Problem A: Functional ECO with Behavioral Change Guidance

Yen-Chun Fang, Shao-Lun Huang, Ching-Yi Huang, Chi-An (Rocky) Wu, Chung-Han Chou

Chih-Jen (Jacky) Hsu, WoeiTzy (Wells) Jong, and Kei-Yong Khoo

Cadence Design Systems, Inc.

1. Introduction

Functional ECO [1-10] is to minimize the changes (patch) on the implementation netlist such that it performs the same function as the new specification. In industrial cases, designers write their specifications with RTL source codes and synthesize the RTL codes into an implementation netlist with physical constraints. As technology node advances, it takes many efforts to generate implementation netlists to meet physical constraints. With time-to-market pressures, when specification changes after the implementation netlist is generated, users must change on implementation netlist directly and minimize the patch for physical constraints. Functional ECO algorithm is the solution to generating and minimizing the patch on the implementation netlist such that the patched netlist is functionally equivalent to the new specification. In industrial cases, the information of the difference between original specification and new specification can be used to minimize patch. In this contest, we look for effective algorithms to utilize the information of behavioral change to minimize the patch.

In this contest, we formulate a problem of "Functional ECO with Behavioral Change Guidance." Given three netlists, *R1*, *R2*, and *G1*, contestants are required to write a program to generate and minimize the *patch* for *G1* such that patched *G1* is combinational equivalent to netlist *R2*. Netlist *R1* is used as auxiliary data but can be the key component for your algorithm to generate the minimal patch. In this problem, netlist *R1* and *R2* are generated from RTL elaboration without optimization. Netlist *G1* is synthesized from *R1*. Contestants can utilize the behavioral change between *R1* and *R2* to minimize the patch for *G1*.

2. Background

Functional ECO is the key technology and challenging problem for shortening the time-to-market design schedule. As technology node advances, designs are getting more and more complicated. In some cases, it may require changing the product specification in the late design cycle. However, re-running the whole design flow is impractical [1, 2]. Especially when the mask has been made, the metal only change can save time and money significantly. To address the issue, Engineering Change Order (ECO) is proposed and has been widely used in industries. In contrast to tedious and unpredictable manual ECO, automatic ECO is more practical. As shown in Figure 1, the automatic ECO is a process to generate the patch for rectifying the old netlist to agree with the netlist of the new specification.

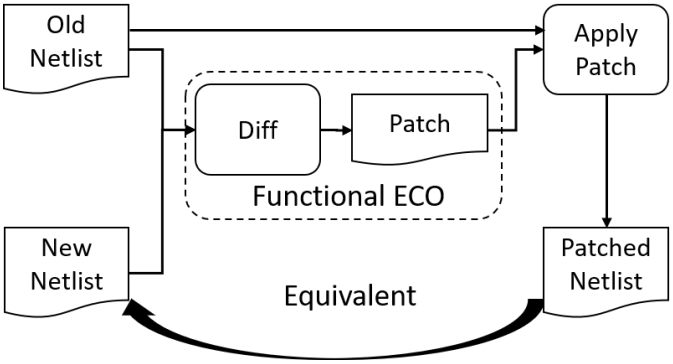

Figure 1. Automatic Functional ECO

Functional ECO is also popular in academic research; several studies [1-10] of automatic function ECO have proposed different algorithms to generate patches with minimized patch size. ICCAD CAD Contest 2012 [11] formulated the problem of functional ECO with benchmark suites. ICCAD CAD Contest 2017 [12] formulated the problem to optimize the patch with physical information. In VLSI design flow, Cadence Encounter Conformal ECO Designer [13] has been widely used for industrial cases for years.

However, existing formal and structural solutions cannot generate properly small patches when two netlists have low similarity even if users can manually edit the netlist with a small change. This observation shows the natural result of the limitation in Boolean methods and structural heuristics. Note that the netlist from RTL elaboration of specification and the implementation netlist are dissimilar because the implementation netlist is synthesized through functional optimization.

As the ECO process develops, the functional ECO problem can be formulated as processing three netlists: original specification, new specification, and implementation netlist. Compared with the old two-netlist ECO, the ECO engine not only analyzes the specification changes but also generates the patch on the implementation netlist.

The original specification (R1) and the new specification (R2) differ in RTL with a few code lines. The implementation netlist (G1) is synthesized and optimized from the original specification (R1), and they are functionally equivalent. We can utilize the above facts to design heuristics and formal solutions to generate the proper ECO patch. In this contest, we formulate the problem "functional ECO with behavioral change guidance" to present challenges and research opportunities for industrial cases. Figure 2 illustrates the problem of this contest.

