behavior matches a given reference specification. The reference specification is often defined by a reference model netlist or "golden" netlist, where we will use the term *model netlist*. In order to automate the modification process, we follow the flow shown in Fig. 4. This flow consists of two distinct steps: netlist localization followed by netlist modification.

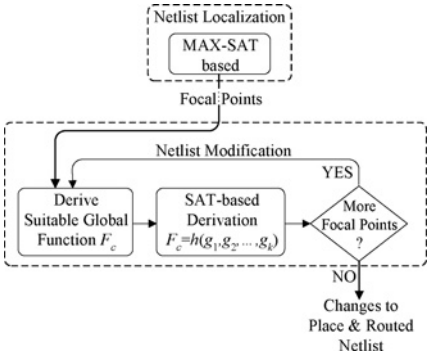


Fig. 4. Automated ECO flow.

Netlist localization finds a set of locations which we will refer to as *focal points*. These focal points help isolate regions that need to be modified. During the *netlist modification* phase, the proposed algorithm will leverage existing logic found in the design to help rewire the nodes at each focal point such that the resulting circuit will match the behavior of the model netlist. This starts by first deriving the proper global function F_c to replace the logic at each focal point. After a new global function is found, a search of the netlist is conducted to find a proper set of subfunctions, $\{g_1, g_2, ..., g_k\}$, to implement the new global function F_c such that $F_c = h(g_1, g_2, ..., g_k)$. Since these subfunctions, which we will refer to as *basis functions*, already exist in the placed-and-routed netlist, the only required changes to the netlist revolves around the implementation of the *dependency function* h. As we will show in Section III-B, implementing h requires much fewer changes than implementing F_c from scratch. This makes the proposed algorithm well suited for ECOs which must preserve the existing placed-and-routed circuit as much as possible.

A. Netlist Localization

Netlist localization is the process of identifying which nodes must be changed in order to change the behavior of the circuit such that it matches the behavior of the model netlist. The approach we use to find these nodes is based upon the work presented in [17]. Here the authors introduce a localization technique based upon MAX-SAT. Like SAT, MAX-SAT seeks a satisfiable variable assignment to a CNF. However, in the case of unsatisfiability, MAX-SAT seeks an assignment which maximizes the number of satisfying clauses. Using the set of unsatisfiable clauses, we can isolate discrepancies between two netlists.

This process is illustrated in Fig. 5. Consider the two circuits in Fig. 5. The top circuit is the model netlist while the bottom circuit is the implementation of the original circuit. Under the input stimulus $\{a = 0, b = 1, d = 1\}$ the original circuit has a response of $\{e = 0\}$ which conflicts with the model response

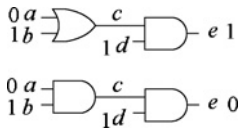


Fig. 5. Maximum satisfiability solution to localization.

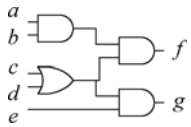


Fig. 6. Example circuit which requires multiple trace vectors to isolate discrepancies using MAX-SAT.

of $\{e = 1\}$. The corresponding CNF for the original circuit and the input/output vectors are shown below

$$\begin{aligned}(\overline{a}) \cdot (b) \cdot (d) \cdot (e) \cdot \\ (a + \overline{c}) \cdot (b + \overline{c}) \cdot (\overline{a} + \overline{b} + c) \cdot \\ (c + \overline{e}) \cdot (d + \overline{e}) \cdot (\overline{c} + \overline{d} + e).\end{aligned} \tag{1}$$

The above equation is unsatisfiable; however, the MAX-SAT problem will attempt to maximize the number of satisfied clauses. One possible solution to this is satisfying all clauses except for clause $(a + \overline{c})$. Since clause $(a + \overline{c})$ is derived from the AND-gate, this indicates that the function output from this gate (i.e., function on wire c) is the source of the discrepancy. Clause $(\overline{a})$ could have also been identified in the MAX-SAT solution which does not help identify the discrepancy in our circuit. To avoid fruitless clauses from being identified, you can adjust the MAX-SAT solver to force such clauses to satisfiable during the MAX-SAT process.

This type of localization is known as trace-based localization where a vector set is used to help identify discrepancies between two netlists. The main limitation with this approach is that if only a small set of vectors or traces are available, many possible locations will be identified; of which, several would not be the true source of the problem. For example, consider Fig. 6. If the input assignment is $a = 1$, $b = 1$, $c = 0$, $d = 0$, and $e = 0$ while the output assignment for the model netlist is $f = 1$ and $g = 0$, this leads to an unsatisfiable solution. When solving MAX-SAT on this construction, this would lead to two possible discrepancies: either the output AND gate feeding pin f, or the OR gate.

We can get around the issue described previously by adding additional vectors. For example, referring back to Fig. 6, adding an additional vector constraint $a = 1$, $b = 1$, $c = 1$, $d = 0$, and $e = 1$, with $f = 0$ and $g = 1$, would remove the OR gate as a possible suspect. Assuming that a large enough set of vectors are available, an iterative approach can be applied to the localization process where the number of potential locations are reduced. This technique is described in more further detail in [18].

In our case, we will often refer to discrepancy points isolated by the MAX-SAT localization process as *focal points*. Also, in many examples, we will assume there only exists one focal point, which we will refer to as *single-location modifications*.

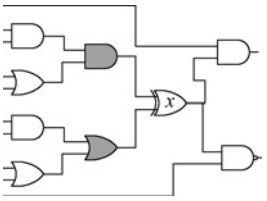


Fig. 7. Illustration of dominator node, x, to reduce the number of focal points to explore.

Even with the heuristic described previously, there exists the potential for returning equivalent focal points. These focal points are considered equivalent if modifying the global function that exists at any of the focal points can potentially remove any discrepancy between the existing circuit and the model netlist. We reduce the number of equivalent focal points returned by using multiple outputs when formulating the MAX-SAT characteristic function. Furthermore, the MAX-SAT solution will minimize the number of unsatisfiable clauses, which in turn reduces the number of focal points in each solution is small. This is illustrated in Fig. 7. Assuming that the two grey nodes are the original sources of the discrepancy between the existing circuit and a model netlist, the node labeled with x will be the node identified by the MAX-SAT problem since this would minimize the number of unsatisfiable clauses.

This property of MAX-SAT proves to be good in the context of ECOs, since this insures a minimally disruptive change in succeeding resynthesis steps after localization.

In the example shown in Fig. 5, the original circuit and model netlist exist as a gate-level netlist. This is not necessary, since we are only interested in the characteristic function of the current circuit and IO response of the model netlist; not their implementation. This is important in our case where our circuit exists as a netlist of LUTs on Altera's Stratix architecture and our model netlist consists of a netlist of AND and inverter gates (AIG) synthesized from a register transfer level (RTL) netlist.

B. Netlist Modification

After netlist localization occurs, the netlist can be modified. Modifications are centered around the *focal points* that were identified during the localization phase. Referring back to Fig. 4, the process of netlist modification requires a series of steps. First, we must define a global function F_c to replace the existing global function at each focal point. Following this, we must decompose the replacement function F_c such that it leverages existing logic within the placed-and-routed netlist. We will show later on that we use a specialized circuit construct to define an implicit representation of the on and offset of F_c. This significantly enhances the scalability of our approach. Once we have found a suitable representation for F_c, we will then decompose F_c into a set of basis functions. Since our basis functions already exist within the placed-and-routed netlist, we have significant control in how we decompose our function, which is a requirement if we want to minimally disrupt the existing plate-and-routed circuit.

