A Signal Correlation Guided ATPG Solver And Its Applications For Solving Difficult Industrial Cases*

Feng Lu, Li-C. Wang, K.-T. (Tim) Cheng
Department of ECE
University of California at Santa Barbara
{lufeng, licwang, timcheng}@ece.ucsb.edu

John Moondanos and Ziyad Hanna
Design Technology
INTEL Corporation
{john.moondanos,ziyad.hanna}@intel.com

ABSTRACT

The developments of efficient SAT solvers have attracted tremendous research interest in recent years. The merits of these solvers are often compared in terms of their performance based upon a wide spread of benchmarks. In this paper, we extend an earlier-proposed solver design concept called (SCGL) Signal Correlation Guided Learning that is ATPG-based into a family of heuristics. Along with this SCGL family of heuristics, we classify benchmark examples according to their performance using the SCGL heuristics. With this study, we identify the class of problems that are uniquely suitable to be solved by using the SCGL approach. In particular, for solving difficult circuit-based problems at INTEL, our SCGL-based ATPG solver is able to achieve at least an order of magnitude speedup over the state-of-the-art SAT solvers. Our conclusion is that SCGL is an unique solver design concept that can complement heuristics proposed by others for solving circuit-oriented difficult problems.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Design Aids

General Terms

Verification, Algorithm

Keywords

Boolean Satisfiability, Boolean Equivalence Checking, ATPG

1. INTRODUCTION

Boolean Satistifiability has attracted tremendous research effort in recent years, resulting in the developments of various efficient SAT solver packages. Based upon particular SAT package architectures, researchers have tried to develop better heuristics to enhance SAT solver efficiency, by either speeding up the Boolean Constraint Propagation (BCP) procedure or finding a better decision ordering (or both).

Popular SAT solvers [1, 2, 3, 4] are designed based upon the Conjunctive Normal Form (CNF) representation. For many applications in CAD, applying SAT to solve a circuit-oriented problem often requires transformation of the circuit gate-level netlist into its corresponding CNF format [5]. In this circuit-to-CNF transformation, the topological ordering among the internal signals is obscured. All signals become (input) *variables* in the CNF format.

For solving circuit-oriented CAD problems, circuit structural information can be very useful. For example, researchers have developed various SAT solvers able to utilize circuit-related information to speed up the state-of-the-art SAT solver performance [6, 7, 8].

Recently, a circuit-based SAT solver was proposed in [9]. This ATPG-based solver utilized *signal topological ordering* and *signal correlation guided learning* (SCGL) to identify an effective decision ordering in the SAT search process. The main ideas were:

(1). If a group of signals can be identified in advance as highly correlated, then in the solver's decision variable selection, they can be grouped together. **(2).** When solving a SAT problem originating from a circuit, sometimes we can solve the problem more easily by following the topological structure. **(3).** When deciding value assignments to signals, it is more effective for the solver to select those values that are more likely to cause conflicts.

The ATPG-based solver includes two types of the SCGL heuristics: *implicit learning* and *explicit learning* [9] which differ in how they affect the decision ordering. In implicit learning, correlated signals are grouped together in the decision variable selection, and correlation information is used to guide the solver to assign values that are more likely to cause conflicts. In explicit learning, a set of K signals $\{s_1, \ldots, s_K\}$ are pre-selected based upon signal correlations. The decision ordering of these signals based on their topological order. Value assignments $\{s_1 = v_1, \ldots, s_k = v_K\}$ are pre-determined so that solving each sub-problem "$s_i = v_i$" ($1 \le i \le K$) is likely to cause conflicts. Moreover, the solver starts by *explicitly* following the topological order to solve each sub-problem in $\{s_1 = v_1, \ldots, s_k = v_K\}$, and then solve the original SAT problem.

In this paper, we extend SCGL into a family of heuristics $\{SCGL_0$, $SCGL_1$, ..., $SCGL_e\}$ based upon the *degree of SCGL learning*. In the two extreme cases, $SCGL_0$ involves no signal correlation learning while $SCGL_e$ follows the explicit learning approach where decision ordering is highly biased by the signal correlation information. In other words, the decision ordering in $SCGL_i$ is less biased by the signal correlation information than $SCGL_{i+1}$. We then apply this family of heuristics to different groups of benchmark examples. Our goal is first to demonstrate the following two principles:

(1). When circuit information is available and signal correlations are accurate, the more we use SCGL to guide the decision ordering, the better the results will be. **(2).** When circuit information is not available and as a result, signal correlations may not be accurate, SCGL becomes an overhead. The less we use SCGL to guide the decision ordering, the better the results will be.

