

國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

半正規的工程改變命令方法

Semi-Formal ECO Method

謝佳霖

Chia-Lin Hsieh

指導教授：黃鐘揚 教授

Advisor: Chung-Yang (Ric) Huang, Ph.D.

中華民國 107 年 7 月

July 2018



摘要

在晶片設計的過程中，如果後期發現了設計瑕疵或有規格改變，工程改變命令是一個普遍使用的方法。我們提出一個兩階段的半正規工程命令改變方法來解決這個問題。我們先在兩個電路中找出最多的功能性配對，接著最佳化這些配對並產生最小的修補邏輯電路來修好有問題的電路。我們利用了 *FRAIG* 技術搭配基於模擬回饋的區域電路功能配對演算法來得到較好的配對，而這些較好的配對會減少所要修好有問題的電路的修補邏輯電路大小。實驗結果顯示我們提出的方法能在合理的時間內產生小的修補邏輯電路來修好有問題的電路。

關鍵詞：工程改變命令、功能性配對、修補邏輯電路

Abstract



Engineering change order (ECO) is a popular technique for rectifying design errors and specification changes in late design stages. We present a two-phase semi-formal patch generation to rectify multiple errors. We first 1) discover the functional matches in two circuits, then 2) optimize and generate a patch circuit from the matches. The ECO engine in this thesis discovers functional and structural matches in two circuits by the *FRAIG* technique and the simulation-guided cut-matching algorithm. Then, the combinational equivalence checking technique combined with a linear-time selection heuristic is processed to minimize the patch size from the matches. The experimental results show that this ECO engine can rectify circuits with small patch size within reasonable runtime.

Index Terms – Engineering change order, functional matches, patch circuit

Contents



| | |
|---|-----------|
| 摘要..... | i |
| Abstract..... | ii |
| Chapter 1 Introduction..... | 1 |
| Chapter 2 Preliminaries..... | 2 |
| 2.1 ECO Problem..... | 2 |
| 2.2 Match-and-Replace Method..... | 3 |
| 2.3 Miter and FRAIG..... | 5 |
| 2.4 Boolean Matching..... | 5 |
| 2.5 Circuit Functional Similarity..... | 6 |
| Chapter 3 Semi-Formal ECO Method..... | 8 |
| 3.1 Overview of Our ECO Method..... | 8 |
| 3.2 Matching Phase — Patch Region Identification..... | 10 |
| 3.2.1 Input-side Merging Frontier Identification..... | 11 |
| 3.2.2 Output-side Frontier Identification..... | 13 |
| 3.3 Replacing Phase — Patch Region Optimization..... | 19 |
| Chapter 4 Experimental Results..... | 23 |
| Chapter 5 Conclusions and Future Work..... | 29 |
| Reference..... | 30 |

List of Figures



| | | |
|----------|---|----|
| Fig. 1. | Interpolation circuit in [7] | 3 |
| Fig. 2. | Interpolation circuit in [9] | 4 |
| Fig. 3. | Matching matrix in [10] | 4 |
| Fig. 4. | Functionally-reduced And-Inverter graph algorithm..... | 6 |
| Fig. 5. | Semi-formal ECO method..... | 8 |
| Fig. 6. | Example of semi-formal ECO method..... | 9 |
| Fig. 7. | Patch replacement considering merged gates..... | 12 |
| Fig. 8. | Merged gate beneath/above input-side merging frontier..... | 13 |
| Fig. 9. | Example of (constrained) cut logic..... | 14 |
| Fig. 10. | Output-side frontier identification algorithm..... | 15 |
| Fig. 11. | Get guided-cut candidates algorithm..... | 17 |
| Fig. 12. | Example of cut generation w.r.t. duplicated merged gates..... | 18 |
| Fig. 13. | Simulation-guided cut-matching algorithm..... | 19 |
| Fig. 14. | Rectification pair selector..... | 20 |
| Fig. 15. | Example of inserting multiplexer on a rectification pair consists of a merged gate..... | 21 |
| Fig. 16. | Rectification pair selection algorithm..... | 22 |

List of Tables



| | |
|---|----|
| Table 1. Performance comparison on various modifications..... | 24 |
| Table 2. Modifications on the testcases..... | 25 |
| Table 3. Performance comparison under checkSizeEarlyReturn..... | 26 |
| Table 4. Performance comparison under checkSizeEarlyReturn and doExtractLeaves..... | 27 |
| Table 5. Performance comparison under checkSizeEarlyReturn and globalLevelEarlyReturn.... | 28 |

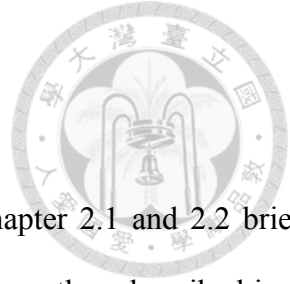


Chapter 1 Introduction

In Modern VLSI design process, late design changes are nearly unavoidable for specification changes or design errors. Considering the time-to-market pressure and cost, it is impractical to restart the whole design flow from scratch. A widely used solution is Engineering Change Order (ECO). Up-to-date ECOs are mostly conducted by manual efforts in practice. This is due to the fact that functional rectification in such late design stages are usually small and local. However, this approach is becoming very time-consuming and error-prone as the design size scales up. Therefore, as an alternative, designers may seek to quickly fix the designs in higher level (e.g., RTL) and then resort to an automatic functional ECO tool to rectify the original designs.

In this thesis, we propose a semi-formal method for the multi-error functional ECO problem. This is a two-phase approach: 1) matching phase: to maximize the matches between the golden and revised circuits in order to identify the minimal regions for rectification; and 2) replacing phase: to generate minimal patch logic to replace the corresponding region in the revised circuit. Different from [18], where a randomly enumerated cut with a selecting-matrix Boolean matching algorithm is applied to identify the backward matching boundary, we adopt the simulation-guided cut-matching process for the search of a backward matching cut. With this approach, we can effectively get a better rectification pair and lead to a smaller patch size within reasonable runtime.

The remained of this thesis is organized as follows. Chapter 2 introduces preliminaries related to this thesis. Chapter 3 describes our framework and introduces algorithms to identify the *patch region*, the *patch region* is then optimized by a linear-time heuristic. Chapter 4 shows the experimental results and Chapter 5 concludes the thesis.



Chapter 2 Preliminaries

In this chapter, we give the basic concepts related to our work. Chapter 2.1 and 2.2 briefly introduces previous works on ECO problem. The technique and concepts are then described in the remaining chapter.

2.1 ECO Problem

There have been many research publications focusing on functional ECO in recent years. [1]–[3] propose fault models to describe the design errors, such as wrong gate-type, missing inverter, misplaced wire, and so on. These algorithms rectify the buggy designs according to their fault models. Therefore, the patch circuits are often compact and predictable. However, these algorithms fail to produce the patch if the functional difference cannot be represented by the fault models, which happens in most cases.

Synthesis-based ECO algorithms [4]–[8] first identify an internal rectification signal by some diagnosis strategies, then apply re-synthesis techniques to generate a patch function for the functional differences. Although these algorithms have enabled the automatic functional ECO flow, their main drawback is that they all rely on a single-fix signal, as in Fig. 1. While in many cases the only possible single-fix signal is the primary output itself, the reported patches may be unacceptably large. To resolve these problems, [9] proposed a partial-fix interpolation-based ECO engine, which is another synthesis-based ECO engine. This engine performs partial rectifications iteratively to fix multiple errors in the design, as in Fig. 2. Although this approach can handle the problem more efficiently, it does not consider the correlations between multiple errors in the design. That is, each error in the design is considered independently. Tang et al. [10] proposed a cofactor reduction algorithm to produce multi-fix rectification functions by interpolation, which considers multiple errors simultaneously.

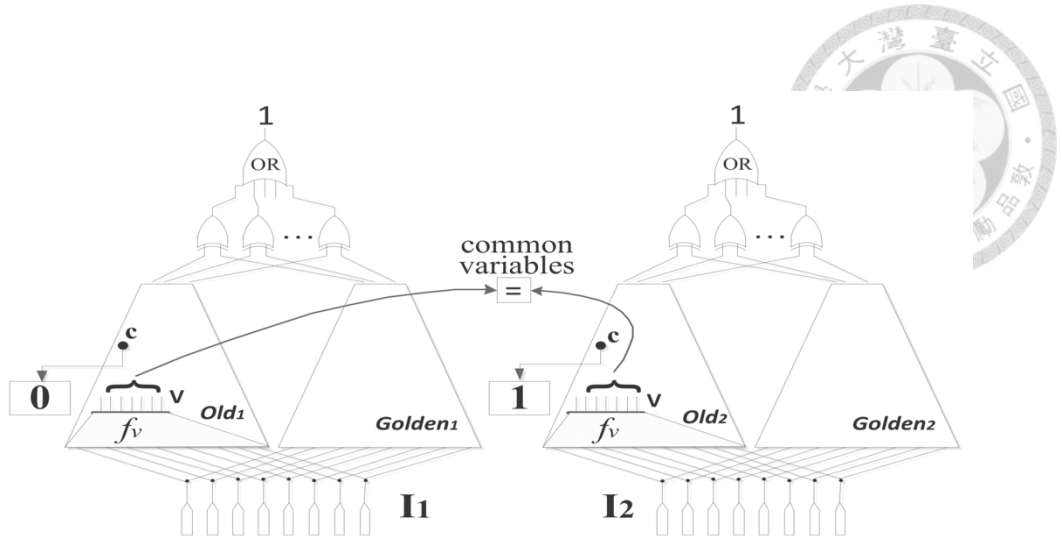


Fig. 1. Interpolation circuit in [7].

Sweeping-based ECO algorithms are proposed in [11]–[14]. They perform a structural comparison between the old and new circuits. Since in practice the functional rectifications in RTL are often corresponding to small and local changes, sweeping-based methods are very reasonable for functional ECOs. DeltaSyn [14] introduces a dual-phase flow to first identify the input-side and output-side boundaries of the changes and then collect the remaining netlist between two boundaries as the gates to be replaced. With well-studied forward sweeping algorithms [15]–[17], it is easy to merge functionally equivalent gates and derive the input-side boundary between two circuits. Nevertheless, a major challenge still remains: finding the output-side matching boundary is very difficult. Thus, the size of the output-side boundary that can be identified is always limited to a small number. What is worse, when functional symmetry resides, it takes much more time to enumerate all the permutations for the proper match.

