QuteSAT: A Robust Circuit-based SAT Solver for Complex Circuit Structure

Chi-An Wu, Ting-Hao Lin, Chih-Chun Lee and Chung-Yang (Ric) Huang
Department of Electrical Engineering
National Taiwan University, Taiwan

Abstract

We propose a robust circuit-based Boolean Satisfiability (SAT) solver, QuteSAT, that can be applied to complex circuit netlist structure. Several novel techniques are proposed in this paper, including: (1) a generic watching scheme on general gate types for efficient Boolean Constraint Propagation (BCP), (2) an implicit implication graph representation for efficient learning, and (3) careful engineering on the most advanced SAT algorithms for the circuit-based data structure. Our experimental results show that our baseline solver, without taking the advantage of the circuit information, can achieve the same performance as the fastest Conjunctive Normal Form (CNF)-based solvers. We also demonstrate that by applying a simple circuit-oriented decision ordering technique (J-frontier), our solver can constantly outperform the CNF ones for more than 15+ times. With the great flexibility on the circuit-based data structure, our solver can serve as a solid foundation for the general SAT research in the future.

1. Introduction

With the dramatic improvements in the last decade, Boolean Satisfiability (SAT) solvers are casting a great impact on the Electronic Design Automation (EDA) industry. Many traditional EDA problems can now be revisited by the powerful SAT solvers.

Most of the advances in the SAT techniques come from the solvers that are designed on the Conjunctive Normal Form (CNF) representation, namely the Product of Sum (PoS) format. The most cited ones include: (1) two literal watching (with lazy update) scheme for efficient Boolean Constraint Propagation (BCP) [1][2], (2) conflict-driven learning and non-chronological backtracking [3][4], (3) quasi-static decision variable ordering and restart [2][5], and (4) careful engineering for a compact and efficient SAT solver [2][6].

Nevertheless, many of the problems in VLSI CAD are circuit structure-oriented in nature. In other words, if we can utilize the circuit structure information, we will be able to devise a better SAT search strategy. However, when we model these problems in CNF, the structure information is lost. In general, attempts to include circuit structure information into CNF-based SAT solvers have been unsuccessful due to the significant overhead [7]. Therefore, realizing the SAT algorithms on the circuit data structure directly, called circuit-based SAT, seems to be a reasonable solution.

Kuehlmann *et al.* implemented a Boolean reasoning engine on the AND/INVERTER graph (AIG) [8]. Their results showed that with simplified circuit structure, the circuit-based SAT can be as efficient as the CNF SAT. However, some circuit information may be lost during the transformation to the AIG. Ganai *et al.* proposed a hybrid SAT solver in which the original problem was represented in the circuit format, and the learned information was recorded as CNF clauses [9]. By applying two-literal-watching scheme on the longer clauses (i.e. the learned clauses), and table lookup method for the gates with smaller amount of fanins, their hybrid BCP can achieve the best results of all. Another circuit-based SAT solver by Lu *et al.* [10] tried to accrue more learning information from the internal signal correlations. Their results show that their circuit SAT can consistently outperform the CNF ones. However, for the circuits that do not have as many internal signal correlations, their solver may not perform as well. On the other hand, the non-clausal SAT solver in [11] generalized the two literal watch technique to the propositional formulas represented in DAG. However, they selected the watched points only among the gate inputs and thus may incur unnecessary complication on the checking of the circuit BCP.

In this paper, we propose a robust circuit-based SAT solver, called *QuteSAT*, in which its baseline solver (i.e. without using circuit information) can perform as well as the state-of-the-art CNF solvers (e.g. zChaff [2] and miniSat [6]), and with just a little help from the circuit information, it can greatly outperform the CNF ones. The key technologies of our solver include: (1) a generic watching scheme that can seamlessly work on all kinds of circuit gates (simple or complex gates), (2) an implicit implication graph that enables efficient conflict-driven learning and (3) careful engineering work to implement most of the advanced SAT algorithm on the circuit data structure. For comparison purpose, we also implement a simple "J-frontier" algorithm to demonstrate that the circuit information can greatly help in the SAT problem.

