Match and Replace — A Functional ECO Engine for Multi-Error Circuit Rectification

Shao-Lun Huang†, Wei-Hsun Lin‡, Chung-Yang (Ric) Huang†‡
† Department of Electrical Engineering / ‡ Graduate Institute of Electronics Engineering
National Taiwan University, Taiwan

***Abstract*—Functional ECO has been an indispensible technique in modern VLSI design flow. This paper proposes an ECO engine in a two-phase approach: a matching phase for rectification pair identification and a replacement phase for pair selection and patch minimization. The rectification pair identification algorithm explores rectification pairs between the original circuit and the golden circuit. The rectification pair selector determines final patches through a linear-time heuristic. A gate-recycle process performs patch minimization for final refinement. The experiments show that this ECO engine outperforms a state-of-the-art interpolation-based engine in both patch quality and runtime.**

I. Introduction

In modern VLSI design process, statistics in [1] points out that over 60% of design failures are due to functional errors. What is worse, more than half of the designs have encountered various sorts of functional problems even after design tape-out or chip fabrication. This in general implies a gigantic monetary or human resource cost in reclaiming or redesigning the chips. Therefore, to avoid the loss caused by the design re-spin, an engineering change order (ECO) approach that aims at rectifying the design problems after synthesis and optimization has become an indispensible technique.

There have been many researches focusing on the functional ECO problem in recent years [2] – [9]. They adopt BDD[2], SPFD[3], SAT[4], error model[5], hybrid method[6][8], and many techniques[7][9] to automatically repair functional faults in circuits. Most of them such as [4][7][8] rely on diagnosis strategies to identify an internal rectification signal and 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 require a single rectification signal. While in many cases the only possible single rectification signal is the primary output itself, the reported patches may contain a good portion of redundant logic or even be larger than the target golden design.

DeltaSyn in [9] proposes a sweeping-based functional ECO algorithm. The authors introduce a dual-phase flow to first identify the input-side and output-side boundaries of the changes and then collect the rest netlist between two boundaries as the gates to be replaced. Since in practice the functional corrections in RTL are often corresponding to small and local changes, this sweeping-based method is a very reasonable algorithm for functional ECOs. Nevertheless, there are still challenges in reality: finding the output-side matching boundary is very difficult. The computational complexity of output-side matching algorithm, which is cut-based Boolean matching introduced in [9], situates in between co-NP and $\sum_2^p$ in the polynomial hierarchy [10]. Thus, the size of the boundary is always limited to a small number. What is worse, when functional symmetry occurs, it takes much more time to enumerate all permutations for the best matches.

In this paper, we propose a "match-and-replace" algorithm for the multi-error functional ECO problem. As illustrated in Fig. 1, our algorithm is a two-phase approach: (1) Matching phase: to maximize the matches between the original and golden circuits in order to identify the minimal regions for rectification; and (2) Replacement phase: for each mismatch in (1), generate a minimal patch logic to replace the corresponding cone in the original circuit. Different from DeltaSyn[9], we enhance the SAT-based Boolean matching algorithm in [10] to a "cut matching" process for the search of a backward matching cut. For each matched pair on the cut, if its corresponding gates are proven equivalent in the forward SAT sweeping process [11][12][13], we can conclude that it has no impact on the design errors and thus can be removed from the set of the rectification candidates. Otherwise, we recursively apply the cut-matching algorithm for this pair in order to further enlarge the matched region. In "Replacement phase", we propose a recycling mechanism to reuse the gates in the original circuit cone for the patch composition. That is, while fanout-free unmatched gates in the original cone are to be disconnected and replaced by some of the patch logic, they can be reclaimed if their sizes and gate types match the other patch gates in the golden circuit. This is essentially equivalent to a rewiring process in the original circuit and can serve as a low cost solution if the patch is small and local so that the physical constraints (e.g. wire length) can be easily met.

