Towards Automated ECOs in FPGAs

Andrew Ling
Electrical and Computer Engineering
University of Toronto
aling@eecg.toronto.edu

Stephen Brown
Toronto Technology Center
Altera Corporation
sbrown@altera.com

Jianwen Zhu and Sean Safarpour
Electrical and Computer Engineering
University of Toronto
jzhu|sean@eecg.toronto.edu

ABSTRACT

During the FPGA design flow, engineering change orders (ECOs) have become an essential methodology to apply late-stage specification changes and bug fixes. ECOs are beneficial since they are applied directly to a place-and-routed netlist which preserves most of the engineering effort invested previously. Unfortunately, designers often apply ECOs in a manual fashion which has an unpredictable impact on the design's final correctness and end costs.

As a solution, we introduce an automated method to tackle the ECO problem. Specifically, we introduce a 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. Our technique is robust enough to handle a wide range of changes. Furthermore, our technique can successfully make late-stage functional changes while minimally perturbing the place-and-routed netlist: something that is necessary for ECOs. When applied to several benchmarks on Altera's Stratix architecture, we show that our approach can automatically apply ECOs in over 80% of the cases presented. Furthermore, our technique does this with a minimal impact to the circuit performance where on average over 90% of the placement and routing wires remain unchanged.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Logic Design - *Automatic synthesis*

General Terms

Algorithms, Design, Verification

Keywords

Boolean Satisfiability, Resynthesis, Optimization, FPGA, PST

1. INTRODUCTION

As FPGA design complexity increases, achieving tight timing and area constraints is becoming very challenging and often requires several design iterations as shown at the top of figure 1. Here, a design described in a Hardware Description Language (HDL) is passed to an FPGA CAD flow such as Quartus II. If design constraints are not met after the CAD flow finishes, the designer must modify their existing HDL code and rerun the CAD flow. To complicate things, even if performance constraints are met, bug fixes or feature changes may still be required. Applying these changes directly at the HDL and using a "from scratch" FPGA recompile is often not an option since this does not guarantee that circuit performance will be maintained. 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 place-and-routed netlist such as the rewiring of LUTs or changing LUT implementations. As a result, 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 figure 1 through the path labeled as *Original Flow*. The feedback process shown in *Original Flow* can often tie up a designer for several months [1, 2, 3, 4] which leads to missed project deadlines. With very short design cycles demanded today, this is very detrimental to a product's success.

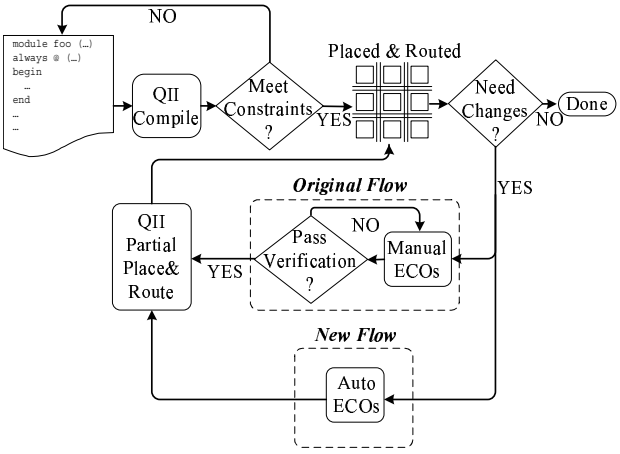


Figure 1: Generalized flow using ECOs.

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 since it requires no manual intervention. This can potentially reduce the time spent on ECOs from weeks to a few hours.

FPGA'09, February 22–24, 2009, Monterey, California, USA.

The proposed flow is shown at the bottom of figure 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 which 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. Following this, the proposed algorithm applies a resynthesis step to alter the circuit. The resynthesis step first defines the function that will replace the existing nodes found in the localization step. Next, the proposed algorithm uses a SAT-based decomposition to break our necessary changes into a set of subfunctions. During decomposition, subfunctions must exist within the place-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, we will show that unlike previous approaches, we can handle multiple changes and do not require a formal verification step to ensure that our modifications are correct.

