CAD Contest Problem A

*Note: Sub-titles are not captured in Xplore and should not be used

1st Guo-Wei Ho
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901135@ntu.edu.tw

2st Yen-Ju Lee
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901075@ntu.edu.tw

3st Meng-Hung Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901103@ntu.edu.tw

4st Pin-Chun Yu
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901005@ntu.edu.tw

***Abstract*—We studied and implemented an algorithm from previous work to solve the ECO problem of CAD contest problem A. The algorithm consist of two phase. In the matching phase, we identify and maximize the functional matched part between two circuits. In the replacement phase, a minimized patch is generated to replace the different part in the error circuit. Since we haven't complete the whole implementation, this report will focus on the implementation details and the work we have achived.**

***Index Terms*—**

I. Introduction

In this report, we present the semi-formal method proposed in [2] and show our study and implementation of the algorithm. The algorithm aimed to solve the multi-error functional ECO problem. Based on a previous work [1], the proposed algorithm adopt a simulation-guided cut-matching algorithm to deal with output side matching.

The algorithm consist of two phase: The matching phase and the replacement phase. The goal of the mathcing phase is to identify the functionally matched part between the revised circuit and the golden circuit.

This process can be further divided into two part: Input side merging and output side matching. For input side, we simply perform strash and FRAIG on the miter circuit to merge the functionally equivalent gates. The major challenge is in the output side matching process and we will be mostly focus on this part. The algorithm proposed tries to find cuts with greater chance to match by performing simulation and exploit the similarity information. After that, the equivalence checking process is modeled as a SAT problem and solved by a SAT solver.

In the replacement phase, we insert a MUX on each rectification pair to help us determine the feasible set of gates to be patched. Finally, we generate the patch and the verilog file from the selected patches. The Overview of the algorithm is shown in Fig. 1.

Identify applicable funding agency here. If none, delete this.

The rest of this report is organized as follows. In section II and III, We define some terms and describe the objective and algorithm in the each phase. Since the algorithm is mostly the same as in the original paper, we only describe the main idea of algorithm and will bring up some implementation details we derived. In section IV, we will go through the work we've done currently. Finally, section V, VI and VII will be consist of our future work, teamwork, and disclaimer for the source code and package we use respectively.

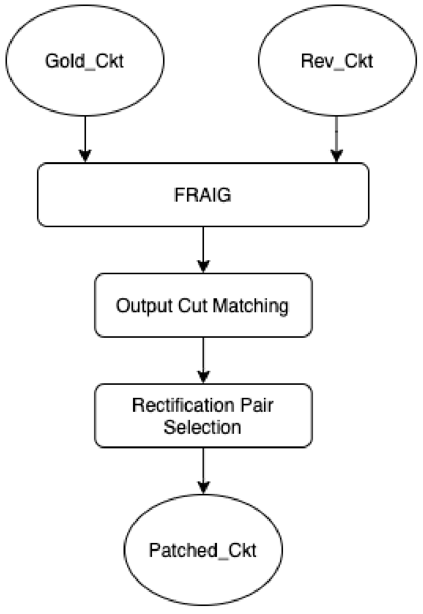

Fig. 1. The overview of our algorithm

II. Matching Phase – Recitification Pair Identification

A. Input Side Merging

The input side merging is to identify as most as possible equivalent gates between the revised circuit and the golden circuit. Since, in our replacement phase, we will fix the different part of circuit by replacing some gates in the revised circuit with the correct function in the golden circuit, we can utilize these functionally equivalent gates to reduce the patch size by using then as the corresponding merged gates in the golden circuit.

B. Rectification Pair

1) Rectification Pair: A set of pairs: $\vec{g} \leftrightarrow \vec{g'}$ (RP) is called a set of rectification pairs if

$$\forall \vec{X}, Rev\_Ckt(\vec{X})|_{\vec{g} \leftarrow \vec{g'}} \equiv Gold\_Ckt(\vec{X})$$

where every $g_i \in \vec{g}$ belongs to the revised circuit, $g'_i \in \vec{g}'$ belongs to the golden circuit, and $\vec{X}$ is the input assignment. When replacing every g_i with g'_i, the revised circuit becomes functionally equivalent to the the golden circuit. Note the the set of PO pairs is a trivial set of rectification pairs.