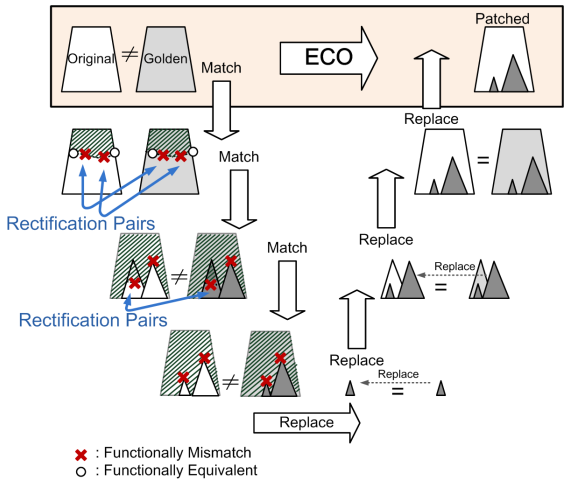


Fig. 1: Match-and-Replace ECO engine

We conduct our experiments on several benchmark circuits and the results show that our ECO engine can efficiently identify the patch logic for multi-error designs. Our patch sizes are only about 0.40% or 1.98% of the original design, with or without patch recycling, respectively. When compared to a state-of-the-art interpolation-based ECO engine [8], which fails in many multi-error cases, we can reduce the patch sizes to 1/22 or 1/4.5, and runtime to 54% or 49%, with or without patch recycling, respectively.

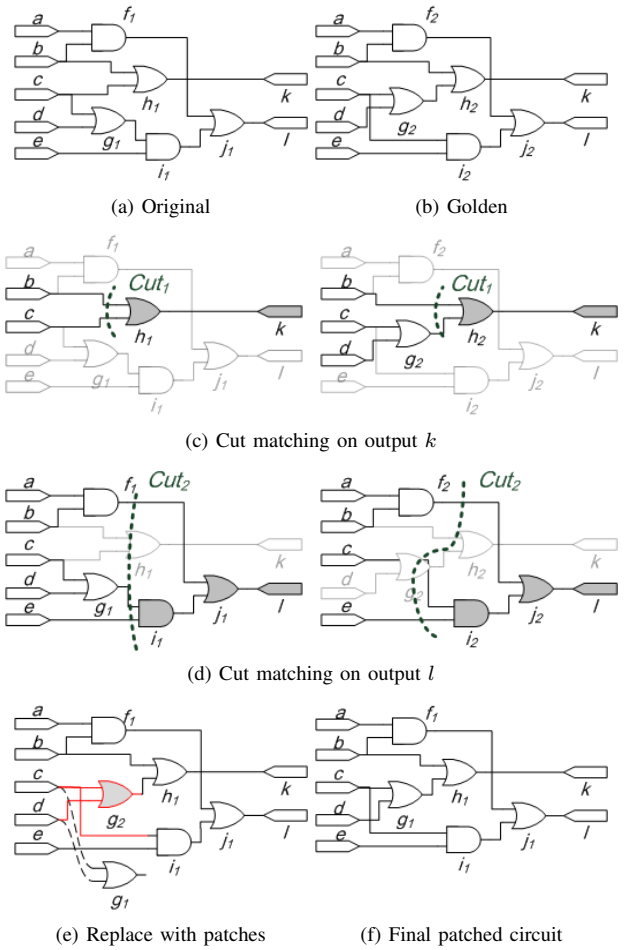
978-1-4577-1400-9/11/$26.00 ©2011 IEEE

II. An Illustrative Example of Our Match-and-Replace ECO Algorithm

Fig. 2 illustrates an example of our Match-and-Replace (M&R) ECO algorithm. The original and the target golden circuits are shown in Fig. 2(a) and 2(b), respectively. In this example, the functional changes are caused by the reconnection of the wires $< c, h_1 >$ and $< g_1, i_1 >$. It can be shown that this change cannot be rectified by a single fix on the original circuit.

In the beginning of the ECO process, the primary inputs are mapped together, and the gates f_1 in the original circuit and f_2 in the golden are merged in the SAT-sweeping step. Our engine then performs the cut matching algorithm on the primary outputs k and l as shown in Fig. 2(c) and Fig. 2(d), respectively. The resulted cuts are the matching pairs $\{b \leftrightarrow b\}$ and $\{c \leftrightarrow g_2\}$ for k, and $\{f_1 \leftrightarrow f_2\}$, $\{g_1 \leftrightarrow c\}$ and $\{e \leftrightarrow e\}$ for l.

We next apply the "rectification pair selection" algorithm on the above rectification pairs. In this example, we conclude that both pairs are needed for the error corrections. We then replace the wire $< c, h_1 >$ by the gate g_2, and the wire $< g_1, i_1 >$ by the primary input c. The resulted patched circuit is shown in Fig. 2(e). Finally, since the disconnected gate g_1 in the original circuit has the same gate type and number of inputs as the patch gate g_2, we can reuse it as the patch in Fig. 2(f). Therefore, the final patch cost (i.e. number of gates added) is 0 in this case.