1) *Suitable Corrective Function F_c:* To understand why we need to find a suitable global function to replace the function

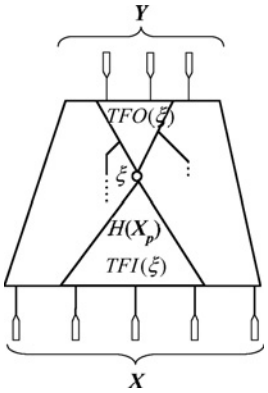


Fig. 8. Illustration of focal point ξ. Modification can be applied by replacing $H(\boldsymbol{X}_p)$ with some arbitrary function $F_c(\boldsymbol{X}_s)$.

at each focal point, consider the circuit illustrated in Fig. 8. Assume node ξ is identified during the localization step. The transitive fanin and fanout for node ξ are labeled as *TFI* and *TFO*. The inputs to the circuit are labeled as the variable set $\boldsymbol{X}$, and the outputs of the circuit are labeled as the variable set $\boldsymbol{Y}$. Also, note that we illustrate some of the input branches to the transitive fanout with dotted lines which will be important later on in Fig. 9. Assuming we can change the global function $H(\boldsymbol{X}_p)$ to any arbitrary function $F_c(\boldsymbol{X}_s)$ where $\boldsymbol{X}_p \subseteq \boldsymbol{X}$ and $\boldsymbol{X}_s \subseteq \boldsymbol{X}$, we can alter the circuit behavior to match our model netlist. Thus, the ECO problem becomes one of finding a suitable global function $F_c(\boldsymbol{X}_s)$ to replace the existing $H(\boldsymbol{X}_p)$. This leads to the following definition and lemma.

Definition 1: Suitable global function. In the case of a single focal point, ξ, identified during the localization step, a suitable global function F_c is defined as a function which can be used to replace the existing global function H for node ξ such that the resulting circuit matches the input-output behavior of a model netlist.

Lemma 1: For a single-location modification, such as the one shown in Fig. 8, the behavior of the original circuit can be altered to match a model netlist if the global function located at focal point ξ is replaced with another suitable global function $F_c(\boldsymbol{X}_s)$.[1]

Our work follows lemma 1 where given a set of focal points returned by the localization step, we attempt to replace the global function at each location with another global function. This requires us to first find a suitable global function, which we label as F_c. We do so by deriving an implicit representation of F_c using a specialized circuit construct. Following this, we search for a set of basis functions $\{g_1, g_2, ..., g_k\}$ from the existing netlist and see if they can be used to implement the new function F_c. This requires a SAT-based functional dependency check described later on in this section. If the basis functions pass the SAT test, we can generate the decomposition $F_c = h(g_1, g_2, ..., g_k)$ where h is known as the dependency function between F_c and the basis functions $\{g_1, g_2, ..., g_k\}$. To understand this in detail, we will first consider the simple case of a combinational circuit containing a single focal point returned from the localization step.

[1]An if-and-only-if condition is not possible, since we can also alter the circuit using corrective function found in the transitive fanout of the focal point ξ; particularly if there are dominator nodes found in the transitive fanout. However, in this paper, we will only use the focal point as the location for resynthesis.

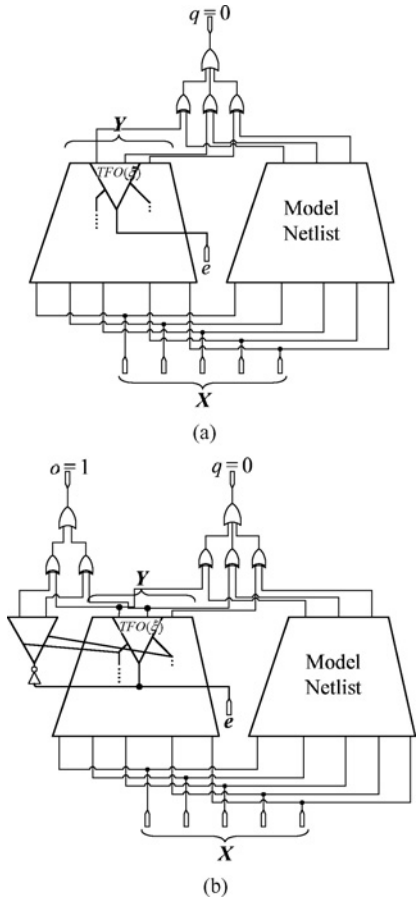


Fig. 9. Illustration of the construction of the SAT problem. (a) First the original circuit is constrained to a *model netlist*. (b) Observability don't care constraints are added to the circuit to improve the flexibility of finding a minimally disruptive corrective function F_c.

2) *Finding an Implicit Representation of F_c:* Deriving a suitable global function, F_c, can be done through brute force means where we explicitly create F_c. However, an explicit representation of F_c, such as a truth table or BDD, will not scale since F_c may depend on several variables (i.e., we must enumerate all 2^n vectors to derive the corrective function, where n is the number of primary input variables of the global function F_c). To avoid this problem we instead choose to construct an implicit representation of F_c. We start off with a construction of the circuit shown in Fig. 9. On the left side of Fig. 9(a), we show a circuit we intend to modify, which looks similar to Fig. 8. However, in Fig. 9(a), we replace the focal point ξ with an input pin e. Doing so gives us the freedom to search for a suitable F_c to replace pin e. Next, we have to constrain the input-output behavior of the circuit to match the model netlist. We do this by attaching the model netlist to our circuit as shown on the left side of Fig. 9(a). Here, we tie the circuit inputs together and connect the outputs with an XOR gate along with an OR gate at the top. Following this, we force the OR gate output to 0 ($q = 0$). As a result of these constraints, any satisfying assignment to the circuit in Fig. 9(a) will be consistent with the input-output behavior of the model netlist.

After constraining the circuit to the model netlist behavior, we seek to remove the "don't care" space of the suitable function F_c. This gives us greater flexibility in choosing a proper F_c. We do this by restricting the input space, $\boldsymbol{X}$, to values where pin e is observable at the outputs. Informally, e, is observable at a PO, y, if for a given assignment, the value of e impacts the value of y. Otherwise, e is not observable at node y. This is appropriate since pin e will ultimately be replaced with F_c. This is shown in Fig. 9(b) which we will refer to as *CircuitConstrain*. In *CircuitConstrain*, we duplicate the transitive fanout of pin e. When connecting the duplicated transitive fanout to the original circuit, we complement input pin e, while all other input branches to the transitive fanout are kept the same. Also, for each output pair of the duplicated transitive fanout and original circuit, we add an `XOR` gate followed by an `OR` gate and set the output of the `OR` gate to 1 ($o = 1$). The circuitry added at the outputs along with the duplicated transitive fanout checks to see if the input pin e is observable at the outputs. For example, if the output of the `OR` gate is 0 ($o = 0$), this implies that the value on pin e does not matter and will not affect the outputs. Based on the previous description of *CircuitConstrain*, we find the following lemma and proposition to be true.