The outline of the paper is as follows: in Section 2, we describe several novel techniques for the efficient circuit-based BCP. Other algorithms that contribute to the success of our solver are presented in Section 3. In Section 4 the experimental results demonstrate the robustness of our solver. Section 5 concludes the paper.

2. Efficient circuit-based BCP

Boolean Constraint Propagation (BCP) is the most time consuming process in the SAT algorithm. It derives the logic implications in the deduction procedure. In a modern SAT solver, BCP may perform millions of implications per second, which is about 80% of the total SAT time [9]. Therefore, an efficient BCP is a critical factor for a powerful SAT solver.

With the "2-literal-watching" scheme, the BCP step implemented in CNF SAT usually yields a higher efficiency. Circuit SAT, on the other hand, usually adopts the "table-lookup" method for logic implications. This method is more efficient for circuits with limited gate types and with small number of fanins. However, for complex circuit structure, the "table-lookup" method may not be as efficient.

We propose several novel techniques to enhance the BCP process for the circuit SAT. Our main contributions include: (1) direct implication graph, (2) generic watching algorithm, and (3) implicit implication recording. Our methods enable the circuit-based SAT solver to have the similar BCP performance as the CNF SAT, and at the same time to maintain the great flexibility for the circuit SAT to work on the circuit netlist structure.

We will first review the similarities and differences between CNF and circuit-based BCPs.

2.1. CNF vs. circuit-based BCP

The BCP process in the CNF SAT is very simple. To satisfy a CNF formula, all the clauses must be asserted. In other words, each clause must have at least one literal assigned to '1'. This monotonic satisfiability check makes the BCP process in CNF SAT very simple and efficient.

Most of the modern CNF SAT solvers utilize the "2-literal-watching" scheme for BCP, which is essentially a "lazy evaluation" technique — any assignment on the non-watched literal will not be processed; only the clauses on the watching list of the implied literal will be updated. This can greatly reduce the number of useless updates in the BCP process.

The circuit-based SAT, on the other hand, usually builds on a netlist of different gate types, and for each gate type, it has even several different implications (Fig. 1). Therefore, it may not be easy to implement the "watching" scheme for the circuit SAT. However, after carefully examining the similarities and differences of the CNF and circuit-based BCPs, we find that the watching scheme can be generalized for complex circuit netlists. This is achieved by several of our novel techniques and will be described in the following subsections.

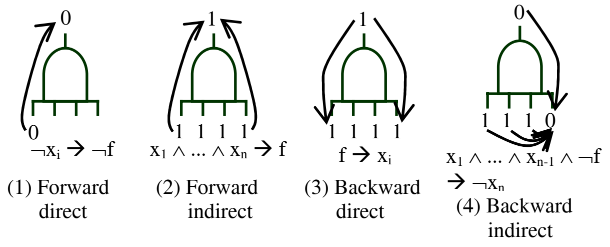
0
0
$\neg x_i \rightarrow \neg f$
(1) Forward direct
1
1 1 1 1
$x_1 \wedge \ldots \wedge x_n \rightarrow f$
(2) Forward indirect
1
1 1 1 1
$f \rightarrow x_i$
(3) Backward direct
0
1 1 1 0
$x_1 \wedge \ldots \wedge x_{n-1} \wedge \neg f \rightarrow \neg x_n$
(4) Backward indirect

Fig 1. Implications of an AND gate

2.2. Direct implication graph

The idea of circuit-based watching scheme may become easier to understand if we separate the direct from the indirect implications.

An implication is called a "*direct implication*" if it can be derived from a single source. On the other hand, it is called an "*indirect implication*" if it takes multiple sources to conclude this implication. For the AND gate in Fig. 1 as an example, the implications "$\neg x_i \rightarrow \neg f$" and "$f \rightarrow x_i$" are direct implications, while the implications "$x_1 \wedge \ldots \wedge x_n \rightarrow f$" and "$x_1 \wedge \ldots \wedge x_{n-1} \wedge \neg f \rightarrow \neg x_n$" are indirect.

