Combinational Logic Loop Analysis in Digital Circuit Simulation

Chenyuan Chu
Peking University
China
2200012860@stu.pku.edu.cn

Yitian Sun
Peking University
China
2200012836@stu.pku.edu.cn

Boxin Zheng
Peking University
China
2200012879@stu.pku.edu.cn

Abstract

Combinational logic loops in digital circuits can cause oscillations, disrupting functionality and halting simulations. These loops are challenging to analyze with static timing tools, often leading to infinite loops in simulators. This paper introduces a novel method, *reverse detection*, to efficiently detect combinational logic loops, identify those prone to oscillation, and determine the minimum number of registers needed to break them. We also propose an approach to eliminate oscillations without registers and a mechanism for detecting oscillations in the original circuit. These methods are scalable, making them suitable for large-scale circuit designs. It is worth noting that the performance of our algorithm is enhanced by a factor of 7 through the introduction of parallel computing. The source code is available at https://github.com/FileTransferrr/EDA-Project.

Keywords: digital circuit design, combinational logic loops, reverse detection, circuit transformation, scalability

1 Introduction

Combinational logic loops in digital circuits cause signals to oscillate indefinitely, preventing the circuit from reaching a stable state. This disrupts functionality, leads to unpredictable outputs, and undermines system reliability and performance. Additionally, such loops increase power consumption and may interfere with other modules. Eliminating these loops is crucial for ensuring stability and correctness in digital circuit design.

The rest of the paper is organized in the following order: Section 2 details the methodology for detecting oscillations in combinational logic circuits, identifying the necessary conditions for oscillation, and determining the minimal number of registers required to eliminate all potential oscillations. Section 3 introduces a general approach to transforming circuits to prevent oscillations under any possible input, without converting combinational logic circuits into sequential ones. Section 4 presents the results of our proposed methods. Section 5 concludes the paper.

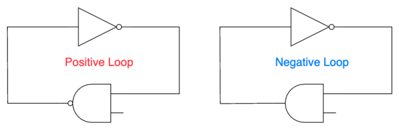

Figure 1. Example of a positive loop and a negative loop

2 Oscillation Detection

2.1 Preparation

To account for the interactions among multiple logic loops, it is necessary to convert a Verilog file into a graph representation and process it in units of *Strongly Connected Components (SCCs)*. To achieve this, we employ *Tarjan's algorithm* Tarjan [3].

2.2 Positive Loop and Negative Loop

We determine a certain loop as a positive or negative loop according to the number of inverters in the loop. If the number of inverters is even, the loop is positive; otherwise, it is negative. It is worth noting that a inverter is not necessarily a *NOT* gate, but can be any gate that inverts the signal, such as a *NAND* gate. A simple example of a positive loop and a negative loop is shown in Figure 1.

2.3 Oscillation Detection

Any combinational logic circuit can be implemented using fundamental logic gates, such as *AND2, OR2, NOT1*, and *NAND2*. For positive logic loops, it is evident that they cannot sustain oscillation. However, negative logic loops do not always exhibit sustained oscillation due to interactions among loops within the same SCC. Recognizing the overestimation in the number of loops capable of sustained oscillation, we implemented a detection methodology. This approach first identifies the necessary conditions for oscillation and then applies these conditions iteratively in a reverse-detection algorithm. Consider a detected negative loop. Given a condition that the signal on the corresponding wire must satisfy, there are three possible scenarios:

1. **The signal originates from the input to the SCC**: In this case, the condition can always be satisfied.
2. **The signal comes from another negative loop within the same SCC**: Here, the condition cannot be satisfied, as a fixed signal within a negative loop is impossible.

