ICCAD-2017 CAD Contest in Resource-aware Patch Generation

Ching-Yi Huang
Cadence Taiwan, Inc
8F., No.2, Lixing 6th Rd.
Science Park, Hsinchu, Taiwan
chingyi@cadence.com

Chih-Jen Hsu
Cadence Taiwan, Inc
8F., No.2, Lixing 6th Rd.
Science Park, Hsinchu, Taiwan
jackyhsu@cadence.com

Chi-An Wu
Cadence Taiwan, Inc
8F., No.2, Lixing 6th Rd.
Science Park, Hsinchu, Taiwan
rockywu@cadence.com

Kei-Yong Khoo
Cadence Design System, Inc.
2655 Seely Avenue
San Jose, CA 95134
khoo@cadence.com

***Abstract*— With a functional Engineering Change Order (ECO) problem, the quality of patch plays an important role in the performance of the patched circuit. In this contest, contestants need to generate *patch functions* that will make two circuits equivalent, while minimizing the *resource cost* of the generated patches. Resource cost is the comprehensive physical cost of all the patches, and minimizing the resource cost implies improving patch quality (timing, power, routing, or area). The resource cost of patches can be modeled as a weighting function with respect to several physical properties of nodes used for patches. In ICCAD 2017 CAD Contest, we have assigned each internal node a reasonable *constant weight* to represent the corresponding physical cost if the node is used for generating patches. Also, the resource cost of the patches is calculated as the *weight summation of patches' support nodes*. This formulation can elegantly identify wanted algorithms for the resource-aware patch generation problem.**

Keywords*— *Engineering Change Order, Patch Generation, Equivalence Checking.

I. Introduction

In a modern design flow, if some functionality has to be changed or functional bugs are found at late stages, restarting the whole synthesis and verification flow is impractical. To save time and cost, automating *Engineering Change Orders (ECOs)* is more practical. As illustrated in Fig. 1, an automated ECO process can identify the differences between the old circuit *F* and the new circuit *G*, and generate a corresponding *patch function* such that *F'* (the resultant circuit after applying the patch to the old circuit *F*) and *G* become equivalent. In addition, patch generation plays an important role in the ECO problem because the quality of patch directly influences the performance of the patched circuit.

In an industrial design flow, the functional ECO tool, e.g., *Cadence Encounter Conformal ECO Designer* [1], has been widely used for industrial cases for years. Although the patch size is an important metric for patch quality, patch generation must consider other physical issues, including timing and power closure, to solve the ECO problem practically.

In academic research, several studies [2-9] have proposed different kinds of algorithms to generate patches. The work has primarily focused on minimizing the patch size. The generated patches, however, may be unusable in industrial problems due to the lack of consideration of physical issues.

Recently, some research addressed the physical issues in functional ECO [10] and some searched for better base functions to generate a target function with better result [11]. In this contest, we will focus on the resource-aware patch generation problem to motivate new ideas for solving the practical ECO problem.

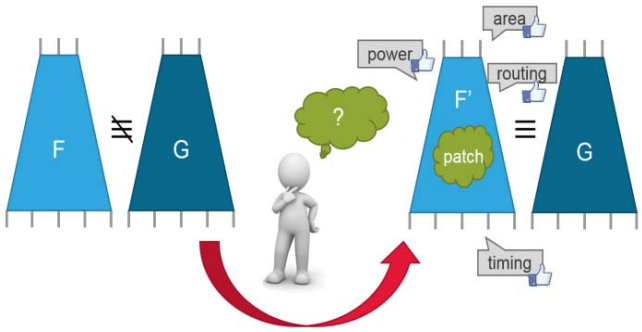


Fig. 1. The resource-aware patch generation problem.

II. Contest Objective

The objective of this contest is to develop a flexible, scalable, and practical resource-aware patch generation algorithm that can be utilized in industry tools. In this contest, we provide benchmarks that are representatives of industrial problems to facilitate the academic research. Although existing work may not provide feasible solutions for these benchmarks, we look forward to inspiring new research into the functional ECO.

