# 電機資訊學院 2022 作 BRAIN PLUS HAND 實作專題競賽

# Verilog Simulation Optimization via Instruction Reduction

隊伍編號：EECS009　組員：樊明膀、劉家豪、許鎧博


## Abstract

Since the complexity of a SOC increases exponentially, functional verification becomes a time-consuming problem in the design cycle. To reduce the main reason contributing to the low simulation efficiency on CPU time, contestants are required to facilitate the simulator by decreasing the number of continuous assignments and bit–selections. The optimized Verilog file must be syntactically correct, inviolate the given limitations, and be functional equivalent to the input Verilog file. Any third-party open-source are not allowed in the program.

Under the constraints, this project provides a data structure to save the parsing instructions rather than common graphs. This data structure similar to a binary tree can represent the circuit directly without further transformation, reducing the complexity during the optimization. In addition to implement traditional optimization methods, this project provides novel strategies to further improve simulation efficiency. After the “Final Test”, we won the ICCAD awards out of all contestants.


## Problem Formulation

Use a C++ program to minimize the total number of assignments and bit/part-selects in the given Verilog and outputs the optimized Verilog file with the same functionality.
Input Format: A combination circuit with two elements in one Verilog assignment and a tb module.
Output Format: Allow to optimize the circuit, combine the output and add additional temporary wire assignments, but need to conform to the constraints and don’t lose the functionality.
Constraint: Backus-Naur Form (BNF) is adopted here as the regulation to LHS and RHS.

```
module dut (out, in);          <LHS> ::=          <RHS> ::=                    <ITEM> ::= <1 or N-bit 2-state constant>
  output[SIZEOUT:0] out;         out[#]             <ITEM>                       | out[#]
  input[SIZEIN:0] in;            | origtmp#         | <UNARY_OP> <ITEM>          | in[#]
  wire origtmp1;                 | xformtmp#        | <ITEM> <BINARY_OP> <ITEM>  | origtmp#
  ...                            | out[MSB:LSB]   <BINARY_OP> ::=              | out[MSB:LSB]
  wire origtmpN;               <UNARY_OP> ::=       &   (bit and)              | in[MSB:LSB]
  assign <LHS> = <RHS>;          ~   (bit negation) | |  (bit or)               | xformtmp#
  …                                                 | ^  (bit or)               | xformtmp#[MSB:LSB]
```

Evaluation: If any of simulation correctness, optimization regulation, and execution time limit is violated, simulation efficiency will NOT be evaluated. If not, the score will be calculated by score = (<OUT_PORT_WIDTH> + <IN_PORT_WIDTH>) / (<COUNT_ASGN> + <COUNT_SELS>).

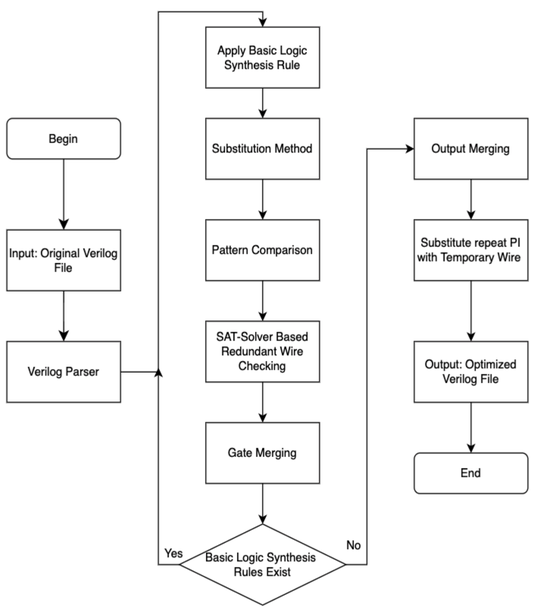
For each “original.v”, the number of (<OUT_PORT_WIDTH> + <IN_PORT_WIDTH>) is fixed. Therefore, the lower the number of (<COUNT_ASGN> + <COUNT_SELS>) indicates fewer CPU instructions needed, the higher the score is.

## Methodology and Comparison to current research

In recent years, researchers have usually used traditional graphs (such as AIG) to do logic optimization. However, we use a data structure similar to the binary tree to represent the circuit rather than AIG.

The complexity of traversing the circuit will be decreased by this data structure, which has good properties to deal with don’t care and pattern comparison implementation

What’s more, we find that there are few kinds of research talking about how to optimize the RTL file which has the XOR gates. In our program, we develop this structure to optimize the Verilog file and the circuit accompanied by the XOR gates. Fig. 1 is the flowchart of our work, which will be elaborated in the next part.



