# EE U0250 SOCV FINAL PROJECT REPORT

*Jian-Heng Liu, Wei-Hsu Chen, Zhong-Yu Sui, Hung-Ling Liu and Chung-Yang (Ric) Huang*

National Taiwan University Taipei 10617, Taiwan

## ABSTRACT

This is a final report for the SOCV (System On Chip Verification) final project, focusing on CAD Contest Problem A, which is the most challenging problem in Boolean matching techniques, known as Projective NP3 (Non-exact Negation and Permutation of outputs and Negation and Permutation of inputs). We will briefly provide an introduction to Boolean matching and contest requirements, and present our SAT-based algorithm. Finally, we will showcase the experimental results.

***Index Terms—*** Boolean matching, equivalence checking

## 1. INTRODUCTION

The goal of boolean matching is to determine if two Boolean functions are equivalent, meaning they produce the same output for some possible input combinations. This problem is fundamental in various areas, including digital circuit design, formal verification, and computer-aided design (CAD). Boolean matching plays a crucial role in optimizing circuit designs, detecting errors, and ensuring correct functionality.

NP3 involves finding a non-exact projective mapping between two functions, which makes the matching process more complex than NPNP-equivalence Boolean matching researches.

In modern digital IC design, the large number of input/output ports are usually caused by the buses or datapaths in the design, and these information of input/output buses may reduce the complexity of solving the Boolean matching problem. Thus, in this contest not only provide circuits file, but also the details of each input/output bus.

In the following sections, we will outline the details of our SAT-based algorithm and present the results of our experiments, demonstrating its effectiveness in handling the NP3 problem in boolean matching.

## 2. ALGORITHM

We first parse the circuit, and derive the corresponded CNF formula. Then, build two 0-1 matrices using the method described in [3]and [2]. These matrices are named $M_I$ and $M_O$, respectively. Finally, we connect the circuit CNF and built matrices and add constraint clauses, use one matching algorithm to iterative find optimal solution.

### 2.1. Circuit Parsing

We take usage of two EDA tools, abc and gv (general verification) which integrates two open source engines, "berkeley-abc" [1] and "yosys". We first read the verilog files of two circuits by a handwritten parser, and convert the test data of the contest into a format that can be read by abc. We then convert verilog file to aig file by abc, and then translated aig into aag file with aag vars to input/output names mapping file. Finally, we read in aag files, convert them into CNF clauses in MiniSat clause format.

### 2.2. Permutation and Negation Matrix

Define a set of Boolean variables is denoted as an upper-case letter, e.g. X; its elements are in lower-case letters, e.g., $x_i \in$ X. Given a function $F(X)$, its outputs are denoted as $< f_1, f_2, ..., f_{|f|} >$, where $|f|$ is the number of outputs of $F(X)$. Given two circuits $C_1$ and $C_2$, they represent two functions $F(X)$ and $G(Y)$ respectively. For simplicity, we assume $|X| = nPI1$, $|F| = nPO1$, $|Y| = nPI2$, $|G| = nPO2$ in the remaining parts of the paper.

$M_I$ shows how the inputs of $C_1$ are mapped to the inputs of $C_2$, where $a_{i,j} = 1(b_{i,j} = 1)$ denotes that the positive (negative) of $x_j$ is mapped to $y_i$ ,and $o_i = 1(z_i = 1)$ denotes $y_i$ is mapped to $constant\ 0(1)$.

Assume the number of $C_1$'s inputs(outputs) $= m$, and the number of $C_2$'s input(output) $= n$ in following matrices.

$$M_I = \begin{array}{c} \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{array} \begin{array}{c} \begin{array}{ccccccc} x_1 & \neg x_1 & x_2 & \cdots & \neg x_m & 0 & 1 \end{array} \\ \begin{pmatrix} a_{11} & b_{11} & a_{12} & \dots & b_{1m} & z_1 & o_1 \\ a_{21} & b_{21} & a_{22} & \dots & b_{2m} & z_2 & o_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n,1} & b_{n,1} & a_{n,2} & \dots & b_{nm} & z_n & o_n \end{pmatrix} \end{array} \tag{1}$$

In $M_O$, $c_{i,j} = 1(d_{i,j} = 1)$ denotes that the positive (negative) of $f_j$ is mapped to $f_i^* \in F^*$, where $f^*$ is further used to check the we can map $F$ to $G$

$$M_O = \begin{array}{c} \\ f_1^* \\ f_2^* \\ \vdots \\ f_n^* \end{array} \begin{array}{c} \begin{array}{ccccc} f_1 & \neg f_1 & f_2 & \cdots & \neg f_m \end{array} \\ \begin{pmatrix} c_{11} & d_{11} & c_{12} & \dots & d_{1m} \\ c_{21} & d_{21} & c_{22} & \dots & d_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & d_{n,1} & c_{n,2} & \dots & d_{nm} \end{pmatrix} \end{array} \quad (2)$$