III. Problem Formulation and Input/Output Format

Given two circuits *F* and *G*, and the weight information of internal nodes in *F*, contestants need to utilize internal nodes in *F* as supports, called *base nodes*, to generate the *patch functions* with *minimum resource cost* at a specific set of *target points* in *F* such that *F'*, the patched circuit, and *G* are equivalent, as Fig. 2 shows. The resource cost is calculated by the weight summation of the used based nodes.

A. Program Requirement

The requested program must be run on a Linux system. The time limit of running each testcase is 1800 seconds. Parallel computation with multiple threads or processes is not allowed. The executable file should be named "*rpgen*" and accepts five arguments:

./rpgen <F.v> <G.v> <weight.txt> <patch.v> <out.v>

- *<F.v>* and *<G.v>* are input files that describe two circuits *F* and *G* in Verilog, respectively.
- *<weight.txt>* is an input file that describes the weight information of internal nodes in the circuit *<F.v>*.
- *<patch.v>* is an output file that describes the patch.
- *<out.v>* is an output file that describes the patched circuit *F'*. Note that *<F.v>*, *<G.v>*, *<patch.v>*, and *<out.v>* are all combinational circuits with no loop.

B. Input Format

<F.v> and *<G.v>* describe gate-level circuits in Verilog. They have only one module, named top, and contain only primitive gates without hierarchical structure. Target points are indicated by wire declaration with names *t_0*, *t_1*, …, *t_n* in *<F.v>*; target points are floating and are inputs of some gates. The format is:

```
module top (<name0>, <name1>, … ); //F.v
input <name0>, <name1>, …;
output <name0>, <name1>, …;
wire <name0>, <name1>, …;
wire t_0, t_1, …; //target points
<primitive gate type> (<name0>, <name1>, …);
<primitive gate type> (<name0>, t_0, <name1>, …);
…
endmodule
```

```
module top ( <name0>, <name1>, … ); //G.v
input <name0>, <name1>, …;
output <name0>, <name1>, …;
wire <name0>, <name1>, …;
<primitive gate type> (<name0>, <name1>, …);
<primitive gate type> (<name0>, <name1>, …);
…
…
endmodule
```

<weight.txt> describes the weight information of internal nodes of *<F.v>* with the following format:

```
<name0> <weight0>
<name1> <weight1>
<name2> <weight2>
…
```

Node name and the corresponding weight are separated by a space character, and different nodes are described in different lines. Internal nodes without assigned weights have ***Infinite (INF)*** weight.

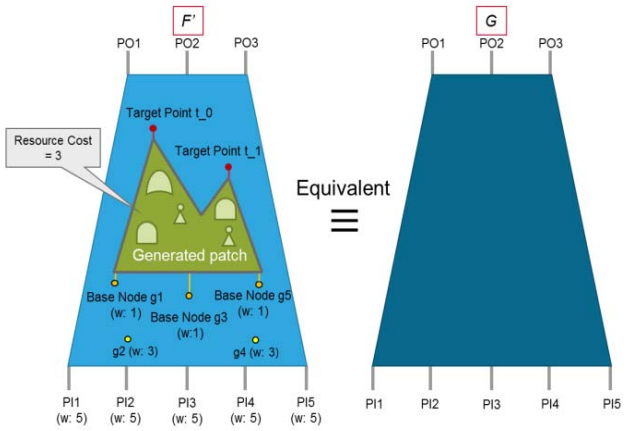


Fig. 2. Problem formulation.

C. Output Format

<patch.v> describes the generated patch. It is a gate-level circuit in Verilog, which has only one module, named *patch*, and only contain primitive gates without hierarchical structure. The format is:

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

Note that only one module can be in *<patch.v>*, even if there are multiple target points, in which case the patch module would have multiple output ports.

