

SOCV Final Project: X-value Equivalence Checking

Sung-Ta Tsai

Dept. of Electrical Engineering

National Taiwan University

Taipei, Taiwan

b05901040@ntu.edu.tw

Abstract—In this report, a technique is proposed to solve X-value Equivalence Checking. This technique transforms X-value netlist into binary-value netlist and solve with SAT solver [1]. During this conversion, some techniques are adopted to reduce netlist size. Conversion of netlist and optimization techniques will be discussed in this report. 66% of the testcases of ICCAD contests can be solved successfully within 2 minutes. Other testcases can get result within runtime limit (1,800 seconds) under conflict limit.

I. PROBLEM ANALYSIS

In this problem, we need to do equivalence checking on two different combinational netlist. However, not only 0, 1 value can occur in each signal, X value is also possible to occur. Recent verification techniques like CNF-based SAT [3] solver can't be directly apply on this problem since they can only handle binary value netlist. Therefore, the main idea to solve this problem is try to convert this problem into binary-value equivalence checking (or even a problem which can be modeled as SAT problem). Then we can apply SAT solver to solve that new problem. To convert this problem into binary-value equivalence checking, two basic ideas are discussed below.

A. Try to replace DC and MUX gates and lbx with primitive gates

Only DC and MUX gates and lbx signal can generate X in the output of their corresponding gates. To avoid dealing with X value, replacing them with primitive gates is necessary. However, the conditions of X value propagating from those gates to PO is much more than original problem. Its likely that the number of gates in the fanout cone of DC, MUX gates become exponential growth with respect to original netlist.

B. Represent all signals with two bits

Since each signal can have 0, 1 or X value, representing each signal with 2 bits rather than 1 bit can make propagation of signals from PI to PO as easy as in binary-value netlist. However, redundant signals may possibly occur in some conditions. If some gates will never generate X value in its output under all kinds of PI assignments, using two bits will be wasted and resulting in complexity growth of the problem. Also, there could be some X value signal generated by MUX or DC gate, but propagation of X value is blocked since side input of gate in its fanout cone is controlling value.

Problem of ideas discussed above is the growth of complexity and redundant signals. But using them can make this problem become solvable by SAT solver. Therefore, I combine both ideas and some heuristics to convert original netlist into binary-value netlist and try to make the structure smaller.

II. PROPOSED TECHNIQUE

The overall algorithm is shown in Fig. 1.

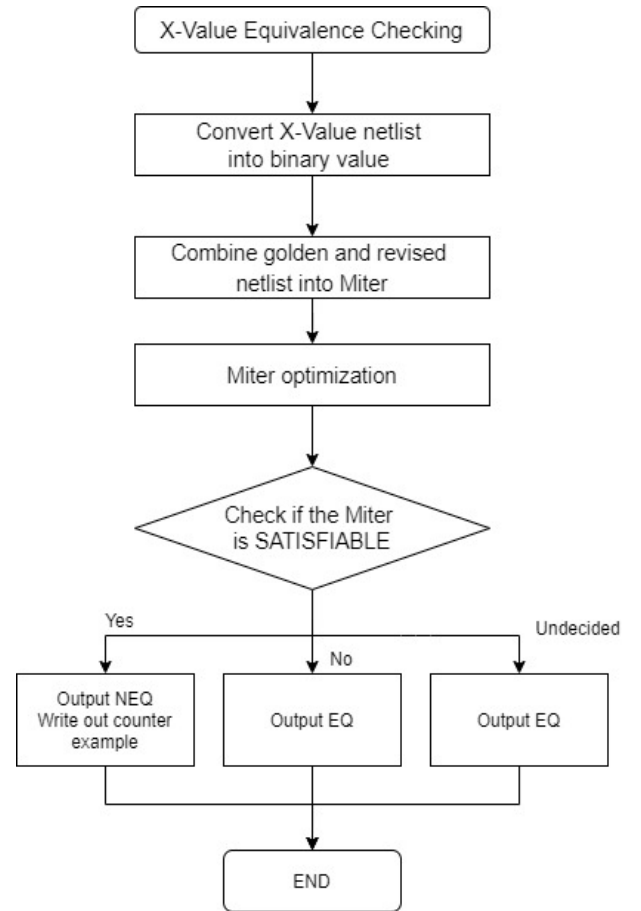


Fig. 1. Flow Chart of proposed technique.

