

SAT-based Combinational Equivalence Checking

Zhuo Huang
zhuang@cise.ufl.edu

Prabhat Mishra
prabhat@cise.ufl.edu

CISE Technical Report #05-007
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, FL 32610, USA

August 31, 2005

Abstract

Combinational equivalence checking is one of the key components in today's hardware verification methodology. Structural similarity of the two designs are exploited by existing BDD, SAT, or ATPG based methods. This report presents a technique for improving the performance of the existing SAT-based combinational equivalence checkers by adding new constraints based on the structural similarity. Our experimental results using ISCAS benchmarks demonstrate the usefulness of this approach.

Contents

- 1 Introduction** **4**

- 2 Preliminary** **4**
 - 2.1 Combinational Equivalence Checking 4
 - 2.2 Structural Similarity 5
 - 2.3 SAT 6

- 3 CEC Algorithm Design** **6**
 - 3.1 SAT representation of Combinational Circuit 6
 - 3.2 Algorithm for SAT-based CEC 7
 - 3.3 Gate Equivalence Checking 10

- 4 Related Work** **10**

- 5 Experiment Results** **11**
 - 5.1 Simulation Experiments 11
 - 5.2 CEC Experiments 12
 - 5.3 Results Analysis 13

- 6 Future Work** **13**

- 7 Conclusions** **13**

- 8 Acknowledgments** **14**

List of Figures

1	An example for structural similarity	5
2	An hardware Model	7

List of Tables

1	Experiments for simulation	11
2	Experiments for getting Structure Similarity (S1)	12
3	Experiments for getting Structure Similarity (S2)	12
4	Experiments for Equivalence Checking(EC) without/with Structure Similarity(SS) .	12

1 Introduction

Verification is an important issue of hardware design. It ensures the implementation satisfies the specification. Equivalence checking is a verification technique to guarantee the correctness of the implementation. Combinational Equivalence Checking (CEC) is a fundamental technique for the equivalence checking for digital circuits. The target of CEC is to verify the equivalence of two combinational circuit designs, which is a significant problem in hardware design and optimization.

Many formal techniques including BDD-based techniques, SAT-based techniques and so on, have been applied for CEC. The SAT-based and BDD-based approaches seems more natural, since the CEC problem is represented in the propositional logic form in most cases and both SAT-based techniques and BDD-based techniques deals with the propositional logic quite well.

Informally speaking, the mechanism of the BDD-based techniques is to translate the two circuit designs into two graphs and then prove that the two graphs are actually same. And the mechanism of the SAT-based techniques is to prove that the outputs for the two designs are always matched. Both techniques are quite successful for the CEC problems. In this report, we proposed an improvement to the existing SAT-based techniques.

The rest of the report is organized as follows. Section 2 presents the basic concepts of the CEC problem and the SAT problems. Section 3 presents the basic algorithms we implemented. Section 4 presents related work about the SAT-based CEC problems. Section 5 describes the experimental results. Finally, Section 7 concludes the report and present the future direction of the work.

2 Preliminary

In this section, we represent some basic concepts of CEC and SAT.

2.1 Combinational Equivalence Checking

Equivalence checking is important for hardware design verification. Suppose we designed a circuit and verified its correctness by running millions of test cases or by running some formal verification tool;and we later did tiny modifications for certain reasons, such as re-organizing for efficiency or for lower power. After the modification, we need to guarantee the new design is also correct. An obvious solution is to run the millions of test cases or the formal verification tool again but it may take weeks or even months. Another solution is to check whether the new design is equivalent to the old one or not. The later approach is known as equivalence checking. It works since we already know the old one is correct. Generally equivalence checking is more efficient than the former solution.

CEC is the simple case of the equivalence checking. Its target is to check whether the two combinational circuits are equivalent. The task would be easy to solve if the two designs have structural similarity. In our work, we only consider the CEC problem for two similar designs.