Lemma 2: CircuitConstrain is satisfiable if and only if the output of the current circuit matches the output of the model netlist and if pin e is observable at the outputs.

Proposition 1: CircuitConstrain is satisfiable if and only if the assignment to e and the assignment to the input vector $\boldsymbol{X}$ is consistent with a suitable F_c. In other words, for all satisfying assignments to *CircuitConstrain*, $F_c(\boldsymbol{X}) = e$.

Proof: Assume that if we have a satisfying assignment to *CircuitConstrain* then the values on e and $\boldsymbol{X}$ do not match any suitable F_c. In other words, assume that for a given satisfiable assignment to e and $\boldsymbol{X}$ then $e \neq F_c(\boldsymbol{X})$. Thus, this implies that the output of the current circuit does not match the model netlist by lemma 1. However, this is not possible since any satisfying assignment to *CircuitConstrain* must match the input-output behavior of the model netlist by lemma 2. Thus, by contradiction any satisfying assignment to e and $\boldsymbol{X}$ implies a match to a suitable F_c. Now let us assume that if the values on e and $\boldsymbol{X}$ match a suitable F_c then we do not have a satisfying assignment to *CircuitConstrain*. In this case, if $e = F_c(\boldsymbol{X})$, then the output response of our circuit will match the model netlist. Also, e must be observable since we can restrict the inputs space of $\boldsymbol{X}$ to the observable space of F_c. However, these two conditions must imply that we must have a satisfying assignment to *CircuitConstrain* by lemma 2. Thus, by contradiction, if the values on e and $\boldsymbol{X}$ match a suitable F_c then we have a satisfying assignment to *Circuit-Constrain*. Hence we have proved that *CircuitConstrain* is satisfiable if and only if the assignment to e and the assignment to the input vector $\boldsymbol{X}$ is consistent with a suitable F_c. ■

3) *Decomposition of F_c Using SAT:* By proposition 1, we can use the circuit in Fig. 9(b) to find a suitable F_c by exploring all satisfying assignments to e and $\boldsymbol{X}$. Doing so will effectively create the entire truth table for function F_c. For example, we would start by setting $\boldsymbol{X}$ to 000...00. We would then find a satisfying value for e. This value found for e would represent F_c(000...00). Next, we would continue for $\boldsymbol{X}$ = 000...01 and so on.

$x_1x_2x_3$	g_1 g_2 g_3	F_c
000	0 0 0	1
001	0 0 1	0
010	0 0 0	1
011	0 1 1	0
100	0 0 1	0
101	0 0 1	0
110	1 0 1	1
111	1 1 1	1

Fig. 10. Functional dependence example.

The problem with the previously described approach is that this requires an exhaustive traversal of the input space $\boldsymbol{X}$, which does not scale. Furthermore, implementing F_c directly is not desirable since this may require significant changes to the existing placed-and-routed netlist. A more practical approach for ECOs would be to search for a set of basis functions, $\{g_1, g_2, ..., g_k\}$, found from the existing netlist and check if this set of basis functions can be used to implement F_c such that $F_c = h(g_1, g_2, ..., g_k)$. Informally, this will be possible if and only if there does not exist a situation where the basis functions $\{g_1, g_2, ..., g_k\}$ are equivalent for a given input pair assignment and F_c is not equivalent for the same input pair. For example, consider Fig. 10. Here we show four functions, g_1, g_2, g_3, and F_c. In this example, all functions are dependent on three variables x_1, x_2, and x_3. Looking at function g_1 and g_2, we see that there exists a given input pair $\{000, 001\}$ where g_1 and g_2 are equivalent, but F_c is not equivalent. Thus, it is impossible to find a function h such that $F_c = h(g_1, g_2)$. In contrast, if we select functions g_1 and g_3, it is impossible to find an input pair where g_1 and g_3 are equivalent and F_c is not equivalent. Thus, in this case, there exists a dependency function h such that $F_c = h(g_1, g_3)$ (in this case $h = g_1 \cdot g_3 + \overline{g_1} \cdot \overline{g_3}$). The key benefit of this process is that by leveraging existing logic within the netlist (i.e., the basis functions g_i) we only need to derive the dependency function h, which has a smaller support set than the global function F_c. This is ideal for ECOs since implementing the dependency function h within the existing circuit has a much smaller and predictable impact than implementing the entire global function F_c from scratch.

To formally check the condition described previously we will adapt a SAT-based approach [19]. This requires constructing a characteristic function that checks for the following condition: for a given set of basis functions $\{g_1, g_2, ..., g_k\}$, does there exist a minterm pair where the basis functions are equivalent and F_c is not equivalent. The primary difference between [19] and our approach is that in [19], their F_c function exists explicitly within the circuit. This is because in [19], their work is applied to logic synthesis and circuit rewiring where subcircuits are tested with SAT to see if they can be replaced with alternate subcircuits; thereby reducing the overall circuit area. In contrast, in this paper we must derive a proper F_c using the circuit construction shown in Fig. 9.

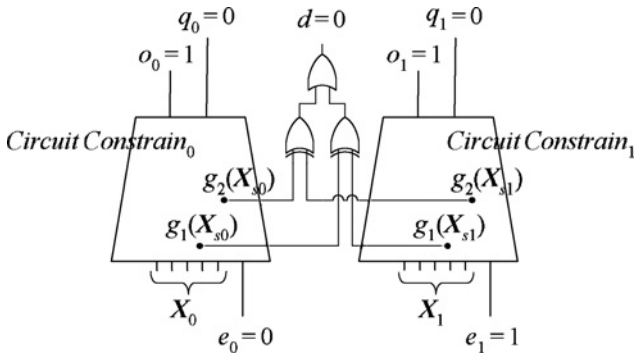


Fig. 11. Deriving global function $F = h(g_1, g_2)$ using dependence check construct.

To create the characteristic function that checks the previously described condition, we construct the circuit shown in Fig. 11. In Fig. 11, we have duplicated *Circuit-Constrain* and labeled them as $CircuitConstrain_0$ and $CircuitConstrain_1$. In each duplicated copy, we have simplified *CircuitConstrain* and have only shown their duplicated input and output pins.

To check if there exists a minterm pair where a set of basis functions are equivalent, we can extract a basis function set from each duplicated circuit and connect them together with an XOR-OR gate network and set the OR gate output to 0 ($d = 0$) as shown in Fig. 11. Here, we have selected two wires g_1 and g_2 as the basis functions. The support set of the basis functions g_1 and g_2 in $CircuitConstrain_0$ ($CircuitConstrain_1$) are labeled as $\boldsymbol{X}_{s0}$ ($\boldsymbol{X}_{s1}$) which is a subset of the variable set $\boldsymbol{X}_0$ ($\boldsymbol{X}_1$). Due to these constraints, a satisfying assignment to Fig. 11 will find an minterm pair assignment to variables $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$ where the basis functions are equivalent.

To check if there exists a minterm pair where F_c is not equivalent, we can also use the circuit construct in Fig. 11. Here, we will constrain each duplicated pin e. Since in *CircuitConstrain*, only satisfiable assignments to e and $\boldsymbol{X}$ are consistent with a suitable global function $F_c(\boldsymbol{X})$ by proposition 1, constraining e to 0 or 1 provides a means to constrain the input $\boldsymbol{X}$ to the offset and onset for $F_c(\boldsymbol{X})$. This is stated more formally in the following lemma.