The ultimate goal of this work is to create an ECO flow which can automatically update the behavior of a circuit in a manner which preserves as much as possible the placement and routing of the circuit while maintaining timing. This can be achieved by satisfying the following three criteria: a) be minimally disruptive to an existing place-and-routed circuit; b) have little or no impact to the performance of a circuit; c) be robust enough to handle a wide range of changes. In later sections, we will show that our approach does indeed satisfy the previous three criteria. In particular, when applied to Altera's Stratix architecture using several benchmarks from the Altera QUIP [5] and the ITC benchmark suite we will show our technique on average leaves over 90% of a place-and-routed circuit unchanged; has a marginal impact to circuit performance where we reduce the frequency by less than 3% on average; and is robust enough to handle over 80% of the changes presented. In all cases, we require no manual intervention by the user.

The rest of the paper is organized as follows: section 2 discusses the background and related work; section 3 describes the proposed algorithm in detail; section 4 shows experimental results; and section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

2.1 Terminology

The combinational portion of a LUT network can be represented as a directed acyclic graph (DAG). 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. Primary inputs (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.

When defining a function at a node v, it is termed a *global function* if the function support set consist 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.

2.2 Related Work

ECOs covers a wide range of work which is either used to incrementally improve the performance of a design [6] or help modify the behavior of a design such that circuit performance is maintained [7, 8, 9, 10, 11, 12]. Our work falls in the latter category where we focus on late-stage ECOs which are applied directly to a place-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 [8] and [9]. Here, using formal methods and random simulation, the authors in [8, 9] 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 is promising where they can automatically apply ECOs in more than 70% of the cases they present.

The technique in [9, 8] requires an explicit representation of any modification, which does not scale to large changes. This is not a problem in ASICs since ECOs requiring major 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. Our approach improves on this where we can handle much larger changes by using a SAT-based approach shown in the following sections.

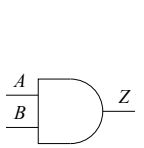
2.3 Boolean Satisfiability (SAT)

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 figure 2

Recent advancement in SAT solvers [13] have improved their runtime performance by an order of magnitude, thus several problems in EDA 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 [14] 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 figure 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. Figure 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 3.1, we will show a SAT-based technique using the characteristic function to isolate errors within a design.

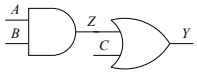
In this paper, we will often refer to a circuit as being *satisfiable* without explicitly showing its characteristic function. We define a circuit as being satisfiable if there is a satisfying assignment to the characteristic function of the circuit. The circuit is unsatisfiable if no such assignment exists. Furthermore, a *satisfying assignment* to a circuit is an assignment which is consistent with the circuit's



A	B	Z	F_{AND}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$F_{\text{AND}} = \underbrace{(A+\overline{Z})}_{\text{clause}} \cdot (B+\overline{Z}) \cdot (\overline{A}+\underbrace{\overline{B}}_{\text{literal}}+Z)$$

Figure 2: Characteristic function derivation for 2-input AND gate.



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

Figure 3: Cascaded gate characteristic function: top clauses from `AND` gate; bottom clauses from `OR` gate.

characteristic function. For example, one possible satisfying assignment to the circuit in figure 3 is $ABCZY = 01101$.

3. AUTOMATED ECOS: GENERAL TECHNIQUE

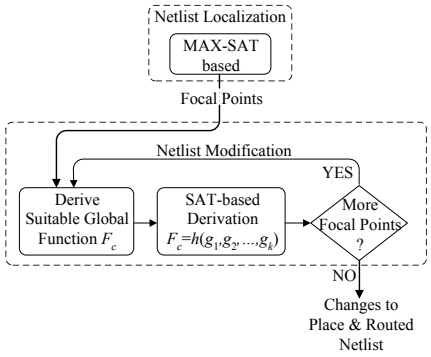


Figure 4: Automated ECO flow.