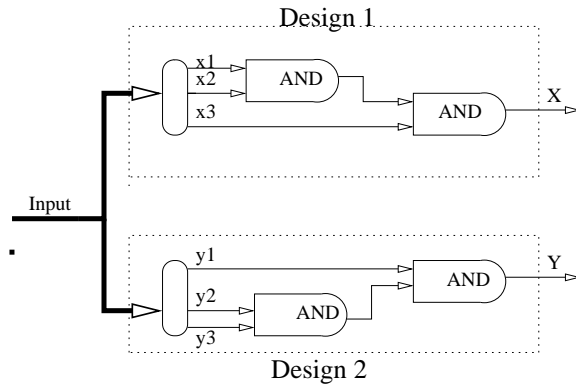


Figure 1. An example for structural similarity

2.2 Structural Similarity

If we only do a tiny modification of the design, the new one is similar to the old one. Such kind of similarity can be used to simplify the CEC problem. Consider an example in Figure 1, which has two designs. Both designs have exactly the same inputs. To prove that Design 1 is equivalent to Design 2, we have to prove that they produce the same output i.e. $X=Y$. We use $\langle X, Y \rangle$ to represent that X and Y are a pair of equivalent signals. The only difference between the two designs is the input of the AND gates. If we already know from the structural similarity that the x_1 is equivalent to y_1 , x_2 is equivalent to y_2 and x_3 is equivalent to y_3 , we can easily prove that X is equivalent to Y . Hence the two designs are equivalent. The equivalence of $\langle x_i, y_i \rangle$ represents the structural similarity of the two designs.

When we talk about the structural similarity, the objects we consider are the inner gates in the designs, such as $\langle x_i, y_i \rangle$. The structure similarity we captured is based on the equivalence between the output values of the gates.

The equivalence of the gates can be captured gradually. In Figure 1 the equivalence of $\langle X, Y \rangle$ are learned based on the equivalence of the $\langle x_i, y_i \rangle$. The equivalence of X and Y can be used in future to check the equivalence of other gates.

Now the problem becomes how to know the two gates are equivalent. It is difficult for an algorithm to read two models in some language, generate a graph like Figure 1, compare it and find out the equivalence. We use another solution in our approach.

Two gates are equivalent if and only if their outputs are always equivalent for all possible inputs. That is easy to check by an algorithm. In Figure 1, if we already know the equivalence of $\langle x_i, y_i \rangle$, we only need to consider the 8 possible value combinations for $\langle x_i, y_i \rangle$ and check whether X always equals to Y or not. Theoretically, this approach can handle large designs. Since the process is done gradually and the designs are similar, each time we just need to deal with a small part of the design.

2.3 SAT

The SAT problem for the propositional logic is a basic problem in the computer science and plays an important role in theory and many different applications. SAT is NP-complete, which means there is no efficient algorithm for it and also means the results of SAT study can be used to different problems.

SAT has been studied for more than forty years and there have been many powerful tools and algorithms developed. These tools and algorithms have also been applied in many different areas, such as Equivalence Checking, Bounded Model Checking, Artificial Intelligence and so on. Although the propositional logic can use \cap (and) \cup (or) and \neg (not) to represent a formula as complex as wanted, most of the SAT tools use a simple form as inputs which is called **Conjunctive Normal Form(CNF)**.

In CNF format, a **Literal** is either a propositional variable or the negation of a propositional variable, such as a , $\neg b$, here 'a' and 'b' are proposition variables; a **Clause** is a disjunction of literals, such as: $(L_1 \cup L_2 \cup L_3 \cup \dots \cup L_n)$. Here L_i are literals; and a **CNF Instance** is a set of clauses. Any propositional formula can be converted into an instance, such as $\{p, \neg p \cup q, \neg q \cup \neg p \cup r\}$.