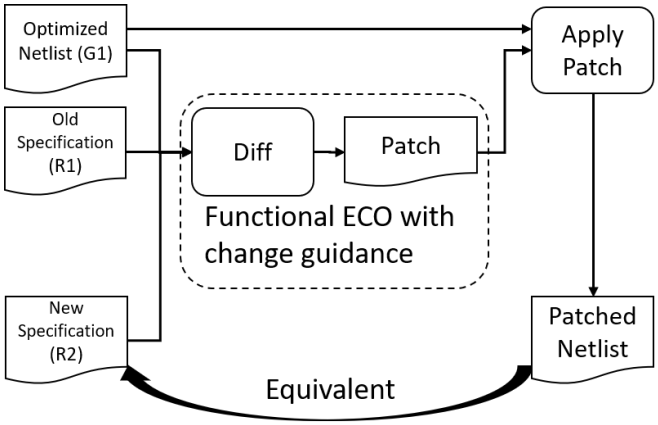

Figure 2. Functional ECO with Behavioral Change Guidance

2.1 Patch and Applying Patch

The patch netlist is a declarative file describing how an implementation netlist changes. We can create the changed (patched) netlist by applying the patch to the original implementation netlist. The netlist patch describes:

1. Which wire in the original netlist does need to change?
2. What is the support for the new function in the original netlist?
3. What is the new function for the changed net?

For the following example in Table 1, in *G1.v*, if we want to change the function of the wire t as $x \wedge y$, we can create the patch file in Verilog as

1. One output port named t, which t is the wire name in *G1.v* we want to change.
2. Two input ports named x and y, which x and y are the wires we want to use as the support for the new function.
3. An AND gate, which performs the new function.

The patch for *G1.v* is as *patch.v* in Table 1, and the patched netlist is as G2.v.

G1.v	patch.v	G2.v

module top(in0, in1, out);
input in0, in1;
output out;
wire x, y, t;
not n0(x, in0);
buf n1(y, in1);
xor n2(t, x, y);
not n3(out, t);
endmodule
	module top_eco(t, x, y);
	output t;
	input x, y;
	and eco1(t, x, y);
	endmodule
		module top(in0, in1, out);
		input in0, in1;
		output out;
		wire x, y, t;
		not n0(x, in0);
		buf n1(y, in1);
		// xor n2(t, x, y);
		and eco1(t, x, y);
		not n3(out, t);
		endmodule

Table 1. Example of a patch for using an AND gate for wire t.

There would be the case that we want to change some wire x but use the wire x as the support. For the example in Table 2, in *G1.v*, if we want to add a *NOT* gate at the output of wire x, we

need to declare x as the output port where wire x in *G1.v* performs the new function, but its support wire is the original function of x. In this case, we use input variable x_in, the wire name appending the notation "_in," to represent the wire x in *G1.v*. Therefore, patch.v can be described in Table 2. We will describe in detail about the patch and applying the patch in Sections 4.4 and 4.5.

G1.v	patch.v	G2.v

module top(in0, in1, out);
input in0, in1;
output out;
wire x, y, t;
not n0(x, in0);
buf n1(y, in1);
xor n2(t, x, y);
not n3(out, t);
endmodule
	module top_eco(x, x_in);
	output x;
	input x_in;
	not eco1 (x, x_in);
	endmodule
		module top(in0, in1, out);
		input in0, in1;
		output out;
		wire x, y, t;
		not n0(**x_in**, in0);
		not eco1(x, x_in);
		buf n1(y, in1);
		xor n2(t, x, y);
		not n3(out, t);
		endmodule

Table 2. Example of a patch for adding an inverter to the output of x.

3. Contest Objective

This problem aims to develop a practical, efficient, and accurate functional ECO program with behavior change guidance. In this contest, we provide industry-scale benchmarks to evaluate contestants' algorithms. We expect novel ideas can be inspired and can be applied in industrial tools. We also hope that this problem can facilitate innovative research on functional ECO.

