ICCAD 2016 Contest: Static Timing Analysis

Kai-Sheng Cheng
Department of Electrical Engineering
National Chung Cheng University
168 University Rd., Chiayi 621, Taiwan
coal511464@gmail.com

***Abstract*—In this project work, our approach is following the problem that released by ICCAD contest organization [1] which try to solve the STA (Static Timing Analysis) problem in design automation field. The problem can be reduced to find the total path in the given circuit and a circuit simulator (Timing graph based), then analysis the critical path e.g. Time slacks, signal rising/falling, true/false path. The proposed try to refer statistical STA based on path-based algorithm which carry out depth-first traversal at critical paths. We skip some part of detail in this problem but solve the critical issue that covering the general goal.**

Keywords: Design automation, Static Timing Analysis.

I. Introduction

STA is one of research topic in electronic design automation which can report critical circuit path and contribute to analyzing transition in each logic unit along the path. Moreover, STA also focusing on checking timing violation that validate timing performance by checking all possible paths. Unfortunately, while conventional STA are very successful but still have limitations.

Nowadays in IC design industry, the complexity and scale raise constantly. Thus a better time performance in STA tool can increase the efficiency of VLSI design flow. Wang, *et al.* [2] proposed a block-based SSTA method which evaluates the false paths to increase the accuracy on timing analysis. Another approach is by employ multi-core machine or multi thread library intends to archive better time performance, but it's a challenge to implement a parallel system flow. Shen, *et al.* [3] proposed a GPU based parallel computing techniques for accelerating SSTA that using PCA (Principal Component Analysis). The problemD in contest is willing to lead the designer to determine the timing requirement which relative to a well-known false path problem. The STA program in contest is for combinational logic circuits under multi-core computing machine, and an input file that specific under Verilog gate-level netlist is given. The evaluation will consider to several benchmarks while verify the program. Furthermore, the Floating-mode rule that depict in contest website can be an reference perform in the program. For those given input circuit only for combinational logic circuits thus optimizing the sequential circuits is not necessary.

According to problem description, designer is going to develop a STA application as well as finding the true path with different input vectors. The designed program should support the Floating-mode in problem description tables, the table shows each cases of true path in combinational logic circuits. In addition, the problem also set a limitation for the predefined time slack. The evaluation process will conclude the final result and the run time spent also concerned.

II. ProblemD Description and solution

In this section, we will make a simple notice to show that which part of implementations are finished and which are not but show the future works. This practice project work almost finished in March. For some purpose, the author is not willing to continue the rest implementation. **More importantly, the contest problem had been modified occasionally during April or May, thus the recent released input files or test cases might not supported to the paper implementation.** The following work are using standard C++11 ISO/IEC 14882:2011 or later standard without any open source or any commercial STA solvers.

A. Terminology

Following the Verilog gate-level netlist, we set the primary input as *Src* named by alphabetically exclude the upper case "M", and primary output as *Sink* with notation "M". For example, {A∼H,L∼Z}[0∼INT_MAX] and M[0∼INT_MAX] are acceptable input/output format receptively. The program parse function is currently not supporting another notations of Src/Sink. All logic unit is defined as AND2, OR2, XOR2, NAND2, NOR2, NOT1, the given circuit are not contain register or any buffer. The time slack in this problem is not a critical issue but more like a threshold or constrain. Each of logic unit will consuming one time slack (gate delay is 1 ns), see Figure 1.

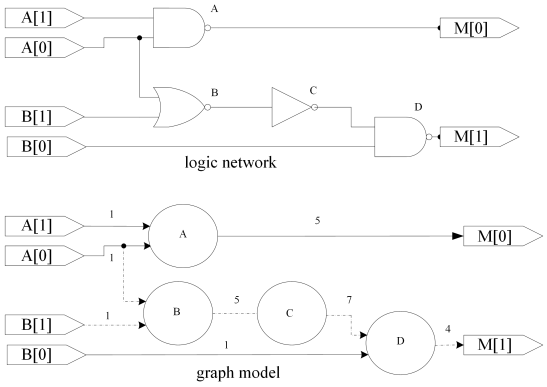

Figure 1: An example of logic model and graph model.

We parse the gate-level file into a 2D vector as well as a timing graph form, see Figure 2. The nodes represent each input/output from logic units, wires are connecting orderly shows the each step in a simulation process. We create a simulator function as a vector table by updating the value. For finding all paths, here we using depth-first search in a timing graph, thus time complexity in this case is n^m, where n indicate the branch factor and m is depth.