2.2 Match-and-Replace Method

Match-and-Replace (M&R) [18] discovers functional and structural matches in two circuits by coordinating the SAT-sweeping and the cut-matching algorithms. In the matching phase, M&R randomly enumerates a cut from a matched gate g in the revised circuit, candidate gates are then selected gradually according to the matched gate g' and the selected cut of g . A matching matrix is

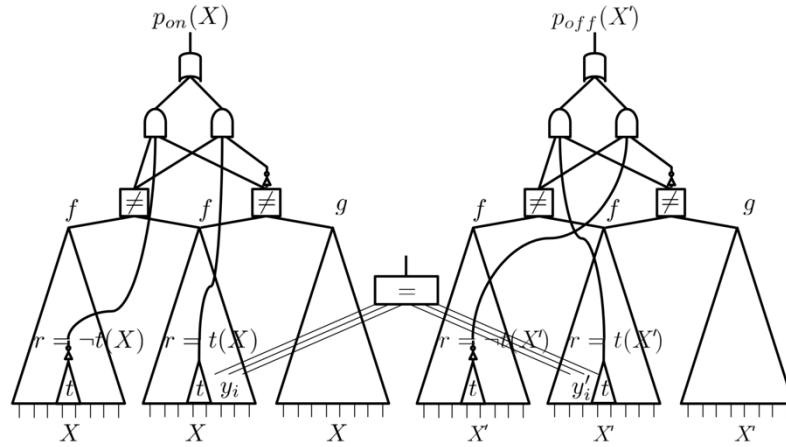


Fig. 2. Interpolation circuit in [9].

constructed and the SAT-based matching are processed to get the matched cut, the matching matrix is shown in Fig. 3. Since the cut from the matched gate g is generated with no clue, we can expect that the matching process is quite difficult and time-consuming, and even though a matched cut is discovered, the quality of the pairing on the matched cut might not be good, resulting in large patch size. We propose a simulation-guided cut-matching algorithm based on the results of the *FRAIG* technique, which gives a clue while matching the cuts. The rectification pair selection algorithm is processed in the replacing phase to get the minimal patch circuit. The experimental results show that we can rectify multiple errors with small patch size within reasonable runtime.

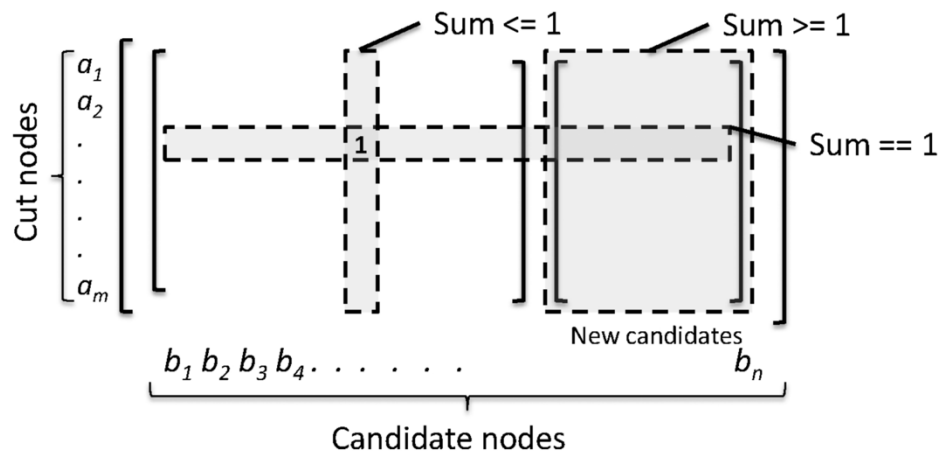
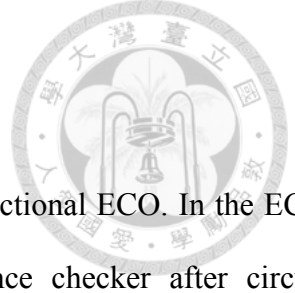


Fig. 3. Matching matrix in [10].



2.3 Miter & FRAIG

Functional equivalence checker is an important formal engine in functional ECO. In the ECO process, we check functional correctness by the functional equivalence checker after circuit rectification. If two circuits are non-equivalent, functional equivalence checker reports a counter-example, which is an error pattern for ECO process.

The equivalence checking problem is transformed to a satisfiability problem. A miter [19] applies an exclusive-or gate (XOR) to two Boolean network F and G which are to be compared. F and G are functionally equivalent if and only if there is no assignment to the inputs of F and G such that the miter evaluates 1. In other words, F and G are functionally equivalent if and only if the miter of F and G is unsatisfiable.

To speed up the equivalence checking process, Functionally-reduced AND-INV graph [16] (*FRAIG*) technique is often applied to identify and merge the equivalent gates. The algorithm is summarized in Fig. 4. Not only we can speed up the equivalence checking process, we can also utilize the merged gates discovered in the *FRAIG* process to speed up the output-side frontier identification process and guide the pairing in the cut-matching algorithm, which will be discussed in chapter 3.2.2.

2.4 Boolean Matching

Boolean matching is the problem of determining whether two Boolean functions are functionally equivalent under permutations and negations of inputs and outputs. It has been broadly adopted in logic synthesis and verification [14], [20]. It has also been applied to top-down network mappings, such as technology mapping [20] and backward matching in functional ECO [14].

Given two functions $f(X)$ and $g(Y)$, which have the same number of inputs, $|X| = |Y|$, Boolean matching under NPN-equivalence determines whether these two functions can be equivalent or complementary to each other under negations and permutations of their input variables. The “N”s’



Algorithm Functionally-Reduced And-Inverter Graph

```
1: Input: Circuit ckt
2: Output: Circuit ckt
3: solver  $\leftarrow$  Init_Proof_Model(ckt)
4: classes  $\leftarrow$  Init_FEC_By_Random_Simulation(ckt)
5: for each gate g in ckt in a topological order do
6:   fec  $\leftarrow$  Get_FEC(classes, g)
7:   if fec = null then continue
8:   for each m in fec do
9:     if Sat_Check_Equivalent(solver, g, m) = UNSAT then
10:      Merge(ckt, g, m)
11:     else
12:       pattern  $\leftarrow$  Get_Sat_Pattern(solver)
13:       classes  $\leftarrow$  Simulate_And_Update_FEC(classes, pattern)
14:     end for
15: end for
```

Fig. 4. Functionally-reduced And-Inverter graph algorithm.

on “NPN” mean negations on the inputs and outputs, respectively, and the “P” signifies the permutations on the inputs. Special cases of NPN-equivalence include NP-equivalence and P-equivalence, where NP-equivalence refers to functional equivalent under input negations and permutations, and P-equivalence implies functional equivalence under input permutations only.

In our ECO approach, we adopt a simulation-guided cut-matching process to explore the output-side boundaries of the circuits, which is an NP-equivalent Boolean matching problem, the only difference is that our cut-matching algorithm requires to search for the matching boundaries since the boundaries are not predefined.

2.5 Circuit Functional Similarity

Cosine similarity [21] is a simple method of measuring similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Given two vectors of



attributes, A and B, the cosine similarity is represented as:

$$similarity = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (1)$$

In the field of machine learning, cosine similarity is widely used in clustering multiple objects into multiple groups, which is a method of unsupervised learning. It is also a common technique for statistical data analysis used in many other fields.

Since multiple design errors or specification changes are to be rectified, gates affected by the buggy gate might be non-equivalent with the gates in the golden circuit. Through our observation, the functional difference between these gates can be observed only in few input assignments, i.e. the response of these gates are usually identical.

Identifying output-side matching boundary in sweeping-based ECO algorithms is very difficult, especially when functional symmetry resides. As stated above, two gates having a highly-similar-response might be equivalent if no errors are to be rectified. That is, the pairing of these gates might lead to smaller patch size. *Functionally Similar Candidates (FSC)* can be exploited to guide the pairing of the gates in the matching algorithm, the definition is as follows:

Definition 1: $FSC_g^\theta(\vec{X})$, *Functionally Similar Candidates* of a gate g is a set of gates $\vec{g} : (g_1, g_2, \dots, g_n)$. For every gate $g_i \in \vec{g}$, the cosine similarity of g 's and g_i 's response under input assignments \vec{X} are above threshold θ .

In our ECO approach, *FSC* can be a good guidance in pairing gates, which can lead to a better pairing and a smaller patch size.



Chapter 3 Semi-Formal ECO Method

In this chapter, the main flow of our ECO method will be introduced followed by an illustrative example in chapter 3.1. The algorithms and theories belong to our method will then be discussed in the remaining chapter.

3.1 Overview of Our ECO engine

Fig. 5 shows the flow of our ECO engine. After reading the golden and revised circuit, we perform *FRAIG* technique to discover and merge functionally equivalent gates. The matching phase is then processed to identify the *patch region*, and the replacing phase will optimize the *patch region* to 1) minimize the final patch circuit, and 2) guarantee all output functions are correct after the rectification. At last, our ECO engine will output the patched circuit. We will verify the functional equivalence of the patched and golden circuit by academic tool ABC [22] using *cec* command.

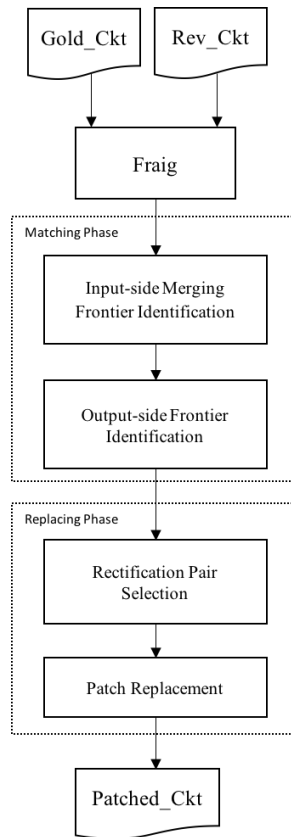
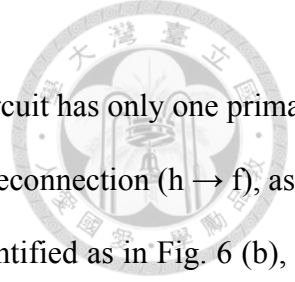


Fig. 5. Semi-formal ECO method.