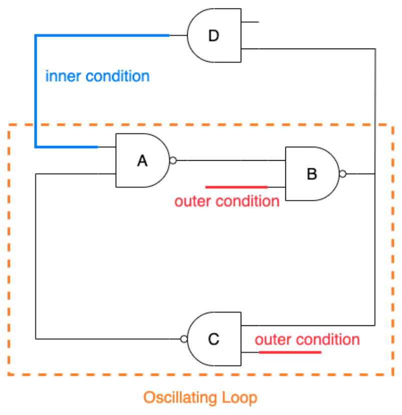

Figure 2. Definition of inner condition and outer condition

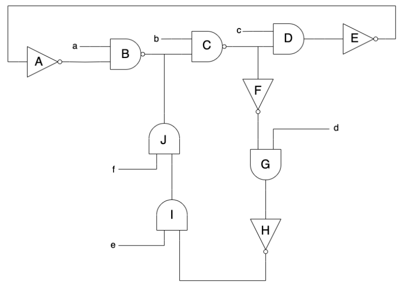

Figure 3. An SCC for Example

3. **The signal originates from a positive loop within the same SCC**: In this scenario, backward detection is applied recursively.

For example, consider the SCC shown in Figure 2. We define the following for an oscillating loop: if the condition of a certain gate within the loop is determined by a signal originating outside the SCC, we refer to it as an *outer condition*. Conversely, if the condition is influenced by signals from within the same SCC, we refer to it as an *inner condition*. Furthermore, for an oscillating loop, the condition required for the SCC to oscillate is proven to be unique. The proof is presented below. Consider the SCC derived from a specific test case, as shown in Figure 3, where all conditions within the oscillating loop are outer conditions. In this context, we denote logic gates with capital letters and input signals outside the given SCC with lowercase letters. The loop condition of interest is to determine the specific inputs to these outer conditions under which the given SCC exhibits oscillation. In the combinational logic circuit shown, the outer conditions are represented by a, b, c, d, e, f. We aim to demonstrate that the vector of outer conditions required for oscillation is unique. For the negative loop composed of gates C, F, G, H, I, J, the necessary condition for this loop to be oscillating is given by $b = 1, d = 1, e = 1, f = 1$(the influence of other loops is momentarily ignored). However, consider the positive loop composed of gates A, B, C, D, E, the signal a should be logic 1, otherwise the output of gate B is fixed by 1, which violates the definition of an oscillating loop. Now focus on the signal c, it can only be logic 1, otherwise the output of gate D is fixed by 0, the output of gate E is fixed by 1, the output of gate A is fixed by 0, and the output of gate B is fixed, which also violates the definition of an oscillating loop in the end.

Table 1. Outer Conditions of Different Gates

Gate Type	Outer Condition (Logic Value)
AND2	1
OR2	0
NAND2	1

2.4 Reverse Detection

Now we promote this theory to any other SCC, which we call *reverse detection*.

definition: We define that the 2-input gates whose off-path input is an outer condition and output directly enters another negative loop in a positive loop (if exists) as the **Backwards Gates** (BGs) and the other 2-input gates with external signal input but do not directly connect another negative loop as **Don't Care Gates** (DCGs).

If an SCC contains multiple loops, they must share some gates. Using the reverse detection method, we can determine all the outer conditions of *BGs*. For *DCGs*, if the outer condition leads to a fixed output, the state of the positive loop becomes fixed, and the logic values of shared edges between the positive and negative loops are also fixed, which contradicts the definition of an oscillating loop. Therefore, the outer conditions of *DCGs* must result in uncertain outputs. The outer conditions for the possible gate types in this problem are summarized in Table 1.

2.5 Break Oscillation

In this section, the concept of partial oscillation is proposed. We define the oscillating part of the given SCC as OSC-SCC. We can extract the OSC-SCC from the given SCC, and the problem is transformed into deleting the minimum number of nodes so that no loop exists in OSC-SCC. Now we can discuss all possible scenarios:

1. **Only one negative loop in an SCC**: Then randomly break an edge in this loop.
2. **More than 1 negative loop in an SCC**: The in-degree of all the nodes in OSC-SCC is no more than 2 because all the gates are 2-input. Therefore it is impossible that more than 1 negative loop share a single edge.

Now the problem is transformed into finding out the gates whose both inputs are in the given SCC. The process of solving this problem is shown in Figure 4.