With a known set of rectification pairs RP, and a pair p: $g_i \leftrightarrow g'_i$ where $p \in RP$, if a set of pairs P is a set of rectification pairs of the pair p, then the set of pairs

$$RP' : RP - \{p\} + P$$

must also be a set of rectification pairs.

2) Cut Function: A cut function $CF^{o}_{(\vec{g})}$ is a part of the circuit the represents the functionality of the output with respect to the cut ($\vec{g}$). Two cut logic (o, o') with respect to $(\vec{g}, \vec{g}')$ are said to be equivalent if

$$\forall \vec{X}, CF^{o}_{(\vec{g})}(\vec{X}) \equiv CF^{o'}_{(\vec{g'})}(\vec{X})$$

where $\vec{X}$ is the input assignment. If two cut functions are equivalent, the set of pairs p: $g \leftrightarrow g'$ is a set of rectification pairs of the pair (o, o').

C. Output Cut Matching

To identify as most as possible rectification pairs, we take each known pair as a cut function, and try to find a match cut $(\vec{c}, \vec{c'})$ such that two cut functions are equivalent. Then, we add the non-equivalent pairs on the cut into our set of rectification pairs. The main algorithm of identifying rectification pairs by cut matching is shown in algorithm**??**

1) Simulatoin Guilded Cut Matching: To find two equivalent cut functions, we have to find a possibly matched cuts and prove the equivalence using the SAT solver. Unlike the method proposed in [1], which use a sat solver to solve a matching matrix, we refer to the method of simulation guided cut matching proposed in [2].

To utilize the similarity of gates as the paper suggested, we've came up with some different approaches. The main idea are the same, but some implementations tends to enumerate more possible cuts, while others rely more on the similarity information.

The method we currently used is described as the following: To find the match cut on a pair (o, o'), we first identify the duplicated merged gates in the fanin cone of two gates within a certain level, and consider them as part of the cut. Then, we run

Algorithm 1 Rectification Pair Identification

```
Input: Rev_Ckt, Golden_Ckt
Output: RPSet
  for each pair in POs do
      RPSet ← pair
  end for
  for each pair($g, g'$) in RPSet do
      Cut ← Find_Match_Cut($g, g'$)
      for each pair($c, c'$) in Cut do
          if $c$ is not equivalent to $c'$ then
              RPSet← pair($c, c'$)
          end if
      end for
  end for
```

the simulation on the candidate gates that can potentially form the left part of the cut. We then compare the similarity of the simulation signal of each pairs of gate, keep those pairs with high similarity, and try to enumerate cuts using those gates. Note that we only consider the good vector when calculating similarity. The whole algorithm to generate a list of candidate cuts is presented in algorithm 2.

Algorithm 2 Get Guilded Cut Cands

```
Input: $g$, $g'$, $level$
Output: candCuts, mergeCut
  mergeCut ← Find_Dup_Merged($g$, $g'$, $level$)
  candGates ← Find_Cand_Gates($g$, $g'$, $level$, mergeCut)
  candPairs ← Find_Similar_Pairs($g$, $g'$, candGateList)
  order candPairs by similarity
  candCuts ← Enumerate_Cuts($g$, $g'$, candPairs)
  order candCuts by checksize
```

Note that the checksize mentioned in algorithm 2 means the length of the cut found, which is in fact the unmerged part of the actual cut.

2) Constraint Cut Logic: Since our goal is to rectify the revised circuit to match the functionality of golden circuit, we can just focus on the equivalence under the care constraint of golden circuit. That is, when checking the equivalence of two cut functions, we can optionally add the part of golden circuit under the cut into our proof instance to reject some don't care assignment on the cut. By doing so, the chance of mathcing two cuts can be increased.