If there is an assignment for the variables that the instance becomes true, we call the instance is **Satisfiable**. The assignment is called a **Solution**. For example, $\{p, \neg p \cup q, \neg q \cup \neg p \cup r\}$ is satisfiable and $\{p = 1, q = 1, r = 1\}$ is a solution. If no solution exists for an instance, we call the it is **Unsatisfiable**. For example, $\{p, \neg p \cup q, \neg q \cup \neg p \cup r, \neg r\}$ is unsatisfiable. The target of SAT algorithms is to find out whether an instance is satisfiable or not and which is the solution if it is satisfiable.

The most SAT algorithms can be divided into two groups, complete algorithms and incomplete algorithms. The complete algorithms are based on the DPLL algorithm[11, 12]. Those works include Zchaff[13], SATO[8], Berkmin[4], Minisat[9], and so on. The incomplete algorithms are mainly based on the local search[2]. It can find a solution quickly for some instances but it cannot prove an instance is unsatisfiable. Those algorithms and tools include Walksat[1] and so on.

Many efficient methods have been studied in the SAT tools, such as Conflict-Driven Lemma Learning, Variable Selection (or so-called Variable Ordering), Boolean Constraint Propagation, Pre-Processing, Restart and so on[10]. Benefited from all of them, The modern SAT tools are strong enough to handle many real-world problems, including CEC problems.

3 CEC Algorithm Design

In this section, we describe the algorithm for the CEC problem. The algorithm use SAT tool as a black box and we choose SATO[8] in our approach. We first describe the SAT representation of the circuit and then describe the algorithm and details.

3.1 SAT representation of Combinational Circuit

In order to use SAT tools to solve the problem, we first need to represent the hardware model in CNF instance. In our work we only consider the combinational circuits. In our work, the combinational circuits are usually represented in gate-level language and we only use the basic

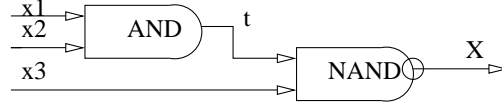


Figure 2. An hardware Model

gates, such as AND OR NOT NAND NOR XOR. Here we give the translation from the basic gates into CNF clauses.

For AND gate with n inputs, such as $out = AND(x_1, x_2, \dots, x_n)$, we generate $n+1$ clauses:

$$\neg x_1 \cup \neg x_2 \cup \dots \cup \neg x_n \cup out$$

$$\neg out \cup x_1$$

$$\neg out \cup x_2$$

...

$$\neg out \cup x_n$$

For OR gate with n inputs, such as $out = OR(x_1, x_2, \dots, x_n)$, we generate $n+1$ clauses:

$$\neg x_1 \cup out$$

$$\neg x_2 \cup out$$

...

$$\neg x_n \cup out$$

$$\neg out \cup x_1 \cup x_2 \cup \dots \cup x_n$$

For NOT gate, such as $out = NOT(x)$, we generate 2 clauses:

$$out \cup x$$

$$\neg out \cup \neg x$$

For XOR gate with 2 inputs, $out = XOR(x, y)$, we generate 4 clauses:

$$x \cup y \cup \neg out$$

$$x \cup \neg y \cup out$$

$$\neg x \cup y \cup out$$

$$\neg x \cup \neg y \cup \neg out$$

For other gates such as NAND NOR, it can be translated in the similar way.

For Example, in Figure 2, the formula for the circuit is $\{\neg x_1 \cup \neg x_2 \cup t, \neg t \cup x_1, \neg t \cup x_2, \neg t \cup \neg x_3 \cup \neg x, x \cup t, x \cup x_3\}$.

Based on the translation from gates to CNF clauses, we can generate CNF representation for any large combinational circuit.

3.2 Algorithm for SAT-based CEC

In Section 2.2, we represent the main idea of the CEC Algorithm based on the structural similarity. We present the algorithm here. It can be divided into three main steps: pre-processing, generating structural similarity information and checking equivalence of the two designs.

We first consider a more general problem. Suppose we have a hardware model M and we want to use SAT tools to handle some problems with M . We need to get the CNF instance F_M of M . In

Section 3.1, we already show how to generate the CNF instance. When we use SAT tools to solve the F_M , we will find many solutions of F_M . Each solution gives us the value of the propositional variables. These variables are corresponding to the input output and inner signals.

