ICCAD-2012 CAD Contest in Finding the Minimal Logic Difference for Functional ECO and Benchmark Suite

CAD Contest

WoeiTzy Jong[1], Hwei-Tseng Wang[1], Chengta Hsieh[2], Kei-Yong Khoo[2]

[1]Cadence Taiwan, Inc., 2F, No. 6-5, Du Sing Rd., Industrial Based Science Park, Hsinchu 300, Taiwan

[2] Cadence Design Systems, Inc., 2655 Seely Avenue, San Jose, CA 95134

{wjong, janew, chengtah, khoo}@cadence.com

ABSTRACT

In this paper, an automatic functional Engineering Change Order (ECO) problem is proposed. The contestants need to implement programs to identify the logic difference of the old netlist and the newly synthesized netlist, which agrees with new specification. How the logic difference can be presented as patch and how we measure the patch quality is explained. The program that can find the minimal patch wins.

Categories and Subject Descriptors

B.7.2 [Hardware, Integrated Circuits]: Design Aids

General Terms

Design, Verification.

Keywords

ECO, Synthesis, P&R.

1. INTRODUCTION

As technology node advances, designs are getting more and more complicated. As the result, bugs are often found in late design stage. Besides, the product specification may also change in late design cycle. Re-running the whole design flow is impractical [2, 3]. Especially when the mask has been made, the metal only change can save time and money significantly. To address this issue, Engineering Change Order (ECO) is proposed. In the manual ECO flow (c.f. Figure 1), the designer first implements the change in RTL, and performs verification tasks to make sure the new fix agree with the new specification. Then the designers try to correlate the old netlist (G1) with the old RTL and try to implement the functional ECO based on the learned correlation. If the modified netlist (G3) is functionally equivalent to new RTL, the ECO is done. Otherwise one more manual iteration is needed to fix the human error. The manual process is very tedious and unpredictable. The contest is trying to find an efficient program that can identify the minimal logic difference automatically.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA.

2. PROBLEM DESCRIPTION AND INPUT/OUTPUT FORMAT

To analyze the difference between G1 and G2 to implement functional ECO, the automatic ECO flow is proposed (c.f. Figure 2). In [1]-[6], various methods have been proposed to rectify G1. The program, gate_diff, should be implemented to take G1 and G2 as input and find the functional ECO solution. The solution will be stored in a Verilog file as patch file. The concept is same as Unix diff and patch. The details about the patch format and patch application process are described in Section 3. The grade will be given based on the patch size. The program should also