Lemma 3: In *CircuitConstrain* of Fig. 9(b), if pin e is assigned a value 0, *CircuitConstrain* is satisfiable if and only if the input vector $\boldsymbol{X}$ is assigned a value found in the offset of the function $F_c(\boldsymbol{X})$. Conversely, if pin e is assigned a value 1, the circuit is satisfiable if and only if the input vector $\boldsymbol{X}$ is assigned a value found in the onset of the function $F_c(\boldsymbol{X})$.

Following from lemma 3, the circuit in Fig. 11 constrains e_0 to 0 and e_1 to 1 to restrict the possible assignments to $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$. Specifically, setting e_0 to 0 constrains all possible assignments of $\boldsymbol{X}_0$ to the offset of F_c, while setting e_1 to 1 constrains all possible assignments of $\boldsymbol{X}_1$ to the onset of F_c. Thus, due to these constraints, a satisfying assignment to Fig. 11 will find a minterm pair assignment to variables $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$ that respectively map to the offset and onset of function F_c. Thus, for this minterm pair, F_c is not equivalent.

Using the previous circuit constraints in Fig. 11, we can create the characteristic function shown in

$$\begin{aligned}\exists \boldsymbol{X}_0 \exists \boldsymbol{X}_1 \; & (CircuitConstrain_{0\ CNF}) \cdot \\ & (CircuitConstrain_{1\ CNF}) \cdot \\ & (e_0 \equiv 0) \cdot (e_1 \equiv 1) \cdot (g_1(\boldsymbol{X}_{s0}) \equiv g_1(\boldsymbol{X}_{s1})) \cdot \\ & (g_2(\boldsymbol{X}_{s0}) \equiv g_2(\boldsymbol{X}_{s1})) \cdot (\boldsymbol{X}_0 \neq \boldsymbol{X}_1).\end{aligned} \tag{2}$$

In (2), assume that $CircuitConstrain_{0\ CNF}$ ($CircuitConstrain_{1\ CNF}$) represents the clauses derived from the characteristic function of $CircuitConstrain_0$ ($CircuitConstrain_1$). Equation (2) asks if there exists a minterm assignment pair to $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$ which map to the off and onset of F_c respectively (i.e., $e_0 \equiv 0$ and $e_1 \equiv 1$), while the basis functions are equivalent. This leads to the following proposition.

Proposition 2: If (2) is satisfiable the basis functions, g_i, cannot be used to implement a suitable global function F_c. If (2) is unsatisfiable, the basis functions, g_i, can implement a suitable global function F_c.

Proof: Assume if (2) is satisfiable then F_c has a decomposition $h(g_1, g_2, ..., g_k)$. If (2) is satisfiable, this implies that there is a vector assignment C_0 to $\boldsymbol{X}_0$ which maps to the offset of F_c and there exists a vector assignment C_1 to $\boldsymbol{X}_1$ which maps to the onset of F_c, which follows from lemma 3. This also implies that vector C_0 and vector C_1 map to the same output vector, $K = k_1k_2...k_k$ in the output space of the basis set $\{g_1, g_2, ..., g_k\}$ due to the g_i equality constraints in (2). However, since we have assumed $F_c = h(g_1, g_2, ..., g_k)$ in the first statement, this implies $F_c(C_0) = h(K) = 0$ and $F_c(C_1) = h(K) = 1$, thus we have a contradiction. Thus if (2) is satisfiable then F_c cannot be decomposed into $h(g_1, g_2, ..., g_k)$. To prove the second statement in proposition 2, we can use a similar argument and will not show it in detail here. ■

If (2) is unsatisfiable, we can derive the correct dependency function h from the proof of unsatisfiability. This requires backtracing through the set of clauses are the source of the unsatisfiable condition and creating a function based on these clauses. In our case, if the function h is small enough (its support set size is less than 20), we create a BDD representation of h. We limit the variable size to 20 since this minimizes the risk of BDD explosion. In the cases where the BDD does get very large, we fail to create a proper implementation of h. Deriving h based on the proof of unsatisfiability is based on the theorem of Craig interpolation and will not be discussed here. For a detailed description on deriving h, please refer to [19] and [20].

If the support size of h is less than the number of inputs to a LUT, we can implement h in a single LUT and the modification process is complete. However, if the support size of h is larger than the number of inputs to a LUT, we must decompose and technology map h to the circuit architecture. In our case, we decompose h into a network of LUTs for the Stratix architecture. Our decomposition method is similar to what is presented in [21]. Here, we recursively extract out a function which contains at most k inputs, where k is the LUT size in question, from function h. We continue until function h

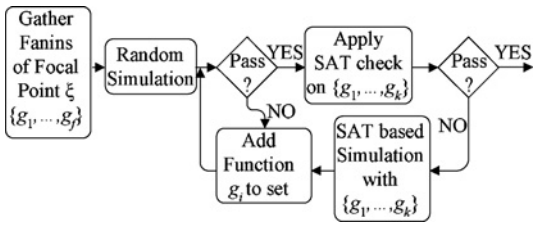


Fig. 12. Searching for basis functions flow.

$x_1x_2x_3$	e	g_1 g_2 g_3
100	0	0 0 1
001	0	0 0 1
010	1	0 0 0
011	0	0 1 1
110	1	1 0 1
101	0	0 0 1

Fig. 13. Random simulation example.

contains at most k inputs. Alternatively, one can modify their decomposition algorithm to map h into any technology. During the decomposition, since timing information is available, we skew the decomposition such that critical wires are placed near the top of the decomposition tree. This is similar to the work presented in [22] and reduces the number of logic levels the critical wires have to go through in order to preserve timing.

4) *Searching for Basis Functions:* In the previous description, we assumed that the set of basis functions necessary in our functional dependency check were available. In practice, we must search the circuit netlist for an appropriate set of basis functions. This flow is illustrated in Fig. 12. First, the fanins to the focal point, ξ, are chosen as the basis functions. Next, we use a random simulation based check to see if this set can potentially be used as a dependency set to implement the corrective function F_c. If the check fails, we search for additional functions which exist within the circuit to add to the set of basis functions. Additional functions are added until our random simulation based check passes. When searching for additional basis functions, we first select functions within the transitive fanin of the focal point ξ. If this fails, we collect the primary outputs found within the transitive fanout of the focal point ξ and select functions found within the transitive fanins of those primary outputs. It should be noted that during this search, we must avoid using basis functions found within the transitive fanout of the focal point to avoid creating combinational cycles within the circuit. Once a set of basis functions pass the random simulation check, we apply our SAT-based decomposition technique. This process continues until we find a suitable set of basis functions to implement F_c such that $F_c = h(g_1, g_2, ..., g_k)$. Although not shown in Fig. 12, if the number of basis functions required grows to more than 20 functions or if the number random simulation calls is more than 10, the search will terminate.