For example, suppose we are dealing with the hardware model in Figure 2, when we use SAT tools to solve the corresponding instance $\{\neg x_1 \cup \neg x_2 \cup t, \neg t \cup x_1, \neg t \cup x_2, \neg t \cup \neg x_3 \cup \neg x, x \cup t, x \cup x_3\}$, a possible solution is $\{x_1 = 0, x_2 = 0, x_3 = 0, t = 0, x = 1\}$. It is corresponding to a possible combination of the signal values in Figure 2, which is the inputs are $\{x_1 = 0, x_2 = 0, x_3 = 0\}$ and the values of the inner and output signals are $\{t = 0, x = 1\}$.

We can easily prove that a solution of F_M means a possible combination for the values of inputs outputs and inner signals of M. Hence, we can call such a solution of F_M is a **Possible State(PS)** of M.

Suppose we want to check whether some property P is always true for hardware model M. We just need to check whether there is any PS of M that doesn't satisfy P. In the other word, we just need to check whether there is any PS of M that satisfies the negation of P. We translate the negation of P into CNF clauses F_{np} and use the sat tools to solve $F_M \& F_{np}$, where F_M is the corresponding formula of hardware model M. If $F \& F_{np}$ is satisfiable and α is the solution found by SAT tool, α satisfies both F_M and the negation of P. It means there exists a PS of M, (α), which satisfies (F_M) but doesn't satisfy P. So α is a counterexample that shows P is not always kept on M. On the other hand, if $F_M \& F_{np}$ is unsatisfiable, there is no such solution that satisfy both F_M and the negation of P. Hence P is kept on M. In this way, we can use SAT tool to judge whether a property P is kept on a hardware model M.

For example, in Figure 2, we want check a property that is $\{x_3 = 0 \rightarrow x = 1\}$. We translate the negation of the property $\{\neg(x_3 = 0 \rightarrow x = 1)\}$ into CNF clauses $\{x_3 = 0, x = 0\}$. Then we use SAT tool to solve $\{\neg x_1 \cup \neg x_2 \cup t, \neg t \cup x_1, \neg t \cup x_2, \neg t \cup \neg x_3 \cup \neg x, x \cup t, x \cup x_3\} \& \{x_3 = 0, x = 0\}$. The tool answers that is unsatisfiable. So we know that the property $\{x_3 = 0 \rightarrow x = 1\}$ is always true in Figure 2.

In the pre-processing part, we generate the CNF instances for the hardware models which are represented in gate level language. The instance of the hardware model will be used for multiple times later. So, We generate it just once in advance for efficiency.

As we discussed in Section 2.2, when we check the equivalence of two hardware models, we can try to explore the structural similarity to improve the performance. In our approach, we use the equivalence of gates to present the structural similarity. In the second step, we try to find out those equivalent gates. It is time consuming to check the equivalence between any two gates by testing all possible inputs. We use an alternate mechanism to do the same job. We simulate the two designs for several times and record those gates that always have the same outputs. After simulation, we divide the gates into several sets based on the outputs of the gates during simulation. Each set contains all the gates that always have the same output. We call the sets as **Unclear Sets**. It is obvious that the real equivalent gates can only exists in the same Unclear Set. During equivalence checking, we only need to handle the gates in the same Unclear Set.

In our algorithm, we also use the SAT tool to do the simulation. In our simulation, all we want to know is the values of inner and output signals under the given inputs. If we want to simulate hardware model M for some input I, we translate M into CNF clauses F_M and translate the inputs

into clauses F_{CI} . F_{CI} is a set of clauses that restrict input signals to some given value. We use the SAT tools to solve the $F_M \& F_{CI}$. As we mentioned before, the assignment we get gives the value of the output and inner signals. These values can be used for our simulation.