An illustrative example is shown in Fig. 6. The golden and revised circuit has only one primary output k and k' respectively. The functional changes are caused by a wire reconnection ($h \rightarrow f$), as in Fig. 6 (a). After the *FRAIG* process, the input-side merging frontier is identified as in Fig. 6 (b), all gates below the input-side merging frontier are considered as don't-care gates. The simulation-guided cut-matching algorithm is then applied to get a set of rectification pairs RP_Set . In Fig. 6 (c), a matched cut is discovered, but only the colored gate is added to RP_Set , since it is the only non-equivalent gate on the matched cut. The rectification pair selection algorithm is processed on the existing RP_Set to get the minimal patch. Only the colored rectification pair is selected to be the final patch. We reconnect the wire from f to h as in Fig. 6 (d), and the revised circuit is now functionally equivalent to the golden. The patch size for this example is 0.

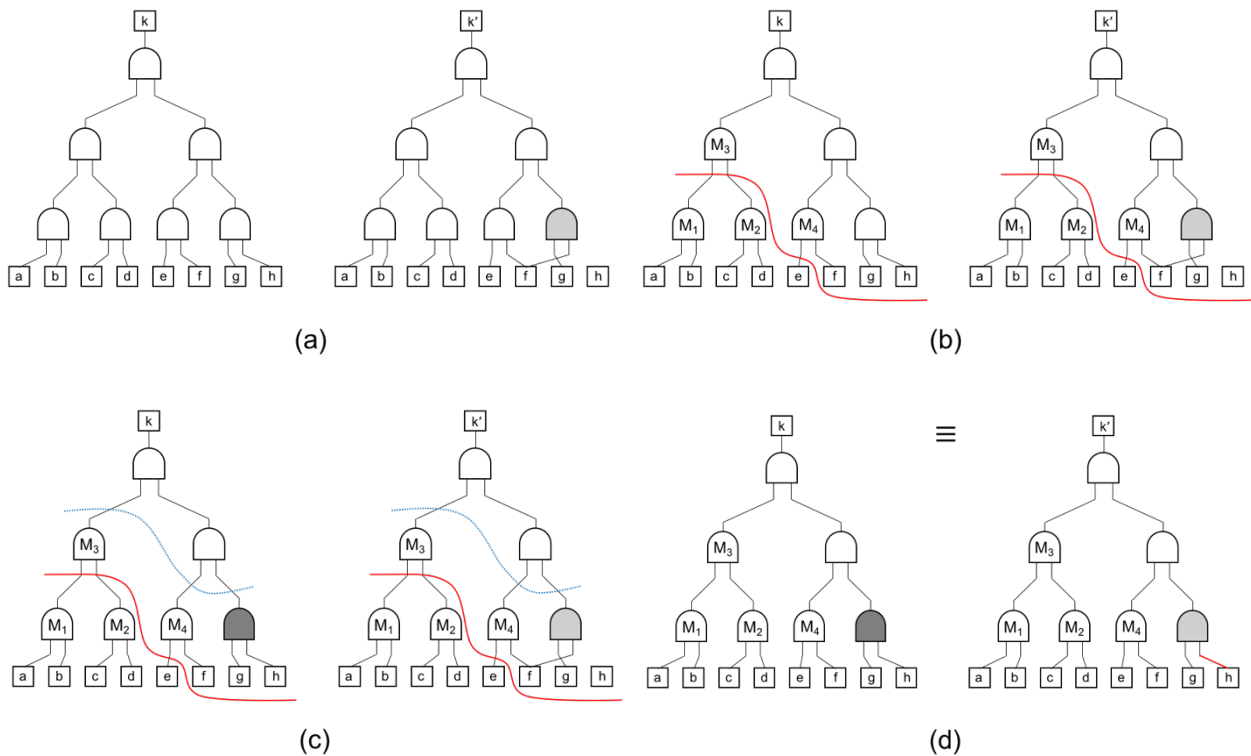
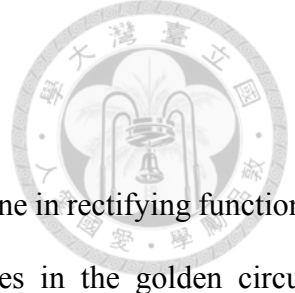


Fig. 6. Example for semi-formal ECO method. (a) Golden and Revised circuit. (b) Identify input-side merging frontier. (c) Identify output-side frontier. (d) Final patched circuit.



3.2 Matching Phase – Patch Region Identification

Patch Region is a set of rectification pairs, which assists the ECO engine in rectifying functional differences by replacing the gates in the revised circuits with the gates in the golden circuit. Rectification pairs in our ECO process are proposed to rectify functional differences between two circuits. In general, they are often in the form of a group of pairs. The definition is shown as follows:

Definition 2: A set of pairs RP_Set : $\vec{g} \leftrightarrow \vec{g}'$ is called a set of rectification pairs if each $g_i \in \vec{g}$ belongs to the revised circuit and $g'_i \in \vec{g}'$ belongs to the golden circuit, and when replacing every g_i with g'_i , the revised circuit becomes functionally equivalent to the golden.

In the rectification pair, g_i is called a patched gate and g'_i is its corresponding patch. By the rectification pair, the functional rectification can be formulated as:

$$\forall \vec{X}, Rev_Ckt(\vec{X}) \Big|_{\vec{g} \xrightarrow{replace} \vec{g}'} \equiv Gold_Ckt(\vec{X}) \quad (2)$$

where \vec{X} is the input assignment.

Please note that it is always possible to derive a set of rectification pairs to fix the differences between the revised and golden circuits. For example, the whole PO pairs naturally form a trivial set of rectification pairs. By directly replacing all outputs in the revised circuit with the outputs in the golden, the revised circuit is obviously equivalent to the golden. However, the size of the patches will be unacceptably large. The following lemma introduces an iterative approach:

Lemma 1: Given a rectification pair $p : g_i \leftrightarrow g'_i \in RP_Set$, $0 \leq i \leq n$ and a set of pairs P_Set : $\vec{h} \leftrightarrow \vec{h}'$, the set $RP_Set - \{p\} + P_Set$ is also a set of rectification pairs if P_Set is a set of rectification pairs of p and disjoint from RP_Set .

Proof: Since

$$\forall \vec{X}, Rev_Ckt(\vec{X}) \Big|_{\vec{g} \xrightarrow{replace} \vec{g}'} \equiv Gold_Ckt(\vec{X}) \quad (3)$$

If a set of pairs P_Set : $\vec{h} \leftrightarrow \vec{h}'$ is a set of rectification pairs of pair p : $g_i \leftrightarrow g'_i$, it can be

formulated as

$$g_i(\vec{X}) \Big|_{\vec{h} \xleftarrow{\text{replace}} \vec{h}'} \equiv g'_i(\vec{X}) \quad (4)$$

by (3) and (4)

$$\begin{aligned} \forall \vec{X}, Rev_{Ckt(\vec{X})} \Big|_{g_1 \xleftarrow{\text{replace}} g'_1, \dots, g_{i-1} \xleftarrow{\text{replace}} g'_{i-1}, g_{i+1} \xleftarrow{\text{replace}} g'_{i+1}, \dots, g_n \xleftarrow{\text{replace}} g'_n, \vec{h} \xleftarrow{\text{replace}} \vec{h}'} \\ \equiv Gold_Ckt(\vec{X}) \end{aligned} \quad (5)$$

By (5), a set of pairs RP_Set' : $RP_Set - \{p\} + P_Set$ must also be a set of rectification pairs.

According the above lemma, we are able to iteratively derive a set of new rectification pairs on the existing pairs.

In this chapter, input-side merging frontier identification and output-side frontier identification are presented in 3.2.1 and 3.2.2 respectively.

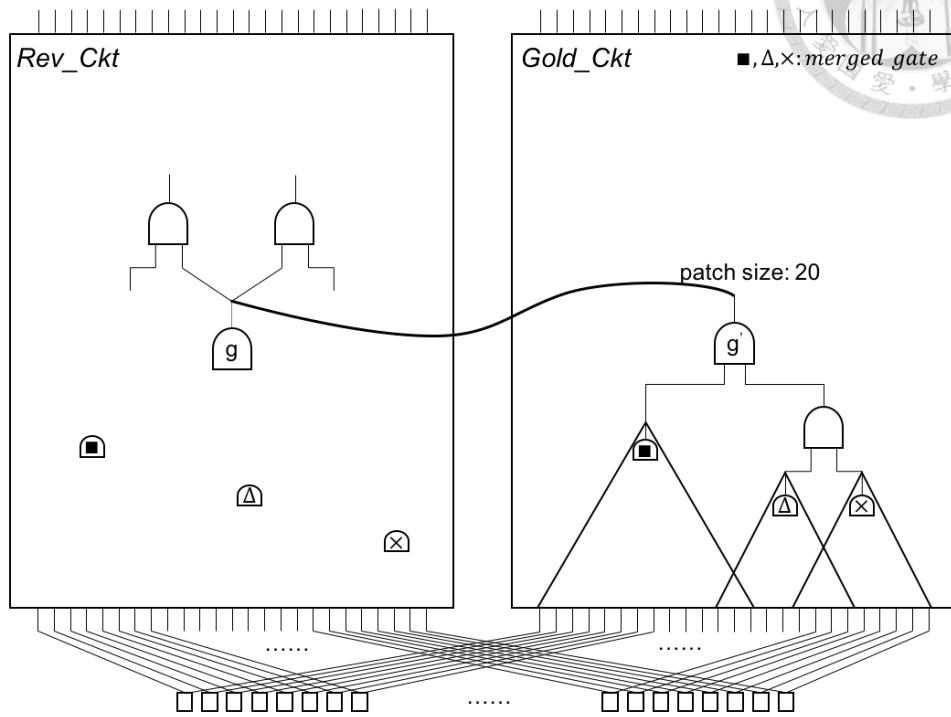
3.2.1 Input-side Merging Frontier Identification