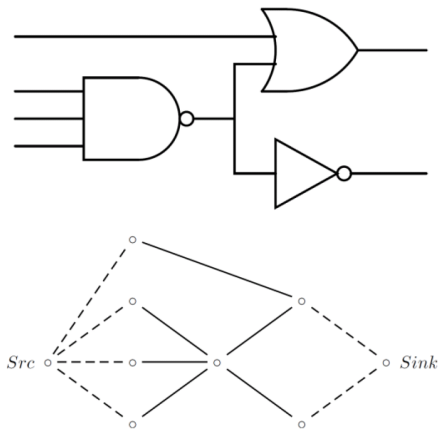

Figure 2: An example of timing graph.

B. Detecting False Paths

This part is not finish yet but with a timing graph, we can back-track the neighborhood nodes in order to detect the false section. Following the instruction in contest website, we only have to updating the simulator once determine whether a section wire are not a true paths, then abort and re-initial the simulator. A path with a long delay will not contribute to the circuit if it is not a true path. In other hand, it is false path. If all the input are not controlling value which case is defined to true path either. The true path cases that is, a section circuit is then set logic value on output side already as well as can imply by an assigned value, see Figure 3.

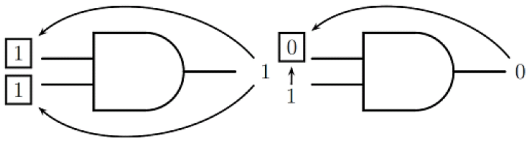

Figure 3: An example of implication.

C. Function Declarations

We here to show the critical functions design, the project had created a class **ProblemD** which as follow:

```
class ProblemD{
//Parameters:
//Circuit description parsed from verilog file.
vector<vector<string>> parse;
vector<vector<string>> allpath; //For store ↩
    allpath.
vector<vector<int>> sim;  //For timing graph and ↩
    simulator.
vector<vector<int>> _allinvec; //Input
vector<vector<int>> _alloutvec; //Output

public:
//Functions:
//——Neutral section———//
Skip...

//——Find all path section———//
bool find_next(vector<pair<int, int>> &pos, int ↩
    row, int col, string tar);
void find_path();

//——Simulator section———//
void _make_inout(char *pch, vector<int> &↩
    _inout_vec, bool isOut);
void Veri_simulator(); //A simulator function
bool logic_output(string func_str, vector<int> &↩
    vec);
void SimTable_update(string tar, bool in); //↩
    Update the simulator
bool check_2DrowReady(vector<vector<int>> &vec);
bool check_1DrowReady(vector<int> &vec);

}
```

III. Experiments and Conclusion

The test case are resealed by ICCAD contest website. As previous mentions, the dataset format which contain in project document is not exact the same as ICCAD released. There have minor change in the dataset but not include circuit description and functions. The circuit layout is just same as ICCAD released. **Each logic unit cost one time unit, but the implementation doesn't consider any Timing Constraint which means we are actually finding the total path. For real problem solving, we can abort a current thread while reaching the Timing Constraint. For a instance, in case2 show in table I, the Timing Constraint is 43 that pretty smaller than gate-level numbers, thus we don't have to find all path for a real contest problem.**

Table I: The public Benchmarks.

Case Name	Gate Numbers	Input Numbers	Output Numbers	Timing Constraint	Slack Constraint
case1	17	4	4	none	none
case2	413	61	26	43	10
case3	95	8	9	31	6
case4	276	42	21	45	6
case5	11	4	4	none	none

Table II: The experiment result for finding total path and generate a single simulation result.

Case Name	Total Paths	Find Total Path duration (s)	A Simulation duration (s)
case1	20	0.09	0.005
case2	6985	134.72	0.13
case3	2080	4.41	0.01
case4	1801	10.23	0.06
case5	16	0.06	0.004

IV. Acknowledgement

Immeasurable appreciation and deepest gratitude for the Prof. Po-Hung Lin's support. This project work is for the course which held by Prof. Po-Hung Lin, the faculty in Department of Electrical Engineering, National Chung Cheng University.

V. License

The source code/paper are available in Github [4]. The author are not willing to publish this article paper, also the Github URL which show in Reference may not available anymore in some days future. Although it is very welcome to someone who intend to using the code or have any distribution for personal use.

References

[1] ICCAD 2016 Contest http://cad-contest-2016.el.cycu.edu.tw/Problem_D/default.html

[2] Sying-Jyan Wang and Tsung-Huei Tzeng and Katherine Shu-Min Li, "Fast and accurate statistical static timing analysis," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2555-2558, June, 2014.

[3] Yiren Shen and Jiang Hu, "GPU acceleration for PCA-based statistical static timing analysis," *IEEE International Conference on Computer Design (ICCD)*, pp. 674-679, October, 2015.

[4] Source code and paper url https://github.com/ram4996/EDA