The goal of our work is to incrementally modify an existing place-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 figure 4. This flow consists of two distinct steps: netlist localization followed by netlist modification.

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 place-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 3.2, 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 place-and-routed circuit as much as possible.

3.1 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 [15]. 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.

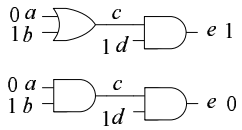


Figure 5: Maximum satisfiability solution to localization.

This process is illustrated in the figure 5. Consider the two circuits in figure 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 of $\{e = 1\}$. The corresponding CNF for the original circuit and the input/output vectors are shown below.

$$\begin{aligned} & (\overline{a}) \cdot (b) \cdot (d) \cdot (e) \cdot \\ & (a+\overline{c}) \cdot (b+\overline{c}) \cdot (\overline{a}+\overline{b}+c) \cdot \\ & (c+\overline{e}) \cdot (d+\overline{e}) \cdot (\overline{c}+\overline{d}+e) \end{aligned} \quad (2)$$

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, false locations will be identified. However, 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.

In our case, we will often refer to discrepancy points isolated by the localization point 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*.

In the example shown in figure 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 an RTL netlist.

3.2 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 figure 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 place-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 place-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.

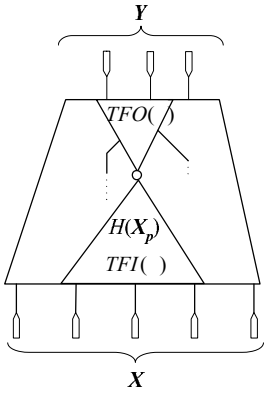


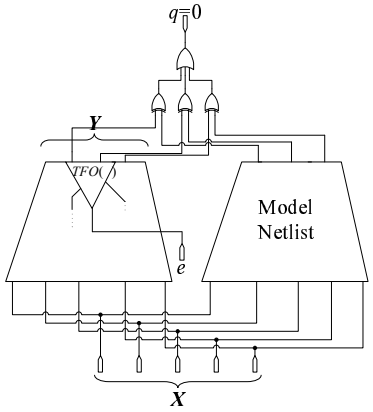
Figure 6: Illustration of localized node ξ. Modification can be applied by replacing $H(\boldsymbol{X}_p)$ with some arbitrary function $F_c(\boldsymbol{X}_s)$.

To understand why we need to find a suitable global function to replace the function at each focal point, consider the circuit illustrated in figure 6. 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 figure 7. 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.

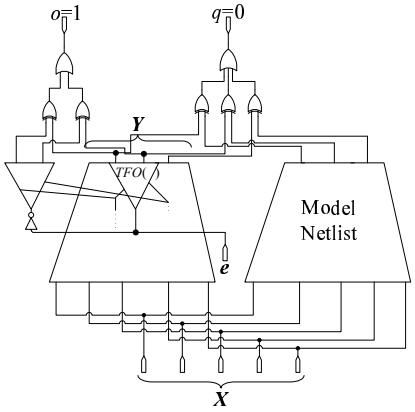
DEFINITION 3.1. *Suitable Global Function: In the case of a single focal point, ξ, identified during the localization step, a suitable global function F_c is defined as a function which can be used to replace an existing global function H found at ξ such that the resulting circuit matches the input-output behavior of a model netlist.*

LEMMA 3.2. *For a single-location modification, such as the one shown in figure 6, the circuit behavior of the original circuit can be altered to match a model netlist if and only if the global function located at focal point ξ is replaced with another suitable global function $F_c(\mathbf{X}_s)$.*

Our work follows lemma 3.2 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.