FRAIG technique discovers and merges functionally equivalent gates. For a rectification pair $g \leftrightarrow g'$, a trivial way to replace g with g' is to patch the whole g' logic to g , which might have a large patch size. Since a merged gate represents a pair of functionally equivalent gate from both circuits, we can redirect the wire on the merged gates found in the g' logic to the functionally equivalent gate g_{eq} to reduce the patch size, Fig. 7 shows a simple example. When we are to replace g with g' , we can patch the whole g' logic to g , resulting in patch size = 20. If we find a merged gate in g' logic, we can redirect the wire to the functionally equivalent gate in the revised circuit to reuse the logic, resulting in patch size = 2. Note that the higher level of the merged gate located, the more patch size we can reduce in the replacing process. The merging frontier is defined as follows:

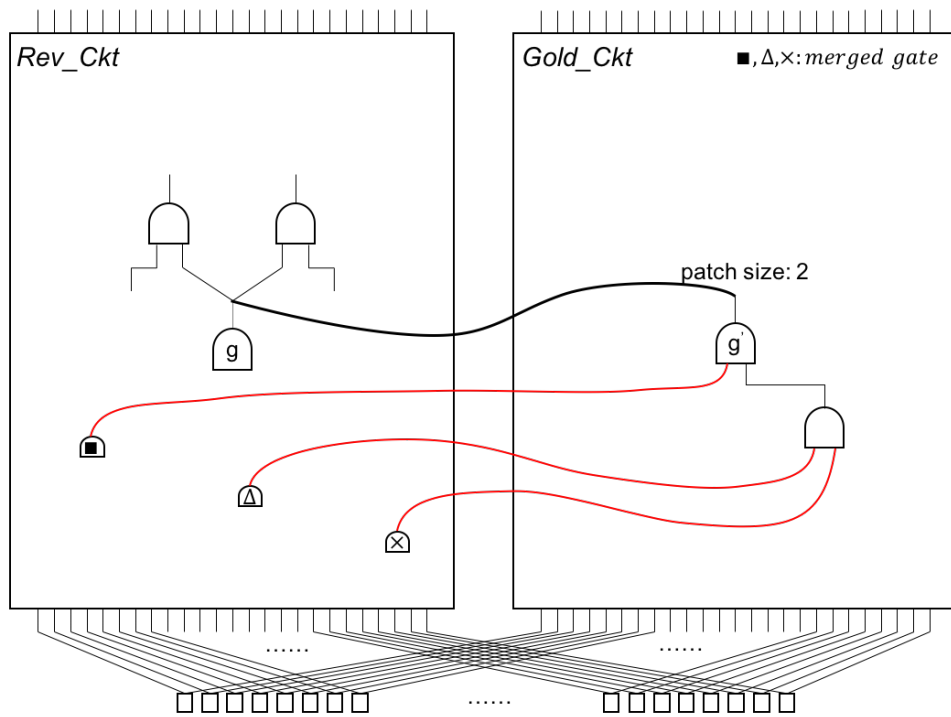
Definition 2: A merging frontier is a set of merged gates $\vec{m} : (m_1, m_2, \dots, m_n)$ if all primary inputs can find a cut $\vec{g} : (g_1, g_2, \dots, g_m)$ in their fanout cone such that $\vec{g} \subseteq \vec{m}$.

Note that a primary input can be a merging frontier of itself since it is naturally a merged gate

in the ECO process.

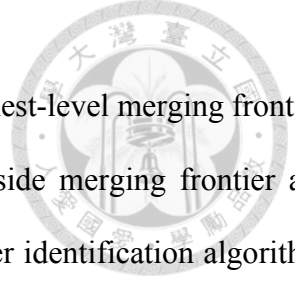


(a)



(b)

Fig. 7. Patch replacement considering merged gates. (a) Replace g with the whole g' logic. (b) Replace g with a portion of g' logic utilizing merged gates.



The input-side merging frontier identification aims at finding the highest-level merging frontier based on the results of the *FRAIG* process. All gates below the input-side merging frontier are considered as don't-care gates, which can speed up the output-side frontier identification algorithm described in 3.2.2. Fig. 8 shows the process of input-side merging frontier identification. For a merged gate g , if we can find a path with non-merged gates all the way to PO, it is considered above the input-side merging frontier. Otherwise, we can find a boundary of merged gates in g 's fanout cone. We mark all the gates within the boundary to be don't-care gates, then recursively call for the merged gates on the boundary to push the frontier to a higher level.

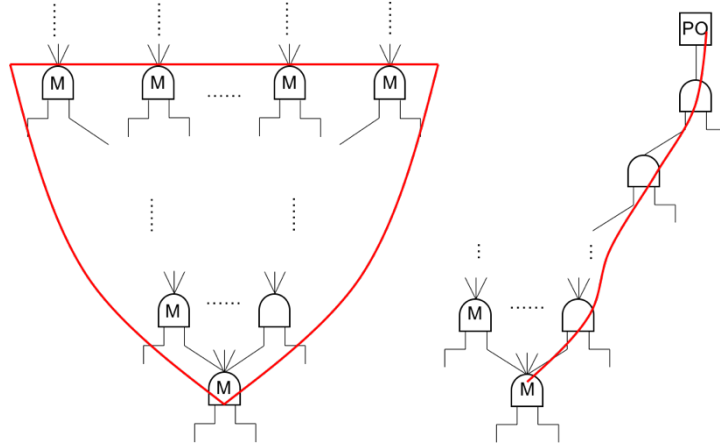


Fig. 8. Merged gate beneath/above input-side merging frontier.

3.2.2 Output-side Frontier Identification

In this section, we iteratively perform simulation-guided cut-matching algorithm to get a set of rectification pairs. We first define the cut logic as follows:

Definition 3: A cut logic $CL_{(l_1, l_2, \dots, l_n)}^o$ is a logic cone that represents the functionality of the output o with respect to (l_1, l_2, \dots, l_n) . A simple example is shown in Fig. 9. The cut logic of g with respect to (e, f) is $CL_{(e, f)}^g(X_1, X_2) = X_1X_2$.

Two cut logic (o, o') with respect to (\vec{l}, \vec{l}') are equivalent if

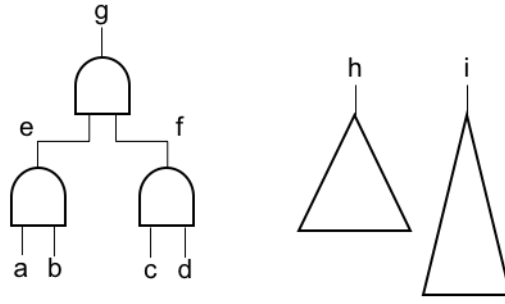


Fig. 9. Example of (constrained) cut logic.

$$CL_{(\vec{l})}^o(X) \equiv CL_{(\vec{l}')}^{o'}(X) \quad (6)$$

The algorithm of output-side frontier identification is shown in Fig. 10. In the beginning of the algorithm, we include all the PO pairs into the *RP_Set* since they are trivial rectification pairs (line 4). After that, we iteratively invoke simulation-guided cut-matching algorithm on the existing rectification pairs to discover more rectification pairs (line 7). The simulation-guided cut-matching algorithm is applied on the rectification pair $g \leftrightarrow g'$. When there are cuts matched in the matching process, we will obtain a set of new matched pairs on the returned matched cuts *Cut_Pair*, we then include the functionally non-equivalent pairs among these matched pairs into the list of the rectification pairs.

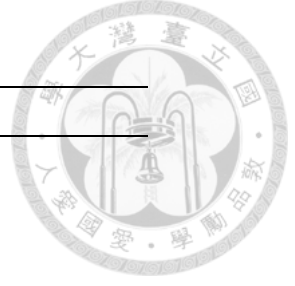
Since we cannot guarantee the *Cut_Pair* generated from the simulation-guided cut-matching algorithm is disjoint from *RP_Set*, we don't remove the original rectification pair as stated in *Lemma 1* and optimize the *RP_Set* in the replacing phase.

We iteratively explore new pairs in the rectification pair list. After we traverse all the rectification pairs and no new matches are derived by the matching process, the identification process terminates and the pairs in the *RP_Set* is the rectification pairs of two circuits.

The cut-matching algorithm is a process to match two cuts in the revised and golden circuits so that the corresponding cut logics are equivalent.

Note that by (6), we ignore the circuit constraint below the pseudo PIs (\vec{l}) , which will increase





Algorithm Output-side Frontier Identification Algorithm

```

1: Input: Gold_Ckt, Rev_Ckt
2: Output: RP_Set
3: for each pair(po, po') in POs do
4:   RP_Set  $\leftarrow$  RP_Set + pair(po, po')
5: end for
6: for each pair(g, g') in RP_Set do
7:   Cut_Pair  $\leftarrow$  Simulation_Guided_Cut_Matching(g, g', Gold_Ckt, Rev_Ckt)
8:   for each pair(n, n') in Cut_Pair do
9:     if n == n' then continue
10:    RP_Set  $\leftarrow$  RP_Set + pair(n, n')
11:   end for
12: end for
  
```

Fig. 10. Output-side frontier identification algorithm.

the difficulty of matching the cut since it is an over-approximation matching. We define the constrained cut logic to increase the matching probability.

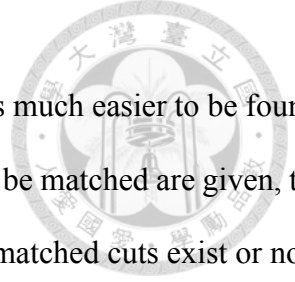
Definition 4: A constrained cut logic $CCL_{(i_1, i_2, \dots, i_n) \xleftarrow{\text{care}} (i'_1, i'_2, \dots, i'_n)}^o$ is a logic cone that represents the functionality of the output *o* with respect to (i_1, i_2, \dots, i_n) , having the care set of the functional circuit constraint of $(i'_1, i'_2, \dots, i'_n)$. Fig. 9 shows an example. The constrained cut logic of *g* with respect to (e, f) , having functional circuit constraint of (h, i) , is $CCL_{(e, f) \xleftarrow{\text{care}} (h, i)}^g(X_1, X_2) = (X_1 \equiv h)(X_2 \equiv i)$.