We connect Boolean variables, $a_{ij}$, $b_{ij}$, $c_{kl}$, and $d_{kl}$, for $i, j = 1, ..., n$ and $k, l = 1, ..., m$, to the meaning as circuit miter, by asserting the formula $\varphi_A$, which is

$$\varphi_A = \bigwedge_{i,j=1}^{i=nPI2,\ j=nPI1} (a_{i,j} \Rightarrow (y_i \equiv x_j))(b_{i,j} \Rightarrow (y_i \equiv \neg x_j)) \wedge \bigwedge_{k=1,l=1}^{k=nPO2,\ l=nPO1} (c_{k,l} \Rightarrow (f_k^* \equiv f_l))(d_{k,l} \Rightarrow (f_k^* \equiv \neg f_l)) \quad (3)$$

For $M_I$, we have two constraints that are 1) each inputs $x_j$ at least map to one input of $y \in Y$ (formula 4), and 2) each input $y_i \in Y$ are mapped to inputs $x_j \in X$ or $constant$ 1 or $constant$ 0 (formula 5)

$$\sum_{i=1}^{nPI2} (a_{i,j} + b_{i,j}) \geq 1, \forall j = 1, 2, ..., nPI1 \quad (4)$$

$$z_i + o_i + \sum_{j=1}^{nPI1} (a_{i,j} + b_{i,j}) = 1, \forall i = 1, 2, ..., nPI2 \quad (5)$$

For $M_O$, an additional constraint would be added: each output function $f_i^* \in F^*$ map to at most one (no matching is also allowed) output function $f \in F$.

$$\sum_{k=1}^{nPO1} (c_{l,k} + d_{l,k}) \leq 1, \forall l = 1, 2, ..., nPO2 \quad (6)$$

These constraints (4), (5), and (6) can be expressed by the formula $\varphi_C$. In non-exact projective NPNP (NP3) Boolean matching, $F(X)$ and $G(Y)$ are equivalent if and only if

$$\exists \vec{a}, \exists \vec{b}, \exists \vec{c}, \exists \vec{d}, \forall \vec{x}, \forall \vec{y}. \ (\varphi_C \wedge \varphi_A \wedge \bigwedge_{k=1}^{output\_2} (g_k \equiv f_k^*)) \quad (7)$$

### 2.3. One Matching

As described in Algorithm 1, we distribute these constraints (including the $F$ and $G$ formulas) to two SAT solvers [4] named MatrixSolver and MiterSolver. We use two solvers in turns to bound searching space and check the candidates.

MatrixSolver solves $\Phi_{<i>}$ , which includes $\varphi_C$ and learned clauses after iteration i. A solution of MatrixSolver would be a candidate mapping relation, while no solution indicates that $F$ and $G$ cannot be matched.

Then, the possible mapping correctness would be verified by MiterSolver, which aims to solve the miter constraint $\Psi$.

$$\Psi = \varphi_A \wedge \bigvee_{k=1}^{nPO2} (g_k \neq f_k^*) \rightarrow UNSAT \quad (8)$$

If the output of MiterSolver is UNSAT, it means no pattern would make mapping fail and therefore a matching solution has been found. We can count the score of the matching solution to decide whether it is the currently best solution. Otherwise, if the output is SAT, it means the possible mapping is wrong because at least one output pair is inequivalent.

After solving MiterSolver, regardless of whether the output is SAT or UNSAT, MatrixSolver will add learned clauses to strengthen $\Phi_{<i>}$ to $\Phi_{<i+1>}$ accordingly and to block the current solution.

Even if a match is found, the algorithm will continue to run until the time limit of the contest or MatrixSolver is unable to find a solution.

**Algorithm 1** One Matching

**Input:** 2 Circuits $C_1$,$C_2$
**Output:** best matches

```
Build mapping matrix M_I, M_O
initial Matrix constraints clause Φ = φ_C
while true do
    if MatrixSolver(Φ) UNSAT then
        Break                                   ▷ no more match
    end if
    M = pattern make MatrixSolver(Φ) SAT
    assign matrix variable in Ψ according to M
    if MiterSolver(Ψ) UNSAT then
        count M's score
        if M's score ≥ best Matches's then
            best Matches= M
        end if
    end if
    add learned Clause to Φ
end while
```

### 2.4. Pseudo-Boolean Constraints

By using the pseudo-Boolean technique [5], SAT problems can be transformed into optimization problems. This technique involves adding a constraint to the learned clause that excludes not only the current solution at the end of each iteration but also all solutions with scores lower than the best score. This ensures that each solution is better than the previous one.

Currently, there are several solvers available for solving PB problems. Among them, we prioritize MiniSat+ because

it provides several methods for translating PB constraints to clauses and achieves good results.

## 3. HOW TO RUN CODES

Please follow the instructions below to run the code, or refer to the README file for more details. If you encounter any issues during execution, please feel free to contact us for assistance.