Based upon these two principles, we can ask the following fundamental question: Does there exist a class of problems that are highly suited for applying the SCGL heuristics? From our experimental analysis, we concluded that indeed there exists such a class of circuit-based problem instances. They originate from the application of equivalence checking, and SCGL provides a very efficient solution for them. To demonstrate the effectiveness of our solver on this class of testcases, we conducted extensive experiments using units from an INTEL design in the Pentium class of microprocessors. For these testcases, an

*This work was supported in part by INTEL/MICRO Project #01-016 and in part by the MARCO/DARPA Gigascale Silicon Research Center (http://www.gigascale.org).

advanced equivalence check point matching approach [12] [13] did not work, and our solver achieves at least an order of magnitude performance improvement over a state-of-the-art SAT solver.

The rest of the paper is organized as follows: In Section 2, we explain the key ideas in SCGL and present the original *implicit* and *explicit* learning. Section 3 extends SCGL into a family of heuristics. In Section 4, we describe the implementation details of our solver. In Section 5 we present general experimental results and summarize our findings. In Section 6, we focus on specific experimental results obtained by applying our solver to difficult combinational equivalence problems at INTEL. Finally, in Section 7 we conclude by offering some comprehensive comments of the results in this paper and a preview of our future work.

2. SIGNAL CORRELATION GUIDED LEARNING (SCGL)

Most SAT problems encountered in CAD originate from circuits. In the traditional approach, a circuit is transformed into its CNF-equivalent form [5] and then a SAT solver is applied on the circuit in CNF. The major disadvantage of this transformation is the loss of the circuit structural information, specifically the topological ordering among the signals. When solving a circuit-originating problem, the topological information can be very useful, and with a CNF netlist, it is not easy to fully take advantage of this information.

2.1 An Illustrative Example

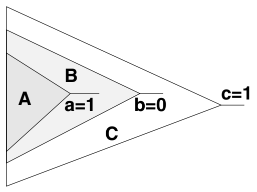

Figure 1: A SAT Process Following Topological Order

Consider the circuit in Figure 1. Suppose we want to solve a circuit SAT problem with the output objective "$c = 1$." If we apply SAT to prove $c = 1$, then potentially, the search space is the entire circuit. Now suppose we can identify, in advance, two internal signals a and b such that "$a = 1$" and "$b = 0$" *individually* are very **unlikely** to happen. Then, we can divide the original problem into three sub-problems: 1) solving "$a = 1$," 2) solving "$b = 0$," and then 3) solving "$c = 1$."

Since "$a = 1$" is unlikely to happen, conflicts are more likely to occur during the solving process. As a result, information can be learned more effectively. In a SAT solver, this information is stored as *learned clauses*, each of which represents a functional sub-space containing no solution. From another point of view, each learned clause specifies a constraint on a set of circuit signals, which has to be true based upon the circuit structure.

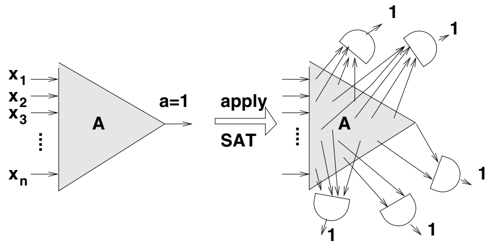

Figure 2: Learned Gates Accumulated by Solving "$a = 1$"

Most importantly, if we assume that solving "$a = 1$" is done only based upon the *cone of logic* headed by the signal "a" (the shaded area A in Figures 1 and 2) then *the learned clauses will be based upon the signals contained in the area A only*. Figure 2 illustrates the results of applying SAT for solving "$a = 1$." Regardless of whether the problem is satisfiable or not, a set of learned clauses can be collected. In Figure 2, they are represented as the *learned AND gates* whose outputs are 1.

As the solver finishes solving "$a = 1$" and starts solving "$b = 0$," all the learned information regarding the circuit area A can be used to help solving "$b = 0$." In addition, if "$a = 1$" is indeed unsatisfiable, then signal "a" can be assigned with 0 when solving "$b = 0$." Similarly, learned information from solving "$a = 1$" and "$b = 0$" can be used to help solving "$c = 1$."

Intuitively, solving the three sub-problems would be much faster than solving the original problem. This is because when solving "$b = 0$," hopefully much fewer (or no) decisions are required to go into area A. Hence, the search space is more restricted within the area B of Figure 1. Similarly, solving "$c = 1$" requires focusing the decision making only on area C. Moreover, the learned clauses accumulated by solving "$a = 1$" would be shorter because they are based upon the signals on area A only. Similar, the learned clauses accumulated by solving "$b = 0$" would be shorter because fewer decisions are made on area A. Therefore, conceptually, this strategy allows us to solve a complex problem *incrementally*.

We make two key observations: 1) the incremental process suggests that we should guide a solver to solve a sequence of pre-selected sub-problems following their topological ordering, and 2) the selection of the sub-problems such as "$a = 1$" and "$b = 0$" should be those most likely to be unsatisfiable. Intuitively, the search space for a likely unsatisfiable problem instance is more constrained and hence, conflict analysis can be more effective and more information can be accumulated in the learned clauses.