With the above definition, we can perform cut-matching algorithm under constrained cut logic:

Definition 5: Given two output (o, o') and two cuts $\{\vec{g}: (g_1, g_2, \dots, g_n), \vec{g}': (g'_1, g'_2, \dots, g'_n)\}$ in the revised and golden circuit respectively, the cut-matching algorithm is a process to verify two cuts which:

$$CCL_{(\vec{g}) \xleftarrow{\text{care}} (\vec{g}')}^o(X) \equiv CCL_{(\vec{g}') \xleftarrow{\text{care}} (\vec{g})}^{o'}(X) \quad (7)$$

We call \vec{g} and \vec{g}' two matched cuts.



Note that (7) fits the patch replacement scenario, so the matched cut is much easier to be found.

Different from Boolean matching where the supports of the circuits to be matched are given, the cut-matching process in our algorithm is required to examine whether the matched cuts exist or not.

M&R randomly enumerates a cut from a matched gate g in the revised circuit, candidate gates are then selected gradually according to the matched gate g' and the selected cut of g , a matching matrix is constructed to find the corresponding cut. As stated before, the matching process is difficult, and the quality of the pairing on the matched cut might not be good, resulting in large patch size.

Since the modified region in the circuit is usually small, we observe that a portion of both circuits remains functionally equivalent, i.e. we can discover many merged gates in the *FRAIG* process.

We exploit those merged gates to get a *Guided_Cut_Pair* (line 5 in Fig. 13), the algorithm is presented in Fig. 11. For gates g and g' , we find duplicated merged gates within a certain level, then we construct the cut according to those merged gates (line 3-4). If the *Guided_Cut_Pair* has different cut size, we will perform possible expansion on the smaller cut. After we equalize the cut size, we invoke simulation and calculate *FSCs*, the pairing of the cut is then decided based on these results (line 5). The *checkSize* will be recorded for further decision of cuts.

In sweeping-based ECO algorithms, one mismatch will limit the finding of further cut matches on the following subcircuits and thereby increase the size of the patch circuit. DeltaSyn [14] considers all the possible combinations to prevent mismatching. However, it would take much time to perform all the matching combinations and thus slow down the whole process. M&R [18] arbitrarily select a set of feasible matches to generate a new set of rectification pairs when two nodes are functionally symmetric. This is a trade-off between the quality of the pairing and runtime. We propose a simulation-guided pairing to get a promising pairing in feasible runtime.

For those non-duplicated merged gates in the cut, we will first perform simulation and extract their response. After performing the calculation of pairwise cosine similarity, *FSCs* of each non-



Algorithm Get Guided Cut Candidates Algorithm

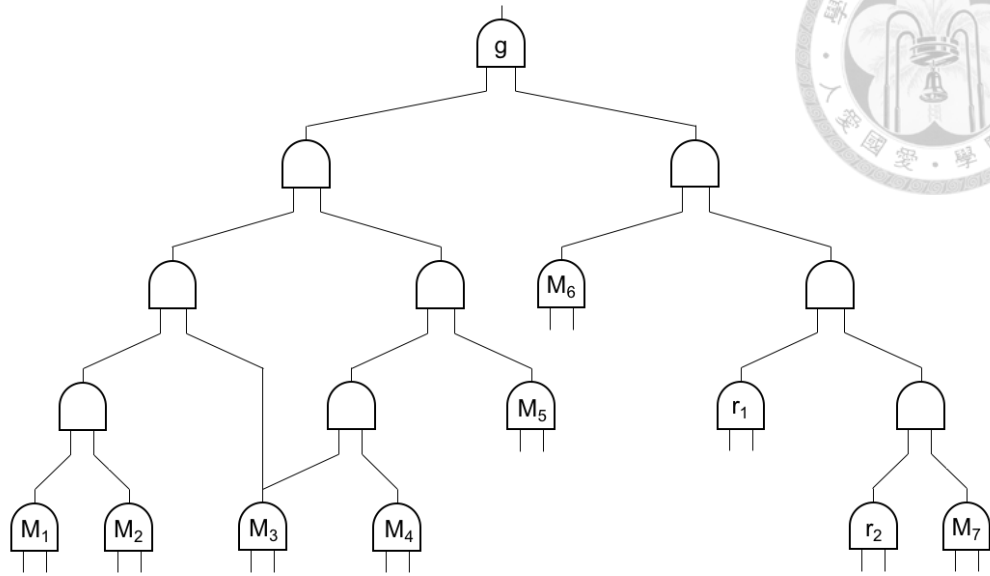
```
1: Input:  $g, g', Gold\_Ckt, Rev\_Ckt, level$ 
2: Output:  $Cut\_Cand, checkSize$ 
3:  $merged\_Lst \leftarrow Get\_Dup\_Merged(g, g', Gold\_Ckt, Rev\_Ckt, level)$ 
4:  $cand\_Lst, cSize\_Lst \leftarrow Constraint\_Cut(g, g, Gold\_Ckt, Rev\_Ckt, merged\_Lst)$ 
5:  $cand\_Lst \leftarrow Reorder\_By\_CktSimilarity(cand\_Lst)$ 
6: for each ( $cand\ c, cSize\ s$ ) in ( $cand\_Lst, cSize\_Lst$ ) do
7:    $Cut\_Cand \leftarrow Cut\_Cand + (merged\_Lst, c)$ 
8:    $checkSize \leftarrow checkSize + s$ 
9: end for
```

Fig. 11. Get guided-cut candidates algorithm.

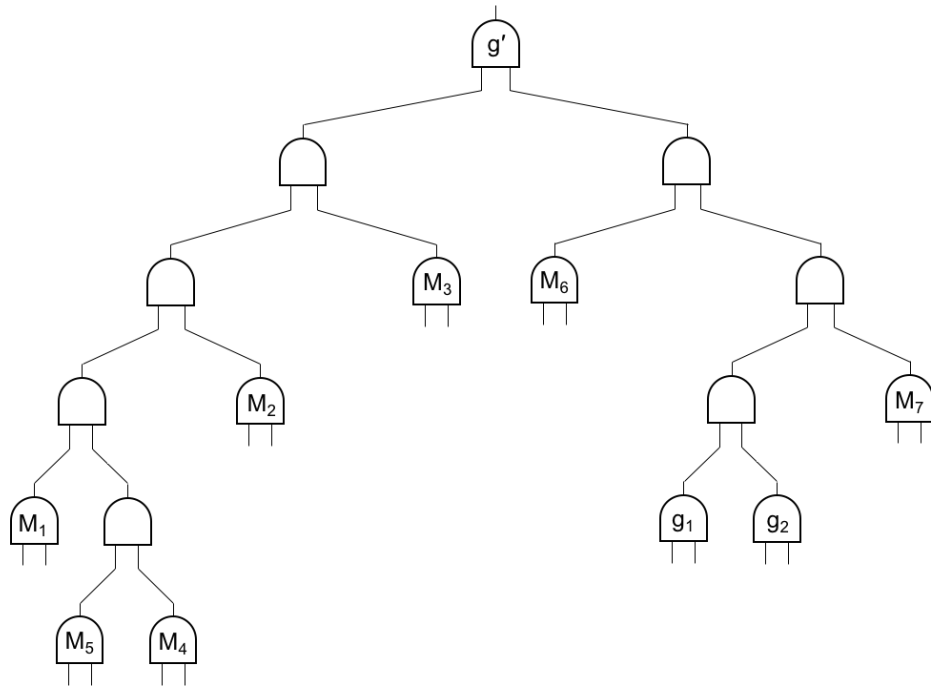
duplicated merged gates are collected. The pairing is then decided by the calculated cosine similarity (line 5). To make the pairing more promising, not only the functional circuit similarity should be considered, but also the structural information, which is the duplicated merged gates in the subcircuits under the pairing, since the duplicated merged gates represent a guidance in the cut-matching process. The functional circuit similarity and the number of duplicated merged gates beneath subcircuits of each possible pairing is calculated and scored by a pre-defined weight, then the guided-pairing is decided by the ranking of the scores.

A simple example is illustrated in Fig. 12. For a certain level of 5, merged gates $M_1 - M_7$ are found in both subcircuits of g and g' . To form a cut, (r_1, r_2) and (g_1, g_2) are added to the $cand_Lst$, and the corresponding $checkSize$ of this cut is 2. By considering the circuit similarity and the structural information, we pair (r_1, g_2) and (r_2, g_1) as they are the more promising pairing.

The flow of the simulation-guided cut-matching algorithm is shown in Fig. 13. To maximize the use of the merged gates, we will gradually increase the level to find more duplicated merged gates (line 5, 15), different location of the duplicated merged gates will result in different cut size and the corresponding $checkSize$. The more duplicated merged gates we find; the more guidance of the cut we will get. Therefore, we tend to pick the cut with the smallest $checkSize$, which is the number of



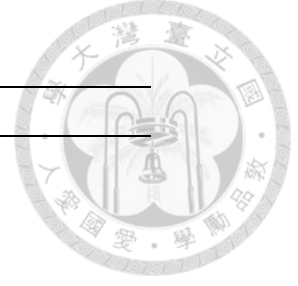
(a)



(b)

Fig. 12. Example of cut generation w.r.t. duplicated merged gates. (a) Revised. (b) Golden.

the non-duplicated merged gates in the cut. Also, the cuts to be checked have a higher chance to be equivalent under *constrained cut logic* if the *checkSize* is smaller since they have more guidance, i.e. less uncertainty. Note that if we get the same *checkSize* in different level, we will pick the cut generated in the deeper level, since it is usually a cut with bigger cut size (line 6-14).