<out.v> describes the patched circuit *F'*. The content of *<out.v>* must be the same as *<F.v>*, except for the added patch instance. The patch must be represented as a module instance and must be added at the end of the top module (before the **endmodule** line), as shown in the following example:

```
module top (<name0>, <name1>, …);
input <name0>, <name1>, …;
output <name0>, <name1>, …;
wire <name0>, <name1>, …;
wire t_0, t_1, …; //target points
<primitive gate type> (<name0>, <name1>, …);
<primitive gate type> (<name0>, t_0, <name1>, … );
…
// patch instance
patch p0 (t_0, t_1, …, <name0>, <name1>, …);
endmodule
```

In summary, we check the following items in the output files:

1. Every line in <*out.v*> above the patch instance must be the same as that of <*F.v*> before the **endmodule** line.
2. The patch instance must be at the last line (just before the **endmodule** line) in <*out.v*>.
3. <*out.v*> and <*patch.v*> must follow the format of primitive gate-level circuit in Verilog.
4. There is only one module in <*patch.v*> and named *patch*.
5. The circuit <*out.v*> with <*patch.v*> has to be functionally equivalent to the circuit <*G.v*>.
6. We evaluate the resource cost based on the inputs of the patch instance in <*out.v*>.
7. We evaluate the patch size according to the gate count in <*patch.v*>.
8. <*patch.v*> and <*out.v*> must be combinational circuits with no loop.

IV. Example

Here we use an example to illustrate the problem description and how to calculate the resource cost. In this example, there are two designs <*F.v*> and <*G.v*> and one weight information file <*weight.txt*> for <*F.v*>, as shown in Figs. 3 and 4.

In this example, we can see that there is a target point t_0 in <*F.v*>. Contestants need to generate a patch, represented by a patch file <*patch.v*> and the patched circuit file <*out.v*>, at t_0 such that the patched circuit can be equivalent to the circuit in <*G.v*>. Here we provide four sample outputs and illustrate the calculation of corresponding resource costs.

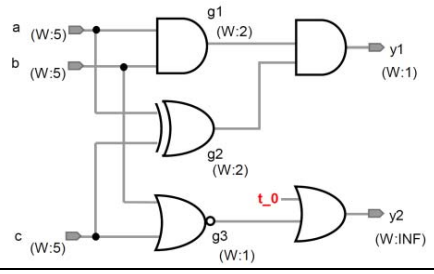
F.v	*weight.txt*
module top (y1, y2, a, b, c); input a, b, c; output y1, y2; wire g1, g2, g3; wire t_0; and (g1, a, b); xor (g2, a, c); nor (g3, b, c); and (y1, g1, g2); or (y2, t_0, g3); endmodule	a 5 b 5 c 5 g1 2 g2 2 g3 1 y1 1
	

Fig. 3. The example of <*F.v*> and <*weight.v*>.

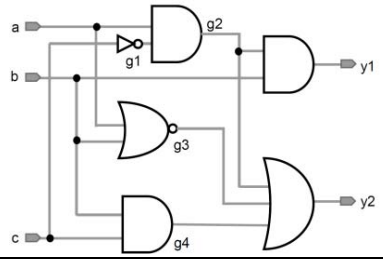
G.v
module top (y1, y2, a, b, c); input a, b, c; output y1, y2; wire g1, g2, g3, g4; not (g1, c); and (g2, a, g1); nor (g3, a, b); and (g4, b, c); and (y1, b, g2); or (y2, g2, g3, g4); endmodule


Fig. 4. The example of <*G.v*>.

Team A, as shown in Fig. 5, uses {a, b, c} as the base nodes to generate the patch at t_0. According to the file <*weight.txt*>, the weights of nodes {a, b, c} are {5, 5, 5}, respectively. Thus, the corresponding resource cost of the patch is 5 + 5 + 5 = 15. Besides, since there are 3 primitive gates in <*patch.v*>, the corresponding patch size is 3.