2.2 Potential Issues

If solving "$a = 1$" and "$b = 0$" does not cause many conflicts, then there is not much information to be learned from the solving processes. In this case, the above strategy may incur some overhead. Usually, this indicates the ineffectiveness of the method used to "guess" that "$a = 1$" and "$b = 0$" are unlikely to happen. Moreover, if solving "$c = 1$" does not depend on signals a and b much, then the above strategy can also incur overhead. To reduce the overheads, one can use the signal correlation information such as "$a = 1$" and "$b = 0$" *implicitly* rather than *explicitly* as described above. Implicitly, we can simply bias the value assignments more toward "$a = 1$" and "$b = 0$" during the solving of "$c = 1$" without first solving the sub-problems.

We note that intuitively, the above incremental strategy would not be effective for solving circuit satisfiability if its CNF form is used. This happens because with a 2-level OR-AND CNF structure, the topological ordering among the signals is lost. With a 2-level structure, the incremental strategy has very little room to proceed. Moreover, since both a and b are primary inputs in the 2-level OR-AND CNF circuit, the order of solving the sub-problems may become solving "$b = 0$" followed by solving "$a = 1$."

In addition to the signal correlations such as "$a = 1$" and "$b = 0$" pair-wise correlations of the forms "$signal_1 = signal_2$" and "$signal_1 \neq signal_2$" are also possible [9]. Then, the key issue becomes how to identify *in advance* those signal correlations for a given problem instance. One straightforward approach would be to utilize a random simulation approach [9].

2.3 Identifying Signal Correlations

Let $\{s_1, \ldots, s_n\}$ be n signals in a given circuit instance. Our goal is to identify four types of signal correlations "$s_i = 0$," "$s_i = 1$," "$s_i = s_j$," and "$s_i \neq s_j$" for all i and all $i \neq j$. For simplicity, we refer to these four types of correlation using the forms "$a = b$" and "$a \neq b$" with the understanding that b can be a signal or the constant "0."

In Algorithm 2.1, we demonstrate a simple procedure to compute the set of equivalence correlations (i.e. "$s_i = s_j$") using random simulation. To identify the remaining types of correlations "$s_i = 0$," "$s_i =$

1," and "$s_i \neq s_j$" we employ similar procedures.

In the algorithm, an *equivalence class* is a subset of signals mutually having the equivalence relationship. Accordingly, two signals will be classified into the same class if their simulation results from all the simulated assignments are identical.

Algorithm 2.1: RANDOM SIMULATION(*Circuit*)

```
comment: C is the initial equivalence class.
$i \leftarrow 0; C \leftarrow$ set of all signals;
$S = \{C\}$; while ($i < 4$) and ($S \neq S'$)
  do {
    $S' \leftarrow S$;
    Produce 32 random input assignments;
    Perform parallel logic simulation [14];
    Based upon the simulation results and
     the current equivalence classes,
      compute the new equivalence classes;
    $S \leftarrow \{$the new equivalence classes$\}$;
    $i \leftarrow i + 1$;
  }
return ($S$)
```

We note that deciding the equivalence classes among all signals can be achieved quite efficiently with a hash table. Hence, run time is actually close to linear instead of quadratic on the number of signals under consideration.

Our random simulation is simple. Each time 32 random input assignments are simulated together using a word (32 bits) in parallel logic simulation [14]. If repeating the simulation step several times does not lead to identifying any new equivalence class(es), then the simulation stops, and the set of the equivalence classes is returned.

2.4 Implicit Learning

There are two ways to utilize the signal correlation information discovered by the random simulation: *implicit learning* and *explicit learning*. In explicit learning, the solver explicitly creates a sequence of sub-problems based upon the signal correlations and solves them one by one following the topological order. As mentioned earlier, this strategy may be too biased and incur unnecessary overhead depending on the characteristics of the problem instance. To avoid such bias, in implicit learning, signal correlations are used to influence the decision variable selection and variable value assignment. This means that instead of creating a sequence of likely unsatisfiable sub-problems for the solver to solve explicitly one by one, correlation information is used only *within the decision variable selection procedure*.

Suppose we are given a solver without any SCGL heuristics. Usually, a solver comes with its own heuristics to select the decision variables. For example, in the latest Chaff package ZChaff [1], VSIDS is used in the selection. With implicit learning, we can add the signal correlations on top of the VSIDS decision variable selection.

For example, suppose a signal s_i is correlated with signal s_j as "$s_i \neq s_j$." During the solving process, whenever s_i gets to be assigned a value, we want to "immediately" make the decision to assign the same value to s_j. In this way, we "group" the two signals together in the solver's value-assignment process, and values are assigned in such a way that they are most likely to cause conflicts. Algorithm 2.2 describes a way to adopt the implicit learning.