Algorithm Simulation-Guided Cut-Matching Algorithm

```

1:  Input:  $g, g', Gold\_Ckt, Rev\_Ckt$ 
2:  Output:  $Cut\_Pair$ 
3:   $level \leftarrow lv\_Init, checkSize\_Min \leftarrow INT\_MAX$ 
4:  while  $level < lv\_Thre$  do
5:     $Cut\_Cand, checkSize \leftarrow Get\_Guided\_Cut\_Cands(g, g', Gold\_Ckt,$ 
                                      $Rev\_Ckt, level)$ 
6:    Sort ( $Cut\_Cand, checkSize$ ) according to  $cSize$ 
7:    for each ( $cut\ c, cSize\ s$ ) in ( $Cut\_Cand, checkSize$ ) do
8:      if  $s > checkSize\_Min$  then break
9:      if  $Check\_Cut\_Func\_Under\_CCL(g, g', c, Gold\_Ckt, Rev\_Ckt)$  then
10:         $checkSize\_Min \leftarrow s$ 
11:         $Cut\_Pair \leftarrow c$ 
12:        break
13:      end if
14:    end for
15:     $level \leftarrow level + 1$ 
16: end while
  
```

Fig. 13. Simulation-guided cut-matching algorithm.

3.3 Replacing Phase – Patch Region Optimization

After identifying the *patch region*, a rectification pair selector is proposed to determine the acceptances of the rectification pairs, since some rectification pairs may be redundant if their functional errors can be covered by other rectification pairs. There are two principal objectives in the rectification pair selection process: 1) to minimize the final patch circuit, and 2) to guarantee all output functions are correct after the rectification.

The rectification pair selector $RPS(\vec{X}, s_1, s_2, \dots, s_n)$ is a miter as shown in Fig. 14. We include n 2-to-1 multiplexers $MUX(s, a, b)$ to connect the matched rectification pairs $\{g_1 \leftrightarrow g'_1, g_2 \leftrightarrow g'_2, \dots, g_n \leftrightarrow g'_n\}$. For each pair $g_i \leftrightarrow g'_i$, we insert a $MUX(s_i, g_i, g'_i)$ on the output of g_i . g_i and g'_i are connected by the inputs of the MUX ; the original fanouts of g_i is driven by the outputs of the

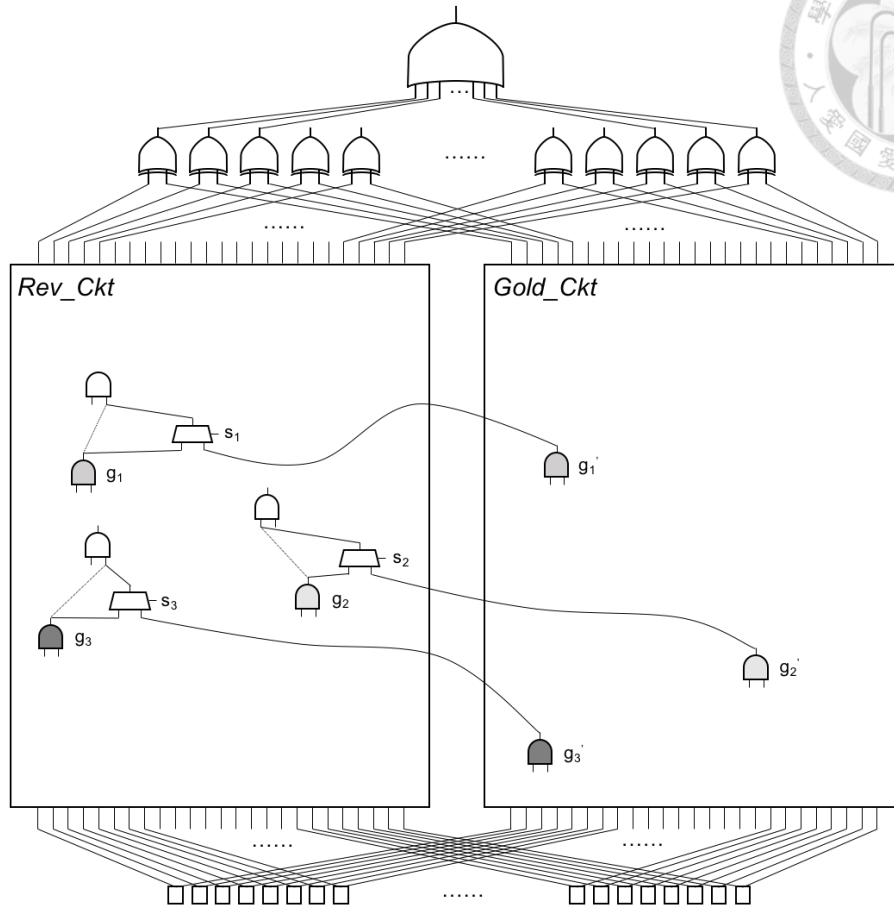


Fig. 14. Rectification pair selector.

MUX ; the selection signal s_i becomes a primary input of the miter.

The assignment of \vec{s} represents the selection of the patch logic. When we assign 1 to $s_i \in \vec{s}$, the fanout of g_i in the revised circuit is driven by g'_i . That is, this specific patch is committed. On the other hand, when s_i is assigned 0, g_i in the revised circuit would not be replaced and keep its original functionalities.

Since we exploit merged gates in the simulation-guided cut-matching process, we would have to insert MUX on a rectification pair $m_j \leftrightarrow g_j$, where m_j is a merged gate. If we do the above connection on the merged gate, we would change the functionality of the golden circuit, which is not allowed in the ECO process. To cope with this problem, we identify a boundary on the fanout cone of the merged gate, which is excluded from merged gates on the boundary, we then duplicate gates

within the boundary and make a special connection on them. Fig. 15 shows an example, if a $MUX(s, M_1, g')$ is to insert on a rectification pair $M_1 \leftrightarrow g'$ by the above connection, the functionality of G_1 and G_2 will be modified if s is assigned 1. After we identify the boundary and duplicate the gates within the boundary, the functionality of G_1 and G_2 is now independent from the newly-duplicated gate D_1 . Inserting MUX on the newly-duplicated gate D_1 not only meet our needs of the RPS , but also retain the functionality of the golden circuit.

The patch selection process can then be formulated as a QBF

$$\exists \vec{s}, \forall \vec{X}, RPS(\vec{X}, \vec{s}) \equiv 0 \quad (8)$$

By the rectification pair selector, we can derive a valid patch by solving the QBF (8). However, exploring the minimal patch circuit, which is an optimization problem, on the QBF is very inefficient. Consequently, we propose a feasible rectification pair selection algorithm to get a quality solution according to the characteristics of the rectification pairs.

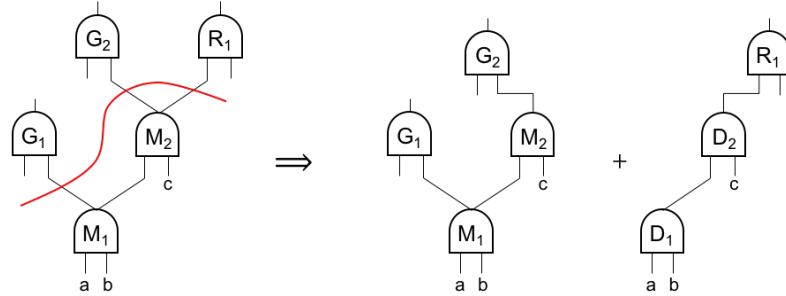


Fig. 15. Example of inserting multiplexer on a rectification pair consists of a merged gate.

The rectification pair selection algorithm is shown in Fig. 16. In the beginning of the flow, we replace all patched gates g_i with g'_i , i.e. $\vec{s} = (1, 1, \dots, 1)$ (line 3-5). Note that the revised circuit must be equivalent to the golden under this assignment since all POs are replaced by the golden. We then attempt to undo a rectification pair $g_j \leftrightarrow g'_j$ by assigning 0 to s_j iteratively. Whenever we undo a rectification pair, we perform the equivalence checking by the miter. If the miter is unsatisfiable, the functional errors are covered by other rectification pairs, and thus the rectification

pair can be discarded. Otherwise, we restore s_j to 1, since the replacement is necessary for the rectification (line 8-11). In our observation, the lower level of the rectification pairs we select, the smaller size of the patch circuit can be generated since they are closer to the input-side merging frontier. Therefore, we undo the rectification pair in a top-down traversal order (line 6). The RP_Set_Opt is the final *patch region* to rectify the revised circuit.

Algorithm Rectification Pair Selection Algorithm

```

1:  Input:  $RP\_Set, Miter$ 
2:  Output:  $RP\_Set\_Opt$ 
3:  for each  $s_i$  in  $\vec{s}$  of  $Miter$  do
4:       $s_i \leftarrow 1$ 
5:  end for
6:  Sort  $RP\_Set$  in a top-down order
7:  for each  $\{pair(g, g'), s_i\}$  in  $\{RP\_Set, \vec{s} \text{ of } Miter\}$  do
8:       $s_i \leftarrow 0$ 
9:      if  $Miter.solve() = SAT$  then
10:          $RP\_Set\_Opt \leftarrow RP\_Set\_Opt + pair(g, g')$ 
11:          $s_i \leftarrow 1$ 
12:      end if
13:  end for

```

Fig. 16. Rectification pair selection Algorithm.



Chapter 4 Experimental Results

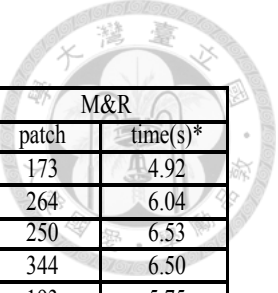
We implement our ECO engine in C++ language and apply MiniSAT [23] as our SAT engine. All of our experiments are conducted on a Linux workstation with 32 GB RAM and 3.0 GHz Intel Core i7 CPU. The correctness of the experiments was verified by the ABC command *cec*.