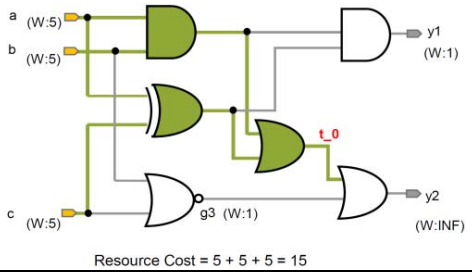
patch.v	*out.v*
module patch (y, a, b, c); input a, b, c; output y; wire w1, w2; and (w1, a, b); xor (w2, a, c); or (y, w1, w2); endmodule	module top (y1, y2, a, b, c); input a, b, c; output y1, y2; wire g1, g2, g3; wire t_0; and (g1, a, b); xor (g2, a, c); nor (g3, b, c); and (y1, g1, g2); or (y2, t_0, g3); patch p0 (.y(t_0),.a(a),.b(b),.c(c)); endmodule
	

Fig. 5. The output files of Team A.

Team B, as shown in Fig. 6, uses {a, b, c} as the base nodes to generate the patch at t_0. According to the file <*weight.txt*>, the corresponding resource cost of the patch is

15. However, since there are 4 primitive gates in <*patch.v*>, the corresponding patch size is 4.

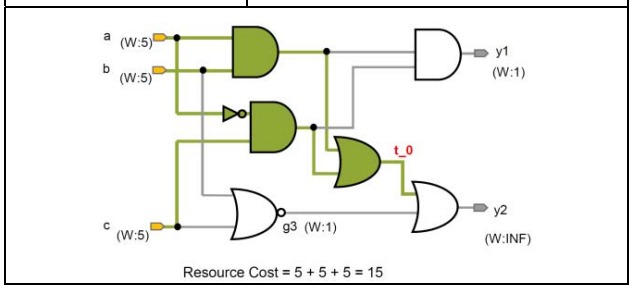
patch.v	*out.v*
module patch (y, a, b, c); input a, b, c; output y; and (w1, a, b); not (w2, a); and (w3, w2, c); or (y, w1, w3); endmodule	module top (y1, y2, a, b, c); input a, b, c; output y1, y2; wire g1, g2, g3; wire t_0; and (g1, a, b); xor (g2, a, c); nor (g3, b, c); and (y1, g1, g2); or (y2, t_0, g3); patch p0 (.y(t_0),.a(a),.b(b),.c(c)); endmodule
	

Fig. 6. The output files of Team B.

Team C, as shown in Fig. 7, uses {g1, g2} as the base nodes to generate the patch at t_0. According to the file <*weight.txt*>, the weights of nodes {g1, g2} are {2, 2}, respectively. Thus, the corresponding resource cost of the patch is 2 + 2 = 4. Besides, the corresponding patch size is 1 because there is only one gate in <*patch.v*>.

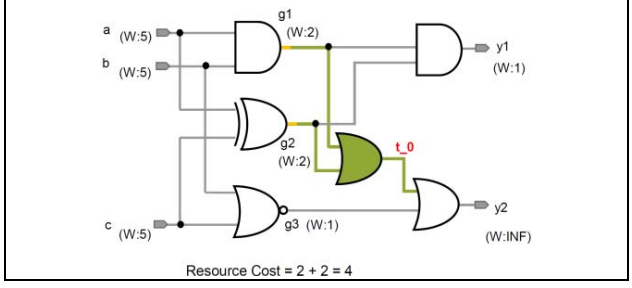
patch.v	*out.v*
module patch (y, a, b); input a, b; output y; or (y, a, b); endmodule	module top (y1, y2, a, b, c); input a, b, c; output y1, y2; wire g1, g2, g3; wire t_0; and (g1, a, b); xor (g2, a, c); nor (g3, b, c); and (y1, g1, g2); or (y2, t_0, g3); patch p0 (.y(t_0),.a(g1),.b(g2)); endmodule
	

Fig. 7. The output files of Team C.