The random simulation check shown in Fig. 12 is used as a quick check to see if the current basis function set can be used to implement F_c without going through the more expensive SAT-based check described in the previous section. Here, using *CircuitConstrain* in Fig. 9(b), we randomly simulate the PIs $\boldsymbol{X}$ and pin e. Vectors which violate the observability ($o = 1$) and equality ($q = 0$) constraint are discarded. Following this, we search through the circuit netlist and find a set of nodes whose vectors do not satisfy (2). For example, consider Fig. 13. Assume that the circuit PIs are variables x_1, x_2, and x_3 and functions g_i are chosen from the existing circuit. Here, we show the value of the functions, g_i, with respect to a random input vector assignment to the PIs and pin e. Using this sample set, we can see that the set $\{g_1, g_2\}$ cannot be used as our basis set since there exists a primary input pair, $\{001, 010\}$, which satisfies (2). In contrast, basis set $\{g_1, g_3\}$ is a possible set since there does not exist any input pair which satisfies (2). Since our random simulation set is not exhaustive, we must apply the SAT-based check described in the previous section to cover any input vectors missed during the random simulation.

If the current basis set fails the random simulation test, we must add additional functions to build a proper set of basis functions. To help guide the search, we use a SAT based simulation to create input patterns that can find additional vector pairs that need to be *differentiated*. The SAT based simulation works by duplicating the circuit and "locking" each pair of duplicated basis functions to equivalent values (i.e., "locking" means adding equality clauses to ensure the duplicated basis function outputs are equivalent) and setting the pair of duplicated input pins e to complemented values. Following this, a SAT solver is run 32 times to find a set of 32 different satisfying values on the other wire which generates a new vector set. Using these new vector sets, we search for functions that can *differentiate* minterms currently differentiated by pin e, but not differentiated by the current basis function set. Informally, a function set can *differentiate* a pair of minterms if their outputs are different for that minterm pair. For example, in Fig. 13, $\{g_1, g_2\}$ cannot differentiate minterms 001 and 010 since $g_1(001) \equiv g_1(010)$ and $g_2(001) \equiv g_2(010)$. Since pin e differentiates this pair, to create a valid basis set from $\{g_1, g_2\}$ for pin e, we would need to pick an additional function g_j that can differentiate vectors 001 and 010 (differentiating pairs $\{100, 010\}$ and/or $\{101, 010\}$ are also possible). Wires whose functions cover the most vectors not differentiated by the current basis set are chosen first. This greedy algorithm helps to minimize the size of the basis function set. The second criteria when choosing basis functions is their timing criticality: basis functions with non-critical wires are chosen over basis functions with critical wires. Alternate selection criteria could also be used such as structural information, however, experimentally we found that our current technique works well in the context of the benchmark set used in the results section.

5) *Partitioning of Circuit and Treatment of Registers:* In order to help reduce the size of (2), we can remove portions of the circuit which do not impact the behavior of the current node being modified. This is illustrated in Fig. 14. Here, we are assuming that node ξ was localized in our netlist localization step. Using node ξ, we collect the set of outputs within the transitive fanout of ξ, as shown in Fig. 14(a). Following this, we collect the transitive fanin of each output, as shown in Fig. 14(b). Since the POs of the implemented netlist may require that their transitive fanin PIs be changed during the modification, the transitive fanin collected in Fig. 14(b) is

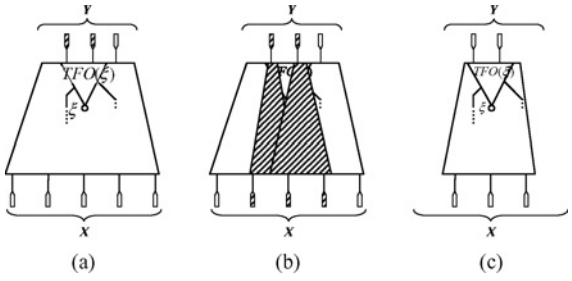


Fig. 14. Circuit reduction. (a) Outputs found in transitive fanout cone of focal point. (b) Transitive fanin cones of outputs in (a). (c) Reduced circuit using cones in (b).

defined by the model netlist. Thus, this requires that our model netlist be a white-box implementation, such as an gate-level netlist derived from an RTL description. Any node which is not part of a transitive fanin of a given output can be removed, as shown in Fig. 14(c). In conditions that the transitive fanout is excessively large and contains several hundred POs, we limit the number of POs kept to under 50. Thus, we are able to remove a significant portion of the circuit to speed up the SAT-based functional dependency check. This can potentially create an incomplete solution since this heuristics removes some information that may be needed to derive a corrective function F_c. However, empirically we rarely found this to be a problem where for our circuit set, 50 outputs were sufficient to find a valid correction.

In the case of sequential circuits, we treat registers as primary inputs and outputs where the input of a register becomes a primary output and the output of a register becomes a primary input. This reduces the complexity of our problem; however, limits the application of our algorithm to circuits which have not been retimed.

6) *Multiple Focal Points:* In the previous sections, we described the situation when only a single focal point is returned by the localization step. In the case of multiple focal points, where we need to find multiple suitable global functions, we apply a step-by-step approach where we modify each focal point one at a time. While modifying a given focal point, ξ, additional focal points are fixed such that they are set to either 0 or 1 using SAT equivalency constraints [i.e., add the clause (z) or $(\overline{z})$ to set z to 1 or 0, respectively, where z is the variable representing the output of the focal point in the characteristic function]. For example, one overly simplistic heuristic would be to set all focal points other than ξ to value 0 (in general, this would not work since this could make the focal point ξ unobservable at the outputs, such as in a case where ξ and another focal point enter into a single AND gate). This ensures that changes seen at the output are due to the focal point ξ, and not due to the other focal points found within the netlist. The fixing of focal points has the possibility of not converging, particularly for complex changes requiring several modifications. However, in practice the step-by-step heuristic works in the majority of cases as shown empirically in Section IV where we have at most four focal points into the design.

IV. Results

The primary goal of ECOs is to apply a functional change to the circuit while minimally disrupting the existing placement and routing while preserving timing. To ensure that the resynthesis technique described in the previous sections can be applied to late-stage ECOs, we attempt to achieve this goal and must ensure that our technique can automatically update the behavior of a circuit in a minimally disruptive manner. Thus, we evaluate our work based on three criteria: how disruptive our technique is on a placed-and-routed netlist, how much impact our technique has on performance, and how robust our technique is in handling changes. Two experiments will be run to quantify these three criteria. The first experiment models a late-stage specification change at the RTL, while the second experiment models a late-stage bug fix. We run our algorithm on circuits from Altera's QUIP [6] and the ITC benchmark suite.

All experiments are run on a PentiumD 3.2 GHz processor with 2 MB of RAM. Circuits are synthesized using Quartus II 7.1 and for Altera's Stratix architecture. To communicate with Quartus II, we use Altera's QUIP TCL interface [6].

A. Specification Changes

In our first set of experiments, we seek to see how disruptive our approach is on an existing placed-and-routed circuit and measure its impact on performance. This is important for late-stage ECOs since we want to preserve as much as possible the engineering effort invested previously when applying our changes. To quantify the disruption of our approach, we apply a HDL specification change both using a full recompile of the design (i.e., compile the HDL code from scratch through the Quartus II CAD flow) and using the proposed ECO approach then compare this against the original placed-and-routed netlist. To get the initial placed-and-routed circuit, we compile the circuit described in Verilog or VHDL using Altera's Quartus II CAD flow. During the experiment, we first compile the circuit described in Verilog or VHDL using Altera's Quartus II CAD flow. In the initial compile, we use Altera's design space explorer to achieve the best possible timing performance. Next, we modify the original HDL code and use this modified code as our new specification (this would be the *model netlist* as described in Section III).