For example, in Figure 2, we want to simulate the circuit with inputs $\{x_1 = 0, x_2 = 1, x_3 = 0\}$. We use SAT tool to solve $\{\neg x_1 \cup \neg x_2 \cup t, \neg t \cup x_1, \neg t \cup x_2, \neg t \cup \neg x_3 \cup \neg x, x \cup t, x \cup x_3\} \& \{x_1 = 0, x_2 = 1, x_3 = 0\}$. The tool gives the solution $\{x_1 = 0, x_2 = 1, x_3 = 0, t = 0, x = 1\}$. $\{t = 0, x = 1\}$ can be used as the simulation result.

When we check the equivalence of gates, the equivalence we proved before can be used to improve the procedure. We need to choose the easy tasks to start with and maintain a list to record the proven equivalence. More details will be discussed in Section 3.3.

In the final equivalence checking part, we test the outputs from the two designs one by one. For any two corresponding outputs from two designs, we get the constraints by using the XOR to the corresponding outputs. The equivalence of inner gates are added into the formula to guide the searching procedure. If the instance including the original designs, the equivalence and the XOR constraints is unsatisfiable, the two outputs are matched.

The algorithm is as follows:

Procedure 1: *basic CEC Algorithm*
Input: Two models in gate representation *two models*.
Output: Matched or Not Matched.
begin
 1. Pre-process: Generate the CNF file CF for SAT tools from the two models.
 2. Generate structure similarity information.
 2.1 Simulate for given times, creating the Unclear Sets.
 2.2 Generate Equivalent Gates List $EGL = \emptyset$
 2.3 Choose and mark an Unclear Set based on certain strategy.
 2.4 If no more Unclear Set can be chosen, goto 3.
 2.5 Generate the constraints C to test the equivalence of the gates from Unclear Set.
 2.6 Use the SAT tool to solve the $CF \& C \& formu(EGL)$;
 2.7 if Satisfiable, modify the Unclear Sets. Goto 2.3.
 2.8 Add the equivalent gates into EGL. $EGL = EGL \cup equivalentgates$ Goto 2.3.
 3. Match outputs.
 3.1 **for** all corresponding outputs:
 generate the constraints CO by XOR.
 Use the SAT tools to solve $CF \& formu(EGL) \& CO$.
 if Satisfiable, answer Not Matched.
 endfor
 3.2 Answer Matched.
end

Here formula(EGL) is a function to translate the equivalence of gates into CNF. The translation is as follows:

Suppose G_1 is equivalent to G_2 , we generate two clauses:

$G_1 \cup \neg G_2$ $\neg G_1 \cup G_2$ These clauses force the value of G_1 and G_2 are always equal. Hence we utilize the previous results in the searching procedure.

3.3 Gate Equivalence Checking

In this section, we will discuss some issues in the second step of the algorithm. These issues include the number of simulation, the constraints to check gate equivalence and the strategy to choose the gates. The main concern is the efficiency.

More simulation means more chances to distinguish variables but we are unable to do all the possible simulations since that will take too much time. In our approach we run 32 or 64 simulations and the experiments in Section 5.1 shows that is good enough.

After simulation, we have several Unclear Sets and each Unclear Set contains several variables. Now we should choose the suitable set and variables to start with. After we check an Unclear Set, we know which variables are equal and which are not. Then we say the Set becomes "clear" and we mark it. Since we use a learning mechanism to use the previous results, the order of the sets is important. The better order implies the better performance.

The simplest mechanism to choose the set is we choose the first set we meet. This mechanism is easy to implement and low cost but the performance is effected by the organization of the original gate representation.

Another mechanism is based on the "distance". Here the distance means how far the gate is from the inputs. In Figure 2, the distance of t is 1 and the distance of X is 2. Generally, we define:

$$distance(OS)=max\{distance(\alpha)+1, \text{ where } \alpha \text{ is an input for the gate whose output is OS}\}$$

We define the distance of an Unclear Set based on the distance of signal. We define:

$$distance(S)=max\{distance(\alpha), \text{ where } \alpha \text{ is a signal in S}\}$$

The procedure always choose an Unclear Set that has the lowest distance. Since the inputs are matched, the lower variables are the easier cases to check.