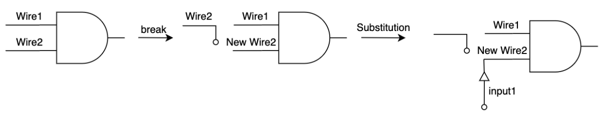

Figure 4. Flow for solving the problem

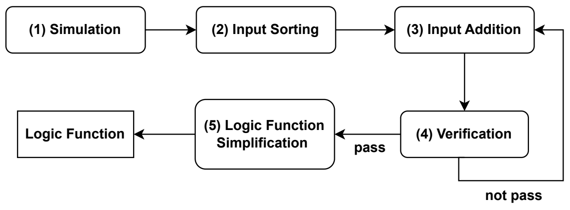

Figure 5. flowchart

3 Circuit Transformation

In Section 2.5, we have discussed how to break the oscillation in the given SCC. Now we will discuss how to transform the original circuit to a new one that meets the following three requirements without introducing sequential devices.

3.1 Requirement

This part focus on transforming the original circuit to a new one, which meets the following three criteria:

1. It can **avoid oscillation** given any possible inputs.
2. Wires in the original circuit have a **one-to-one mapping** in the new circuit.
3. An output signal can **predict oscillation** in the original circuit.

3.2 Basic intuition

We treat the first two requirements as a single task, referred to as *Loop Elimination*, while the third requirement is addressed as a separate task, *Addition of Detection Signals*. Our approach is straightforward:

1. Disconnect specific wires in the original circuit to eliminate potential oscillation loops;
2. To ensure the circuit's functionality remains intact, we introduce auxiliary circuits that replicate these connections, using the overall circuit's input as their input;
3. To minimize the size of the auxiliary circuits, we select a subset of the overall circuit's input to serve as the input for these auxiliary circuits;
4. The construction of oscillation detection signals follows a similar approach.

3.3 Complete Flow

Interestingly, we come up with a general process based on the aforementioned approach which can be applied to both tasks. The flowchart is shown in Figure 5.

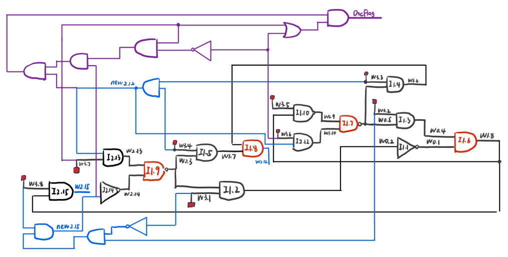

Figure 6. Final Circuit Diagram

We will now provide an explanation of the flowchart:

1. First, we need to use simulator such as iverilog.
2. The input wires are sorted based on importance:
   a. Task 1: Importance is determined by the distance to disconnected wires, always located at the intersection of oscillating loops. Distance is measured in gate hops (1 per gate), with ties broken by ASCII order.
   b. Task 2: Importance is based on the distance to core gates, located at critical junctions of oscillating loops. Distance is measured similarly, with ties also resolved by ASCII order.
3. Top-ranked wires are added to the auxiliary circuit's logic function, aiming to minimize input variables while satisfying simulation results. Using all inputs guarantees correctness but increases complexity, so we prioritize minimal input usage.
4. To verify the logic function, we test if selected inputs can consistently represent a wire's value. If all possible combinations yield consistent target wire values in simulation, they form the minimal input set. Otherwise, a new input is added, and the process repeats.
5. Once the minimal input set is determined, the complete logic function is constructed by summing the minterms from simulation results and simplifying using methods like Quine-McCluskey McCluskey [1] or ESPRESSO McGeer et al. [2].

3.4 Implementation

Following the above process, we successfully transform the original circuit which is oscillating into a new circuit withpout introducing registers. The final circuit diagram is shown in Figure 6, where the purple part represents the auxiliary circuit for the oscillation detection signal *OscFlag*, and the blue part is the auxiliary circuit used to supplement the disconnected wires.