1) *Modification Descriptions:* Modifications are primarily constrained to control-path code or modifications to `case` and `if-else` statements where we change less than three lines of code. We restrict our changes to control logic since we find from literature, most ECOs are applied to control logic or enable signals, as opposed to datapath changes [3] (although in many real world design flows, datapath changes are possible). An example of such changes is shown in Fig. 15. After our code is modified, we perform a full recompile of the modified HDL code and record the performance impact. We also apply the required HDL changes directly to the optimized circuit using our ECO approach and record its performance impact and measure the disruption to the placed-and-routed netlist.

To quantify the disruption to the existing netlist due to a full recompile and from the ECO compile, we measure the percent change in timing and the percentage of LUTs and wires disrupted in the original placed-and-routed circuit. We define a LUT or wire as being disrupted if it satisfies one of three conditions: 1) it needs to be moved from the

TABLE I
IMPACT OF MODIFICATIONS ON CIRCUIT PERFORMANCE ON A STRATIX CHIP

	Before		*After*$_{FULL}$		*After*$_{ECO}$					
Circuit	*LUTs*	$CLK_{(ns)}$	*LUTs*	$CLK_{\%\Delta}$	*LUTs*	$CLK_{\%\Delta}$	Δ_P	Δ_R	n_{focal}	n_{basis}
aes_core	5359	7.96	5359	−0.25%	5359	2.26%	0.08%	0.16%	1	6
huffman_enc	633	5.12	634	−0.39%	640	4.10%	7.36%	9.32%	3	5
usb_phy	177	3.2	180	−0.62%	179	0.31%	9.25%	10.35%	1	4
fip_cordic_rca	467	21.6	467	6.39%	470	4.86%	3.54%	5.32%	2	8
mux64_16bit	1277	5.86	1277	4.10%	1277	1.37%	2.23%	3.52%	4	15
ahb_arbiter	862	9.45	862	9.95%	864	1.16%	2.34%	3.54%	3	5
barrel64	1043	8.59	1042	−0.23%	1043	0.81%	1.30%	2.50%	1	6
Average				2.70%		2.13%	3.73%	4.95%		

TABLE II
AUTOMATED CORRECTION RESULTS ON A STRATIX CHIP

	Before		*After*$_{ECO}$					
Circuit	*LUTs*	$CLK_{(ns)}$	*LUTs*	$CLK_{\%\Delta}$	Δ_P	Δ_R	n_{focal}	n_{basis}
aes_core	5359	7.95	5359	0.00%	1.12%	1.28%	1	4
huffman_enc	633	5.08	633	0.20%	3.16%	3.61%	2	6
usb_phy	177	3.13	177	0.00%	13.56%	10.33%	2	8
fip_cordic_rca	467	21.6	471	2.31%	12.85%	14.68%	2	6
mux64_16bit	1277	5.86	1277	0.51%	4.70%	5.37%	1	7
ahb_arbiter	865	8.97	972	16.83%	6.94%	7.93%	2	15
barrel64	1041	8.58	1041	−4.66%	5.76%	6.59%	2	9
b03	59	3.37	59	0.89%	16.95%	19.37%	3	9
b04	184	6.56	184	−9.60%	32.61%	37.27%	3	8
b08	58	3.68	58	5.71%	8.62%	9.85%	2	9
b09	50	3.97	50	−3.02%	16.00%	11.43%	2	7
b10	78	3.79	79	−17.41%	10.26%	11.72%	2	8
b11	167	6.68	187	4.64%	9.58%	10.95%	1	7
b12	378	6.17	381	3.40%	4.23%	4.84%	3	8
b14	3014	8.17	3015	2.57%	1.99%	2.28%	2	10
b15	6012	9.17	6023	6.98%	1.00%	1.14%	3	7
Average				0.18%	9.33%	9.91%		

```
//Original                    //Modification
...                           ...
if(busreg[j-1] == 1)          if(busreg[j-1] == 1)
 begin                           grant = j-1;
   grant = j-1;               else
   lock = hlock[j-1];             grant = inter1;
 end                          if(busreg[j-1] == 0)
else                            lock = hlock[j-1];
 begin                        else
   grant = inter1;                lock = inter2;
   lock = inter2;             ...
 end                          ...
...
```

Fig. 15. Example changes in HDL code.

original placement location or routing track from the original HDL compile; 2) it is deleted from the original netlist; or 3) it is newly introduced into the netlist. These results are shown in Table I. The first column shows the circuit name, *Circuit*, followed by three major column groups. The first group, *Before*, reports the circuit size in terms of LUTs and timing numbers during the initial compile of the design (i.e., before applying any changes). Timing numbers report the minimum clock period in terms of nanoseconds. The next group of columns headed by *After*$_{FULL}$ shows the number of LUTs and percent change in timing of a full recompile of the modified HDL code with respect to the numbers shown in the *Before* column. The final group headed by *After*$_{ECO}$ shows the relative impact of the proposed approach on circuit performance. Again, we show the number of LUTs and percent change in timing with respect to *Before* after modifying the design using the proposed ECO approach. Two additional columns, Δ_P and Δ_R, are shown which respectively show the percent disruption to the placement and routing after our changes are applied. This is not shown in the group *After*$_{FULL}$ since in the full recompile of the circuit, all LUTs and wires will be modified. The last row shows the average ratio of the percentages shown. Finally, we also show the number of focal points returned, n_{focal}, and the number of basis functions required to apply the ECO, n_{basis}.

As our results show, the results of a "from scratch" recompile shown in columns *After*$_{FULL}$, are fairly acceptable on average where performance decreases by 2.70%. However, when examining the individual performance impact of each circuit, we see that the results varies by a large amount from −0.62% to 9.95%. This illustrates the unpredictable nature of a full recompile on circuit performance. Furthermore, during a full recompile, the entire netlist is disrupted, thus all of the placement and routing effort invested previously is lost.

In contrast, the proposed ECO approach has a much more predictable impact to performance where the performance change varies between 0.31% and 4.86%. Also, the average disruption of our approach is relatively low where we keep over 95% of the placement and routing unchanged. Based on what is reported in industry [2], we feel that this number is acceptable to most designers, which is a necessary condition for the practical use of our approach. In terms of runtime, the proposed ECO approach is approximately 2x faster than performing a full compile. However, this discrepancy in runtime has more to do with the internal algorithms of Quartus II than our work and thus, detailed numbers are not shown. Furthermore, runtime comparisons would not reflect the reduction in time from going from a manual process, which often takes several days, to the proposed automatic ECO approach, which takes a few hours to complete.

During our experiments, we found that the class of changes we applied did not create drastic changes to the netlist. Specifically, during the localization process, all of our HDL changes would result in less than five focal points being returned. In cases where five or more focal points were returned, we found that our approach had difficulty finding a non-disruptive modification to the placed-and-routed netlist.