The causality of the direct implications in a circuit depends only statically on the netlist structure, not dynamically on the other implications during the decision process. In other words, the relationship of the direct implications can be derived during the circuit preprocessing step, and it will remain unchanged in the entire search process. In addition, there will never be a conflict arisen from the direct implications. Therefore, we can record the direct implication relations as a graph in the circuit parsing step, and then in the BCP process, whenever a gate gets an implication, we can immediately obtain its direct implications by tracing the direct implication graph.

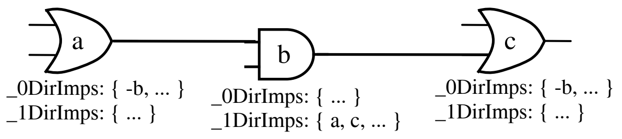

Fig 2. Direct implications graph

Please note that the direct implications actually correspond to the 2-literal clauses in the CNF, and many CNF SAT solvers also have special treatments for these clauses. For example, miniSat does not store the 2-literal clauses. Instead, it records them directly in the "watched list". This is similar to our "direct implication graph" idea.

2.3. Watching scheme for primitive gates

The primitive gates here refer to AND, OR, NAND, and NOR. Buffers and inverters are collapsed as attributes to the gate inputs. We will use an n-input AND gate to illustrate the watching scheme for the primitive gates. The rules for other primitive gates can be deduced similarly.

Consider the CNF of an n-input AND gate. It consists of n "2-literal clauses" and one "(n+1)-literal clause":

AND: $(x_1 + \neg f)\ldots(x_n + \neg f)(\neg x_1 + \ldots + \neg x_n + f)$

Since the 2-literal clauses have been recorded in the direct implication graph, we only need to consider the single clause for the indirect implication. It is worthwhile to note that the two indirect implications for the AND gate, namely "$x_1 \wedge \ldots \wedge x_n \rightarrow f$" and "$x_1 \wedge \ldots \wedge x_{n-1} \wedge \neg f \rightarrow \neg x_n$" in Fig. 1, actually corresponds to the same CNF clause $(\neg x_1 + \ldots + \neg x_n + f)$. Therefore, we can use an unified watching scheme to handle these two indirect implications.

The idea can then become straightforward — just adds the gate output 'f' and inputs 'x_1'... 'x_n' to the watched candidate list and select two of them as the watched pointers. The watched value for the output 'f', as suggested by the CNF clause, should be '0', while the watched values for the gate inputs 'x_1'... 'x_n' should be '1'. However, since there is no separated data structure for positive and negative literals in circuit SAT, we will record the "watching gates" as two lists ("watching-0" and "watching-1") in the gate data structure.

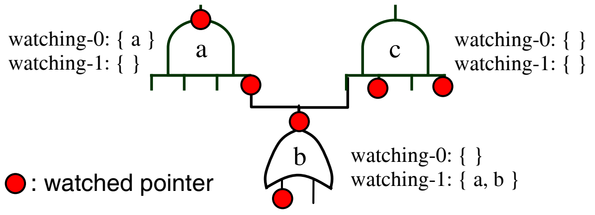

Fig 3. Watched pointers and watching gate list

2.4. Watching scheme for complex gate

The complex gates here refer to arbitrary Boolean function that can be represented by Boolean variables. We will illustrate the watching scheme for the complex gates by three examples: XOR, multiplexer (MUX), and Pseudo-Boolean constraint (PB) gates. The generic algorithm will be presented in the next subsection.

2.4.1. XOR gate An n-input XOR gate corresponds to 2^n (n+1)-literal clauses in CNF, each of which requires 2 watched pointers in the CNF SAT. The indirect implication of an XOR gate occurs only when there are n assignments on the gate output and inputs, be them '0' or '1', and when this happens, all the clauses will be satisfied at the same time. Therefore, these clauses can actually be watched together by two pointers on the gate.

We introduce the "watching-known" concept to our watching scheme. In other words, if a gate x is in the "watching-known list" of another gate y, when the value of gate y becomes known (i.e. 0 or 1), it will notify gate x to update its watched pointers and potentially generate a new indirect implication.