A. Convert X-Value netlist into binary value

Since each signal can have 3 possible values, representing each signal with two bits is necessary. Table 1 shows how to convert 0, 1, X value into 2 bits in my algorithm. Since

reducing the number of gates after mapping can help to reduce the structure of new netlist, I use Karnaugh Map to reduce the logic of each gate. Note that 10 will never occur in the new netlist. Table 2 shows some examples of mapping each gate from original netlist to binary-value netlist, which is based on the mapping rule in Table 1.

The pseudo code for this part is shown in Algorithm 1. By the mapping rules defined in Table 1 and Table 2, golden and revised netlist can be converted to binary-value netlist respectively.

Algorithm 1: Convert X-Value into binary value netlist

Input: original golden and revised netlist

Output: new golden and revised netlist with only binary-value signal

- 1 Read original golden and revised netlist respectively
 - 2 Parse each line to get each gate information
 - 3 Map each gate from 1 bit to 2 bits using TABLE II
 - 4 Print out new golden and revised netlist into a single file
-

If a gate with original input signals a and b , original output signal c , new gate will have signals $a_1, a_2, b_1, b_2, c_1, c_2$. $_1$ signal means the first bit of new gate signal, and $_2$ signal means the second bit. Mapping of most primitive gates like AND, OR, NAND, NOR, BUF, NOT would only make the number of gates in new netlist become twice as original netlist. However, mapping XOR and MUX gate can be complex and increase lots of gates in binary-value netlist.

TABLE I
SIGNAL MAPPING

Original value	New value	
	First bit	Second bit
0	0	0
1	1	1
X	0	1

B. Combine golden and revised netlist and Construct Miter

Use the binary-value netlist generated in part A to create miter. Link each PI of golden and revised netlist together. Note that each PI values should be either 00 or 11 since only 0 and 1 is possible to occur in PIs in this problem, so linking first bit and second bit of each PIs is necessary and can help to reduce the structure afterwards. For example, if a PI named in , signals including in_1, in_2 of golden netlist and in_1, in_2 of revised netlist should be linked together.

Next, check if PO in golden netlist is compatible with the corresponding PO in revised netlist. If PO in golden netlist get X value, we don't need to check what the value is in revised netlist. Otherwise, each PO in revised netlist should be equivalent to the value in golden netlist. After checking compatibility, each PO should get a final signal

that represents whether this PO is compatible in some PI assignment. If compatible, final signal will have value 0, otherwise 1. Oring all final signals of each PO to form a final output. If final output can be SAT, counter example is found and will return NEQ. Otherwise if UNSAT, return EQ. Detailed description of proving this miter will be in the part D.

The pseudo code for this part is shown in Algorithm 2, which shows the detailed construction of miter.