(a) Attaching circuit to model netlist



(b) Attaching observability constraints for $CircuitConstrain$

Figure 7: Circuit to be modified

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. 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 figure 7. On the left side of figure 7(a) we show a circuit we intend to modify, which looks similar to figure 6. However, in figure 7(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 figure 7(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 figure 7(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 figure 7(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 3.3. *$CircuitConstrain$ 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 3.4. *$CircuitConstrain$ is satisfiable if and only if the assignment to e and the assignment to the input vector* **X** *is consistent with a suitable F_c. In other words, for all satisfying assignments to $CircuitConstrain$, F_c(**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 3.2. However, this is not possible since any satisfying assignment to $CircuitConstrain$ must match the input-output behavior of the model netlist by lemma 3.3. 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 3.3. Thus, by contradiction, if the values on e and $\boldsymbol{X}$ match a suitable F_c then we have a satisfying assignment to $CircuitConstrain$. 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. □

By proposition 3.4, we can use the circuit in figure 7(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.

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 place-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 figure 8. 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.

$x_1x_2x_3$	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

Figure 8: Functional dependency example

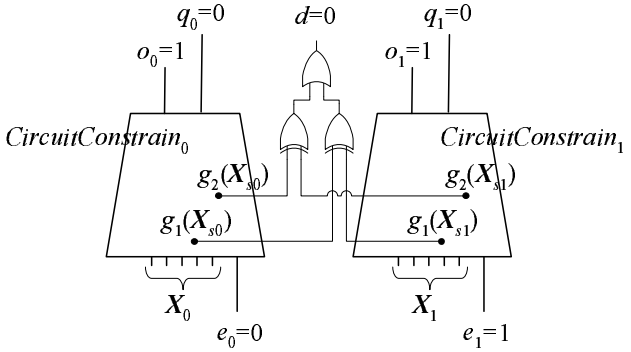


Figure 9: Deriving global function $F = h(g_1, g_2)$ using dependency check construct.

To formally check the condition described previously we will use a SAT-based approach [16]. 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. To create this characteristic function, we construct the circuit shown in figure 9. In figure 9, we have duplicated $CircuitConstrain$ 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 figure 9. 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 figure 9 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 figure 9. 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 3.4, 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 3.5. *In $CircuitConstrain$ of figure 7(b), if pin e is assigned a value 0, $CircuitConstrain$ is satisfiable if and only if the input vector $\mathbf{X}$ is assigned a value found in the offset of the function $F_c(\mathbf{X})$. Conversely, if pin e is assigned a value 1, the circuit is satisfiable if and only if the input vector $\mathbf{X}$ is assigned a value found in the onset of the function $F_c(\mathbf{X})$.*

Following from lemma 3.5, the circuit in figure 9 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 figure 9 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 figure 9, we can create the characteristic function shown in equation 3.

$$\begin{aligned}\exists \boldsymbol{X}_0 \exists \boldsymbol{X}_1 \; & (CircuitConstrain_{0\ CNF}) \cdot \\ & (CircuitConstrain_{1\ CNF}) \cdot \\ & (e_0 \equiv 0) \cdot (e_1 \equiv 1) \cdot (g_1(\boldsymbol{X}_{s0}) \equiv g_1(\boldsymbol{X}_{s1})) \cdot \\ & (g_2(\boldsymbol{X}_{s0}) \equiv g_2(\boldsymbol{X}_{s1})) \cdot (\boldsymbol{X}_0 \neq \boldsymbol{X}_1)\end{aligned} \tag{3}$$

In equation 3, assume that $CircuitConstrain_{0\ CNF}$ ($CircuitConstrain_{1\ CNF}$) represents the clauses derived from the characteristic function of $CircuitConstrain_0$ ($CircuitConstrain_1$). Equation 3 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 3.6. *If equation 3 is satisfiable the basis functions, g_i, cannot be used to implement a suitable global function F_c. If equation 3 is unsatisfiable, the basis functions, g_i, can implement a suitable global function F_c.*

PROOF. Assume if equation 3 is satisfiable then F_c has a decomposition $h(g_1, g_2, ..., g_k)$. If equation 3 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 3.5. 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 equation 3. 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 equation 3 is satisfiable then F_c cannot be decomposed into $h(g_1, g_2, ..., g_k)$. To prove the second statement in proposition 3.6, we can use a similar argument and will not show it in detail here. □

If equation 3 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 [17] and [16].

