CAD Contest 2022 Problem A: Learning Arithmetic Operations from Gate-Level Circuit

B08901061 Yung-Chin, Chen, B08901062 Chia-Hsiang, Chang, B08901048 Yu-Chen, Chen
Department of Electrical Engineering, National Taiwan University

Abstract — Reverse engineering, which aims to reconstruct high-level description of a given gate-level netlist, is essential for maintaining design security and detecting malicious third-party hard IP cores. In this work, we proposed a method for determining if-else conditions using the Quine-Mccluskey procedure. Our design solves netlists consisting of general operations efficiently using equivalence checking. Our design derives the high-level description of the netlist by the following steps: (1) Converting the netlist into a graph. (2) Performing equivalence checking between the netlist and the output function candidates to form a two dimensional boolean table. (3) Applying the Quine-Mccluskey procedure is to find the minimum cover of the table. (4) Finding the control signals according to the cover and solving its high-level logic. This design reaches maximum reduction for 8 test cases provided by CAD contest out of 20 while maintaining extremely low runtime.

I. PROBLEM INTRODUCTION

The technologies of netlist reverse engineering help chip engineers solve the problems of heavy reliance on third-party resources and protect the products from malicious attacks like hardware Trojans. The goal of netlist reverse engineering is to develop a tool to extract high-level functions on a given gate-level circuit. In this problem, given a flatten Verilog netlist which contains only primitive gates (and, or, nand, nor, not, buf, xor, xnor), we have to extract the datapaths (arithmetic operations) from the netlist and translate them into equivalent RTL expressions. The operations include addition, subtraction, multiplication, conditioning, and/or, bit selection and concatenation [1].

The input file is a Verilog format file, containing primitive gates, wires names and constant values. The output file should still be in Verilog format, and the names of original declarations must not be changed. The evaluation of the solution is based on "reduction rate", which equals to

$$(1 - \frac{Cost}{Gate\ count\ in\ input\ netlist}) \times 100\%$$

The *cost* is defined on the original problem description file. The greater the reduction rate is, the higher score the tool gets. Moreover, the generated RTL must be functionally equivalent to the input netlist, or the tool will not be evaluated.

II. RELATED WORK

A. Reverse Engineering

Some approaches develop netlist analysis tools to automate the procedure of reverse engineering. The research in [2] constructs an automation tool named REFSM to catch circuit control logic. REFSM employs a 3-SAT solver to transform boolean expressions into finite state machine transition graphs. The procedures include finding non-State Registers, evaluation of Boolean expressions, Construction of FSM and Splitting FSM.

The research in [3] uses both structural and functional analysis to generate high-level netlist. The strategy includes two stages. The first stage uses topological and functional analysis to find potential module boundaries. Then, the second stage identifies these potential modules by functional analysis.

B. Quine-McCluskey Algorithm

Some researchers use the Q-M algorithm to improve the requirements of reversible logic synthesis. The research in [4] applies and improves the Q-M algorithm. The Q-M algorithm involves 3 steps:

Step 1: finding prime implicants.
Step 2: finding essential prime implicants.
Step 3: If the essential prime implicants cover the function, express the function by the prime implicants. If not, find some other prime implicants that are needed to cover the function.