Table 1 shows our experimental results. The first column shows the name of the testcases and the number of gates. 4 of the testcases are from the benchmark of iwls2017 programming contest [24], the other 2 testcases are from EPFL benchmark [25]. The second and third column shows the number of primary inputs/outputs and the max/average level of the circuits. The fourth column shows the number of three modifications to the circuits. “RW” means gate rewiring. We choose two gates *g* and *h*, and randomly pick one of *g*’s fanin to reconnect the wire to *h*. We will skip the rewiring causing combination loop. “INV” means inverter insertion. An inverter is applied and the output function of the selected gate is inverted. “TC” means gate type change. We change the type of the selected gate, which is an AND gate in our framework, to an XOR gate. Table 2 shows the information of our testcases and illustrates the modification on them. The remaining column shows the final patch size and the runtime of our ECO engine and M&R.

We can see that changing the gate type is usually more difficult than inserting an inverter, since changing an AND gate to an XOR gate will naturally involve the inverter insertion process. Also, the runtime of our ECO engine mainly depends on the level of the circuit, as well as the location of the buggy gate, since the backward cut-matching algorithm is performed in a level manner.

Note that the size in the fifth column is an upper bound of the patch size, since we do not do any optimization on the patch logic, i.e. the newly duplicated gates in the replacing phase. We can get an even smaller patch if we perform some optimization technique on the patch logic, e.g. the floating gate recycling technique proposed in [18].

The experimental results show that our ECO engine can rectify circuits with small patch size

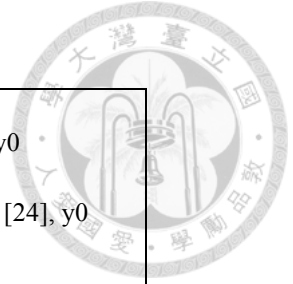


| | pi/po | max/avg lv | | | | Semi-Formal | | M&R | |
|---------------|---------|------------|----|-----|----|-------------|---------|-------|----------|
| | | | RW | INV | TC | patch | time(s) | patch | time(s)* |
| y0 (8053) | 128/147 | 25/17 | 5 | 0 | 0 | 24 | 34.62 | 173 | 4.92 |
| | | | 6 | 0 | 0 | 59 | 29.67 | 264 | 6.04 |
| | | | 2 | 0 | 2 | 50 | 27.90 | 250 | 6.53 |
| | | | 2 | 2 | 0 | 46 | 50.26 | 344 | 6.50 |
| | | | 2 | 2 | 2 | 104 | 42.54 | 193 | 5.75 |
| y1 (5326) | 128/94 | 25/17 | 5 | 0 | 0 | 5 | 23.67 | 379 | 8.03 |
| | | | 6 | 0 | 0 | 39 | 22.56 | 345 | 5.27 |
| | | | 2 | 0 | 2 | 5 | 12.02 | 113 | 2.77 |
| | | | 2 | 2 | 0 | 14 | 12.77 | 178 | 3.31 |
| | | | 2 | 2 | 2 | 45 | 28.56 | 273 | 5.15 |
| y2 (1415) | 207/108 | 48/15 | 10 | 0 | 0 | 19 | 9.20 | 85 | 1.61 |
| | | | 10 | 0 | 2 | 59 | 11.08 | 86 | 1.63 |
| | | | 10 | 2 | 0 | 38 | 4.45 | 39 | 1.63 |
| | | | 10 | 2 | 2 | 79 | 5.77 | 93 | 2.17 |
| y3 (2819) | 512/130 | 227/225 | 5 | 0 | 0 | 58 | 336.10 | 1372 | 20.23 |
| | | | 6 | 0 | 0 | 70 | 513.38 | 1471 | 366.04 |
| | | | 2 | 0 | 2 | 18 | 978.50 | 1603 | 374.79 |
| | | | 2 | 2 | 0 | 34 | 166.25 | 1462 | 27.01 |
| | | | 2 | 2 | 2 | 58 | 698.04 | 1312 | 315.77 |
| y4 (11839) | 256/129 | 88/87 | 5 | 0 | 0 | 0 | 38.39 | 212 | 3.19 |
| | | | 6 | 0 | 0 | 25 | 37.84 | 283 | 3.34 |
| | | | 2 | 0 | 2 | 2 | 38.43 | 154 | 3.30 |
| | | | 2 | 2 | 0 | 0 | 38.07 | 214 | 3.03 |
| | | | 2 | 2 | 2 | 6 | 38.49 | 307 | 3.47 |
| y5 (1147) | 147/141 | 14/7 | 10 | 0 | 0 | 14 | 1.93 | 54 | 0.42 |
| | | | 10 | 0 | 5 | 45 | 3.26 | 101 | 0.55 |
| | | | 10 | 5 | 0 | 31 | 4.19 | 135 | 0.57 |
| | | | 10 | 5 | 5 | 46 | 2.83 | 98 | 0.49 |
| | | | | | | 993 | 3210.79 | 11593 | 1183.51 |

Table 1. Performance comparison on various modifications.

within reasonable runtime. Note that the runtime between two engines is not a fair comparison. The result of M&R is obtained in a much more powerful machine (Intel Xeon E5 CPU @ 2.70GHz with 400 GB RAM) due to some technical issues. We believe that we can obtain a close or even a better runtime on the same machine.

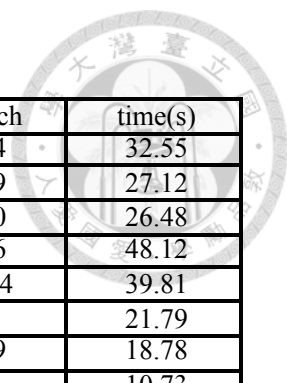
In the simulation-guided cut-matching algorithm, we update the cut if the *checkSize* is smaller or equal to the current cut, since we have more guidance on the cut if the *checkSize* is smaller. That is, cuts having bigger *checkSize* will always be abandoned even though they have equivalent *constrained cut logic*. To increase the performance on runtime, we ignore the cuts having bigger *checkSize* in *Constraint_Cut* (line 4 in Fig. 11). The ignored cuts will not go through *Reorder_By_*



| | |
|--|--|
| | Name: y0 Source: [24], y0 |
| | Name: y1 Source: [24], y1 |
| | Name: y2 Source: [24], y2 |
| | Name: y3 Source: [24], y3 |
| | Name: y4 Source: [25], round_robin_arbiter |
| | Name: y5 Source: [25], i2c |

Table 2. Modifications on the testcases.

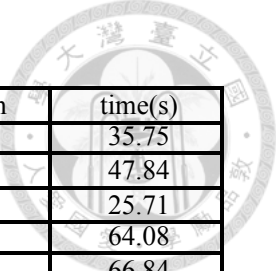
CktSimilarity (line 5 in Fig. 11) and *Check_Cut_Func_Under_CCL* (line 9 in Fig. 13), which will save some time. Table 3 shows the overall improvement, every case takes advantage of this method, and result in a 29.1% improvement in average on runtime.



| | pi/po | max/avg lv | RW | INV | TC | patch | time(s) |
|---------------|---------|------------|----|-----|----|-------|---------|
| y0 (8053) | 128/147 | 25/17 | 5 | 0 | 0 | 24 | 32.55 |
| | | | 6 | 0 | 0 | 59 | 27.12 |
| | | | 2 | 0 | 2 | 50 | 26.48 |
| | | | 2 | 2 | 0 | 46 | 48.12 |
| | | | 2 | 2 | 2 | 104 | 39.81 |
| y1 (5326) | 128/94 | 25/17 | 5 | 0 | 0 | 5 | 21.79 |
| | | | 6 | 0 | 0 | 39 | 18.78 |
| | | | 2 | 0 | 2 | 5 | 10.73 |
| | | | 2 | 2 | 0 | 14 | 12.79 |
| | | | 2 | 2 | 2 | 45 | 26.95 |
| y2 (1415) | 207/108 | 48/15 | 10 | 0 | 0 | 19 | 6.40 |
| | | | 10 | 0 | 2 | 59 | 8.60 |
| | | | 10 | 2 | 0 | 38 | 3.98 |
| | | | 10 | 2 | 2 | 79 | 4.92 |
| y3 (2819) | 512/130 | 227/225 | 5 | 0 | 0 | 58 | 302.09 |
| | | | 6 | 0 | 0 | 70 | 474.96 |
| | | | 2 | 0 | 2 | 18 | 562.93 |
| | | | 2 | 2 | 0 | 34 | 148.45 |
| | | | 2 | 2 | 2 | 58 | 295.98 |
| y4 (11839) | 256/129 | 88/87 | 5 | 0 | 0 | 0 | 38.38 |
| | | | 6 | 0 | 0 | 25 | 37.84 |
| | | | 2 | 0 | 2 | 2 | 38.51 |
| | | | 2 | 2 | 0 | 0 | 38.25 |
| | | | 2 | 2 | 2 | 6 | 38.67 |
| y5 (1147) | 147/141 | 14/7 | 10 | 0 | 0 | 14 | 1.62 |
| | | | 10 | 0 | 5 | 45 | 3.20 |
| | | | 10 | 5 | 0 | 31 | 3.71 |
| | | | 10 | 5 | 5 | 46 | 2.69 |
| | | | | | | 993 | 3210.79 |

Table 3. Performance comparison under checkSizeEarlyReturn.