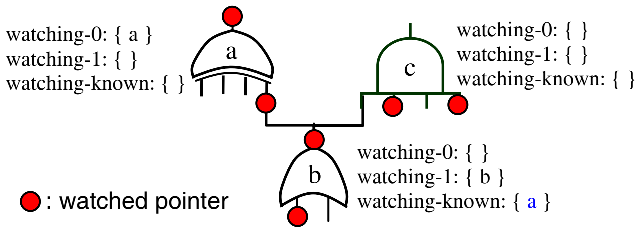

Fig 4. Introducing "watching-known" list

2.4.2. MUX gate The implications and corresponding CNF clauses of a MUX gate are as shown below.

MUX function: f = ¬s ∧ a + s ∧ b	
Implication	CNF clauses
$\neg s \rightarrow (f = a)$	$(s + f + \neg a)(s + \neg f + a)$
$s \rightarrow (f = b)$	$(\neg s + f + \neg b)(\neg s + \neg f + b)$
$(a = b) \rightarrow (f = a)$	$(\neg f + a + b)(f + \neg a + \neg b)$

There are six 3-literal clauses for a MUX gate, with each variable appearing twice in the positive and negative literals. Therefore, an intuition for the MUX gate watching scheme is to pick 2 watched variables from { f, s, a, b } for "watched known". However, this may miss some indirect implications. For example, if we are watching f and s (both have unknown values), while a and b have been assigned to the same known value, then we will miss the indirect implication that f should have the same value as a.

Therefore, we need to have a 3-watching scheme for a MUX gate. Although this may sound inefficient (i.e. 3 watched variables out of 4), yet compared to the six CNF clauses and 12 watched literals, our circuit-based BCP for watching MUX will perform better.

2.4.3. Pseudo Boolean gate A pseudo Boolean (PB) constraint is said to be in the normal form as:

$$\sum_{i=0}^{n} a_i \dot{x}_i \geq b \text{ for } a_i, b \in Z^+, \dot{x}_i \in \{0,1\}$$

where $\dot{x}_i$ denotes a literal x_i or x_i', and the terms $a_i \dot{x}_i$ are usually sorted by the coefficients in the descending order.

PB constraints are mostly used in describing a linear system and representing the 0-1 integer programming problem. To perform the BCP on the PB constraints, for example, miniSat+ converted the PB inequalities to CNF clauses so that the PB satisfiability can be checked by the CNF SAT solver [12]. However, the transformation from PB to CNF constraints usually results in significant overhead and may lose much implicability. On the other hand, Chai and Kuehlmann in [13] applied the "watching scheme" for PB constraint propagation. However, due to the overhead in updating the watched pointers and the number of watches, they decided to discard the watching scheme and use "counters" to implement the PB constraint propagation instead.

In order to be consistent in the BCP algorithm with other part of the circuit, we choose to stay with the "watching scheme" for the PB constraints. Different from [13], we do not change the number of watches during the decision process. The price we pay is that we may need to watch more variables by considering the worst case. For example, in the PB constraint "$4x_1 + 3x_2 + 2x_3 + x_4 >= 3$", we need 3 watched pointers for watching value '0' (rules will be described in the next subsection), while in [13] their method will pick 2 or 3 watches, depending on the values assigned. If we are watching { x_1, x_2, x_3 }, and x_4 = '0', then the new assignment x_3 = '0' will trigger the watch update. However, we cannot find another non-0 variable to update the watched pointer, and we cannot deduce any indirect implication yet either. Nevertheless, we find that our method is a good balance between the "counters" method, where all the variables are watched, and the dynamic number of watches approach, where the overhead in maintaining watches is big.

2.5. Generic watching algorithm

Considering the watching scheme on a gate, the main difference between circuit and CNF SAT is that in CNF we can have different watched literals for different clauses, while in circuit SAT these clauses need to be watched on the same gate by choosing pointers from the gate itself and its inputs. In other words, we need to design the circuit-based watching algorithm that can take care of all the watched literals for all the clauses at the same time.