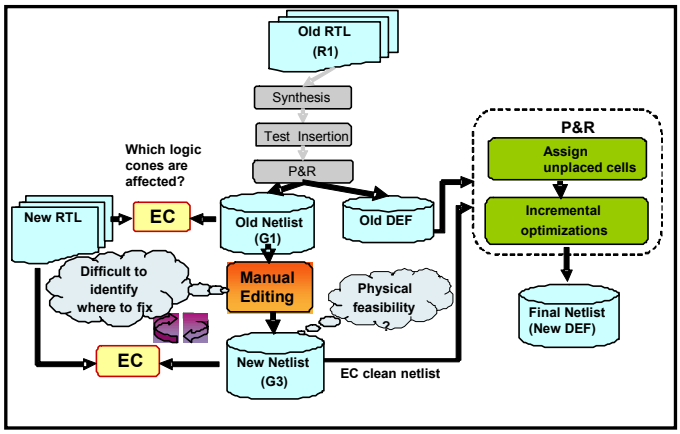

Figure 1. Manual ECO Flow

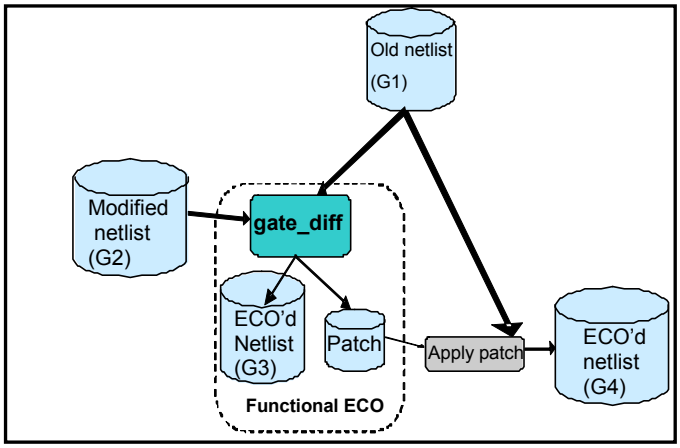

Figure 2. The automatic ECO flow

g1.v (input file)

```
module top(x1, x2, x3, o1);
  input x1, x2, x3;
  output o1;
  wire x1, x2, x3;
  wire o1;
  wire n2;
  and g1 (n2, x1, x2);
  and g3 (o1, x3, n2);
endmodule
```

g2.v (input file)

```
module top(x1, x2, x3, o1);
  input x1, x2, x3;
  output o1;
  wire x1, x2, x3;
  wire o1;
  wire n2;
  and g1 (n2, x1, x2);
  or g3 (o1, x3, n2);
endmodule
```

patch.v (output file)

```
module top_eco(o1,n2,x3);
  output o1;
  input n2,x3;
  wire o1, n2, x3;
  or eco_i1(o1, n2, x3);
endmodule
```

g3.v (output file)

```
module top(x1, x2, x3, o1);
  input x1, x2, x3;
  output o1;
  wire x1, x2, x3;
  wire o1, n2;
  and g1 (n2, x1, x2);
  or eco_i1(o1,n2,x3);
endmodule
```

Figure 3. The patch example

generate the ECO'd netlist G3. The G3 should be logically equivalent to G2 and the only difference to G1 is recorded in the patch file. The patch will also be used by our independent tool to judge whether the patch can generate a good ECO'ed netlist G4 that is logically equivalent to G2.

- **Input/output format**

Both input and output are in Verilog format.

Input:

- g1.v: G1 netlist (c.f. Figure 2), a Verilog gate-level netlist with primitives only.
- g2.v: G2 netlist (c.f. Figure 2), a Verilog gate-level netlist with primitives only.

Both G1 and G2 contain only one design module, named "top", without any design hierarchy and DFFs/DLATs. To simplify the problem, only Verilog primitives are allowed in g1.v and g2.v. Program gate_diff only needs to implement gate-level parser targeting modules with only port/wire declarations and primitives.

Output:

- patch.v: A Verilog file contains the difference between G1 and G2. The format will be explained in Section 3. Only one patch module, named "top_eco", should be contained in the patch.v. How the patch can be patched into G1 will be explained in Section 3.
- g3.v: G3 netlist in Figure 2, a Verilog gate-level netlist with primitives only. No module instances are allowed, i.e. the g3.v should just be a design module "top" without any hierarchy.

g1.v (input file)

```
module top(x1, x2, x3, o1);
  input x1, x2, x3;
  output o1;
  wire x1, x2, x3;
  wire o1;
  wire n2;
  and g1 (n2, x1, x2);
  or g2 (o1, x3, n2);
endmodule
```

g2.v (input file)

```
module top(x1, x2, x3, o1);
  input x1, x2, x3;
  output o1;
  wire x1, x2, x3, o1;
  wire  n1, n2, n3;
  and g1 (n2, x1, x2);
  not g3 (n1, x1);
  and g4 (n3, n1, x3);
  or g2 (o1, n3, n2);
endmodule
```

patch.v (output file)

```
module  top_eco(x3,  x3_in,
x1);
  output x3;
  input x3_in, x1;
  wire n1, x3, x3_in, x1;
  not eco_g3(n1, x1);
  and eco_g4(x3, n1,
    x3_in);
endmodule
```

g3.v (output file)

```
module top(x1, x2, x3, o1);
  input x1, x2, x3;
  output o1;
  wire x1, x2, x3;
  wire o1, n1, n2, n3;
  and g1 (n2, x1, x2);
  not eco_g3 (n1,x1);
  and eco_g4 (n3,n1,x3);
  or g2 (o1, n3, n2);
endmodule
```

Figure 4. This example shows how the patch input port name with postfix "_in" works.