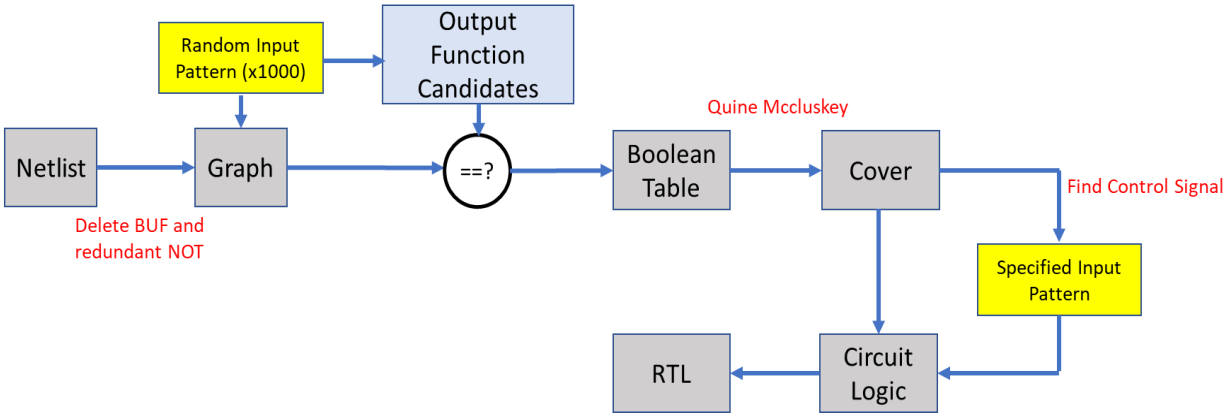


Figure 1. Dataflow of our proposed reverse engineering solver

In step 3, we first need to find the dominance relationship between prime implicants. We delete the implicants that are dominated, and check if there're new essential prime implicants. Repeat this process until finding a cyclic core, and find some necessary remaining prime implicants to cover the function.

III. SOLUTION

A. Overview

Figure 1. presents the dataflow of our proposed reverse engineering solver. Firstly, The given netlist is transformed into a graph, where redundant BUFs and NOTs are removed. Secondly, random patterns are generated to check whether the output value of the netlist is equal to the predetermined output functions. A two-dimensional boolean table can thus be constructed. Subsequently, The Quine-Mccluskey procedure is applied on the boolean table to find a cover. Next, control signals are heuristically determined. We then generate input patterns by varying the control signals to see how they affect the terms in the cover. the high-level behavior of the circuit can therefore be determined. Lastly, the RTL file is written out.

B. Solution Details

1) Graph

In order to conveniently process the circuit, the given netlist is first converted to a graph. Each gate, primary input wire, primary output wire is represented by a vertex. Each wire is represented by a single-direction edge, pointing from its output gate to its input gate. An example of gate conversion is shown in Figure 2. Every gate stores information of its operation type, its input gate(s) and output gate. This representation allows us to traverse the netlist efficiently.

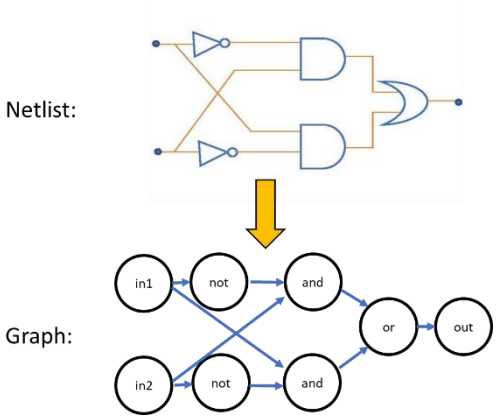
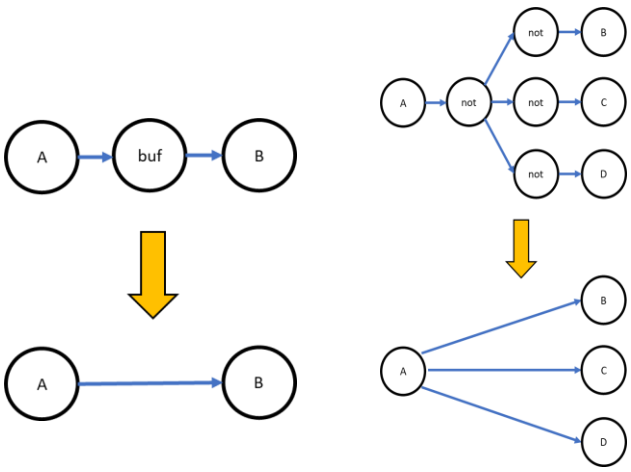


Figure 2. Graph converted from netlist



(a) (b)

Figure 3. Remove redundant gates. (a) Remove BUFs. (b) Remove redundant NOTs.