We should note that we used Quartus II's TCL ECO interface to change and legalize the circuit. Unfortunately, the TCL ECO interface does not provide an incremental placement flow. Instead, Quartus II will apply a limited placement algorithm to new or modified nodes. Here, the ECO placer will apply a timing driven placement and move nodes to existing free locations within the FPGA; however, it will not displace pre-existing nodes within the netlist. As a result, this may hurt timing significantly since circuitous routes can be created by new nodes. To overcome this limitation, we manually unplace a small region surrounding each new node using the Quartus II TCL interface. This gives the placer much more flexibility in where to place nodes and reduces the impact of newly introduced nodes. However, if given access to a full incremental placement algorithm, we feel that our timing and disruptive impact would be less than what is shown in Table I.

B. Error Correction

In our second set of experiments, we apply the proposed ECO technique to fix errors inserted into a design. The errors we inserted were confined to LUT rewiring of inputs or change LUT masks where we alternated between the two types of changes. When rewiring the LUTs, we randomly reassign three inputs of the LUT and when changing the LUT mask, we randomly toggle 50% of the mask bits. To stress the approach, multiple errors are inserted into the design such that the errors interact with each other. Errors interact when they are inserted at nodes whose transitive fanouts overlap with each other. This is necessary to ensure that the complexity of the correction increases as more errors are added to the design. After errors are inserted into the design, Quartus II's placed-and-route is rerun on the circuit to achieve the best performance possible.

Table II shows our results for various circuits when five errors are inserted into the design. The first column *Circuit* lists the circuit name. This is followed by two column groups: *Before* and $After_{ECO}$. *Before* lists the number of LUTs and timing numbers before the application of the ECO fix. Timing numbers report the minimum clock period in terms of nanoseconds. $After_{ECO}$ lists the total number of LUTs after the ECO fix and the relative change in timing with respect to the timing reported in the *Before* column. Also, additional columns Δ_P and Δ_R are shown which respectively show the percent disruption of the placement and wires after the ECO fix is applied. The final row shows the average of the percentages. Finally, we also show the number of focal points returned, n_{focal}, and the number of basis functions required to apply the ECO, n_{basis}. Unlike the previous results shown in Table I, comparisons with a full recompile is not done in this case since a designer would be required to fix the error manually by hand, which in many of the cases could not be practically solved by hand in an efficient manner. The proposed technique, however, can find a fix automatically.

Table II shows that the proposed ECO technique has a marginal impact to timing where we reduce timing on average by 0.18%. This is potentially misleading since for some circuits, such as `b04` and `b10`, we improve timing by a large amount. This can occur in some rare cases since the decomposition of the dependency function h is timing driven and has the potential to shorten the critical path. After removing these outliers, the average impact to timing is 1.72%. Another outlier, `ahb_arbiter`, proved difficult to correct where we found that in this circuit, the proper alterations required more than 15 basis functions to correct it, which ultimately disrupted its timing significantly. Considering that the proposed technique applies ECO changes in an automated fashion within minutes, we feel that 1.72% is an acceptable tradeoff of timing for improved designer productivity where the designer would manually fix these problems in a span of several days. Also, we found that many of the fixes occurred on paths that were critical or near critical. Thus, without a timing driven decomposition of h and a timing driven search for basis functions, circuit performance would not be maintained.

In terms of average disruption to the existing netlist, we disturb approximately 10% of the placed-and-routed netlist during our correction. We should note that this disruption is skewed by the fact that for smaller circuits, a large relative disruption cannot be avoided. For example, in the extreme case where a circuit contains only five LUTs, fixing five errors in the circuit would disrupt 100% of the netlist. If circuits with less than 200 LUTs are removed from Table II, the average disruption to the placement and routing drops to approximately 5%.

However, as the number of errors increased, the circuit correction problem became significantly difficult. This was primarily due to the localization step. For complex fixes, the localization step often returns several focal points, which can be much more numerous than the number of errors injected into the design. This significantly complicates the correction process both in terms of deriving a suitable global function F_c and minimizing the disruption to the existing netlist. Fortunately, the likelihood that large errors are not detected early in the design flow is low and, as a result, applying large changes with late-stage ECOs is not a likely occurrence.

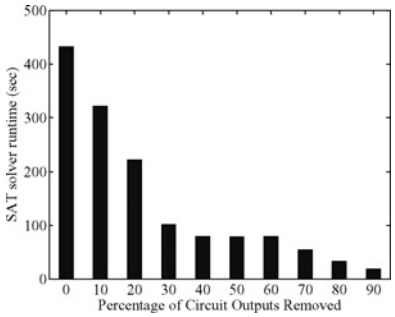


Fig. 16. Impact of output removal on SAT solver runtime for circuit b15.

We also examined the impact of the primary output removal heuristic described in Section III-B5. This is shown in Fig. 16 where we show the percentage of outputs removed vs. the runtime of the SAT solver when correcting circuit b15. During this experiment, we inserted bugs into nodes whose transitive fanouts contained more than 300 primary outputs and registers. Following this, we removed a percentage of outputs or registers and attempted to rectify the circuit. After rectification, we recorded the runtime of the SAT solver. As Fig. 16 shows, we are able to reduce the runtime of the SAT solver by more than 10x in some cases. Also, removing outputs usually did not prevent a valid correction from being found. Only until more than 95% of the outputs were removed from the circuit did the correction fail our verification check. This was due to a lack of information provided by the model netlist after pruning most of the outputs.

C. Runtime of MAX-SAT Problem

We also wanted to examine the scalability of the MAX-SAT flow described in Section III-A. This is illustrated in Table III where we have injected four errors within each design. It is interesting to note that the MAX-SAT solver has the most difficulty for the larger ITC circuits such as b15. This is due to the fact that the ITC circuits are flattened, unlike the circuits in the QUIP benchmark suite, which leads to much more clauses in the MAX-SAT formulation and a much more difficult problem structure.

TABLE III
RUNTIME RESULTS FOR THE MAX-SAT FLOW DURING THE NETLIST LOCALIZATION STEP

Circuit	Size (LUTs)	Runtime (s)
barrel64	1041	21
fip_cordic_rca	467	4
mux64_16bit	1277	11
aes_core	5359	91
huffman_enc	633	100
b03	59	5
b04	184	21
b08	58	1
b09	50	1
b10	78	5
b11	167	1
b12	378	146
b14	3014	266
b15	6012	389

V. LIMITATIONS OF APPROACH

Although our approach to ECOs is quite robust, there are some limitations to its application. First, if the number of errors or changes required are quite large, our approach will not be appropriate. The reason that large changes cannot be handled in our approach is due to two aspects; either our localization step fails to find a small number of focal points or our correction step cannot find a suitable decomposition for a given corrective function F_c. To understand this, consider the case where we injected multiple errors into our design. In this experiment, our MAX-SAT-based localization step will return several possible focal points. As the number of injected errors increases, our MAX-SAT formulation may give "false positives" due to the number of interacting errors in the design. As for correcting multiple errors, the complexity of the correction problem increases as the number of errors increases. This is because we are using an iterative approach where we attempt to fix each error one at a time. This leads to problems since some corrections require simultaneous modifications to two or more focal points, which our approach currently does not support. Without this ability, single point corrections may require more than 20 basis functions to implement the dependency function h: something that does not decompose well for the purposes of ECOs.