(a) Original (b) Golden

(c) Cut matching on output k

(d) Cut matching on output l

(e) Replace with patches (f) Final patched circuit

Fig. 2: An examples for M&R ECO

III. The Matching Phase — Rectification Pair Identification

The rectification pair identification in our engine aims at exploring correlations between original and golden circuits. The rectification pairs are two signals in two circuits. They help us to rectify functional differences between two circuits by replacing the nodes in the original circuit with the nodes in the golden circuit. We briefly define the rectification pair in the following subsection. After that, the rectification pair identification algorithm is proposed in Section III-B. And other matching techniques are discussed in Section III-C and III-D.

A. Rectification Pair

The rectification pairs in our ECO process are to rectify the difference between two circuits. In general, they are often in the form of a group of pairs. The definition is shown as:

Definition 1: A **rectification pair** of nodes is one node n_o in the original circuit and the other n_g in the golden. A group of pairs are **rectification pairs** if and only if replacing all n_o in these pairs with their n_g makes the original and the golden circuits turn into functionally equivalent.

To represent the replacement in the circuit, we write $m \xleftarrow{R} n$ n to mean that a signal m is replaced by n.

Given a circuit $Cir(X)$ with primary inputs X, $Cir(X, m \xleftarrow{R} m'(X), n \xleftarrow{R} n'(X))$ represents that the internal signals $m(X)$ and $n(X)$ in $Cir(X)$ are replaced by $m'(X)$ and $n'(X)$.

According to the definition, we say that $pair(p_1, p_1')$ to $pair(p_n, p_n')$ in two circuits $Ori(X)$ and $Gold(X)$ are rectification pairs if these n pairs satisfy the formulas:

$$\forall X, Ori(X, p_1(X) \xleftarrow{R} p_1'(X), ...p_n(X) \xleftarrow{R} p_n'(X)) \equiv Gold(X) \quad (1)$$

For example, the PO pairs are rectification pairs by default. It is because that we can get a brute-force but correct solution for functional rectification if we directly replace whole circuit from outputs with the golden circuit.

Theorem 1: In ECO process, given a rectification pair $\{p \leftrightarrow p'\}$, if a group of pairs **Ps**:$\{np_1 \leftrightarrow np_1', np_2 \leftrightarrow np_2', ..., np_k \leftrightarrow np_k'\}$ can rectify the functional difference between p and p', these Ps are also rectification pairs in ECO process.

Proof: We define a group of rectification pairs **RPs**, which $\{p \leftrightarrow p'\}$ is included in and this group can rectify the original circuit by replacement. We can generate a circuit $Ori'(X)$ which is a revised $Ori(X)$ by replacing all rectification pairs in **RPs** except $\{p \leftrightarrow p'\}$. And $Ori'(X)$ follow the follow equation:

$$\forall X, Ori'(X, p(X) \xleftarrow{R} p'(X)) \equiv Gold(X) \quad (2)$$

And

$$p(np_1 \xleftarrow{R} np_1', np_2 \xleftarrow{R} np_2', ..., np_k \xleftarrow{R} np_k') \equiv p' \quad (3)$$

By (2)(3)

$$\forall X, Ori'(X, np_1(X) \xleftarrow{R} np_1'(X), ..., np_k(X) \xleftarrow{R} np_k'(X)) \equiv Gold(X) \quad (4)$$

A new group of rectification pairs **RPs**$\cup$**Ps** $-$ $\{p \leftrightarrow p'\}$ must also be a group of rectification pairs in ECO process. ■

Definition 2: A *Cut function* in a circuit Cir,represented as $CF_{Cir}(CUT)$, is a modified output function of Cir which its input signal are moved from the PIs of Cir to the cut CUT.

As $Cir1$ in Fig. 3(a), the original output function is $CF_{Cir1}(a,b,c,d) = (a+b)(c+d)$. And the cut function on cut:$\{a,b,f\}$ is $CF_{Cir1}(a,b,f) = (a+b)f$.

Theorem 2: Given two circuit $CirF$ and $CirG$, if $CF_{CirF}(CUT_F) \equiv CF_{CirG}(CUT_G)$ while $CUT_F = \{f_1, f_2, ..., f_n\}$ and $CUT_G = \{g_1, g_2, ..., g_n\}$, a revised circuit $CirF(X, f_1(X) \overset{R}{\leftarrow} g_1(X), ...f_n(X) \overset{R}{\leftarrow} g_n(X))$ is functionally equivalent to $CirG(X)$.