3. PATCH AND APPLYING PATCH

The examples are shown in Figure 3 and Figure 4. The g1.v (G1 in Figure 2) is the Old Netlist. The g2.v (G2 in Figure 2) is the New Netlist.

The program should identify the gate-level difference between g1.v and g2.v and generate the patch file (patch.v).

The patch module is used to contain the delta logic between the G1 and G3. To simplify the problem, the module is always named as "top_eco".

Patch module name: top_eco

Patch output ports: the set of nets in G1 that needs to be attached to new functions. In Figure 3, the patch module has only one output port, namely "o1".

Patch input ports: the set of nets in G1 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, rename the input port to wire_name +"_in" (c.f. x3 of patch.v in Figure 4). In Figure 3, the input ports are n2 and x3.

Patch function: the new function is described by the logic contained in the patch module. In Figure 3, the patch contains only one OR gate. To simplify the problem, only the primitive, the built-in gate defined in IEEE Standards 1364-2001, is allowed to build the function.

The patch will be attached to G1 by disconnecting the nets in G1 which have same names as patch output ports. Then these nets are connected to the ports of same names in the patch. Finally, the

Name	# Inputs	# Outputs	# Primitives
Design1	88	3	54
Design2	16	1	101
Design3	32	6	161
Design4	19	32	171
Design5	20	13	272
Design6	28	15	223
Design7	69	32	338
Design8	37	32	770
Design9	57	71	775

Figure 5. Statistics of ICCAD 2012 Contest benchmarks.

input ports of the patch will be connected to the nets of same names in G1.

4. BENCHMARK

9 different designs are chosen as the contest benchmarks. The design statistics are shown in Figure 5. The following properties hold for each design.

- No hierarchy, only one flat module named top.
- Verilog primitives only.
- No sequential cells.
- No unconnected pins.
- No power or grounded nets.
- No timing information.

There are total 19 tests created by injecting random logic changes to the designs. Please note that multiple logic changes may exist in one test.

5. EVALUATION

Programs will be evaluated by the following ordered criteria,

1. Correctness: The correctness is a prerequisite. It will be treated as failed if the result is wrong.
2. The program should finish within 1 hour for one testcase on the specified machine. It will be treated as failed if the patch is not generated within 1 hour.

   The cost of a primitive = number of inputs of the primitive - 2. For example, 4 input AND gate will have cost 2. Buffer/Inverter will have cost -1.

   Patch size = number of wires + cost of primitives in the patch. Ultimately, this problem is to minimize the patch size.

   In Figure 3, the patch size is 3. In the example of Figure 4, it is also 3.

   The score for a given testcase is normalized according to the following.

   1) The score of 0 is given if patched is not generated or not correct.

   2) The program generates the minimal patch size gets score 1.

   3) Given the best patch size, among all the contesters, is N. A program generates patch size M will get score N/M.
3. 19 tests will be performed and the scores for each program will be recorded. The program with highest accumulated score wins.

If there is a tie, the program with less total run time wins.

6. CONCLUSIONS

To help contesters to focus on the core problem, no sequential logic is presented in the benchmarks. Only primitive gates without timing information are used to construct the netlist. Then, a method to justify the quality of the patch is proposed. We hope that this benchmark suite can bring up the interest of the academic community for this topic. Therefore, the state-of-the-art for automatic functional ECO can be advanced.

7. REFERENCES

[1] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of digital errors with PRIAM", Proc. IEEE/ACM Int. Conf. Computer-Aided Design, pp.30 -33 1989.

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

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

[4] Eric Lehman, Y. Watanabe, Joel Grodstein and H. Harkness, "Logic Decomposition During Technology Mapping", IEEE, Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp.813-834, vol. 16, no. 8, August 1997.

[5] S.-Y. Huang, K.-C. Chen and K.-T. Cheng, "AutoFix: A Hybrid Tool for Automatic Logic Rectification," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, pp. 1376-1384, vol. 18, no. 9, pp. 1376-1384, Sept. 1999.

[6] Bo-Han Wu, Chun-Ju Yang, Chung-Yang Huang, Jie-Hong Roland Jiang: A robust functional ECO engine by SAT proof minimization and interpolation techniques. Prof. IEEE/ACM International Conference on Computer Aided Design (ICCAD), Nov. 2010.