- **tar zxvf 01_CADContestProblemA.tgz**
- **cd 01_CADContestProblemA**
- **sudo bash SETUP.sh**
- **bash INSTALL.sh**
- **make**
- **./bmatch <input> <match>**

## 4. EXPERIMENTAL RESULTS

Due to the simplicity of our algorithm and the lack of advanced optimization techniques, it runs for a long time when solving large-scale circuits, exceeding the time requirements for the CAD contest.It's a reminder for us to make improvements.

## 5. FUTURE WORK

Based on the algorithm proposed in section 2, we believe that there are many ways to improve the speed of finding the answer and eliminate some unnecessary solutions.

### 5.1. All Matching

[2]The difference between all matching and one matching is that all matching solve all constraints at once. When the solver's output is UNSAT, the corresponding $\Phi_i$ characterize all matching. If the solver's result is SAT, there is matching cannot be achieved, so learned clauses need to be added to strengthen $\Phi_i$ and continue to the next iteration.

To find all solutions, all matching is much faster than one matching. The advantage of one matching is to find a feasible solution in short time. Since not all circuit can be totally search in limited time, we still use one matching in our implement. However, we might estimate the possible run time for all matching, and apply it to suitable sized circuits in future.

**Algorithm 2** All Matching

**Input:** 2 Circuits $C_1$,$C_2$
**Output:** best matches

```
Build mapping matrix
Φ = φ_C
constraints clause φ = Φ ⋀ Ψ
while True do
    if Solver(φ) UNSAT then
        find highest scored matching characterized
    else
        add learned Clause Φ
        φ = Φ ⋀ Ψ
    end if
end while
```

### 5.2. Output Functional Constraints

Given a function $f$, $FuncSupp(f(\vec{x}))$ represents a set of $\vec{x}$ where for any $x_i$ picked from the set, $\frac{df}{dx_i} = f_{x_i=1} \oplus f_{x_i=0}$ always equals true.

Under the restriction of NP3, if $|FuncSupp(f_i)|$ is greater than $|FuncSupp(g_j)|$, it means that the output function $g_j$ of $C_2$ cannot map the output function $f_i$ of $C_1$ [3]. This is because, even if the input mapping is one-to-one, all the inputs of $g_j$ cannot match all the inputs of $f_i$. Based on the limitations of Function Support, we can first remove some impossible mappings by assigning 0 or 1 to certain variables of $M_O$.

### 5.3. Usage of Buses

Bus information can help reducing searching space, e.g. bus with different size is probably not the same bus, we can add correspond clause to $\Phi$.

Though Bus information may trim off some possible solutions, we still can consider Bus information first and try to get some solutions. After we search all solutions under bus constraints, we can loose the constraints to search for more solutions. Since we have added learnt clause with constraints, the time to search all solutions still can be reduce.

## 6. KEY RESEARCH CONTRIBUTIONS

| ***name*** | ***works*** | ***percentage*** |
|---|---|---|
| Jian-Heng | SATsolver | 25% |
| Wei-Hsu | SATsolver | 25% |
| Zhong-Yu | circuit parser | 25% |
| Hung-Ling | code outline/report | 25% |

Basically, everyone has participated in each work. Only the people who have a higher participation rate are listed.

| Test Case | #C1 Input | #C2 Input | #C1 Output | #C2 Output | Score | Run time |
|---|---|---|---|---|---|---|
| 1 | 5 | 5 | 4 | 4 | 8 | <1 |
| 2 | 12 | 12 | 4 | 4 | 4 | 55 |
| 3 | 41 | 41 | 32 | 32 | 0 | 3600 |
| 4 | 12 | 12 | 4 | 4 | 2 | 3600 |
| 5 | 178 | 178 | 123 | 123 | 0 | 3600 |
| 6 | 82 | 86 | 26 | 26 | 0 | 3600 |
| 7 | 12 | 14 | 8 | 8 | 0 | 3600 |
| 8 | 32 | 36 | 7 | 7 | 0 | 3600 |
| 9 | 241 | 256 | 120 | 120 | 0 | 3600 |
| 10 | 183 | 207 | 108 | 108 | 0 | 3600 |

**Fig. 1**. Experimental results

## 7. CONTACT INFORMATION


- Jian-Heng: b08901032@ntu.edu.tw
- Wei-Hsu : b08901181@ntu.edu.tw
- Zhong-Yu : johnsonsui3@gmail.com
- Hung-Ling: b08505024@ntu.edu.tw


## 8. REFERENCES


[1] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. volume 6174, pages 24–40, 07 2010.

[2] Chih-Fan Lai, Jie-Hong R. Jiang, and Kuo-Hua Wang. Boolean matching of function vectors with strengthened learning. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 596–601, 2010.

[3] Chak-Wa Pui, Peishan Tu, Haocheng Li, Gengjie Chen, and Evangeline F. Y. Young. A two-step search engine for large scale boolean matching under np3 equivalence. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 592–598, 2018.

[4] Niklas Sörensson and Niklas Eén. Minisat v1.13 - a sat solver with conflict-clause minimization. 2005.

[5] Niklas Sörensson and Niklas Eén. Translating pseudo-boolean constraints into sat. 2006.