Proof: Since $CF_{CirF}(CUT_F) \equiv CF_{CirG}(CUT_G)$, the difference between $CirF$ and $CirG$ are caused by signals on CUT_F and CUT_G. Once we replace f_1 to f_n with g_1 to g_n, CUT_F becomes equivalent to CUT_G. Thus, Cir_F and Cir_G are functionally equivalent. ■

According to Definition 1 and Theorem 2, we can derive rectification pairs from matched cuts on two circuits. Moreover, we can further adopt cut matching algorithm to recursively expand rectification pairs from PO to PI by theorem 1. As the example in Fig. 3, $\{h \leftrightarrow i\}$ is a rectification pair from previous step and $h(e,f)$ and $i(g,c)$ are both AND2 gates. Thus, the cut functions $CF_{Cir1}(e,f) = CF_{Cir2}(g,c)$. We derive that $\{e \leftrightarrow g\}$ and $\{f \leftrightarrow c\}$ are two new rectification pairs in this example.

(a) Cir1 (b) Cir2

Fig. 3: $\{e,f\}$ and $\{g,c\}$ is a pair of matching cut for (h,i). (e,g) and (f,c) are their rectification pairs.

B. Overview of the Rectification Pair Identification Algorithm

Rectification pairs are pairs of nodes in two circuits as in Definition 1. Here, we proposed a rectification pair identification algorithm as Fig. 4.

Algorithm 1 *Rectification Pair Identification Algorithm*

```
1: Input: ECO circuits: Original, Golden
2: Output: Rectification pairs: RectifyPair
3: for all pair(po, po') in POs do
4:    RectifyPair ← RecifyPair + pair(po, po');
5:    for all pair(po, po') in RectifyPair do
6:       CutPair ← Cut_Matching(p, p', Original, Golden);
7:    end for
8:    for all NewPair(n, n') in CutPair do
9:       if n ≠ n' then
10:         RectifyPair ← RectifyPair + NewPair(n, n');
11:      end if
12:   end for
13: end for
```

Fig. 4: Rectification pair identification algorithm

Fig. 4 shows the algorithm for matching rectification pairs. We mark all PO pairs as initial rectification pairs and search further rectification pairs from these initial pairs. When the cut matching algorithm returns a matching cut, we obtain a set of new matching pairs. We recursively derive new rectification pairs until no new rectification pair identified.

C. Cut Matching Algorithm

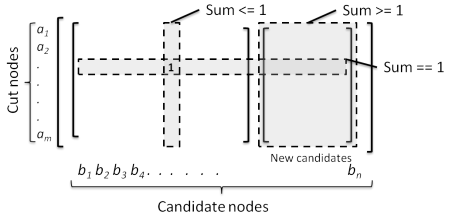
The cut matching algorithm in rectification pair matching algorithm is a revised cut-based Boolean matching algorithm. We extend the SAT-based Boolean matching algorithm in [10] to get matched cuts.

Definition 3: Given an original circuit and a golden circuit, cut matching aims to satisfy the formula:

$$\exists Cut_1 \exists Cut_2, CF_{Ori}(Cut_1) \equiv CF_{Gold}(Cut_2) \tag{5}$$

Then, Cut_1 in Ori and Cut_2 in $Gold$ are two matched cut.

In short, cut matching is a process to identify a matching cut between original and golden circuit. Different from the well-known Boolean matching problem, the cuts in our algorithm are not predefined and thus we have to search for the cuts and check whether they can be matched at the same time. We insert three extra steps before ordinary Boolean matching in [10]: In the beginning, a cut $Cut1$ is selected from the original circuit. On the other hand, a set of candidate nodes are selected from the golden circuit according to $Cut1$. When the matching engine returns a success, we get two matched cuts. Then, we recursively call the cut matching algorithm again from any of the rectification pairs. This process continues until we cannot derive any matched cut pair. Our Boolean matching algorithm aims at getting a P-equivalence match in two circuits. In our matching flow, our algorithm not only explores P-equivalence on the inputs of the circuits but any possible cuts in two circuits. Therefore, the matching problem is extended from cut and cut into cut and set of nodes. To model the cut-nodes mapping with SAT problem, we describe the relations of the matching with a $0-1$ matrix. Here, a matching relation between a m-input cut and n nodes is modeled as the $m \times n$ $0-1$ matrix:



In this matrix, $\{a_1, a_2, ..., a_m\}$ are nodes on $Cut1$. $\{b_1, b_2, ..., b_n\}$ are nodes in the candidate set. There are $m \times n$ binary variables e_{11} to e_{mn} in this matrix. The variable e_{ij} assigned 1 means that a_i and b_j are matched. The SAT solver is assigned to get $m \times n$ assignments to represent a feasible m to m matching in this matrix. The constraints of the feasible matching can be represented as:

$$\sum_{j=1}^{n} e_{ij} = 1, \qquad i = 1, ..., m \tag{6}$$

$$\sum_{i=1}^{m} e_{ij} \leq 1, \qquad j = 1, ..., n \tag{7}$$

$$\sum_{i=1}^{m} \sum_{j=k}^{n} e_{ij} = 1, \quad b_k \, to \, b_n \, are \, new \, candidates. \tag{8}$$