Algorithm 2: Combine golden and revised netlist and Construct Miter

Input: golden and revised netlist with only binary-value signal

Output: Miter that needs to be solved by SAT solver

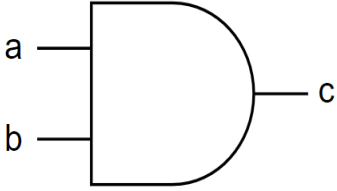
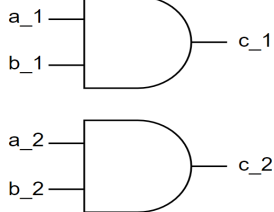
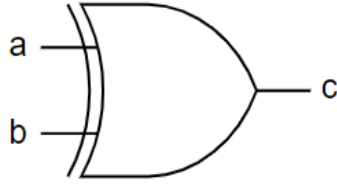
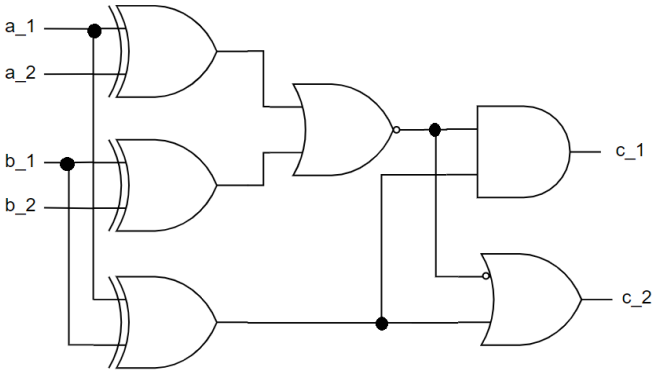
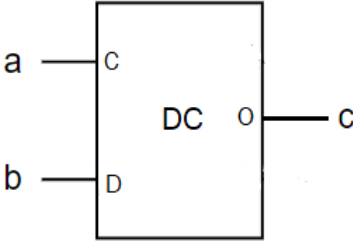
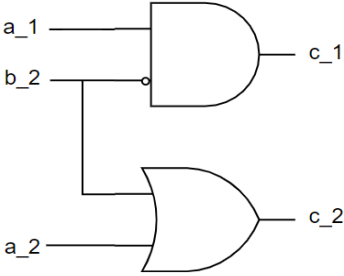
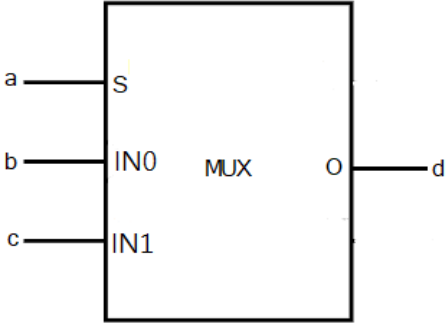
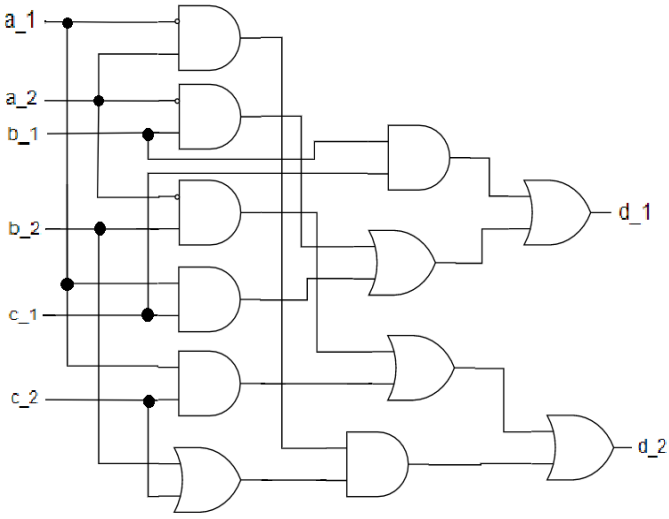
- 1 **for each** $PI \in netlist$ **do**
 - 2 Link the golden and revised netlist signal all be linked together.
 - 3 **end**
 - 4 **for each** $PO \in netlist$ **do**
 - 5 AND first bit signal of that PO in golden netlist and the inverse second bit signal of that PO in revised netlist.
 - 6 // If golden PO is 11 and revised PO is 00, incompatible occurs.
 - 7 AND the inverse of second bit signal of that PO in golden netlist and first bit signal of that PO in revised netlist.
 - 8 // If golden PO is 00 and revised PO is 11, incompatible occurs.
 - 9 OR the above two signals in line 5, 7.
 - 10 OR first bit signal and the inverse of second bit signal of that PO in golden netlist.
 - 11 // Block the conditions that golden PO is 01, always compatible
 - 12 AND the above two signals in line 9, 10 to become the final signal of that PO.
 - 13 **end**
 - 14 Or all final signals to become a single output.
 - 15 Write out this netlist (Miter).
-

C. Miter optimization

Use the created miter to solve SAT can get the final answer. However, there are still lots of redundant signals in the miter. Before starting proving this miter, I use some circuit optimization techniques to remove some redundant gates. To make optimization easier, transform the miter into AND-INVERTER Graph, where there can only be AND gates with 2 inputs and inverters. Now consider the following conditions that some gates are redundant in the netlist.