Since BUFs and consecutive NOTs are logically redundant for a netlist, we remove these gates to simplify the graph. The operations are described in Figure 3.

2) Equivalence Checking

Figure 4. shows the equivalence checking scheme. Given the graph representing the netlist and a set of

output function candidates, the input pattern is randomly generated. We compare the output of the netlist with outputs of the output function candidates to form a one-dimensional boolean table that takes the length of the number of function candidates. We repeat the process for 1000 times to construct a two-dimensional table.

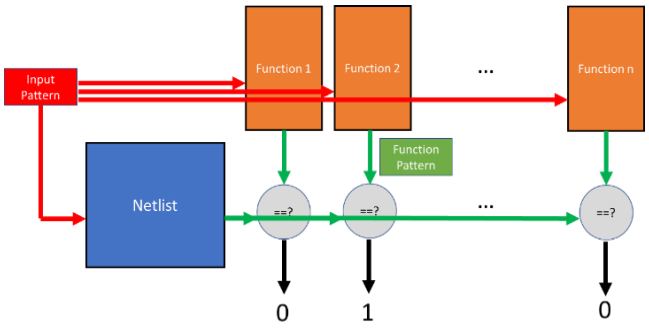


Figure 4. Equivalence Checking Scheme

3) Output Function Candidates

In order to perform the equivalence checking scheme above, output function candidates have to be determined beforehand. We set our function candidate set *F* in terms of the following formula:

$$F = \vec{S} \cdot \vec{T} + constant$$

$\vec{T}$ consists of products of inputs. Our selection of $\vec{T}$ varys with respect to the number of inputs *n*, which is shown as follows:

$if\ n = 2, \vec{T} = (a, b, ab, a^2, b^2, a^3, b^3)$
$if\ n = 3, \vec{T} = (a, b, c, ab, bc, ca, a^2, b^2, c^2, abc, a^3, b^3, c^3)$
$if\ n = 4, \vec{T} = (a, b, c, d, ab, ac, ad, bc, bd, cd)$
$if\ n = 5, \vec{T} = (a, b, c, d, e, ab, ac, ad, ae, bc, bd, be, cd, ce, de)$
$else, \vec{T} = (a, b, c, d, e, \ldots\ldots, z, \ldots)$

$\vec{S}$ has the same length as $\vec{T}$ and each element takes on value values of {-1, 0, 1} to denote the existence and sign of the term. The constant term is calculated by setting all inputs to zero and observing the output of the netlist. For instance, if *n*=2 and $\vec{S}$ = {1, -1, 0, 0, 0, 0, 0} with constant = 3, we check whether the netlist has the same function as a-b+3.

The size of the function candidate set is $3^{|T|}$, which grows exponentially with respect to |T|. In order to reduce the space complexity, we check functions that only have less than 5 terms.

4) Quine-Mccluskey Procedure

Given a two-dimensional boolean table, where each column represents a function candidate and each row represents an input pattern, we have to find the target function that satisfies all input patterns. In regard to target functions that include if-else conditions, a single function candidate cannot satisfy all input patterns. For these cases, we find a set of function candidates that jointly satisfy all input patterns. This is equivalent to finding a minimum cover of the boolean table. For example, {0, in1*in2, in2*in3, in3*in1} is a valid cover of the boolean table for Figure 5., we thus know that the target function is composed of these four function terms controlling by if-else conditions.

To find the minimum cover of the table, we apply Quine-Mccluskey Procedure. We keep searching for dominant columns and update the table after removing the dominant columns and dominated rows from it. If no column dominance exists in the table, we heuristically choose the column that satisfies the most input pattern cases.