As stated above, duplicated merged gates represent a guidance while matching the cut. We first extract duplicated merged gates within a certain level, then construct the cut according to those gates. Note that if the quality of the previous pairings on the matched cuts are not good as expected, or the level to extract is not deep enough, we might extract no duplicated merged gates. If no duplicated merged gates are found in this level, we will skip this level and continue to search for deeper levels. We can extract the whole leaves within this certain level if no duplicated merged gates are found, which increases the probability of matching the cut, hence increase the size of the set of rectification pairs. However, the quality of the rectification pair is unstable, and we will have much more rectification pairs to perform simulation-guided cut-matching algorithm, resulting in higher runtime.



| | pi/po | max/avg lv | RW | INV | TC | patch | time(s) |
|---------------|---------|------------|----|-----|----|-------|---------|
| y0 (8053) | 128/147 | 25/17 | 5 | 0 | 0 | 24 | 35.75 |
| | | | 6 | 0 | 0 | 59 | 47.84 |
| | | | 2 | 0 | 2 | 50 | 25.71 |
| | | | 2 | 2 | 0 | 46 | 64.08 |
| | | | 2 | 2 | 2 | 104 | 66.84 |
| y1 (5326) | 128/94 | 25/17 | 5 | 0 | 0 | 6 | 35.70 |
| | | | 6 | 0 | 0 | 39 | 21.41 |
| | | | 2 | 0 | 2 | 5 | 10.64 |
| | | | 2 | 2 | 0 | 14 | 13.15 |
| | | | 2 | 2 | 2 | 45 | 37.20 |
| y2 (1415) | 207/108 | 48/15 | 10 | 0 | 0 | 220 | 10.54 |
| | | | 10 | 0 | 2 | 204 | 11.77 |
| | | | 10 | 2 | 0 | 72 | 7.29 |
| | | | 10 | 2 | 2 | 107 | 6.14 |
| y3 (2819) | 512/130 | 227/225 | 5 | 0 | 0 | * | >1800 |
| | | | 6 | 0 | 0 | * | >1800 |
| | | | 2 | 0 | 2 | 18 | 645.67 |
| | | | 2 | 2 | 0 | 34 | 191.36 |
| | | | 2 | 2 | 2 | * | >1800 |
| y4 (11839) | 256/129 | 88/87 | 5 | 0 | 0 | 0 | 38.05 |
| | | | 6 | 0 | 0 | 25 | 37.59 |
| | | | 2 | 0 | 2 | 2 | 38.45 |
| | | | 2 | 2 | 0 | 30 | 38.74 |
| | | | 2 | 2 | 2 | 6 | 38.55 |
| y5 (1147) | 147/141 | 14/7 | 10 | 0 | 0 | 14 | 1.22 |
| | | | 10 | 0 | 5 | 45 | 3.47 |
| | | | 10 | 5 | 0 | 42 | 4.01 |
| | | | 10 | 5 | 5 | 46 | 3.24 |
| | | | | | | 1257 | 6834.44 |

Table 4. Performance comparison under `checkSizeEarlyReturn` and `doExtractLeaves`.

Table 4 shows the experimental results. We can see that no cases take advantage from this heuristic, since the pairing is unstable. But we can expect that there should be some cases other than these can take advantage from this heuristic and get smaller patch size. 3 cases from y3 exceed time limit (1800s). The average runtime increases by 112.86%, and the patch size increases by 26.59%.

To maximize the use of the merged gates, we will gradually increase the level to get the matched cuts, then the cut with the smallest *checkSize* and the deepest level will be the final matched cut. We can take the first matched cut as the final matched cut to improve the runtime, but might sacrifice the patch size, since the first matched cut might not have a small *checkSize*. The number of the duplicated merged gates in the first matched cut will also be small due to the shallow level. Table 5 shows the



experimental results. Some cases get smaller patches and some do not. Although the average runtime has a 9.17% improvement, the patch size increases 146.73% in average.

| | pi/po | max/avg lv | RW | INV | TC | patch | time(s) |
|---------------|---------|------------|----|-----|----|-------|---------|
| y0 (8053) | 128/147 | 25/17 | 5 | 0 | 0 | 94 | 30.68 |
| | | | 6 | 0 | 0 | 184 | 34.39 |
| | | | 2 | 0 | 2 | 72 | 29.05 |
| | | | 2 | 2 | 0 | 58 | 49.96 |
| | | | 2 | 2 | 2 | 91 | 40.93 |
| y1 (5326) | 128/94 | 25/17 | 5 | 0 | 0 | 173 | 26.32 |
| | | | 6 | 0 | 0 | 299 | 24.00 |
| | | | 2 | 0 | 2 | 77 | 17.52 |
| | | | 2 | 2 | 0 | 55 | 14.38 |
| | | | 2 | 2 | 2 | 97 | 26.59 |
| y2 (1415) | 207/108 | 48/15 | 10 | 0 | 0 | 280 | 7.27 |
| | | | 10 | 0 | 2 | 96 | 7.81 |
| | | | 10 | 2 | 0 | 108 | 3.17 |
| | | | 10 | 2 | 2 | 61 | 3.94 |
| y3 (2819) | 512/130 | 227/225 | 5 | 0 | 0 | 158 | 245.55 |
| | | | 6 | 0 | 0 | 142 | 579.90 |
| | | | 2 | 0 | 2 | 51 | 1211.16 |
| | | | 2 | 2 | 0 | 71 | 123.16 |
| | | | 2 | 2 | 2 | 89 | 237.17 |
| y4 (11839) | 256/129 | 88/87 | 5 | 0 | 0 | 0 | 38.29 |
| | | | 6 | 0 | 0 | 25 | 37.96 |
| | | | 2 | 0 | 2 | 3 | 38.91 |
| | | | 2 | 2 | 0 | 0 | 37.77 |
| | | | 2 | 2 | 2 | 9 | 38.29 |
| y5 (1147) | 147/141 | 14/7 | 10 | 0 | 0 | 29 | 2.35 |
| | | | 10 | 0 | 5 | 45 | 3.03 |
| | | | 10 | 5 | 0 | 31 | 3.88 |
| | | | 10 | 5 | 5 | 52 | 2.93 |
| | | | | | | 2450 | 2916.35 |

Table 5. Performance comparison under checkSizeEarlyReturn and globalLevelEarlyReturn.

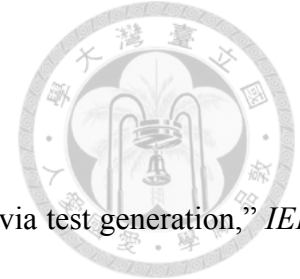
Chapter 5 Conclusions and Future Work



We propose a two-phase semi-formal ECO method. In the matching phase, the *FRAIG* technique followed by the simulation-guided cut-matching algorithm identifies the *patch region*. The *patch region* is then optimized in the replacing phase by a rectification pair selector combined with a linear-time heuristic. The experimental results show that our ECO engine can rectify multiple errors with small patch size within reasonable runtime.

For future work, we plan to design a data structure, which can efficiently store and access a very-long bit-sequence, so that we can make use of this data structure to exploit more simulation-based heuristics to get more better-quality rectification pairs. Besides, a QBF solving algorithm is planned for selecting better rectification pairs from *RP_Set* to get a smaller patch circuit.

Reference



- [1] M. Abadir, J. Ferguson, and T. Kirkland, "Logic design verification via test generation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 7, no. 1, pp. 138–148, Jan. 1988.
- [2] P.-Y. Chung and I. Hajj, "Accord: Automatic catching and correction of logic design errors in combinational circuits," in *Proc. Int. ITC*, Sep. 1992, pp. 742–751.
- [3] A. Veneris and I. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1803–1816, Dec. 1999.
- [4] C.-C. Lin, K.-C. Chen, and M. Marek-Sadowska, "Logic synthesis for engineering change," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 3, pp. 282–292, Mar. 1999.
- [5] Y.-S. Yang, S. Sinha, A. Veneris, and R. Brayton, "Automating logic rectification by approximate SPFDs," in *Proc. ASP-DAC*, Jan. 2007, pp. 402–407.
- [6] A. Ling, S. Brown, S. Safarpour, and J. Zhu, "Toward automated ECOs in FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 1, pp. 18–30, Jan. 2011.
- [7] B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. Jiang, "A robust functional ECO engine by SAT proof minimization and interpolation techniques," in *Proc. IEEE/ACM Int. Conf. ICCAD*, Nov. 2010, pp. 729–734.
- [8] K. H. Chang, I. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 184–188, Jan. 2008.
- [9] K.-F. Tang, C.-A. Wu, P.-K. Huang, and C.-Y. Huang, "Interpolation-based incremental ECO synthesis for multi-error logic rectification," in *Proc. 48th ACM/EDAC/IEEE DAC*, Jun. 2011, pp. 146–151.
- [10] K.-F. Tang, P.-K. Huang, C.-N. Chou and C.-Y. Huang, "Multi-patch generation for multi-error logic rectification by interpolation with cofactor reduction," *Design, Automation & Test in*



- Europe Conference & Exhibition (DATE)*, 2012, pp. 1567-1572.
- [11] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng, "Autofix: A hybrid tool for automatic logic rectification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1376–1384, Sep. 1999.
- [12] D. Hoffmann and T. Kropf, "Efficient design error correction of digital circuits," in *Proc. ICCD*, Sep. 2000, pp. 465–472.
- [13] D. Brand, "Incremental synthesis," in *Proc. IEEE/ACM ICCAD*, Nov. 1994, pp. 14–18.
- [14] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," in *Proc. IEEE/ACM ICCAD*, Nov. 2009, pp. 789–796.
- [15] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. 34th ACM/IEEE DAC*, Jun. 1997, pp. 263–268.
- [16] A. Mishchenko, S. Chatterjee, and R. Brayton, "Fraigs: A unifying representation for logic synthesis and verification," *EECS Dept., UC Berkeley, Tech. Rep.*, 2005.
- [17] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't-cares," in *Proc. 43rd ACM/IEEE DAC*, Jul. 2006, pp. 229–234.
- [18] S.-L. Huang, W.-H. Lin, P.-K. Huang and C.-Y. Huang, "Match and replace: A functional ECO engine for multi-error circuit rectification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no. 3, pp. 467-478, March 2013.
- [19] D. Brand, "Verification of large synthesized designs," in *Proc. IEEE/ACM ICCAD*, Nov. 1993, pp. 534–537.
- [20] J. Cong and Y.-Y. Hwang, "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology map- ping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1077–1090, Sep. 2001.
- [21] https://en.wikipedia.org/wiki/Cosine_similarity



- [22] <http://people.eecs.berkeley.edu/~alanmi/abc/>
- [23] N. Sořrensson and N. Ee', "Minisat v1. 13: A SAT solver with conflict- clause minimization," in *Proc. SAT*, 2005, pp. 53–54.
- [24] <https://github.com/msoeken/iwls2017-contest>
- [25] L.Amaru', P.-E.Gaillardon, and G.DeMicheli, "The epfl combinational benchmark suite," in *Proc. 24th International Workshop on Logic & Synthesis (IWLS)*, no. EPFL-CONF-207551, 2015.