3) Implementation Issue: However, adding the constraint cut logic as stated above also require a way larger SAT model, which could slow down the proving process and use more memory. To speed up the cut function proving process, we can first check the equivalence without constraint, and add the constraint if the miter is satisfiable. We planned to decide whether to prove with or without constraint by the size of the circuit and the number of cut found in the previous step. Also, in order to reuse the SAT model without rebuild and keep the learned clauses in the proof instance, we also planned to proof the cut we found in a approximately top-down order, and only rebuild the model when the necessary. Note that the circuit under the cut in the revised circuit cannot be added to the SAT model since the constraint can exclude some assignment in the care set of the cut in golden circuit. This could possibly make a satisfiable circuit UNSAT under revised circuit's constraint and cause a wrong patch to be generated.

III. Replacement Phase – Patch Generation

A. Rectification Pair Selection

To generate a patch, we want to choose a small subset of rectification pairs from all the pairs we've found in the previous phase, while making sure that the subset of rectification pairs is feasible. To do so, we build a rectification pair selector, which is a miter with some inserted MUXs. Each MUX enable the selection between a patched gate and its corresponding patch.

An example of a rectification pair selector is illustrated in Fig. 2 from [1]

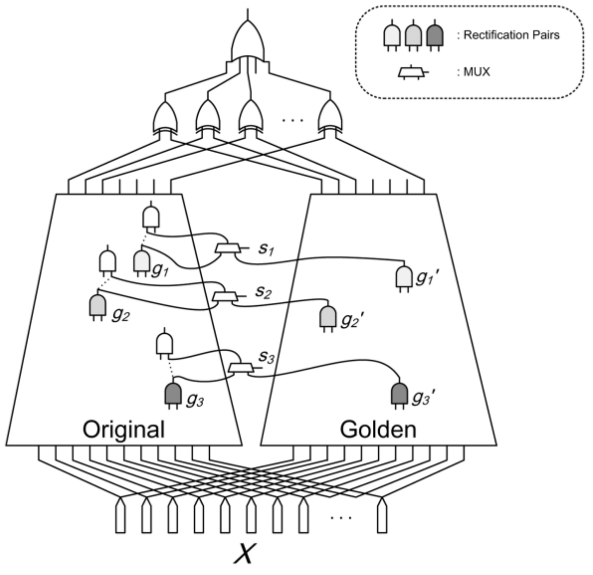

Fig. 2. Rectification pair selector

A feasible patch is an assignment on the selection signals such that, when assigned, the output 1 on the selector is still unsatisfiable. The selection process can be formulated as a QBF

$$\exists s_1, s_2, ..., s_n, \forall \vec{X}, RPS(\vec{X}, s_1, s_2, ..., s_n) \equiv 0$$

How ever, finding the minimal patch on this QBF problem is inefficient, so [1] proposed a heuristic algorithm : Select all patches initially, and undo the patch in a top-down order. When a patch is undone, check the satisfiability of the selector. If the removal make the selector satisfiable, the patch is considered nessessary and will be included in the final patch. In this case, we add the patch back and keep checking the next patch.

IV. Progress and Results

Currently we have finished most of the part in the proposed algorithm and generate a patch. But some issues still exist in our program.

Precisely, the whole framework including the data structure and the functionality of the algorithm is done. We are also able to generate a patch that is, at least, not trivial in some of our testcases. For example, in the following test case "`test3`" from [1], we are able to identify the rectification pair $(i1, i2)$ to fixed the output l.