4. Problem Formulation and Input / Output Format

4. 1 Problem Formulation

To simplify the problem, we use the gate-level netlists generated from RTL elaboration of specifications to represent the old specification *R1* and new specification *R2.* Therefore, the problem formulation is: Given the gate-level netlist *R1* that is generated from the RTL elaboration of original design specification, the gate-level netlist *R2* that is generated from RTL elaboration of new design specification, and the gate-level netlist *G1* which is synthesized and optimized from *R1*, contestants' program needs to generate and minimize patch for *G1* such that the patched *G1* is functionally equivalent to the netlist *R2*. We compare the cost of the patch for evaluating the winner.

4.2 Program requirement

The requested program must be run on a Linux system. The time limit of running each test case is 3600 seconds. Parallel computation with multiple threads or processes is not allowed. The executable file should be named "eco" and accept four arguments:

```
./eco R1.v R2.v G1.v patch.v
```

Argument *R1.v*, *R2.v,* and *G1.v* are the input files, and *patch.v* is the output file for your program. For the testcase in this problem, the netlists generated from the original specification and new specification will be named *R1.v* and *R2.v,* respectively. The implementation netlist synthesized and optimized from the original specification will be named *G1.v*.

4.3 Input format and Output format

The netlists *R1.v*, *R2.v*, *G1.v* and *patch.v* are the combinational netlists composed of

- primitive gates (and, or, nand, nor, not, buf, xor, xnor)
- constant values (1'b1, 1'b0)

The netlist is in Verilog language and in the flattened view. The format is as follows:

```
module <modulename> ( <name0>, <name1>, … );
input <name0>, <name1>, …;
output <name0>, <name1>,…;
wire <name0>, <name1> , …;
<gate type> <name>( <name0>, <name1>, … );
…
endmodule
```

The *module name* for *R1.v*, *R2.v* and *G1.v* are "*top*". The *module name* for *patch.v* is "*top_eco*". There is no wire name ending with "_in" in *R1.v*, *R2.v*, and *G1.v*.

4.4 Patch Semantic

The output patch describes how *G1*.v is changed:

- **Output ports**: The set of wires in *G1.v* that needs to be attached to new functions.
- **Input ports:** The set of wires in *G1.v* that needs to be used as the support of new functions. If a wire needs to be both input and output of the patch module, you need to add appending string "_in" for the input port, e.g., x_in in patch.v in Table 2.
- **Patch function:** The new function is described by the logic contained in the patch module. To simplify the problem, the patch netlists only allow the primitive gates {*and, nand, or, nor, xor, xnor, not, buf*}.

4.5 Applying Patch

The contestant needs to apply the patch and validate the patch correctness. Applying patch is a process followed by the three steps:

The patch output ports are composed of the wires in G1. Given the patch with output ports o_1, o_2, ..., o_m and input ports i_1, i_2, ..., i_n.

1. Disconnect the wire o_1, o_2, to o_m in *G1.v*.
2. Reconnect the wire o_1, o_2, to o_m with the patch output o_1, o_2, …, o_m.

3. Connects the input port $i_1, i_2,...i_n$ of the patch with $i_1, i_2,...i_n$ in *G1.v.* The input port with postfix "_in" would be connected to their original wire. For example, *a_in* in patch.v would be driven by *a* in *G1*.

5. Evaluation

Programs will be evaluated by the following criteria as the same in CAD Contest 2012[13]:

1. **Correctness**: Correctness is a prerequisite. A test case will be treated as failed if the result is wrong.
2. **Time limit:** The main program must finish or generate the output circuit within 3600 seconds; otherwise, the team gets a score of 0 for that test case.
3. **Cost of the patch**: we compare the cost of the patch for evaluating the score for each case.

$$cost\ of\ patch = number\ of\ wires + cost\ of\ primitives + number\ of\ constant$$

The cost of a primitive = number of inputs of the primitive - 2. For example, a 4-input AND gate is of cost 2. A buffer/inverter is of cost -1. The wire is the wire variable of Verilog used in the patch, and the wire includes the input and output. Constant means *1'b0* and *1'b1*. Note that even if multiple *1'b0s or 1'b1s* are used, we only count them once. That is, the number of constants will always be equal to or less than 2 (*1'b0+1'b1*).

4. **Score of a test case**: A team can get a score of $\frac{M}{N}$ for the case where the team gets the cost N for the patch and M is the minimal cost of the valid patches among the teams.
5. **Winner:** The team who gets the highest total score wins the contest. If there is a tie, total runtime is used for tie-breaking.

6. Example

Tables 3, 4, and 5 show a complete example of this problem. Given 3 netlists, *R1.v*, *R2.v*, and *G1.v*, as shown in Table3, you can find that *R1.v* and *R2.v* is similar, and *G1.v* is optimized from *R1.v* and functional equivalent to *R1.v*. Tables 4 and 5 show two different examples of patches, respectively.