Algorithm 2.2: SELECT DECISION VARIABLE(*variables*)

```
Suppose $s$ is just being assigned a value $v$ by implication (BCP)
if ($\exists s', s'$ is correlated with $s$) and ($s'$has not yet been assigned a value)
  then {
    select $s'$as the next decision signal
    if (it is equivalence correlation)
      then $s' \leftarrow \overline{v}$
      else $s' \leftarrow v$
  }
  else {
    use VSIDS (or other heuristics) to select a signal$s'$
    if ($s'$ is correlated with 0)
      then { $s' \leftarrow 1$ if it is equivalence correlation, or
             $s' \leftarrow 0$ otherwise }
  }
return ($s'$)
```

2.5 Explicit Learning

In explicit learning, a sequence of likely unsatisfiable sub-problems are created based upon the signal correlations identified by the random simulation. Internally, the solver would try to solve each sub-problem one by one following the topological order. After finishing solving all the sub-problems, all the learned information is stored in the learned gates (learned clauses) which are then used to solve the original SAT objective.

When solving each sub-problem, we need to decide whether or not to complete the solving. To limit the sub-problem solving process, one can count the number of learned gates accumulated. For example, the solver can move on to the next sub-problem whenever k new learned gates are found. If we allow the solver to complete for each sub-problem, then explicit learning becomes more like a commonly-used equivalence check point matching approach for solving equivalence checking problem. In check point matching, potential equivalent points are identified in advance. And then, following the topological order, each set of potential equivalent points are checked whether they are equivalent and if they are, to get merged together.

3. THE FAMILY OF SCGL HEURISTICS

Implicit learning and explicit learning are two SCGL approaches. However, depending on the degree of biasing the decision variable selection toward the signal correlations, we can derive a family of heuristics based upon these two approaches.

Limit the correlation size For equivalence correlation, "$s_i = s_j$", the simulation may identify a group of more than two mutually equivalent signals. For simplicity, we call this group size the *correlation size*. If the correlation size of a group is large, this may be due to the ineffectiveness of the random simulation. Hence, whether or not to use the correlations becomes questionable. In this case, we can limit the correlation size to a small number such as 3. For those groups whose correlation sizes are larger than 3, they are ignored.

Limit the location of correlations Suppose two signals have the correlations "$s_1 = a$" and "$s_2 = b$" and topologically, one is on the path of the other. Then, it is possible that one is the cause of the other. Hence, apply explicit learning to solve both sub-problems become redundant and may incur unnecessary overhead. To limit this overhead, one can identify a *cut* of the circuit and apply explicit learning to only those signals that fall between the primary inputs and the cut points. Another reason to use such a cut is that for a circuit with a long depth, it is unlikely that random simulation can uncover true signal correlations for signals close to the primary outputs. Hence, a cut will ensure that we do not apply explicit learning to those signal correlations falsely identified by the random simulation.

$SCGL_i$ In the case of implicit learning, suppose k signals are mutually equivalent. In Algorithm 2.2, if one signal is selected, there are $k-1$ possible signals to be selected. To exhaust the learning from this k-signal correlation, the variable selection may be invoked $(k)(k-1)$ times. This is because there are $(k)(k-1)/2$ pairs and for each pair, there are two ways to assign the pair of signals with different values. As described earlier, depending on the characteristics of a given problem instance, using signal correlations can be beneficial or can be an overhead. To avoid the bias of using signal correlations too much, we can further limit the implicit learning. In *$SCGL_i$* heuristic, a group of correlated signals can only be used in the variable selection procedure for no more than i times.

For example, *$SCGL_0$* does not involve the implicit learning at all and hence, depends only on the original decision variable selection heuristic such as VSIDS. In *$SCGL_1$*, each group of correlated signals can only be used once. Once it is used, regardless of which signals are selected, the group of correlated signals is dropped and will never be used again in Algorithm 2.2.

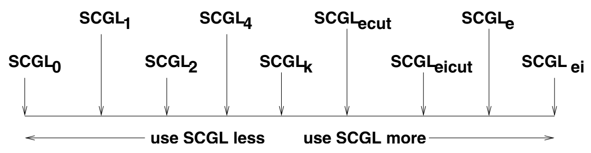

Figure 3: The Family of SCGL Heuristics

Figure 3 summarizes the family of SCGL heuristics discussed above. The more we go to the left, the less the signal correlation information would affect the decision variable selection. In the extreme case, $SCGL_{ei}$ combines both explicit learning and implicit learning without any limit.

- $SCGL_0$: This is the baseline solver without using any SCGL information.
- $SCGL_{1,2,4,k}$: Each group of correlated signals can only be used 1,2,or 4 times in the decision selection procedure. Then, it is dropped. $SCGL_k$ has no such limit and hence it is the original implicit learning. The maximum group size considered is 3.
- $SCGL_{ecut,eicut}$: A cut is enforced to avoid learning on signals close to primary outputs. No limit on the group size. *ecut* does not combine with implicit learning while *eicut* does.
- $SCGL_{e,ei}$: *e* and *ei* are similar to *ecut* and *eicut*, respectively except that no cut is enforced.