Team D, as shown in Fig. 8, uses {g1, g2} as the base nodes to generate the patch at t_0. Thus, the corresponding resource cost is 4. However, the circuit in <*out.v*> with patch <*patch.v*> is not equivalent to the circuit in <*G.v*>. Thus, it's an invalid patch and Team D will not get any score for this case.

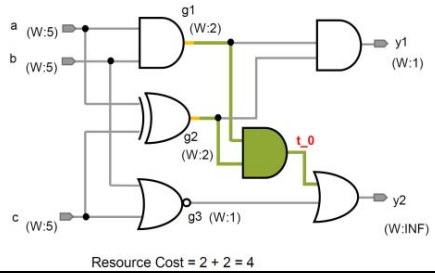
patch.v	*out.v*
module patch (y, a, b); input a, b; output y; and (y, a, b); endmodule	module top (y1, y2, a, b, c); input a, b, c; output y1, y2; wire g1, g2, g3; wire t_0; and (g1, a, b); xor (g2, a, c); nor (g3, b, c); and (y1, g1, g2); or (y2, t_0, g3); patch p0 (.y(t_0),.a(g1),.b(g2)); endmodule
	

Fig. 8. The output files of Team D.

V. Evaluation Methodology

For each case, the result will be evaluated by the following criteria:

1. Correctness: The program has to be finished normally. Output files must comply with the output format. The resultant circuit <*out.v*> with the patch <*patch.v*> has to be functionally equivalent to <*G.v*>. Any violation gets score of 0 for that testcase.
2. Time limit: The program has to be finished within 1800 seconds; otherwise, the team gets score of 0 for that testcase.
3. Scoring according to the rank: The teams that pass the above correctness and time limit checking get their scores by their ranks for that testcase. The teams with the rank 1~6 will get scores of {10, 7, 5, 4, 3, 2}, respectively. The remaining teams get a score of 1. Teams are ranked based on the following criteria:
   a. We rank teams according to the resource cost, i.e., the total weight of used base nodes. The smaller is better.
   b. If the resource cost ties, we rank the teams by the patch size, i.e., the gate count in a patch. The smaller is better.

c. If teams still tie, we rank them according to the runtime. The less is better.

For example in Section IV, Team D gets score of 0 because the patched circuit is not equivalent to the circuit in <*G.v*>. Next, the resource cost of {Team A, Team B, Team C} is {15, 15, 4}, respectively, and Team C is the first since Team C has the least resource cost. However, because Team A and Team B have the same resource cost, we rank them according to the patch size. The patch size is evaluated by the gate count of a patch, so the patch sizes of Team A and Team B are 3 and 4, respectively. Thus, Team A is the second and Team B is the third. Finally, the scores of {Team A, Team B, Team C, Team D} are {7, 5, 10, 0}, accordingly.

The team earning the highest accumulated scores for all the benchmarks wins the contest.

VI. Problem Guidance

1. Contestants need a simple Verilog primitive gate-level parser.
2. Contestants need an equivalence checker to verify the result.
3. Contestants can utilize, modify, or apply ideas from academic papers regarding the ECO problem and patch generation. Refer to REFERENCES.
4. To inspire contestants, we provide a baseline algorithm for single-fix problems and provide a tutorial with the above example. Refer to the ICCAD 2017 CAD Contest website [12].

VII. Benchmark Suite

To address different difficulties and sizes of the industrial problems, we created the ICCAD 2017 CAD Contest benchmark suite from ISCAS, ITC99 in IWLS 2005 benchmarks [13], OpenCore [14], LGSynth'93 [15], and some datapath parts from complex industrial designs. The circuits in RTL were synthesized into gate-level netlists through *Cadence Genus™ Synthesis Solution* [16]. *Genus™ Synthesis Solution* provides the advanced engine to create the most optimized netlists.