We constrain one node on $Cut1$ must map to one node in the candidates. The constraints are shown as (6). On the other hand, for each candidate node, we ensure that one candidate node is mapped to at most one node on the cut. That is, the summation of each column must be equal or less than 1 as (7). Besides, when the cut matching flow inserts new nodes into the candidates set, we add the

constraints as in (8) to prevent a redundant search in original matrix. This constraint requests that at least one node must be included in new candidates.

D. Challenges on Functional Symmetry

Another challenge in sweeping-based ECO is the functional symmetry. When performing Boolean matching algorithm, the input symmetric nodes are totally equivalent. We cannot differentiate them because each node is treated as an independent input gate.

Input mismatch may lead to subsequent mismatch and thus increase the patch size. However, if we identify the correct matching by brute-force search, the matching engine has to enumerate all possible combinations and thus the matching time must become unacceptable.

In our work, when the matching algorithm encounters functional symmetries, we first try to reduce the symmetric nodes by checking functional equivalence. If there are still symmetric inputs, we take down the matching pairs into rectification pairs but return a matching fail and thus our engine can skip the doubtful matches and search for further cut again. As Fig. 2(c), $Cut1$ is the inputs of an OR gate, which are functional symmetric in the cut function. We first check whether there are two nodes functionally equivalent in these symmetric nodes. In this case, b and b are the same primary input and can be excluded. Thus, the symmetric problem is solved. If there are still symmetric nodes, we generate another cut on the original circuit and continue to search for matching on the original PO k.

IV. The Replacement Phase — Rectification Pair Selection and Patch Minimization

In the replacement phase, we propose a rectification pair selector to select final patched cones from rectification pairs. On the other hand, a patch minimization method is also proposed to reduce final patch sizes.

A. Rectification Pair Selection

After we get rectification pairs from Section III, our pair selector starts to determine which nodes should be replaced with final patches. In Section III, we have demonstrated how to repair circuits by pair replacing. However, the rectification pairs in the early stages may be dominated by later pairs and turn into redundant. In addition, one patched node may be rectified with different nodes in different iteration. The rectification becomes more complex. To get a correct patch set, the rectification pair selection algorithm in this section helps us correctly find the rectification nodes in rectification pairs.

The structure of our rectification pair selector is shown as Fig. 5. The original circuit and the golden circuit are connected. We merge PIs and connect POs with XOR gates. After that, we connect each rectification pair with a 2-bit MUX. The original fanout drives are replaced by the MUX gate. With these MUX gates, we incrementally change the network connection when we check the equivalence on the miter. After all selection signals assigned, we perform an equivalent checking to check whether this set of patches rectify the errors in the original circuit.

To find an optimal solution in rectification pairs is a NP-hard problem. However, there are some relations between pairs and we can adopt these relations to speed up our searching. As the example in Fig. 2, when we perform a replacement $c \overset{R}{\leftarrow} g_2$, we do not need to perform the replacement $h_1 \overset{R}{\leftarrow} h_2$ anymore. According to the dependency between rectification pairs, we propose a linear-time heuristic to pick up rectification pairs and keep the circuit rectifiable.

As the cut matching algorithm, the rectification pairs found from PO to PI and the nodes found in the late stage dominate the earlier

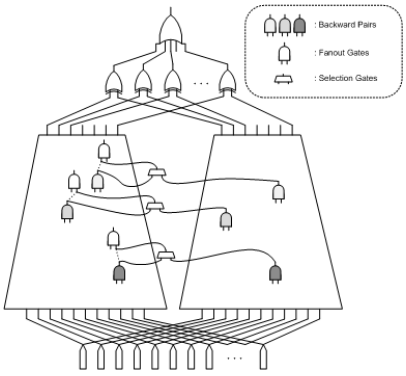


Fig. 5: A miter with selectors

nodes. When the lower nodes are rectified, we do not need to rectify the upper nodes again. Therefore, we give the higher priority to the rectification pairs closer to PIs. According to the priority, our selector linearly picks up a set of rectification pairs to repair the circuit. This heuristic is simple: First, we replace all selection signals of rectification pairs. Then, we undo the replacement from in *breadth-first searching order* from POs to PIs. The experiments show that our searching order is efficient while keeping a good quality.