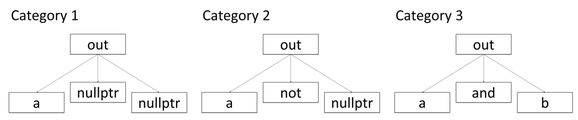

<Fig.1 Flowchart>

## Implementation

1. **Verilog Parser and Data Structure** – After parsing the input Verilog file, we divided them into three categories to map input assignments into the data structure.

Category 1:
assign out = a;
Category 2:
assign out = ~a;

Category 3:
assign out = a & b;
assign out = a | b;
assign out = a ^ b;




<Fig2. Data Structure >

2. **Apply Basic Logic Synthesis Rules** – In Boolean algebra, we all know several Boolean identities are used to minimize the Boolean function. First, we will list some common constant operations of the Boolean function in the following table. Then, optimize the circuit with the rules.

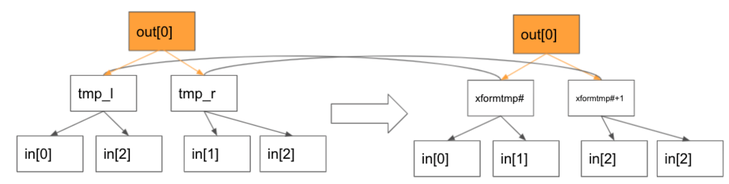
| | | |
|---|---|---|
| A\|0 = A | A&0 = 0 | A^0 = A |
| A\|1 = 1 | A&1 = A | A^1 = ~A |
| A\|2 = 1 | A&2 = 0 | A^2 = 2 |

1. out[0] = origtmp1;
   origtmp1 = in[0] & in[1]; ⇨ out[0] = in[0] & in[1];
2. out[0] = origtmp1 | origtmp2;
   origtmp1 = in[0]; origtmp2 = in[2]; ⇨ out[0] = in[0] | in[1 ];

<Fig3. Substitution Method

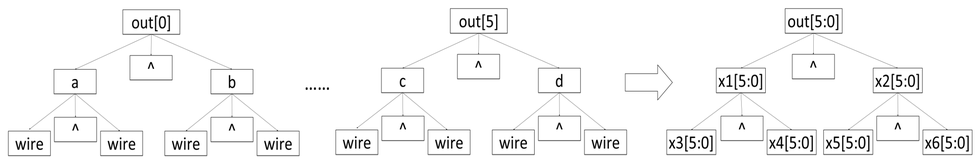
3. **Substitution Method** – After minimizing the inner data structure, we hope to substitute some variables with the corresponding example equations as Fig. 3.
4. **Pattern Comparison** – We offer several Boolean equations that can be minimized as the equations below. If we use tree traversal to find the subtree patterns of the equations, then we can rewrite the structure of the subtree to optimize the circuit. In addition to those equations, it is likely to get an optimized function with the help of commutative law as Fig. 4.

1. a|(ab) = a
2. a&(a|b) = a
3. a|(~a&b) = a|b
4. a&(~a|b) = a&b
5. ~a&b | a&~b = a^b
6. ~(~a) = a (double negation)
7. a&(~a) = 0




<Fig.4 Commutative Law>

5. **SAT-Solver-Based Redundant Wire Checking** – In the EDA field, we will use testing (stuck-at-fault) to check the possible faults in the chips. For a combination circuit, an undetectable fault is corresponding to a redundant wire. Undetectable faults do not change the functionality of the circuit.
6. **Gate Merging** – Compare two circuits and merge gates that have the same functionality to reduce redundancy.
7. **Output Merging** – Merge the continuous output to reduce the number of bit-selection as the Fig. 5.




<Fig.5 Output Merging>

8. **Substitute Repeat PI with Temporary Wire** – If the input signal is used in multiple assignments, we can use a temporary wire to substitute the input in order to reduce the number of bit-selection.

## Experimental Result

The following experimental results in the public 9 testcases were conducted in the Linux environment provided by the CAD Contest as Fig .6, which represents the score comparison between the original Verilog file and the optimized one.

One of these approaches can get a score almost ten times better than the original case. Even some test cases can be simplified to the simplest form of them by our work. Furthermore, our group won the ICCAD awards among all contestants.

| Test Case | original | optimized | Test Case | original | optimized |
|---|---|---|---|---|---|
| 1 | 0.365854 | 0.566038 | 6 | 0.000499875 | 0.0376689 |
| 2 | 0.410256 | 0.610687 | 7 | 0.397772 | 0.642467 |
| 3 | 0.404218 | 0.633609 | 8 | 0.523256 | 1.95652 |
| 4 | 0.0714286 | 0.571429 | 9 | 0.106956 | 0.123919 |
| 5 | 0.444444 | 4 | | | |

<Fig. 6 Comparison to experimental results>