If there are no DC, MUX gate and 1'bx signals in the fanin cone of a gate, this gate must receive 0 or 1 value. That is, this gate must receive 00 or 11 value, where 2 bits signal

TABLE II
GATE MAPPING

Gate type	Original gate	Gate Mapping
AND		
XOR		
DC		
MUX		

must have the same value. Since each PI signal in the miter are linked together respectively, those gates actually have the same inputs. To reduce miter size, use structure hashing to remove those redundant gates.

Some signals may have same value under all PI assignments. However, they don't exactly corresponds to the same inputs. In this case, use random simulation to detect possible equivalent internal signals (FEC groups) and use SAT solver to transform AIG into functionally-reduced AIG. Since proving all FEC pairs is quite difficult, I set conflict limit 100 in SAT solver. After using fraig, do structure hashing again to reduce miter size.

The pseudo code for this part is shown in Algorithm 3, which shows the optimization step of miter. In part C and part D, I use *ABC framework* [1] to implement the algorithm.

Algorithm 3: Miter optimization

Input: Miter

Output: Miter being optimized

- 1 Read the Miter. // command: *read_verilog*
 - 2 Transform miter into AND-INVERTER Graph.
 - 3 Do structure hashing on this aig. // command: *strash*
 - 4 Do fraig on this aig. // command: *fraig*
 - 5 Do structure hashing on this aig. // command: *strash*
 - 6 Sweep all unused gates in this aig.
-

D. Prove the Miter and output result

Solve the SAT problem on the miter. I use the proving command *iprove* [2] in *ABC framework* [1]. Before proving miter, use some synthesis technique such as network balancing, rewriting and refactoring to reduce miter size. Then start to prove this miter for 6 iterations with different conflict limit in Table 3. Note that fraiging and rewriting are performed at the same time to further reduce miter size.

TABLE III
ITERATION CONFLICT LIMITS

Iteration conflict limits	
Iteration	conflict limits
1	5,000
2	10,000
3	20,000
4	40,000
5	80,000
6	160,000

Since most of the design can't be proved under only 160,000, the above process usually can't prove the miter to be UNSAT. If the miter can't be solved in the above process, solve with CNF-based SAT solver with conflict limit 1,000,000 since total runtime limit of each testcase is 1,800 seconds. In this process, solver would remove 50% of the learned clauses for

every new 1,000 learned clauses. If this miter returns UNSAT, output EQ. If this miter returns SAT, output NEQ and counter example. If this proving process still can't get the answer (UNDECIDED), output EQ as well.

The pseudo code for this part is shown in Algorithm 4.

Algorithm 4: Prove the Miter and output result

Input: Miter being optimized

Output: EQ or NEQ

- 1 Prove the Miter. // command: *iprove*
 - 2 **if** the miter is proved to be UNSAT **then**
 - 3 output EQ
 - 4 **else if** the miter is proved to be SAT **then**
 - 5 output NEQ
 - 6 output counter example // command: *write_cex*
 - 7 **else**
 - 8 output EQ // The answer is UNDECIDED
 - 9 **end**
-

III. EXPERIMENTAL RESULTS

Experimental results of public testcases of ICCAD contest executing on the virtual machine of ICCAD contest are shown in Table 4. In case 1, 3 and 8, the netlist can't be proven to be EQ or NEQ by the conflict limit (UNDECIDED). Other cases can get the output result within 2 minutes. The average execution time of all testcases is 4 minutes.

In all NEQ cases (2, 4, 7, 9), they are proven to be NEQ in the 6-iteration solving step. That is, most NEQ cases can be solved within 160,000 conflicts. EQ cases like case 5 and 6 are also proven to be EQ within 6-iteration solving step. However, some cases like case 3 and 8 can't be proved within 1,800 seconds (contest runtime limit). Case 1 can be proved to be EQ in 21 minutes but conflict limit occurs earlier.