The selection process is shown in Fig. 6. In the beginning of the flow, we first replace all signals n_o in rectification pairs $\{n_o \leftrightarrow n_g\}$ with their n_g in line 4. Here, we achieve the replacements by setting 1 to each selection signal. In this step, all primary outputs are replaced by the golden and thus the patched circuit must be equivalent to the golden circuit. After that, we recover the replacement by assigning 0 to selection signal in *breadth-first searching order* as line 8. After each assignment changes, we perform an equivalence checking to ensure the equivalence. If it keeps equivalent, the node is rectified by lower patch, and we can leave out the replacement from final patches. Otherwise, we commit the nodes replaced as line 11, and these rectification pairs will be included in final patches.

Algorithm 2 *Rectification Pair Selection Algorithm*

```
1: Input: Miter, RectifyPair
2: Output: Patches
3: for all pair(n1, n2) in RectifyPair do
4:     Replace(n1, n2)
5: end for
6: Sort Rectifypair in BFS order
7: for all pair(n1, n2) in RectifyPair do
8:     Undo_Replace(n1, n2)
9:     res = Check Equivalence on Miter
10:    if res == Non-Equivalent then
11:        Replace(n1, n2)
12:        Patches ← Patches + pair(n1, n2)
13:    end if
14: end for
```

Fig. 6: Rectification pair selection algorithm

B. Patch Minimization

Skillfully reusing floating gates in the original circuit is an efficient method of reducing patch sizes. After ECO process, there are some gates replaced by new patch circuits and turn into floating and useless. In the circuit design process, these floating gates finally become spare resources for further functional or timing improvements [14]-[18]. On the other hand, if we allow a small wire length overhead, it is feasible

to locally map useless gates for reducing patch sizes. A case study in our experiments shows as Fig. 7. The replaced patch circuit contains 7 gates. We reuse these floating gates to the local patch gates and then the patch size is reduced to 3.

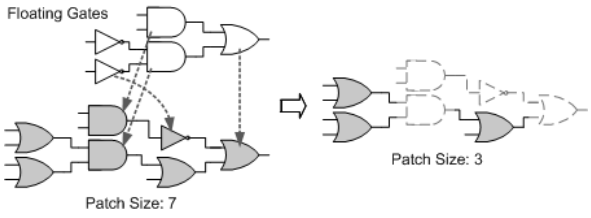


Fig. 7: Gate recycling in patch circuit

V. Match-and-Replace ECO Engine

In this section, we demonstrate the overall flow of our ECO engine. The rectification pair identification algorithm is shown in Section III and the pair selection and the patch minimization method are in Section IV. Besides, we also propose two additional optimization methods for further improvement in this section.

A. An Overview of M&R ECO Engine

The overview of our ECO engine is shown in Fig. 8. In the beginning of our engine, we devote to match rectification pairs. After getting matching pairs, we replace rectification pairs in a miter. Then, we perform the patch selection algorithm to decide which rectification nodes require to keep replaced and which are redundant. In the end of the flow, we get a minimal patch to rectify the original circuit.

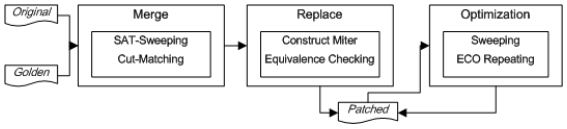


Fig. 8: M&R ECO engine

B. Further Improvements on the Patch Size

Our M&R ECO always return a quality patch circuit after the ECO flow. Nevertheless, still several methods could be applied to improve the patch size.

1) Merging Equivalent Signals: SAT sweeping algorithms [11][12][13] are fast and effective methods of reducing redundancies in a circuit. After inserting patches, we adopt a final sweeping process to merge functionally equivalent gates in final patched circuits. In our replacement step, we have checked the redundancy on new patch gates before inserting patches. However, some nodes are functionally changed and may be equivalent to other nodes after patched. In this case, functional sweeping can merge them and reduce the patch size.

2) Reiterating M&R ECO Process: In our M&R ECO engine, the patched circuit is much similar to the original circuit than the golden one. If we replacing the golden with the patched circuit and perform M&R ECO again, the patched circuit in the second iteration will be more closer to the original circuit and the final patch size is guaranteed to be compact than previous result. The experiment in Section VI-C shows that reiterating M&R get about 50% improvements in some cases.