Now we discuss how to handle variables in an Unclear Set. We start with an Unclear Set with only two variables first. Suppose we want to check whether α is equal to β in design model M, we can test whether any of $F(M)\&\{\alpha = 1, \beta = 0\}$ and $F(M)\&\{\alpha = 0, \beta = 1\}$ is satisfiable or not. If NONE of them is satisfiable, we know α is equal to β . Another solution is we check $F(M)\&\{\alpha \cup \beta\}\&\{\neg\alpha \cup \neg\beta\}$. If it is NOT satisfiable, we also know α is equal to β .

There could be more than two signals in one Unclear Set, we have two choices to handle that. One is from [3], each time we choose two variables to test. Another way is based on following observation. If we want to check whether $x_1, x_2, x_3, \dots, x_n$ are always equal in model M, we just need to check whether $F_M\&(x_1 \cup x_2 \cup x_3 \cup \dots \cup x_n)\&(\neg x_1 \cup \neg x_2 \cup \neg x_3 \cup \dots \cup \neg x_n)$ is satisfiable. If it is satisfiable, those signals are equivalent. Otherwise the solution shows some of them is 1 (which satisfies $(x_1 \cup x_2 \cup x_3 \cup \dots \cup x_n)$) and some of them is 0 (which satisfies $(\neg x_1 \cup \neg x_2 \cup \neg x_3 \cup \dots \cup \neg x_n)$), so they are not equivalent. The experiments of two mechanisms is in Section 5.2.

4 Related Work

SAT-based methods becomes practical for real hardware designs in recent years[15]. In this section, we briefly introduce several works about SAT-based CEC.

E. Goldberg et al. show in [3] that the state-of-the-art SAT-based CEC method gets an exciting speedup by using structure informations and the performance is comparable to the BDD-based

commercial CEC tool. In [17], a technique integrating both SAT and BDD is introduced. It uses a functional learning and internal equivalences to reduce the search space. That actually captures the structure information in a similar way.

The research group in UCSB use Circuit SAT tool to handle the same problem [6]. The tool CSAT, which they developed, utilizes circuit topological information and signal correlations to enforce the searching. That is also based on the structure information.

R. Arora and M. Hsiao represents another way to capture structure information [16]. It first generates the basic implications between nodes by the implication lemma of each gate and then use those basic implications to generate strong two-node implications based on spanning. Those New two-node implications are translated into SAT clauses and the clauses are used to enhance the SAT procedure. It is also a way to capture structure information.

5 Experiment Results

We present experiment results based on the algorithm we mentioned in Section 3. In our approach, we use SATO[8] as the SAT tool to handle all the issues we mentioned above.

5.1 Simulation Experiments

In this section, we try to learn how the number of simulation affects the total procedure. It is obvious that there would be more chances to distinguish two inequivalent variables that after more simulations. Every simulation takes some time so we need to know how many simulations are suitable.

In Table 1, we show the simulation results for four problems from ISCAS-85[5]. We skip those

Table 1. Experiments for simulation

Problem Names	Total variables	Remained Variables/Unclear Sets				
		n=8	n=16	n=32	n=64	n=128
C432	353	301/70	299/127	296/136	295/143	295/145
C499	445	340/76	340/100	331/98	320/97	309/105
C1355	1133	1028/155	1028/357	1019/383	1008/385	1007/401
C1908	1791	1706/130	1699/289	1697/320	1691/327	1688/332
C3540	3339	3241/201	3180/586	3143/720	3128/802	3121/873

variables that we found unequal to any other variables and count the number of the remained variables and the number of the Unclear Sets. Those numbers decide the difficulty of the later procedure.

Table 1 shows that the average size of each Unclear Set is reduced while the number of simulation is increased and the number of variables in the Unclear Set is also reduced. According to the results in Table 1 we needn't do too many simulations. In most cases 32 simulations are enough. In later experiments, we use 32 simulations.