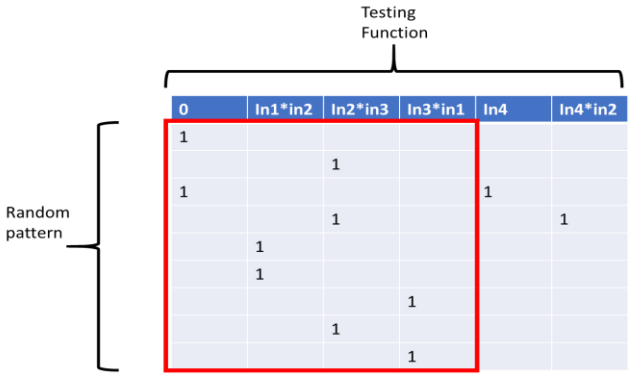


Figure 5. Finding cover of Boolean table

5) Finding Control Signals

For a given if-else condition function, inputs should either be part of control signals or be part of controlling function terms under general situation. If an input does not contribute to any controlling function terms in the cover, it should be a control signal. Hence, we assume that control signals are inputs that do not appear in the cover. For instance, in Figure 5., the cover is {0, in1*in2, in2*in3, in3*in1}, where in4 is not used, we therefore consider it a control signal.

6) Determining the RTL Logic

After finding both the control signals and controlling function terms, we have to observe how control signals control the terms. Here, we apply the same equivalence checking scheme as (2) except that we only vary the input pattern of the control signals and keep other inputs fixed. the RTL logic of the given netlist can hence be determined.

IV. RESULTS

Table 1. shows the performance of our design under the test cases provided by CAD contest 2022. This design successfully solves 8 given netlists with more than 94% of reduction rate out of 20 cases. The solved cases are maximumly simplified and consist of only high-level operations. The runtimes of most cases are less than a minute, which is a lot smaller than 8 hours (the runtime constraint of this problem in the contest). Some output results are shown below in Figure 6.

Testcases	Reduction rate(%)	Runtime(s)
01	**94.2857**	2.075
02	**96.5517**	2.147
03	**99.0415**	2.211
04	**99.862**	64.928
05	2.7907	0.355
06	**99.9395**	4.51
07	**99.7540**	0.719
08	12.7967	20.111
09	40.4	6.639
10	**99.7743**	4.915
11	**99.8329**	4.38
12	41.614	11.756
13	20.623	0.057
14	0	2.300
15	42.6182	1.73
16	20.2497	1.322
17	11.4073	0.066
18	15.625	0.343
19	54.1646	0.179
20	18.8586	0.831

Table 1. Performance on test cases

```
module top(in1, in2, in3, out1);
input wire [31:0] in1;
input wire [31:0] in2;
input wire [31:0] in3;
output wire [64:0] out1;
assign out1 = in2 + in3 + in1 * in1;
endmodule
```

```
module top(in1, in2, in3, out1);
input wire [18:0] in1;
input wire [18:0] in2;
input wire [18:0] in3;
output wire [19:0] out1;
assign out1 = in1 + in2 + in3 + 102;
endmodule
```

Figure 6. Some output results of reverse engineering solver

V. FUTURE WORKS

This design regard the netlist as a blackbox and observe only the output of the netlist. Since the space complexity grows exponentially with the number of function candidate terms we considered, not all the possible terms can be considered if the netlist have many inputs. Thus, the performance degrades fast when the number of inputs increase. Intending to reduce the number of inputs we considered at a time, partitioning the netlist is required.

We proposed a method to solve huge netlists using the above reverse engineering solver and netlist partitioning scheme. First, we calculate transitive fan-ins of all gates and find sub-netlists that only contains limited fan-ins. Second, we apply our reverse engineering solver to the sub-netlists to find their high-level description. Then, the outputs of the sub-netlists are treated as the inputs of the new sub-netlists. Last, The process ends when the whole netlist is converted to high level logic description. For instance, in Figure 7, the netlist is partitioned into four sub-netlists. The reverse engineering solver is applied to each sub-netlists. By partitioning the netlist, the space complexity issue is resolved and more function candidate terms can be considered in the equivalence checking step, which can further enhance the solver's performance.