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. However, 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 [18] and reduces the number of logic levels the critical wires have to go through in order to preserve timing.

3.2.1 Searching for Basis Functions

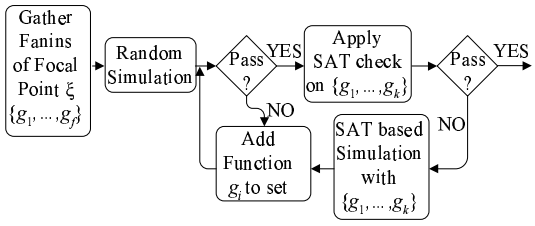


Figure 10: Searching for basis functions flow.

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 figure 10. 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. 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 figure 10, 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.

$x_1x_2x_3$	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

Figure 11: Random simulation example.

The random simulation check shown in figure 10 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 figure 7(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 equation 3. For example, consider figure 11. 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 equation 3. In contrast, basis set $\{g_1, g_3\}$ is a possible set since there does not exist any input pair which satisfies equation 3. 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 vector generation to create input patterns that can find additional vector pairs that need to be *differentiated*. The SAT based vector generation works by duplicating the circuit and "locking" each pair of duplicated basis functions to equivalent values 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 figure 11, $\{g_1, g_2\}$ 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.

3.2.2 Partitioning of Circuit and Treatment of Registers

In order to help reduce the size of equation 3, we can remove portions of the circuit which do not impact the behavior of the current node being modified. This is illustrated in figure 12. Here, we

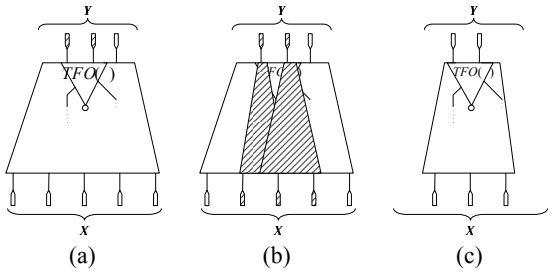


Figure 12: Circuit reduction.

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 figure 12(a). Following this, we collect the transitive fanin of each output, as shown in figure 12(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 figure 12(b) is 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 figure 12(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.

3.2.3 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 "locked down" such that they are set to fixed values. 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 locking down 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 4 where we have at most 4 focal points into the design.

4. RESULTS

To ensure that the resynthesis technique described in the previous sections can be applied to late-stage ECOs, we must ensure that our technique can automatically update the behavior of a circuit while minimally disrupting the existing place-and-routed netlist and preserve timing. Thus, we evaluate our work based on three criteria: how disruptive our technique is on a place-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 [5] and the ITC benchmark suite.

All experiments are run on a PentiumD 3.2 GHz processor with 2MB 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 [5].

4.1 Specification Changes

In our first set of experiments, we seek to see how disruptive our approach is on an existing place-and-routed circuit and 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. During this experiment, we apply a late-stage specification change both using a full recompile of the design and using the proposed ECO approach. During the experiment, we first implement a 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 3). 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. An example of such changes is shown in figure 13. After our code is modified, we perform a full recompile of the modified HDL code and record the performance impact. For comparison, 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 place-and-routed netlist.

```
//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                          ...
...
```

Figure 13: Example changes in HDL code.

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 place-and-routed circuit. These results are shown in table 1. 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 with respect to *Before* as a result of a full compile of the modified HDL code. The final group headed by $After_{ECO}$ shows the 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.

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% to 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 [1], 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 5 focal points being returned. In cases where 5 or more focal points were returned, we found that our approach had difficulty finding a non-disruptive modification to the place-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 only move nodes to existing free locations within the FPGA and 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. 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 1.

4.2 Error Correction

In our second set of experiments, we apply the proposed ECO technique to fix errors inserted into a design. Errors include rewiring of LUTs or changing the LUT functions of various nodes. 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.

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

Table 1: Impact of modifications on circuit performance on a Stratix chip.