5.2 CEC Experiments

In this section, we show the experiments of Combinational Equivalence Checking(CEC). First we show the effects of different mechanisms. In Table 2 and Table 3, Original approach means the approach in[3], and our approach means we generate two constraints for any large set. S1 is the strategy to choose the first Unclear Set and S2 means to choose the Unclear Set with smallest distance. More informations are in Section 3.3.

Table 2. Experiments for getting Structure Similarity (S1)

Problem Names	Original approach				Our approach			
	Num of Test	Building time	Searching time	Total time	Num of Test	Building time	Searching time	Total time
C432	307	0.52	0.05	0.57	152	0.25	0.02	0.27
C499	408	1.24	0.27	1.51	156	0.51	0.21	0.72
C1355	1125	7.83	0.50	8.33	532	3.79	0.75	4.54
C1908	2570	28.31	2.36	30.67	477	5.12	2.62	7.74
C3540	2270	48.26	6.97	55.23	1053	22.61	10.73	33.34

From Table 2 and Table 3, we can see there are no distinguishable difference between the two

Table 3. Experiments for getting Structure Similarity (S2)

Problem Names	Original approach				Our approach			
	Num of Test	Building time	Searching time	Total time	Num of Test	Building time	Searching time	Total time
C432	308	0.68	0.04	0.72	152	0.26	0.04	0.30
C499	411	1.22	0.36	1.58	157	0.48	0.31	0.79
C1355	1134	7.78	0.46	8.24	533	3.76	0.85	4.61
C1908	2574	27.99	1.05	29.04	477	5.35	2.59	7.94
C3540	2277	48.58	7.75	56.33	1053	22.22	10.03	32.25

mechanisms to choose Unclear Sets. Since the time of computing the distance is not counted, choosing the first unclear set seems to be a better solution. The reason is when a design is generated, there is a highly chance it is generated under some order.

Table 4 shows the total equivalence checking time with or without using structure similarity. We

Table 4. Experiments for Equivalence Checking(EC) without/with Structure Similarity(SS)

Problem Names	EC w/o SS	SS time	EC with SS	Total time	Answer
C432	0.2	0.27	0.01	0.28	Matched
C499	8.05	0.72	0.3	1.02	Matched
C1355	26.13	4.54	0.3	4.84	Matched
C1908	31.15	7.74	0.2	7.94	Matched
C3540	2761.83	32.25	0.44	32.69	Matched

use the best time for getting structure similarity here.

5.3 Results Analysis

In this section, we do a brief analysis of our results.

In Section 5.2, we shows that the structure information can reduce the searching time for CEC problems. Our constraints can generate the same structure information as the constraints in [3]. It can reduce the testing number comparing to [3].

The total running time includes the building time and the searching time. The building time is reduced in our approach since we use fewer sub problems. But our approach increases the searching time. One reason is the constraints used in [3] are unit-clauses and SAT tools use those unit-clauses to simplify the problem while reading the problem in. We need more understanding about SAT tools to evaluate the time better.

Since most of the SAT instances generated in the problem is easy to solve, the building time is important in this problem. It can be shown in our experiments the building time is the greater part of the total time.

6 Future Work

In this section, we will discuss some possible ways to improve our work. Our experiment results are still not as good as the results in [3, 6]. There are the future directions of this work.

1. Reduce the Building time. In Section 5.2, we find the building time seems to be the bottle neck of the whole problem. We can reduce the building time by using more flexible SAT flames, sharing data structures and informations between sub problems and so on.

2. Try a better way to capture structure similarity. In Section 5.2, we can see the mechanism based on distance still has the potential to be improved. To do that, we need more understanding about the total procedure.

3. Try other SAT and circuit SAT tools to see whether we can improve the performance. A better way is to develop a SAT or circuit SAT tool just for our needs.