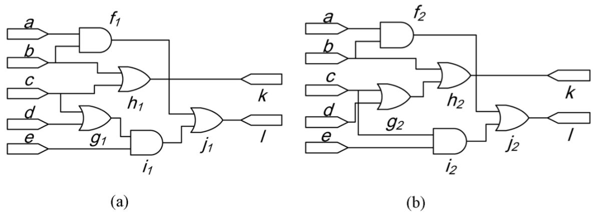

Fig. 3. Example for our engine

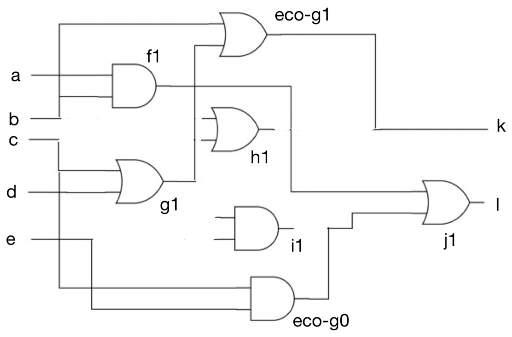

Fig. 4. Output patch for the example

One of the major issues in our program is that we have trouble finding further match cuts in some testcases. We noticed this because the rectification pairs found in "`test2`" are all very close to the original PO. And was further confirmed by studying other testcases.

The main reason is that we pair the corresponding duplicated merged gates on a cut directly during the matching phase. While in some cases, for example, our testcase "`test4`", even with an obvious match cut near the PO, the resulting cut function just won't match if the corresponding duplicated merge gates are paired together. This is also the reason why we can't identify the rectification pair $(c, g2)$ in Fig. 3 since the primary input c is considered merged. Though we don't have time to fixed this before this submission, this would be the next primary task for our project.

Finally, the table in Fig. tab shows our results on the test cases. The testcase test1 and test2 are the public testcases from CAD Contest Problem A. We also have "`test4`" and "`test5`" designed specifically to test the merged gate pairing problem and are not listed in the table.

testcase	test1	test2	test3
patch cost	5	38	6
# nontrivial RP found	0	6	1
# trivial RP replaced (PO)	0	7	1

Fig. 5. patch cost, number of nontrivial rectification pairs found, and the number of trivial RP (PO) replaced by other RP

V. Conclusion and Future Work

As stated in section IV, our primary goal is to fixed the issue we've found and try to find more rectification pairs. There's also lot of room for improvement in terms of performance since we planned to deal with the performance detail after we complete the basic algorithm.

Additionally, in the paper, a final process of reusing floating gates is proposed to reduce the patch size, but we haven't implement it yet. Also, besides the similarity information, the paper also proposed to utilize the structural information. Or, precisely, the duplicated merged gates in the fanin cone of a pair of gates. We plan to put the topic we mentioned above into our work to further improve the result.

VI. Teamwork

Yen-Ju Lee, Meng-Hung Chen, and Pin-Chun Yu are not in this class. The tasks in this project can be divided into 6 Parts, and the team member that work on each part are listed in Fig. VI

Discussion and paper survey	Guo-Wei Ho, Yen-Ju Lee, Meng-Hung Chen, Pin-Chun Yu
Designing the program structure	Guo-Wei Ho, Pin-Chun Yu
Integrating other tools	Guo-Wei Ho, Yen-Ju Lee
The matching phase	Guo-Wei Ho, Yen-Ju Lee
The replacement phase	Guo-Wei Ho, Meng-Hung Chen
Written report	Guo-Wei Ho, Yen-Ju Lee, Meng-Hung Chen

Fig. 6. The distribution of each member in each task

VII. Disclaimer

The source code of the main part of our program came from one of our team members Guo-Wei Ho's previous work, which was used as the final project in the course "Data Structure and Programming" in National Taiwan University. The source code was originally provided by Prof. Chung-Yang (Ric) Huang. It also includes a modified version of MINISAT, a minimalistic, open-source SAT solver, and was used in our equivalence checking process. The synthesis tool ABC is used for input side merging (FRAIG) and network conversion. Since it's easier to build SAT Model with AIG networks, we convert the original circuits, miter, and selectors to AIG network with ABC to simplify the whole routine. Besides, the official AIGER package is used to convert AIG to AAG formant.

References

[1] Huang, S.-L., Lin, W.-H., Huang, P.-K., Huang, C.-Y. (2013). Match and replace: A functional ECO engine for multierror circuit rectification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 32(3), 467–478.

[2] Chia-Lin Hsieh, "Semi-formal ECO method," M.S. thesis, National Taiwan University, Jan. 2018, pp. 1–32.