Let's first review the principle of the CNF-based watching scheme — whenever there are value assignments on the non-watched literals, or '1' assignments on the watched literals, we need not to do anything. We can delay the evaluation of the clause until one of the watched literals is assigned '0', and then we will try to find a non-0 non-watched literal to move the watched pointer. If we fail to update the watched pointer, we will either conclude an indirect implication, or find a conflict assignment.

Following the same spirit, we design our generic circuit-based watching algorithm as follows:

<u>Generic watching algorithm for circuit SAT</u>:

1. Let the output and input pins of the gate be the watched candidates. Let '**n**' be the size of the watch candidate set.
2. Determine the 'watched value" for each pin: if assigning a value '*v*' on this pin may eventually lead to an indirect implication on other pin(s), then '*v*' is the watched value of this pin. For example, assigning a '1' to the input of an AND, or '0' to a variable of a PB constraint may eventually lead to an indirect implication. Therefore, '1' and '0' are their watched values, respectively. On the other hand, for an XOR or MUX gate, both '0' and '1' may lead to indirect implications. Therefore, their watched value is "known".
3. Find a minimum subset of watched candidates so that (a) assigning watched values on all the variables of the subset will produce an indirect implication, but (b) removing any of these assignments will void the implication. Let '**k**' be the size of this subset.
4. We will need (**n – k + 1**) watched pointers.
5. Whenever there are assignments on the non-watched pins, or non-watched value assignments on the watched pins, we do nothing. The update of the watched pointer is called only when there is a watched-value assignment on the watched pin.

For example, for a 4-input AND gate, we have 5 (= 4+1) watch candidates and need 4 assignments on the output or inputs to trigger the indirect implication. Therefore, the number of watched pointers is 2 (= 5 – 4 + 1). As for a MUX gate, the minimum number of assignments to generate an indirect implication is 2. Therefore, we need 3 (= 4 – 2 + 1) watched pointers for it. On the other hand, for the PB constraint "$4x_1 + 3x_2 + 2x_3 + x_4 >= 3$", the minimum subset that satisfies the step 3 in the above algorithm is { x_1, x_2 } (checking from the variables with largest coefficients). Therefore, we need (4 – 2 + 1) = 3 watch variables for this constraint.

It can be shown that the circuit BCP based on our watching scheme is safe and efficient because we will not miss any indirect implication or assignment conflict. In addition, for the primitive gates, it functions the same way as the CNF SAT. It can also seamlessly work on complex gates and generic Boolean functions like the PB constraints.

3. Engineering an efficient circuit SAT

In order to implement an efficient circuit-based SAT solver, we adopt most of the advanced CNF SAT algorithms, and then further improve it by the circuit specific techniques. The outline of our SAT algorithm is similar to most of the CNF-based SAT (e.g. [3]). We will just highlight some of our key improvements in this section.

3.1. Decision variable ordering

Decision variable ordering has exponential impact on the SAT runtime. Most of the modern SAT solvers adopt the quasi-static decision ordering approach [2][5] in which a good initial decision ordering is further improved by the learned information. We adopt the similar concept for our circuit SAT — the initial decision ordering is determined by the number of fanins and fanouts of the gates, and then the order is further adjusted by the learned gate during the search process. The experimental results show that the above baseline algorithm can achieve similar performance as the state-of-the-art CNF solvers.

Moreover, in order to testify the superiority of the circuit-based SAT solver, we also implement a simple "J-frontier" algorithm to prune out the irrelevant decision variables. Slightly different from the definition in ATPG, our J-frontiers are gates ("J-gates") whose justifications are necessary (not just sufficient) to satisfy the original SAT problem. After the BCP of a decision, we update the J-frontier by replacing the satisfied J-gates with their fanins that can determine the gate's output values. (Fig. 5) Note that this update is very cheap because we can easily figure out which fanins are the new J-gates by looking at the watched and "antecedent" (described later) pointers.