4. Use similar technique to improve sequential circuit equivalence checking. Equivalence checking for sequential circuit has become an important issue of hardware designers. Those studies includes [7, 14]. It is meaningful to capture the structure similarity for sequential circuits since we can reduce the searching space by using such information. Although it is more difficult to capture, we hope our work can be useful for sequential circuit equivalence checking.

7 Conclusions

The research on the SAT algorithms and tools is a hot area in the passed years. In this paper we have revisited the application of SAT tools to CEC. Our experiments show that the structure similarity captured by the SAT tools is useful to improve the performance of the equivalence checking and the new constraints can reduce the testing number. The possible improvement can be done by sharing the data structure and information between the SAT instances generated in this problem.

8 Acknowledgments

We would like to thank Prof. Robert Brayton, Dr. Alan Mishchenko, Prof. Li-C Wang, and Dr. Lu Feng for their help and suggestions.

References

- [1] B. Selman, H. Kautz, B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Interlligence (AAAI'94)*, pages 337–343, 1994.
- [2] B. Selman, H. Lvesque, D. Mitchell. a new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Interlligence (AAAI'92)*, pages 440–446, 1992.
- [3] E. Goldberg M. Prasad R. Brayton. Using sat for combinational equivalence checking. In *Proceedings of the conference on Design, automation and test in Europe (DATE'01)*, pages 114–121, 2001.
- [4] E. Goldberg, Y. Novikov. Berkmin.a fast and robust sat-solver. In *Proceedings of the 39th Design Automation Conderence (DAC'02)*, pages 142–149, 2002.
- [5] F. Brglez, and H. Fujiwara. A nutral netlist of 10 combinational benchmark circuits. In *Proceedings of IEEE Int'l Symp. Circuits ans Systems.*, pages 695–698, 1985.
- [6] F. Lu, L.-C. Wang, K.-T. Cheng and R. C.-Y. Huang. A circuit sat solver with signal correlation guided learning. In *Proceedings of the conference on Design, automation and test in Europe (DATE'03)*, pages 10892–10897, 2003.
- [7] F. Lu, M.K.Iyer g. Parthasarathy,L.-C. Wang, K.-T. Cheng, and K.C.Chen . An efficient sequential sat solver with improved search strategies. In *Proceedings of the conference on Design, automation and test in Europe (DATE'05)*, pages 1102–1107, 2005.
- [8] Hantao Zhang. Sato: An efficient propositional prover. In *Proceedings of the 14th International conderence on Automated Deduction (CADE-14)*, pages 272–275, 1997.
- [9] <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat>. *MINISAT*.
- [10] L. Zhang and S. MALik. The quest for efficient boolean satisfiability solvers. In *Proceedings of the 14th International Conderence on Computer Aided Deduction (CADE-14)*, pages 295–313, 2002.
- [11] M. Davis and H. Putnam. A computing prodedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
- [12] M. Davis, G. Logemann, D. Loveland. A machine program for theorem proving. *Communications of The ACM*, 5:394–397, 1962.
- [13] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. Sato: An efficient propositional prover. In *Proceedings of the 38th Design Automation Conderence (DAC'01)*, pages 530–535, 2001.
- [14] M.K. Lyer, G. Parthasarathy, and K.-T. Cheng. Satori - a fast sequential sat engine for circuits. In *Proceedings of the 2003 IEEE/ACM international Conference of Computer-Aided Design(ICCAD03)*, pages 320–325, 2003.

- [15] P. Bjesse, T. Leibard, and A. Mokkedem. Finding bugs in an alpha microprocessor using satisfiability solvers. In *Proceedings of the 13th International Conference on computer Aided Verification*, pages 695–698, 2001.
- [16] R. Arora, M. Hsiao. Using global structural relationships of signals to accelerate sat-based combinational equivalence checking. *Journal of Universal Computer Science*, 10(12):1597–1628, 12 2004.
- [17] S. Reda, and A. Salem. Combinational equivalence checking using boolean satisfiability and binary decision diagrams. In *Proceedings of the conference on Design, automation and test in Europe (DATE'01)*, pages 122–126, 2001.