R1.v	R2.v	G1.v
module top (a, b, c, o1); input a, b, c; output o1; wire a, b, c, o1, n1; and g2 (o1, a, n1); and g1 (n1, b, c); endmodule	module top (a, b, c, o1); input a, b, c; output o1; wire a, b, c, o1, n1; or g2 (o1, a, n1); and g1 (n1, b, c); endmodule	module top (a, b, c, o1); input a, b, c; output o1; wire a, b, c, o1; and g1 (o1, a, b, c); endmodule

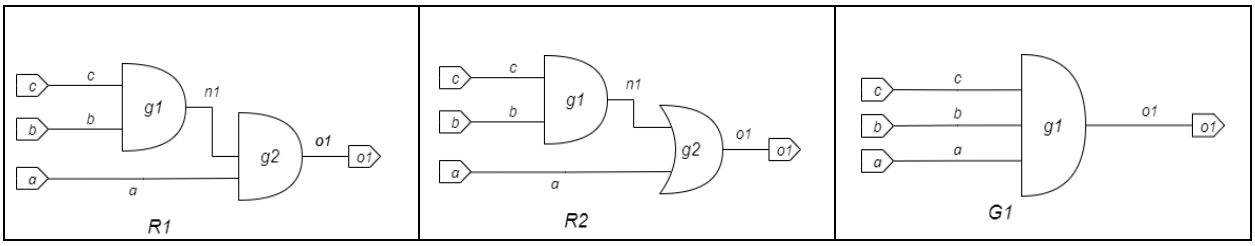

Table 3. The input netlists of the example.

patch.v	Patched G1.v
module top_eco (a, b, c, o1); input a, b, c; output o1; wire a, b, c, n1, o1; or eco_g2 (o1, a, n1); and eco_g1 (n1, b, c); endmodule	module top (a, b, c, o1); input a, b, c; output o1; wire a, b, c; // AND g1 (o1, a, b, c); **or eco_g2 (o1, a, n1);** **and eco_g1 (n1, b, c);** endmodule

Table 4. The output patch and patched netlist for the example in Table3.

patch.v	Patched G1.v
module top_eco (o1, a, o1_in, a_in); output o1, a; input o1_in, a_in wire a, a_in, o1, o1_in; or eco_g2 (o1, o1_in, a_in); buf eco_g1 (a, 1'b1); endmodule	module top (a, b, c, o1); input a, b, c; output o1; wire a, b, c, o1, o1_in, a_in; // AND g1 (o1, a, b, c); **or eco_g2 (o1, o1_1, a);** **buf eco_g1 (a_1, 1'b1);** and g1 (o1_1, a_1, b, c); endmodule

Table 5. Another output patch and patched netlist for the example in Table3.

6.1 Cost of the Patch

For the example in Table 4, the number of wires is 5, including {*a, b, c, n1, o1*}. The cost of primitives is 0 since *eco_g2* and *eco_g1* are two-input primitives (cost = 0). Thus, the cost of the patch is 5.

For the other example in Table 5, the number of wires is 4, which includes {*a, a_in, o1, o1_in*}. The cost of primitive is -1 since *eco_g2* is a two-input primitive (cost = 0) and eco_g1 is a buffer (cost = -1), The number of constants is 1, which is *1'b1*. Thus, the cost of the patch is 4 + (-1) +1 = 4.

A team generating the patch shown in Table 4 can get a score of 0.8, i.e. (4/5), if the minimal cost is 4 among the teams.

6.2 Applying Patch for the Case in Table 5

When applying the patch in Table 5, the first step is disconnecting the wires *a* and *o1* in *G1*, as shown in Figure 3(a). We can regard the wires *a* and *o1* as disconnected and divided into two parts, e.g., *a (in)* and *a (out)* parts, for ease of explanation.

Next, we connect the output of *eco_g1* to the wire *a* (*in*) and connect the output of *eco_g2* to the wire *o1* (*in*), as shown in Figure 3(b).

Finally, we connect the input *o1_in* of *eco_g2* to wire *o1* (*out*) and the input *a_in* of *eco_g2* to *a* (*out*), as shown in Figure 3(c).