To cover the industrial ECO problems, we introduced several ECO scenarios to the benchmark suite. The ECO scenarios we used for the benchmark suite considered different numbers of fix points, different fix points' distances to primary inputs/primary output, and different problem sizes. Moreover, we assigned each internal node of a design a reasonable constant weight based on the considered ECO scenario. For this Contest problem, the constant weight is sufficient to represent the corresponding physical cost if the node is used as the base node. The final weight distributions of considered ECO scenarios can be roughly categorized into the following types:

T1: Distance-aware distribution – A.

T2: Distance-aware distribution – B.

T3: Path-aware distribution.

T4: Locality-aware distribution.

T5: The distribution composed of T1 + T3.

T6: The distribution composed of T2 + T3.

T7: The distribution composed of T1 + T4.

T8: Highly mixed and undulating distribution

The benchmark suite includes a half of single-fix problems and a half of multiple-fix problems with 2, 4, 8, and 12 target points. Different types of weight distribution are also evenly distributed to the benchmarks, which can effectively evaluate the capability and flexibility of contestants' algorithm for minimizing the resource cost when generating the patches. Table I summarizes the detailed information of the benchmark suite. We released 14 cases as the open cases for contestants to evaluate and design the algorithm, and used 6 hidden cases in the final evaluation.

TABLE I. Information of the benchmark suite.

	# of Target Points				
Distribution	1	2	4	8	12
T1	u1, u3, u7, u12			(u17)	
T2	(u15)	u5, u10			
T3	u8		u9		
T4	u4, (u18)			u11	
T5 (T1 + T3)	u13				
T6 (T2 + T3)					u14
T7 (T1 + T4)			(u19)		
T8	u2	u6, (u16)	(u20)		

*Hidden cases are in bracket symbol.

VIII. Conclusion

In the functional ECO problem, the quality of patches plays an important role in the performance of the patched circuit. In ICCAD 2017 CAD Contest, we formula a patch generation problem with the awareness of resource cost. Contestants need to generate patch functions that make two circuits equivalent while minimizing the resource cost of the generated patches, which implies better patch quality. In this contest, we provide benchmarks that are representatives of industrial problems with several ECO scenarios to facilitate academic research on this domain. We expect that contestants can provide creative and practical solutions which can be utilized in industry applications and bring more research interests.

References

[1] 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.

[2] 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.

[3] 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.

[4] S. Krishnaswamy, H. Ren, N. Modi and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," International conference on Computer-Aided Design (ICCAD) – Digest of Technical Papers, 2009, pp. 789-796.

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

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

[7] K.-F. Tang, C.-A. Wu, P.-K. Huang and C.-Y. Huang, "Interpolation-based incremental ECO synthesis for multi-error logic rectification," Design Automation Conference (DAC), 2011, pp. 146-151.

[8] 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.

[9] 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 techniques," International Conference on Computer-Aided Design (ICCAD), 2010, pp. 729-734.

[10] A.-C. Cheng, I. H.-R. Jiang and J.-Y. Jou, "Resource-aware functional ECO patch generation," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 1036-1041.

[11] A. Petkovska, D. Novo, A. Mishchenko and P. Ienne, "Constrained interpolation for guided logic synthesis," International Conference on Computer-Aided Design (ICCAD), 2014, pp. 462-469.

[12] Problem A of ICCAD 2017 CAD Contest, http://cad-contest-2017.el.cycu.edu.tw/Problem_A/default.html.

[13] IWLS 2005 Benchmarks, http://iwls.org/iwls2005/benchmarks.html.

[14] OpenCore Benchmarks, http://opencores.org/.

[15] ACM/SIGDA benchmarks, https://people.engr.ncsu.edu/brglez/CBL/benchmarks/Benchmarks-upto-1996.html.

[16] Cadence Genus™ Synthesis Solution and Conformal LEC, http://www.cadence.com/products.