4. IMPLEMETATION DETAIL

The latest Chaff package ZChaff provided the baseline for our development. In our baseline solver, we implemented all the ideas in ZChaff, including the VSIDS decision variable selection, clause removal, watched literal [1], and UIP based conflict analysis [2]. In addition, our solver adds the justification frontier (J-node) [14] into the consideration of decision variable selection. Although our baseline solver borrows almost all the ideas from ZChaff, the implementation is different from ZChaff in several aspects:

- The input to the solver is assumed to be in a circuit format (such as the ".bench" format). After the circuit is read in, we transformed it into a netlist based upon only the 2-input AND primitive. In the netlist, we allow inverters to be associated with the AND gate inputs as attributes. Lookup tables are used for fast implications on the AND primitive [8].

  If an input is in its CNF form, we first convert it into a 2-level OR-AND circuit. Then, the circuit will be given to our circuit solver. We note that this could add some overhead to the representation of the problem.
- Since our ultimate goal is to develop a circuit SAT solver applicable to sequential circuits directly, internal circuit representation and data structures were designed for later extension to the sequential domain. For instance, "FRAME" objects were used to contain dynamic information that is valid within a time frame during sequential time frame expansion [14]. This adds additional overhead to our code, as comparing to ZChaff.
- We implement *restart* based upon the number of backtracks. When this number exceed 2^{11} the solver will restart.
- In ATPG terminology, a justification frontier (J-node) is a gate whose output has received a value, and some of its inputs need further decision(s) to justify the value[14]. In our implementation, only inputs to J-nodes are considered in the calculation of VSIDS for decision variable selection. However, we note that our definition of the J-node includes all the learned gates. Therefore, initially the restriction on J-nodes for decision making would make our solver behave differently from the ZChaff. However, after many learned gates are accumulated, effectively the two would follow the same VSIDS heuristic. Nevertheless, we note that since our solver makes different decisions at the beginning, the entire decision sequence can be very different from ZChaff.

5. EXPERIMENTAL RESULTS

For initial experiments, we collect three groups of examples. A "C-" example is originated from an ISCAS benchmark circuit. In the case denoted as "C-.eq" we constructed an equivalence checking circuit model by taking two copies of the same circuit. Each pair of corresponding primary outputs are XORed and all the outputs of the XOR go to an AND gate. The SAT problem is to ask if the output of the AND gate is 1. In each case, it is unsatisfiable. In the case denoted as "C-.opt" the two copies are structurally different where one copy is logically optimized with a synthesis tool. A "V-" example is a satisfiable testcase taken from [10]. A "P-" example is an unsatisfiable testcase taken from [10]. All experiments ran on a Pentium-4 2.4G machine with 1.5G RAM under Linux Mandrake 2.4.3.

5.1 Results on "C-" Testcases

		SCGL Family								
Cases	ZChaff	0	1	2	4	*k*	*ecut*	*eicut*	*e*	*ei*
c3540.opt	26	97	66	58	115	11	0.2	0.2	0.3	0.3
c5315.opt	84	35	22	37	20	15	0.3	0.3	0.3	0.3
c7552.opt	262	60	50	37	25	13	1.2	1.2	1.2	1.1
c3540.eq	35	105	82	78	54	27	0.2	0.2	0.3	0.3
c5315.eq	39	23	53	49	13	6	0.4	0.5	0.2	0.2
c7552.eq	190	46	34	43	37	16	1.3	1.3	0.9	0.8
s38417.eq	420	124	90	85	82	78	11	11	4	5
s38584.eq	316	152	143	144	141	148	54	55	40	41
Total	1372	642	540	531	487	314	68.6	69.7	47.2	49
c6288.eq	*	*	*	*	*	*	0.2	0.2	0.9	0.9

*Aborted after 1 hours.

Table 1: Results (secs) For UNSAT "C-" Testcases

Table 1 summarizes the results on "C-" testcases. The performance trend from the SCGL family of heuristics can clearly be observed. Based upon this trend and the illustration in Figure 3, we can conclude that this class of the examples are suitable for using the SCGL heuristics. "c6288.eq" represents an extreme case where only explicit learning heuristics can solve it. And when a right heuristic is applied, the run time is only less than a second.

5.2 Results on "V-" and "P-" Testcases

		SCGL Family								
Cases	ZChaff	0	1	2	4	*k*	*ecut*	*eicut*	*e*	*ei*
Vliw001	356	124	393	196	296	176	648	534	670	748
Vliw002	336	200	324	112	352	500	272	1510	615	1147
Vliw003	381	233	252	237	328	364	608	820	682	1070
Vliw005	1088	244	93	977	249	382	375	786	709	1129
Vliw006	308	428	310	348	310	572	434	755	789	1016
Vliw008	550	153	46	184	433	287	256	461	700	1243
Vliw011	305	150	98	65	63	125	242	633	525	893
Total	3324	1532	1516	2119	2031	2406	2835	5499	4690	7246

*Aborted after 1 hours.