To enhance the effectiveness of these optimization methods, we can try a permutation on the patched circuit. Re-synthesis does not guarantee to get a smaller patch size. However, with re-synthesis, we have more opportunity to sweep the nodes in the patch circuit and further reduce the patch size.

VI. Experimental Results

In the experiments, the modified circuits are treated as a golden circuit. ECO engines are requested to rectify the original circuit. We apply MiniSAT [19] as our SAT engine. All of our experiments are conducted on a Linux workstation with 32GB RAM and 2.5GHz Intel Xeon CPU.

A. Comparison to Interpolation-based ECO

We compare our algorithm and Interpolation-based ECO with 42 testcases from ISCAS'89 benchmarks. The performance comparison are shown as Table I and Fig. 9. We demonstrate that our M&R ECO outperforms in both runtime and performance even in cube permutation. When we turn on locally recycling process, the patch size gets further 80% improvement in only slightly runtime overhead (0.1 to several seconds). Moreover, 20 testcases are rectified with taking 0-size patches with the locally recycling process. That is, these testcases are rectified by internal connection only.

TABLE I: Performance comparison with [8]

ECO Engine	*Patch Size*	*Overhead*	*Runtime(s)*	*Patch=0*
[8]	5239	8.85%	551.53	0
M&R	1161	1.92%	253.79	9
M&R+recycle	232	0.40%	299.43	20

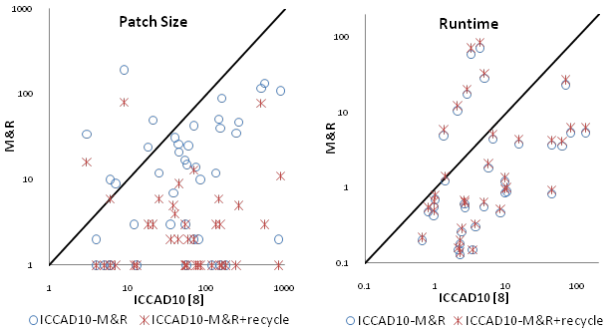


Fig. 9: Performance comparison with [8]

B. Performance Comparison on Different Structural Changes

In this experiment, we focus structural changes on 4 ISCAS'89 cases. We randomly perturb the testcases with 1) node permutation, 2) function inversion, 3) type changing and 4) node replacement. In Table II, the first column shows the name of the testcase and its circuit size. And the second column shows the permutations on the testcase. The column "NS" means node swapping. "INV" means inverter inserted. "TC" means gate type changing. And "FR" means that a function is replaced by other function. The patch size and runtime are shown in "Patch" and "T". The "x" in patch size means the ECO engine aborts.

The numbers in the columns are times of the perturbation. The result shows that M&R ECO is more efficient than interpolation-based ECO when the changes are small and local. As Table II, our M&R ECO always gets smaller patch circuits with less runtime whether the recycling method turn on or not.

TABLE II: Performance comparison on ISCAS'89 cases

Testcases	*Perturbation*				*Results*					
					[8]		M&R		M&R+recycle	
	NS	INV	TC	FR	Patch	T(s)	Patch	T(s)	Patch	T(s)
s6669	5	0	0	0	4	21.86	0	2.69	0	2.69
(4887)	6	0	0	0	x	x	129	187.35	120	189.12
	2	2	0	0	3	27.04	0	2.7	0	2.7
	2	0	2	0	3	27.88	1	2.74	1	2.74
	2	0	0	2	x	x	25	8.17	19	8.12
s9234.1	5	0	0	0	102	37.15	25	4.81	11	4.81
(3521)	6	0	0	0	102	32.21	25	4.81	11	4.81
	2	2	0	0	x	x	14	4.12	12	4.11
	2	0	2	0	x	x	12	4.05	11	4.02
	2	0	0	2	x	x	19	3.07	14	3.11
s13207	5	0	0	0	59	90.27	0	7.69	0	7.7
(5446)	6	0	0	0	x	x	0	7.67	0	7.68
	2	2	0	0	57	77.58	0	7.39	0	7.39
	2	0	2	0	57	78.56	0	7.42	0	7.41
	2	0	0	2	57	83.41	0	7.27	0	7.27
s13207.1	5	0	0	0	55	106.7	17	9.41	15	9.42
(5440)	6	0	0	0	x	x	28	10.04	25	10.06
	2	2	0	0	63	95.73	1	6.95	0	6.95
	2	0	2	0	66	108	1	6.98	0	6.96
	2	0	0	2	75	98.39	9	7.2	0	7.18

C. Effectiveness of Further Improvement

In our previous experiments, most patches are compact and quality. However, we can improve some results by our further improvements. The improvements are shown as Table III. The experiment shows that final sweeping or applying ECO engine iteratively is able to refine patches.