4 Results

4.1 Settings

All testcases were tested on a laptop with an Intel i9-13900HX CPU. To speed up the computation, we used 20 of the 32

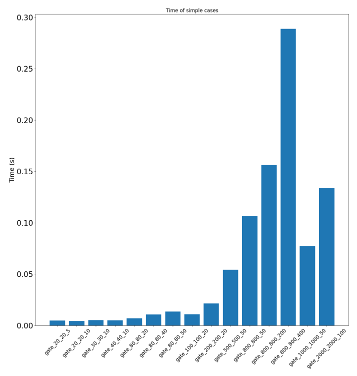

Figure 7. Test Results For Simple Cases

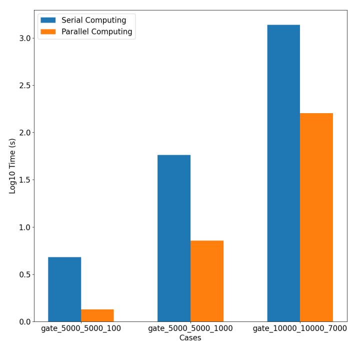

Figure 9. Enhenced Performance by Parallel Computing

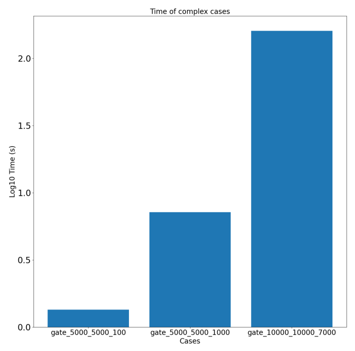

Figure 8. Test Results For Complex Cases

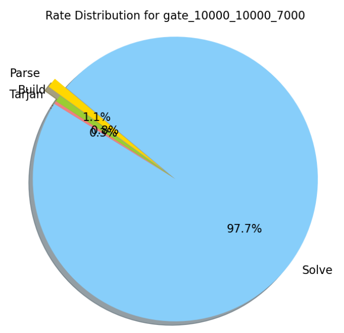

Figure 10. Time Percentage Distribution

threads for parallel computing. For all test cases, we classified them into two categories: simple and complex. The largest simple testcase contains 32,000 logic gates, while the complex testcase contains more than one million logic gates.

4.2 Results

It is worth mentioning that all test cases can get 100% solution accuracy under our algorithm. The most complex of the simple testcases can be completed in **milliseconds**, while the most complex testcases can be completed in about **150 seconds** with the optimization of our algorithm. The test results are shown in Figure 7 and 8. Moreover, the performance of our algorithm is enhanced by **over 7 times** through the introduction of parallel computing. The comparison is shown in Figure 9. For the largest test case, we made a pie chart to analyze the time percentage consumed by each subtask, which is shown in Figure 10. We found that the oscillation detection algorithm consumed the vast majority of the time. However, for such a large-scale combinational logic circuit, a solution time of about 150 seconds is completely acceptable.

5 Conclusion

This paper presented a novel reverse detection approach for detecting and resolving combinational logic loops in digital circuits. By efficiently identifying oscillation-prone loops and proposing minimal modifications to eliminate them, our method ensures circuit stability and scalability for large designs. Additionally, we introduced a circuit transformation technique to prevent oscillations without altering functionality, coupled with an effective oscillation detection mechanism. The results validate the approach's accuracy and efficiency, offering practical solutions for enhancing digital circuit design and EDA tool development.

References

[1] E. J. McCluskey. 1956. Minimization of Boolean functions. *The Bell System Technical Journal* 35, 6 (1956), 1417–1444. https://doi.org/10.1002/j.1538-7305.1956.tb03835.x

[2] Patrick McGeer, Jagesh Sanghavi, Robert Brayton, and Alberto Sangiovanni Vincentelli. 1993. ESPRESSO-SIGNATURE: A new exact minimizer for logic functions. In *Proceedings of the 30th international design automation conference*. 618–624.

[3] Robert Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160. https://doi.org/10.1137/0201010