Table 2: Results (secs) For SAT "V-" Testcases

Table 2 summarizes the results on the "V-" testcases. It is interesting to observe that the trend reverses in these examples. Based upon this reversing trend, we conclude that SCGL heuristics are not feasible for them. Hence, the more we depend on the SCGL, the worse the results would be. We note that since the input format of these examples contains CNF formulae, it is possible that our solver could not utilize the circuit topological information properly.

Table 3 summarizes the results for "P-" testcases. In this table, no trend can be observed. This can be due to three reasons: 1) This class

		SCGL Family								
Cases	ZChaff	0	1	2	4	*k*	*ecut*	*eicut*	*e*	*ei*
3Pipe1	62	90	86	88	75	87	40	65	40	65
3Pipe2	87	154	168	171	177	126	139	125	127	143
4Pipe1	753	1207	1313	1465	1306	1606	777	1966	1518	2520
4Pipe2	2505	3533	3151	3006	2820	*	*	*	2936	*
4Pipe3	*	*	*	*	*	*	*	1919	*	*

*Aborted after 1 hours.

Table 3: Results (secs) For UNSAT "P-" Testcases

of the examples are not suitable for SCGL heuristics. 2) Since the input format of these examples are not entirely circuit-based, it is unclear that the topological ordering used in our solver is indeed the topological ordering in the original circuits where those testcases were derived from. From our past experience [9], correct topological ordering information is crucial for identifying the correlations in the random simulation as well as for explicit learning. 3) We discovered that most of the correlations identified in the random simulation for these examples are false correlations. Hence, the random behavior can be a result of the ineffectiveness of the random simulation for uncovering the signal correlations for these examples.

One may argue that for those "C-" testcases, although SCGL heuristics are effective, these examples are "easy" problems if an advance equivalence check point matching approach [13] is used. We emphasize that our SCGL heuristics are different from equivalence check point matching. However, the fundamental question remains: Does there exist a class of problems where the SCGL heuristics provide a better solution while the equivalence check point matching does not work? In Section 6, this class of testcases will be discussed.

5.3 A different implementation

In Table 4, we alter our baseline solver by two ideas borrowed from Berkmin. First, the variable count used to select decision variable is based upon the number of learned gates that the variable has participated to produce (not the variable appears in). Second, the re-start strategy and clause removal follow the more aggressive strategies as those in Berkmin [11]. In table 4, we summarize the comparison results by using Berkmin's *equivalence learning* (EL) heuristic. Berkmin's results are slightly better than our new baseline, comparable to implicit learning heuristics but worse than our explicit learning heuristics. However, when comparing its results to ZChaff's, the speedup is noticeable.

			SCGL Family					
Cases	ZChaff	Berkmin	0	*k*	*ecut*	*eicut*	*e*	*ei*
c3540.opt	26	2	40	3.3	0.2	0.2	0.3	0.3
c5315.opt	84	3	9	2.3	0.3	0.3	0.3	0.3
c7552.opt	262	6.5	34	5.6	1.2	1.2	1.2	1.1
c3540.eq	35	2.8	23	4.3	0.2	0.2	0.3	0.3
c5315.eq	39	3.4	6	2.2	0.4	0.5	0.2	0.2
c7552.eq	190	7	26	5.1	1.3	1.3	0.9	0.8
s38417.eq	420	116	79	69	11	11	4	5
s38584.eq	316	145	138	126	54	55	40	41
Total	1372	285.7	355	217.8	68.6	69.7	47.2	49
c6288.eq	—	—	—	—	0.2	0.2	0.9	0.9

*Aborted after 1 hours.

Table 4: Results (secs) For UNSAT "C-" Testcases

In table 5, we discover that in general, for "V-" testcases, the run times improve significantly if the solver depends less on the SCGL. EL is not used in the Berkmin results. This further confirms the infeasibility of these testcases for using the SCGL heuristics.

6. SOLVING HARD INDUSTRIAL CASES

In this section, we report our experience with applying the SCGL solver to difficult test cases from an INTEL Pentium III class microprocessor. These testcases come from hard Combinational Equivalence Checking (CEC) problems of HDL specifications against their gate level implementation. In particular, from the generic results of

		SCGL Family				
Cases	Berkmin	0	1	2	4	*k*
Vliw001	165	227	462	230	354	429
Vliw002	117	47	186	160	177	351
Vliw003	213	15	202	168	472	339
Vliw005	96	61	132	213	217	316
Vliw006	150	126	168	303	260	420
Vliw008	103	11	87	102	168	256
Vliw011	68	40	67	160	64	323
Total	912	527	1304	1336	1712	2434

*Aborted after 1 hours.