To avoid this issue, our MAX-SAT formulation can implicitly reduce the number of focal points. This was discussed previously in Section III-A and illustrated in Fig. 7. Here, if the source of the discrepancy between the existing circuit and the model netlist is dominated by a given node, MAX-SAT will return a solution focused on the dominator node, which reduces the number of focal points. Furthermore, if a given focal point has primary outputs within its transitive fanout which are dependent solely on a single focal point (i.e., only one focal point exists in the transitive fanin of the primary output), we can apply our SAT based resynthesis on the single focal point and ignore all other focal points. However, even with these conditions, a large number of focal points is still possible, such as in cases during architectural changes or multiple interacting bugs. Many of these cases would not be handled by ECOs in the first place, but if an ECO is required, a mixed manual and automated approach could be used where the user can partition the design manually, followed by running our approach on each partition.

VI. CONCLUSION

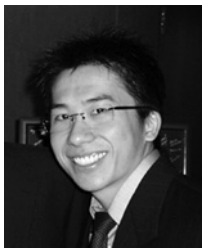
Improving the ECO experience through automation is necessary to reduce the complexity of using FPGAs. In this paper, we have shown an approach to automate the ECO experience. We have shown that our technique is robust and applicable to commercial FPGAs where we apply our work to Altera's Stratix architecture. We have also shown how we can integrate our flow with Altera's commercial CAD suite Quartus II. All of these factors provide a promising foundation toward the full automation of ECOs.

ACKNOWLEDGMENT

The authors would like to thank the help of T. Czajkowski with the integration of Quartus with our academic tools through his PST tool kit developed at the University of Toronto, Toronto, ON, Canada. They would also like to thank T. Yang from the University of Toronto for their insightful comments.

REFERENCES

[1] Altera Corporation, "Engineering change management with the chip planner," San Jose, CA, Tech. Rep. QII52017-9.1.0, Nov. 2009.
[2] K. Morris, "Time for a change: Mentor mondernizes the ECO," *FPGA and Structured ASIC*. Portland, OR: Techfocus Media, May 2006.
[3] S. Golson, "The human ECO compiler," in *Proc. SNUG Conf.*, Trilobyte Systems, 2004, pp. 1–57.
[4] R. Goering. (2007). Post-silicon debugging worth a second look. *EE Times* [Online]. Available: http://www.eetimes.com
[5] D. Platzker, "FPGA design meets the Heisenberg uncertainty principle," SOCcentral, Tech Pro Communications, Lawrenceville, NJ, 2005.
[6] Altera Corporation. *Quartus II University Interface Program* [Online]. Available: http://www.altera.com/education/univ/research/unv-quip.html
[7] J. Cong and M. Sarrafzadeh, "Incremental physical design," in *Proc. ISPD*, 2000, pp. 84–93.
[8] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of digital errors with PRIAM," in *Proc. ICCAD*, 1989, pp. 30–33.
[9] K. Chang, I. L. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," *IEEE J. Technol. Comput.-Aided Design*, vol. 27, no. 1, pp. 184–188, Jan. 2008.
[10] Y.-S. Yang, S. Sinha, A. Veneris, and R. K. Brayton, "Automating logic rectification by approximate SPFDs," in *Proc. ASP-DAC*, 2007, pp. 402–407.
[11] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng, "AutoFix: A hybrid tool for automatic logic rectification," *IEEE J. Technol. Comput.-Aided Design*, vol. 18, no. 9, pp. 1376–1384, Sep. 1999.
[12] Mentor Graphics, "Precision synthesis: Product overview," in *Advanced FPGA Design Synthesis*. Wilsonville, OR: Mentor Graphics, 2005.
[13] *Smart Compile Technology: Smart Guide*, Xilinx Corporation, San Jose, CA, Xilinx Press Release, 2008.
[14] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," in *Proc. ICCAD*, 2009, pp. 789–796.
[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. DAC*, 2001, pp. 530–535.
[16] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, no. 1, pp. 4–15, Jan. 1992.
[17] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah, "Improved design debugging using maximum satisfiability," in *Proc. FMCAD*, 2007, pp. 13–19.
[18] S.-J. S.-R. Pan, K.-T. Cheng, J. Moondanos, and Z. Hanna, "Generation of shorter sequences for high resolution error diagnosis using sequential SAT," in *Proc. ASP-DAC*, 2006, pp. 25–29.
[19] C.-C. Lee, J.-H. R. Jiang, C.-Y. R. Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental SAT solving," in *Proc. ICCAD*, 2007, pp. 227–233.
[20] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. CAV*, vol. 2725. 2003, pp. 1–13.
[21] A. Mishchenko, R. K. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks," in *Proc. ICCAD*, 2008, pp. 38–44.
[22] V. Manohararajah, D. P. Singh, and S. D. Brown, "Post-placement BDD-based decomposition for FPGAs," in *Proc. FPL*, Aug. 2005, pp. 31–38.



Andrew C. Ling (M'02) received the B.A.Sc. degree in computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2003, the M.A.Sc. degree in computer and electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2005, under the supervision of Professor S. D. Brown, and the Ph.D. degree in computer and electrical engineering from the University of Toronto in 2009 under the supervision of Professors S. D. Brown and J. Zhu.

In 2005, he was part of the Physical Synthesis Team, Altera Toronto Technology Center, Toronto, and currently helps lead the CAD Team in a semiconductor startup based in Toronto. His current research interests include computer-aided design algorithms, field-programmable very large scale integration technology, and computer architecture.

Stephen D. Brown (M'90) is currently a Full Professor with the University of Toronto, Toronto, ON, Canada, and is the Director of the Worldwide University Program Altera. From 2000 to 2008, he was the Director of Research and Development with the Altera Toronto Technology Center, Toronto. He is an author of more than 60 scientific publications and is the co-author of three textbooks: *Fundamentals of Digital Logic with VHDL Design* (New York: McGraw-Hill, 2004), *Fundamentals of Digital Logic with Verilog Design* (New York: McGraw-Hill, 2002), and *Field-Programmable Gate Arrays* (Kluwer Academic, 1992). His current research interests include computer-aided design algorithms, field-programmable very large scale integration technology, and computer architecture.

He received the Canadian Natural Sciences and Engineering Research Council's 1992 Doctoral Prize for the Best Doctoral Thesis in Canada. He has received a number of awards for excellence in teaching.

Sean Safarpour (M'03) received the B.A.Sc. degree in computer engineering from the University of British Columbia, Vancouver, BC, Canada, and the M.A.Sc. and Ph.D. degrees in computer engineering from the University of Toronto, Toronto, ON, Canada.

He is currently the Chief Technology Officer with Vennsa Technologies, Toronto, where he is in charge of research and development. His current research interests include design debugging, formal verification techniques, and formal engines such as SAT, QBF, and SMT solvers.

Jianwen Zhu (M'95) received the B.E degree in electrical engineering from Tsinghua University, Beijing, China, in 1993, and the M.S. and Ph.D. degrees in computer science from the University of California, Irvine, in 1996 and 1999, respectively, where he co-invented the SpecC Language.

He was an Assistant Professor in January 2000, and an Associate Professor in July 2005, with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, where he is affiliated with both the Computer Engineering Group and the Electronics Group. He leads the Synthesis Research Group, University of Toronto, to produce theories, languages, algorithms, and software prototypes for that matter.