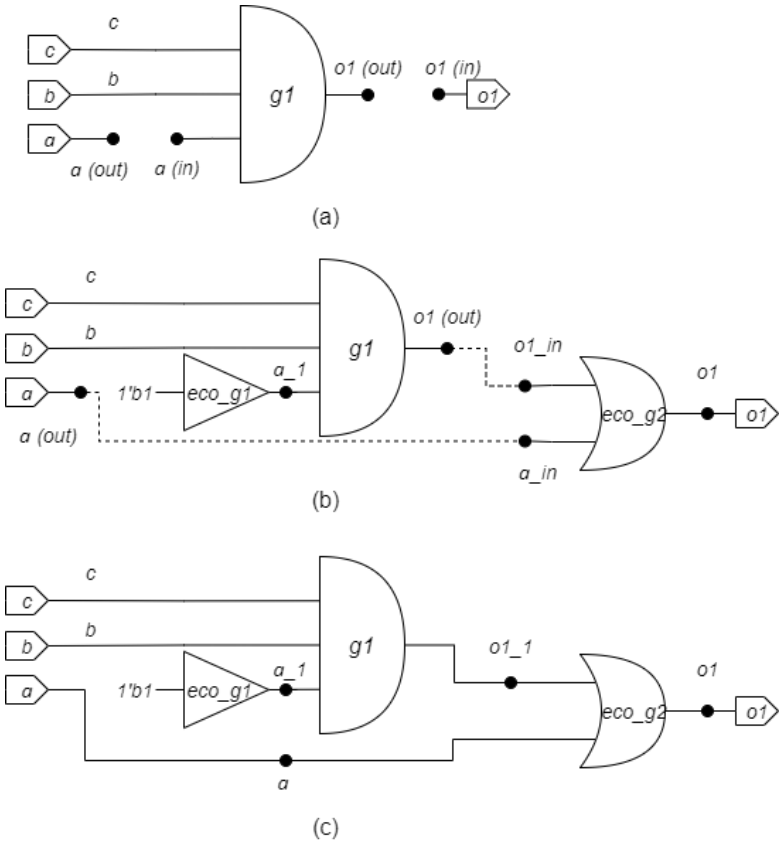

Figure 3. Example of applying patch in Table 5. (a) disconnecting wires *a* and *o1* in *G1*. (b) connecting patch's output port. (c) connecting patch's input ports.

7. Testcases

We will provide several test cases soon. We plan to have the same number of public cases (evaluate in alpha, beta, and final tests) and hidden cases (only for final evaluation).

8. Reference

[1]. H.-T. Liaw, J.-H. Tsaih, and C.-H. Lin, "Efficient automatic diagnosis of digital circuits," *in Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, pp.464 -467, 1990.

[2]. P.-Y. Chung and I. Hajj, "Diagnosis and correction of multiple logic design errors in digital circuits," *IEEE Trans. on VLSI Systems (TVLSI)*, vol. 5, no. 2, pp. 233 –237, June 1997.

[3]. K.-H. Chang, I. L. Markov, and V. Bertacco, "Fixing Design Errors With Counterexamples and Resynthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 1, pp. 184-188, Jan. 2008.

[4]. 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.

[5]. 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 (ICCAD)* – Digest of Technical Papers, 2009, pp. 789-796.

[6]. C.-C. Lin, K-.C. Chen and M. Marek-Sadowska, "Logic synthesis for engineering change," *in Proc. Design Automation Conference (DAC),* pp. 647-652, 1995.

[7]. C.-H. Lin, Y.-C. Huang, S.-C. Chang and W.-B. Jone, "Design and design automation of rectification logic for engineering change," *in Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 1006-1009, 2005.

[8]. 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. Design Automation Conference (DAC),* pp. 146-151, 2011.

[9]. 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," *in Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1567-1572, 2012.

[10]. B.-H. Wu, C.-J. Yang, C.-Y. Huang and J.-H. R. Jiang, "A robust functional ECO engine by SAT proof minimization and interpolation," *in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* pp. 729-734, 2010.

[11]. W. Jong, H.-T. Wang, C. Hsieh, and K.-Y. Khoo, "ICCAD-2012 CAD contest in finding the minimal logic difference for functional ECO and benchmark suite: CAD contest," *in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 342-344, 2012.

[12]. C.-Y. Huang, C.-J. Hsu, C.-A. Wu and K.-Y. Khoo, "ICCAD-2017 CAD contest in resource-aware patch generation," *in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* pp. 857-862, 2017.

[13]. Cadence Encounter Conformal ECO Designer, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/functional-eco/conformal-eco-designer.html