Table 5: Results (secs) For SAT "V-" Testcases

the previous section, we decided to limit the following evaluation **only** to the $SCGL_{eicut}$ heuristic.

Circuit Characteristics: With recent advancements in the field of CEC [12, 13] one does not expect to see many difficult signals in a microprocessor for CEC. The number of logic levels that can exist within a pipestage of modern microprocessor designs is limited and therefore sophisticated CEC techniques prove highly efficient. However with increased market segmentation for microprocessors, our experience has been that a few specific signals can prove particularly hard for certain CEC technologies. Here, we studied the performance of our SCGL solver on testcases coming from four units of an INTEL microprocessor design of the Pentium class. These test cases were derived from combinational equivalence checking of RTL against gate level implementations of signals in these four units that belong to the memory cluster. Circuit C_m is a memory address generation circuit which contains multiple arithmetic units such as adders and multipliers. Circuit C_n performs memory address translation based on operand type, while circuit C_o implements a read-only interface to memory. Finally, circuit C_a performs memory command stream control.

Why They Are Difficult: These combinational equivalence problems have proven particularly difficult for BDD based techniques. Traditional monolithic BDD comparison does not work regardless of complicated variable ordering schemes including dynamic re-ordering. This is reasonable given the arithmetic logic that is included in these circuits. In addition, divide and conquer techniques based on BDDs which incorporate key point matching have also failed on many signals in these circuits. Intuitively, this is due to the fact that these circuits contain many re-convergent signals which create false negatives for cut-point techniques that do not offer false negative elimination. On the other hand, employing cut-point based techniques that offer false-negative elimination by employing parametric representations for functions, such as *normalized BBDs* [13], does not work because of the degree of reconvergence. However, the existence of significant reconvergence is a clear indication that signals in the netlists are highly correlated and this is something that should be exploited to speed up the process of a decision procedure. Also there is no significant structural similarity between the two netlists being compared because the one is coming from the RTL compilation process and the other is coming from the actual gate-level netlist.

Results Comparison: For the plots in Figures 4 and 5 we have to mention that the horizontal axis corresponds to a numerical index that was assigned to the various CEC problems in C_m, C_o, C_a and C_n, so that they would appear with increasing time requirement for the $SCGL_{eicut}$ heuristic. The plot in Figure 4 depicts the time performance comparison between our SCGL ATPG- based solver, ZChaff and Berkmin (the latter using the strategy for equivalence problems, i.e. "s 1"). The comparison takes into account only those pairs of signals that gave rise to extremely difficult combinational equivalence problems. In particular, the signals that appear in the plots of this section are only those that ZChaff, Berkmin and $SCGL_{eicut}$ could finish within 2 hours of CPU time. The experiments were conducted using Pentium III workstations running Red Hat Linux 6.2 with 4G of RAM. From the graphs one can conclude that the performance of $SCGL_{eicut}$ is at least two orders of magnitude better than the performance of ZChaff and at least one order of magnitude faster than Berkmin. We can also see that $SCGL_{eicut}$ could finish 40% more signals than ZChaff and 8% more signals than Berkmin.

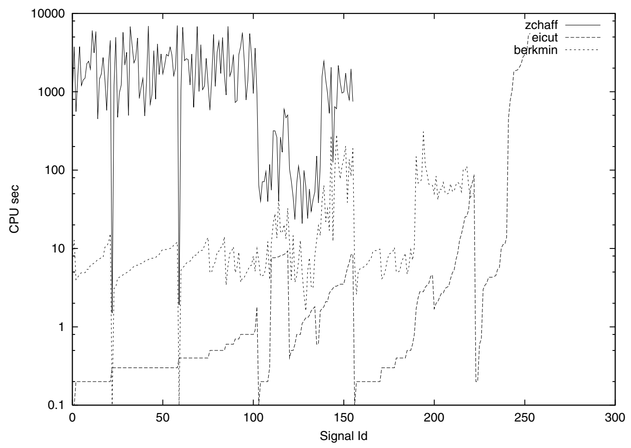

Figure 4: SCGL vs. ZCHAFF vs. Berkmin CPU time comparison on hard signals form circuit C_m, C_o, C_a and C_n

We also include the plot of Figure 5 where we compare the number of decisions that were required by ZChaff vs. the corresponding number for the $SCGL_{eicut}$ heuristic. Unfortunately, this number does not get reported by Berkmin in the information it prints. Clearly we would like to eliminate the possibility to attribute the results of Figure 4 to an extremely efficient coding implementation of $SCGL_{eicut}$. So we examined the number of decisions that each tool requires to prove that the signals are equivalent. As we see from Figure 5, the improved time performance of $SCGL_{eicut}$ can be directly attributed to the reduced number of decisions it requires. Evidently, the clauses learned by $SCGL_{eicut}$ bound the search space very efficiently, so that we could derive the equality of the outputs more efficiently by using far fewer decisions. By observing the similar shapes of the graphs in Figures 4 and 5 we can validate the conjecture that it is exactly the number of decisions in the solvers that determines the actual run-time requirements. Due to lack of space, we are omitting corresponding graphs comparing the number of implications and the number of learned clauses, but suffice it to say that they were found to be similar to those graphs in Figure 5.