Nonetheless, the partitioning scheme still have the following issues: (1) The output ports of the sub-netlists might have different orders. We have to check whether there exists a permutation of outputs that align with the outputs of any function candidates. (2) It might be possible that high-level descriptions for a given sub-netlist does not exist since there can be simplifications of gates when connecting two or several high-level operations together.

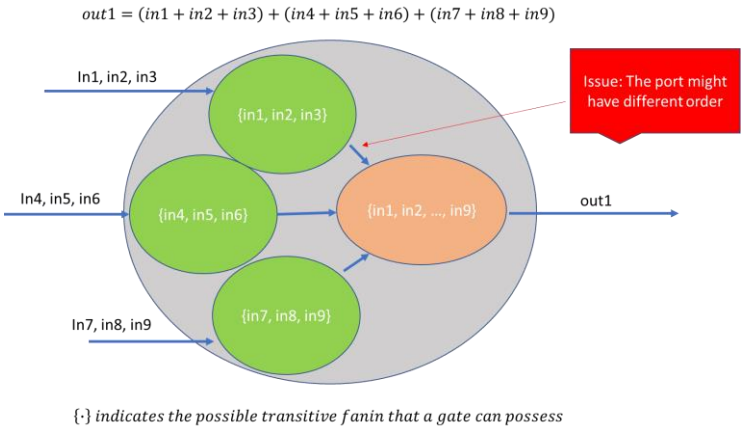


Figure 7. Solve the huge netlists by dividing sub-netlists

VI. CONCLUSION

We presented a reverse engineering solver that efficiently reconstructs the high-level logic given its gate-level netlist. In addition, we proposed a method that

finds and describes the if-else condition of the target function using Quine-Mccluskey procedure. Our work first converts the netlist into a graph, then performs equivalence checking between the netlist and the output function candidates to form a two dimensional boolean table. Quine-Mccluskey procedure is applied to find the minimum cover of the table. The cover is later used for determining the control signals and controlling terms. Last, high-level logic of the netlist is determined and written to the output file. This work successfully solves 8 test cases provided by CAD contest out of 20 while keeping a fast runtime. Partitioning the given netlist and applying our reverse engineering solver to each sub-netlist is left for future work.

VII. JOB DIVISION

Yung-Chin, Chen

- Code Implementation
  - Convert netlist to graph
  - Design Output function candidate sets
  - Compare netlist with output function candidates
  - Find control signals given minimum cover
- Other
  - Propose Quine-Mccluskey procedure for solving if-else condition
  - Propose netlist partitioning for reverse engineering solver (Future Work)
  - Make presentation slides and deliver oral presentation
  - Report writing

Chia-Hsiang, Chang:

- Code Implementation
  - Delete redundant BUF and NOT
  - Apply Quine-Mccluskey procedure to find minimum cover
  - Compute netlist high-level logic and write output file
- Other
  - Propose Quine-Mccluskey procedure for solving if-else condition
  - Propose netlist partitioning for reverse engineering solver (Future Work)
  - Report writing

Yu-Chen, Chen:

- Code Implementation
  - Calculate reduction rate

REFERENCES

[1] “CADContest 2022 Problem A” Problem description, released on 13 May, 2022

[2] T. Meade, S. Zhang and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), 2016, pp. 655-660.

[3] P. Subramanyan et al., "Reverse Engineering Digital Circuits Using Structural and Functional Analyses," in IEEE Transactions on Emerging Topics in Computing, vol. 2, no. 1, pp. 63-80, March 2014.

[4] S. Zhao, C. Wang and J. Sun, "Improvement and implementation of quine-McCluskey algorithm for synthesis of reversible logic circuits," 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2017, pp. 640-642.