TABLE IV
EXPERIMENTAL RESULTS

TestCases	Result	Runtime
Case 1	EQ ¹	8 min 27 sec
Case 2	NEQ	0 min 02 sec
Case 3	EQ ¹	6 min 48 sec
Case 4	NEQ	0 min 09 sec
Case 5	EQ	0 min 39 sec
Case 6	EQ	0 min 39 sec
Case 7	NEQ	0 min 33 sec
Case 8	EQ ¹	16 min 51 sec
Case 9	NEQ	1 min 56 sec
Average	NEQ cases	0 min 40 sec
	EQ cases	6 min 40 sec
	Total cases	4 min 00 sec

¹Result is UNDECIDED.

IV. DISCUSSIONS

A. Use Table 1 to map X, 0, 1 to binary-value and optimization steps

Use this mapping can help to reduce the structure using *strash* command since 0 becomes *00* and 1 becomes *11*. Strash can do faster than *fraig*, making all gates that have no MUX, DC gate within their fanin cone can be simplified to the original netlist. Use *fraig* to remove the same signal can also be simpler after doing *strash*. Also, I tried to add some synthesis techniques in Miter optimization step, such as *rewrite*, *balance*, *satclp*. However, adding those commands make the execution time of NEQ cases becomes longer and does not improve EQ/UNDECIDED cases. Therefore I decide only to adopt *strash* and *fraig* to reduce the miter structure in this step. Other synthesis techniques are used in *iprove*.

B. Solve the Miter with 2 steps

This algorithm is based on [2] and is implemented in *iprove* command. First step is to solve the miter for 6-iteration and incremental conflict limit. This step also tries to reduce the Miter structure using synthesis techniques and *fraig* as well. Second step is to prove by Satsolver with regular removal of learned clauses. Removal of some useless or too old learned clauses can help to increase proving efficiency. This proving method is much faster than using original MiniSat (command: *sat*), where most of the cases can't be proved within 1,800 seconds.

C. Possible Improvement ways

By the experimental results, runtime of cases that can be proved to be EQ or NEQ can all be done within 2 minutes. Using more synthesis techniques would make the runtime longer, but other cases still can't be proved within 1,800 seconds. Since the objective of this contest is to reduce total running time and get the correct answer, there are some possible ways to improve the overall performance.

Try to lower conflict limits. This may improve the total runtime performance but raise the risk of outputting false answer.

Search for better methods to do miter optimization. How to balance the increasing runtime on solved cases and reducing runtime on unsolved cases is a big challenge.

Choose another mapping way or even another idea to transform this problem into a SAT problem. It would be very difficult and the result is not guarantee to be better.

V. CONCLUSIONS

The main idea of the proposed technique is transform the original X-value netlist into binary-value netlist. Then solve this new netlist with SAT-based proving techniques. About 66% of the cases can be solved successfully within 2

minutes. Other cases can't be proved within runtime limit, so the overall algorithm still needs to improve to get better results.

VI. DISCLAIMER OF THIS PROJECT

A. Use of third partys codes

In reducing and solving Miter, I use *ABC: A System for Sequential Synthesis and Verification* [1] to implement the functions (codes in *abc/*). Converting the original netlist into binary-value netlist is done by myself (codes in *src/*), but I use some utility functions in *src/util.h* and *src/util.cpp*, which are from NTU DSNP Class's code [4] to help me do parsing. Also, this project is not used in any other classes.

B. Teamwork

My team (register Team Number *cada0030*) has only one member (myself). There is no other team member that is in or NOT in SOCV class.

VII. FEEDBACKS AND SUGGESTIONS TO SOCV CLASS

I learned lots of things related to verification in SOCV class. Class videos and homeworks really helps me to understand and get familiar with class contents. But it would be better that each week can cover almost the same amount of slides since I felt the loading of each week varies greatly. Overall, I think professor taught well and TA helped everyone in each homework a lot.

REFERENCES

- [1] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*, Release 80410. <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [2] Mishchenko, A., Chatterjee, S., Brayton, R., Eén, N.: Improvements to combinational equivalence checking. In: Proc. ICCAD 06, pp. 836843 (2006)
- [3] E. I. Goldberg, M. R. Prasad and R. K. Brayton, "Using SAT for combinational equivalence checking." Design, Automation and Test in Europe. (DATE 2001)
- [4] NTU DSNP open source code on github, <https://github.com/ric2k1/DSNP.open/tree/master/Homework>