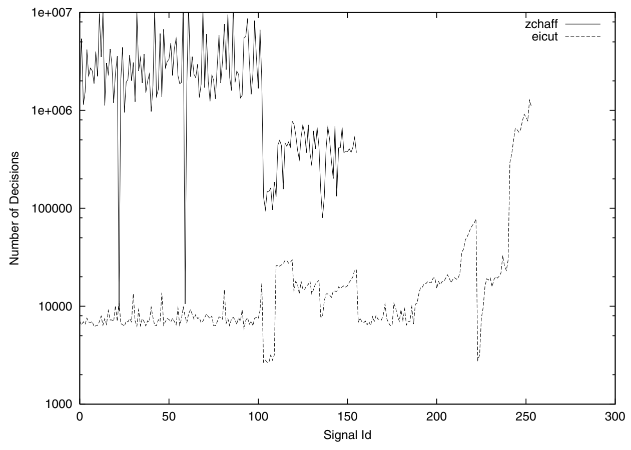

Figure 5: Number of decisions for SCGL vs. ZCHAFF on hard signals form circuit C_m, C_o, C_a and C_n

Finally, we must remind that in Figures 4 and 5 we have presented only comparisons based on CEC problems involving signals that were equivalent. We have also conducted experiments, on signals from the pre-tape-out versions of the four units we have been examining in this section, which were *not* equivalent due to design errors. Although BDD based techniques were failing on these signals as well, both ZChaff and $SCGL_{eicut}$ were able to handle them within a few seconds, without any tool showing any appreciable advantage over the other in those testcases.

7. CONCLUSION AND FUTURE WORK

In conclusion, this paper describes our implementation of an ATPG-based SAT solver whose design philosophy is to take advantage of the signal correlations and circuit topological information. We propose a new solver design concept called *incremental learn-from-conflict* where the solver learning process is carefully guided by signal correlations. We differentiate between the *implicit* learning and the *explicit* learning approaches both implemented in our solver. We discuss their strengths and weaknesses, and compare their performance to other state-of-the-art SAT solvers. Although we do not consider that our ATPG-based solver is superior to ZChaff (and Berkmin) in general on solving all problem instances (especially if the input format is CNF-based), we do conclude that if our solver is able to take advantage of the circuit structural information, significant performance improvements can be obtained. For the future work, we will continue the development of our ATPG-based solver to handle sequential circuits directly.

8. REFERENCES

[1] M.Moskewicz, C.Madigan, Y.Zhao, L.Zhang, and S.Malik, Chaff: Engineering an efficient SAT solver. *Proc. DAC 2001.*

[2] L.Zhang, C.Madigan, M.Moskewicz, and S.Malik. Efficient conflict driven learning in a Boolean satisfiability solver. *ICCAD 2001.*

[3] H. Zhang. SATO: An Efficient Propositional Prover. *Proc. of International Conference on Automated Deduction*, Vol 1249, LNAI, 1997, pp. 272-275

[4] J.P.Marques-Silva and K.A.Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans on Computers, vol.48, pp. 506-521 1999.*

[5] T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. In *IEEE Transactions on Computer-Aided Design*, pages 4-15, Jan, 1992.

[6] A.Kuehlmann, M.Ganai, and V.Paruthi. Circuit-based Boolean Reasoning. *DAC 2001.*

[7] Slawomir Pilarski and Gracia Hu. SAT with Partial Clauses and Back-Leaps. In *Proc. ACM/IEEE Design Automation Conference 2002*

[8] 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. In *Proc. DAC 2002*

[9] Feng Lu, Li-C. Wang, Kwang-Ting Cheng, and Ric Huang. A circuit SAT solver with signal correlation guided learning. In *Proc. European Design and Test Conference 2003*

[10] M.N. Velev. http://www.ece.cmu.edu/ mvelev Benchmark Suites, October 2000.

[11] Evgueni Goldberg, Yakov Novikov. BerkMin: A fast and robust Sat_Solver. *DATE*, pages 142-149, March 2002.

[12] A. Kuhlmann and F. Krohm. Equivalence Checking using Cuts and Heaps. *Proceedings Design Automation Conference, 1997.*

[13] John Moondanos, Carl Seger, Ziyad Hanna and Daher Kaiss. CLEVER: Divide and Conquer Combinational Logic Equivalence VERification with False Negative Elimination. *Proceedings Computer-Aided Verification Conference, 2001.*

[14] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman, Chapters 3,5, and 6: Logic Simulation, Fault Simulation, Test Generation, *Digital Systems Testing and Testable Design*, W.H.Freeman, 1990.