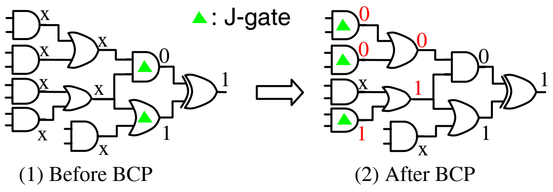

Fig 5. J-frontier

3.2. Learning

Modern SAT solvers apply various learning techniques to prune out the search space [3][14]. In CNF SAT, learned information is stored as clauses such that the SAT algorithms can be applied on both the original and the new clauses. In circuit SAT, on the other hand, the learned information is usually recorded as attached AND gates with tied '0' at the outputs. Therefore, the circuit SAT algorithms can then be executed on the same data structure.

Note that circuit SAT also provides the flexibility in storing and optimizing the learned information into different formats. For example, multi-level logic or more general constraints, etc. We will explore in this direction in the future.

3.3. Implicit implication graph

During the learning process, it is essential to figure out the causes of the implications. An intuitive way is to store this information as an *explicit* implication graph. However, this requires an extra amount of storage, and may lead to great overhead in the backtracking process.

On the other hand, modern CNF SAT solvers utilize an "antecedent" (also called "reason") pointer to record the implication source [2][6] — each implied "variable" has a pointer to the "clause" that produces this implication. The implication sources can then be obtained by excluding the corresponding literal of this variable from the antecedent clause.

Note that this idea may not be easily adopted by the circuit SAT solvers because there is no differentiation between "clause", "variable", and "literal" in the circuit database. The implication sources may be a gate, or a list of gates. How do we represent it with just an "antecedent" pointer?

We found that this is feasible by recording a flag for the implication type (DIRECT or INDIRECT) and with the watched pointers. Our algorithm for retrieving the implicit implication graph during the learning process is as follows:

<u>Implicit implication graph construction</u>:

1. If the implication type is "DIRECT", then the antecedent pointer is the single implication source.
2. If the implication type is "INDIRECT", then the implication sources are the watched candidates of the antecedent gate that are (a) non-watched variables, and (b) watched variables with watched values, excluding the implied pin.

For example, suppose the PB constraint "$4x_1 + 3x_2 + 2x_3 + x_4 >= 3$" acquires the implications in the order: { $x_1 = 0$, $x_2 = 0$ }, and the watched pointers are { x_2, x_3, x_4 }. Then both of the indirect implications, "$x_3 = 1$" and "$x_4 = 1$", have the antecedent pointer to this PB constraint, and the implication sources are: { $x_1 = 0$, $x_2 = 0$ }.

With the proposed implicit implication graph approach, we can perform the learning on circuit SAT very efficiently and with very low overhead.

4. Experimental results

We conduct our experiments on the Equivalence Checking (EC) problems for the benchmark circuits. The EC problem is to verify the functional equivalence between the original circuit and its synthesized/optimized revision. Without the use of the circuit information, it could be very difficult because functional representation could be arbitrarily dissimilar on both circuits.

The benchmark we use include ISCAS 85/89 and ITC99 circuits. We use the logic synthesis tool ABC [15] to map the circuit into two different cell libraries, one including complex gates, and the other with primitive gates only. The CNF representation is based on the Tseitin transformation. We select two state-of-the-art CNF SAT solvers: zChaff [2] and miniSat [6], and one well-known circuit-based SAT solver: NIMO [10], for comparison. All the experiments are conducted on a 3.2GHz Linux machine with 2GB memory.

Table 1. Equivalence checking (EC) experiments

Time: seconds	without circuit info			with circuit info		
	QuteSAT	zChaff	miniSat	QuteSAT -J	NIMO -u	NIMO
C2670	0.16	0.24	0.19	0.04	0.02	0.01
C3540	8.36	7.20	6.49	0.38	0.58	0.01
C5315	2.39	2.62	2.48	0.27	0.61	0.02
C7552	3.71	7.55	22.5	0.39	1.38	0.05
S13207	1.31	1.67	1.04	1.01	0.28	0.06
S35932	25.4	29.24	21.5	0.67	66.8	0.16
S38417	36.2	85.59	30.9	3.14	8.62	0.45
S38584	29.8	48.46	33.9	20.8	67.9	0.78
B12	0.69	1.43	0.69	0.2	0.24	0.03
B14	2529	>3600	793.4	16.7	3380	0.48
B15	116	168.8	83.0	15.9	158.5	2.37
B17	737	>3600	665.5	54.4	>3600	14.4
B20	>3600	>3600	3185	76.6	>3600	2.17
ave rank	1.69	2.69	1.39	N/R	N/R	N/R