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

Table 2: Automated correction results on a Stratix chip.

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 place-and-route is rerun on the circuit to achieve the best performance possible.

Table 2 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 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. Unlike the previous results shown in table 1, 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 2 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 place-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 2, the average disruption to the placement and routing drops to approximately 5%.

When inserting less than five errors into the design, we found we were able to correct over 80% of the circuits. 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.

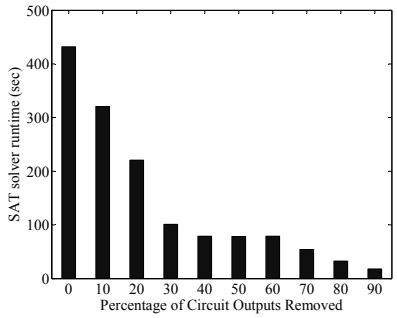


Figure 14: Impact of output removal on SAT solver runtime for circuit `b15`.

As a final experiment, we examined the impact of the the primary output removal heuristic described in section 3.2.2. This is shown in figure 14 where we show the percentage of outputs removed versus 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 figure 14 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.

5. CONCLUSIONS

Improving the ECO experience through automation is necessary to reduce the complexity of using FPGAs. In this work, 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.

6. ACKNOWLEDGMENTS

The authors would like to thank the help of Tomasz Czajkowski with the integration of Quartus with our academic tools through his PST tool kit developed at the University of Toronto. We would also like to thank Terry Yang from the University of Toronto and Kevin Chung and Paul Kundarewich from Xilinx for their insightful comments.

7. REFERENCES

[1] K. Morris, "Time for a change: Mentor mondernizes the ECO," *FPGA and Structured ASIC*, May 2006.

[2] S. Golson, "The human ECO compiler," Trilobyte Systems, 2004.

[3] R. Goering, "Post-silicon debugging worth a second look," EEtimes, 2007. [Online]. Available: http://www.eetimes.com/

[4] D. Platzker, "FPGA design meets the heisenberg uncertainty principle," SOCcentral, 2005.

[5] Altera Corporation, *Quartus II University Interface Program*. [Online]. Available: http://www.altera.com/education/univ/research/unv-quip.html

[6] J. Cong and M. Sarrafzadeh, "Incremental physical design," in *International Symposium on Physical Design*, 2000, pp. 84–93.

[7] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of digital errors with PRIAM," in *International Conference on Computer-Aided Design*, 1989, pp. 30–33.

[8] K. Chang, I. L. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," *IEEE Journal on Technology in Computer-Aided Design*, vol. 27, no. 1, pp. 184–188, 2008.

[9] Y.-S. Yang, S. Sinha, A. Veneris, and R. K. Brayton, "Automating logic rectification by approximate SPFDs," in *Asia-South Pacific Design Automation Conference*, 2007, pp. 402–407.

[10] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng, "AutoFix: a hybrid tool for automatic logic rectification," vol. 18, no. 9, pp. 1376–1384, Sept. 1999.

[11] Mentor Graphics, "Precision synthesis: product overview," Advanced FPGA Design Synthesis, 2005.

[12] Xilinx Corporation, "SmartCompile technology: SmartGuide," Xilinx Press Release, 2008.

[13] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of the 38th Design Automation Conference (Design Automation Conference'01)*, 2001. [Online]. Available: citeseer.ist.psu.edu/moskewicz01chaff.html

[14] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[15] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah, "Improved design debugging using maximum satisfiability," in *Formal Methods in Computer-Aided Design*, 2007, pp. 13–19.

[16] 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 *International Conference on Computer-Aided Design*, 2007, pp. 227–233.

[17] K. L. McMillan, "Interpolation and SAT-based model checking," vol. 2725, pp. 1–13, 2008.

[18] V. Manohararajah, D. P. Singh, and S. D. Brown, "Post-placement BDD-based decomposition for FPGAs," in *Field-Programmable Logic*, Aug. 2005, pp. 31–38.