TABLE III: The examples for further improvement

Testcase	*Method*	*Original*		*After Opt.*	
		Patch	Time(s)	Patch	Time(s)
s713	Sweep	52	17.28	46	17.29
s991	Repeat	100	0.818	49	0.822
s1488	Repeat	88	59.33	71	63.5
s1494	Repeat	109	39.7	50	71.3

VII. Conclusion

This paper proposed a matching-based functional ECO engine. The M&R ECO succeeds to get small patches in any kind of functional changes. Our rectification pair matching algorithm efficiently identifies functional relations between circuits. The patch selector properly decides patch nodes in the circuits. And the patch minimization effectively optimizes the patch sizes. Comparing with other previous works, M&R ECO is more flexible and efficient than others.

Acknowledgment

This work is supported in part by National Science Council under grants NSC 99-2221-E-002 -211 -MY3.

References

[1] J. Jaeger. Virtually every ASIC ends up an FPGA. *EE Times*, December 7, 2007.

[2] C.-C. Lin, K.-C. Chen and M. Marek-Sadowska. Logic synthesis for engineering change. *IEEE Trans. on CAD of Integrated Circuits and Systems,* 1999.

[3] Y.-S. Yang, S. Sinha, A. G. Veneris, and R. K. Brayton. Automating logic rectification by approximate SPFDs. In *Proc. Asia and South Pacific Design Automation Conference,* 2007.

[4] A. C. Ling, S. D. Brown, S. Safarpour and Jianwen Zhu. Toward Automated ECOs in FPGAs. *Proc. FPGA,* 2009.

[5] K.-H. Chang, I.L. Markov and V. Bertacco. Fixing design errors with counterexamples and resynthesis. *IEEE Trans. on CAD of Integrated Circuits and Systems,* 2008.

[6] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng. AutoFix: A hybrid tool for automatic logic rectification. *IEEE Trans. on CAD of Integrated Circuits and Systems,* 1999.

[7] D. Hoffmann and T. Kropf, Efficient design error correction of digital circuits. *Proc. International Conference on Computer Design*, 2000.

[8] B.-H. Wu, C.-J. Yang, C.-Y. R. Huang and J.-H. R. Jiang. A Robust Functional ECO Engine by SAT Proof Minimization and Interpolation Techniques. *Proc. International Conference on Computer-Aided Design*, Nov. 2010.

[9] S. Krishnaswamy, H. Ren, N. Modi, and R.Puri. DeltaSyn: an efficient logic difference optimizer for ECO synthesis. In *Proc. International Conference on Computer-Aided Design*, 2009.

[10] C.-F. Lai, J.-H. R. Jiang, and K.-H. Wang, BooM: A Decision Procedure for Boolean Matching with Abstraction and Dynamic Learning. In *Proc. DAC*, June 2010.

[11] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Proc. Design Automation Conference*, 1997.

[12] A. Mishchenko, S. Chatterjee and R. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," *EECS Dept., UC Berkeley, Tech. Rep,* vol. 3, 2005.

[13] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. L. Sangiovanni-Vincentelli. SAT sweeping with local observability don't-cares. In *Proc. Design Automation Conference*, 2006.

[14] K.-H. Ho, J.-H. R. Jiang, and Y.-W. Chang. TRECO: Dynamic technology remapping for timing Engineering Change Orders. In *Proc. Asia and South Pacific Design Automation Conference*, Jan. 2010.

[15] Y.-M. Kuo, Y.-T. Chang, S.-C. Chang, and M. Marek-Sadowska. Engineering change using spare cells with constant insertion. In *Proc. International Conference on Computer-Aided Design*, Nov. 2007.

[16] N. A. Modi and M. Marek-Sadowska. ECO-Map: Technology remapping for post-mask ECO using simulated annealing. In *Proc. International Conference on Computer-Aided Design*, Oct. 2008.

[17] H.-R. I. Jiang, H.-Y. Chang, L.-G. Chang, and H.-B. Hung. Matching-based minimum-cost spare cell selection for design Changes. In *Proc. Design Automation Conference*, Ruly 2009.

[18] S.-L. Huang, C.-A. Wu, K.-F. Tang, C.-H. Hsu, and C.-Y. Huang. A robust ECO engine by resource-constraint-aware technology mapping and incremental routing optimization. In *Proc. Asia and South Pacific Design Automation Conference*, 2010.

[19] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, p. 53, 2005.