We categorize our experiments into two sets: one without the use of circuit information — CNF SAT and our baseline solvers belong to this category. The other utilizes the circuit information in different ways — our "-J" option turns on the J-frontier technique, and the default NIMO applies signal correlation learning. As for the "-u" option of NIMO, it turns off the "explicit learning". However, based on our observation[1], it is still somehow taking the advantage of the circuit information.

The experimental results show that our baseline solver can achieve the comparable performance with the fastest CNF solvers. To the best of our knowledge, this is the first time that a circuit-based SAT solver can be as efficient as the CNF one while still retains the *complete* circuit information. This is mainly owing to our generic watching scheme for general gate types and the implicit implication graph that facilitates the fast conflict-driven learning.

On the other hand, the J-frontier method is a heuristic that should be effective not only for the EC problem but also for the general SAT problems like property checking. It concentrates on the most relevant search space, and with the advanced technique like "decision restart", our experimental results show that our circuit SAT can constantly beat the CNF SAT solvers for 15+ times.

The other circuit-based SAT solver "NIMO" utilizes the signal correlation learning which is especially useful for the EC problem. By learning the internal equivalent pairs of signals (like most of the commercial EC tools do), it is no surprise that their solver can achieve the best result.

[1] We conducted EC on structurally identical circuits and found that NIMO (even with –u) made 0 decision (of course no conflict) and finished in 0 time. This is impossible for all the other SAT solvers that do not use circuit (hashing) information.

5. Conclusion and future work

With the robust circuit-based SAT solver, we can conduct more advanced research on the general SAT problems. The possible directions include: (1) utilization of more structural information such as signal correlation, (2) circuit-based proof core extraction, (3) unbounded circuit-SAT model checker and (4) combining circuit SAT with PB or ILP solvers.

6. References

[1] H. Zhang, "SATO: An efficient propositional prover", Intl. Conf. on Automated Deduction 1997, pp. 272-275.

[2] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver", DAC 2001, pp. 530 – 535.

[3] J.P. Marques-Silva and K.A. Sakallah, "GRASP-A search algorithm for propositional satisfiability", T. Comp., vol. 48, pp. 506 – 521, 1999.

[4] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver", ICCAD 2001, pp. 279 – 285.

[5] E. Goldberg , Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver", DATE 2002, pp. 142 – 149.

[6] N. Een and N. Sörensson, "MiniSat: A SAT solver with conflict clause minimization", *SAT '05.*

[7] M. Prasad, A. Biere, A. Gupta, "A Survey of Recent Advances in SAT-Based Formal Verification", Software Tools for Technology Transfer 2005. pp. 156-173.

[8] A. Kühlmann, M. Ganai, V. Paruthi, "Circuit-based Boolean Reasoning", DAC 2001, pp. 232 - 237.

[9] M.K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik, "Combining strengths of circuit-based and CNF-based algorithms for a high performance SAT-solver", DAC 2002, pp. 747-750.

[10] http://cadlab.ece.ucsb.edu/downloads/nimo.html, "Sequential Circuit SAT Solver Homepage".

[11] C. Thiffault, F. Bacchus, and T. Walsh, "Solving Non-clausal Formulas with DPLL search", SAT 2004.

[12] N. Een and N. Sörensson, "Translating pseudo-Boolean constraints into SAT". JSAT 2006, pp. 1-26.

[13] D. Chai and A. Kuehlmann, "A fast pseudo-boolean constraint solver", TCAD, 24(3):305–317, 2005.

[14] S. Shuo and M. Hsiao, "Success-driven learning in ATPG for preimage computation", Design & Test of Computers 2004, pp. 504- 512.

[15] http://www.eecs.berkeley.edu/~alanmi/abc, "ABC: A system for sequential synthesis and verification"