Forward Image Computation with Backtracing ATPG and Incremental State-Set Construction [*]

Kameshwar Chandrasekar
ECE Department, Virginia Tech
Blacksburg, VA 24061
kamesh@vt.edu

Michael S. Hsiao
ECE Department, Virginia Tech
Blacksburg, VA 24061
hsiao@vt.edu

ABSTRACT

Image computation is a fundamental step in formal verification of sequential systems, including sequential equivalence checking and symbolic model checking. Since conventional Reduced Ordered Binary Decision Diagram (ROBDD) based methods can potentially suffer from memory explosion, there has been a growing interest in using Automatic Test Pattern Generation (ATPG) / Boolean Satisfiability (SAT) based techniques in recent years. While ATPG has been successful for computing pre-image, image computation presents a very different set of problems. In this paper, we present a novel backtracing-based ATPG technique for forward image computation. We carefully alter the ATPG engine to compute the image cubes and store them incrementally in a Zero-Suppressed Binary Decision Diagram (ZBDD). In order to improve the efficiency of image computation, we propose three heuristics: (i) gate-observability based decision selection heuristics to accelerate ATPG, (ii) search-state based learning techniques supported with a proof for correctness, and (iii) on-the-fly state-set minimization techniques to reduce the size of computed image set. Experimental results on ISCAS '89 and ITC '99 benchmark circuits show that we can achieve orders of magnitude improvement over OBDD-based and SAT-based techniques.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*verification*

General Terms

Verification, Algorithm

Keywords

Image computation, ZBDDs, Model Checking, ATPG

[*]This work is supported in part by NSF Grants 0196470, 0098304, and 0305881

1. INTRODUCTION

Symbolic methods are widely used for formal verification of hardware systems. At the core of these formal methods, there is an image/pre-image computation step that performs state space traversal. In essence, computing image/pre-image requires the computation of the set of all next/previous states that can be reached from a given set of states in one cycle. Traditionally, Reduced Ordered Binary Decision Diagrams (ROBDD) were used, since they are canonical and can be efficiently manipulated. However, these methods suffer from potential memory explosion for large circuits and are generally suitable for only small and medium-sized circuits. Due to the increasing complexity and growing size of designs, several alternatives to ROBDD based methods have received attention in recent years.

In [1, 27], non-canonical structures such as Reduced Boolean Circuit (RBC) and Boolean Expression Diagrams (BED) are integrated with SAT solvers to perform efficient variable quantification for image/pre-image computation. The transition relation of the circuit is represented as an RBC/BED and the authors propose efficient quantification rules for specific sub-structures. However, in general, the length of the formulas may grow exponentially in size due to naive quantification. In [12], SAT solvers are integrated with BDDs to perform image computation. The transition relation of the circuit is represented in Conjunctive Normal Form (CNF). The quantification of the variables is performed during solution enumeration. The SAT solvers are used to provide a disjunctive decomposition of clauses and later BDDs are used to solve the end problems. This technique was improvised in [11, 13], where the inactive clauses are avoided dynamically and efficient variable selection heuristics are proposed.

In [18], an efficient pre-image computation technique is proposed for symbolic model checking using a pure SAT solver. The transition relation is represented as a CNF and variable quantification is performed during solution enumeration. After each solution is found, the state-cube is enlarged by re-building the implication graph to prune a larger search space. This technique was further improved in [16] by using a justification procedure on the circuit for state-cube enlargement. In [19], an approximate image is computed using the interpolants derived from refutation proofs of unsatisfiable CNF instances. This approximate image can be used to restrict the search space during SAT based unbounded model checking. In [25], an ATPG engine is used for pre-image computation. The transition relation is represented as a Boolean circuit and an ATPG engine is invoked to enumerate all the solutions that represents the complete pre-image. In order to prune the search-space, success-driven learning that avoids re-searching overlapping solution subspaces was introduced. Success-driven learning was further augmented in [4, 5] to prune larger search-spaces. However, ATPG based pre-image computation techniques are devoid of inbuilt conflict-driven learning that is inherent in SAT solvers. In [2], success-driven learning was integrated into a SAT-solver to take advantage of both success-driven learning and conflict-driven learning. In a recent approach for SAT-based model checking in [17], the transition relation is represented by an AND-INVERTER graph and efficient solution enumeration is performed by a specialized SAT solver. After obtaining each solution, the state-cube is enlarged by specifying the unspecified inputs and it is demonstrated to prune a larger search-space.

It should be noted that some of the above techniques, [2, 4, 16, 17, 18, 25], are specific to pre-image computation and cannot be directly extended to compute the image. While pre-image computation can be very useful, the pre-image space may potentially contain many unreachable states. On the other hand, all the states that are traversed during image computation from a legal initial state are guaranteed to be reachable. In this regard, we focus on image computation for sequential circuits using an ATPG engine. The novel features of this work are as follows:

1. We propose a novel backtracing-based ATPG for forward image computation. The image cubes are incrementally stored in a Zero-Suppressed Binary Decision Diagram, instead of adding them one by one.

2. We use a new decision selection heuristic and search-state based learning technique for the purpose of image computation.

3. We use cube minimization techniques 'on-the-fly' to reduce the size of the final state-set ZBDD.

Experimental results on ISCAS '89 and ITC '99 benchmark circuits show that we can achieve orders of magnitude improvement over OBDD-based and SAT-based techniques.

The rest of the paper is organized as follows. We introduce the preliminaries of image computation and ZBDDs in Section 2. In Section 3, we illustrate the novel image computation technique that stores the image cubes incrementally in a ZBDD. In Section 4, we propose new heuristics to improve the efficiency of the image computation technique. Experimental results for large ISCAS '89 and ITC '99 circuits are presented in Section 5. We conclude the paper in Section 6.

2. PRELIMINARIES

2.1 Image computation

Given a Transition Relation $T(X,I,X')$ and a set of initial states $S(X)$, the image of $S(X)$ can be computed by,

$$Image(X') = \exists_{x,i}\, T(X,I,X')\, .\, S(X)$$

where,

- X represents initial state elements
- I represents primary inputs
- X' represents next state elements

Essentially, the above equation can be used for image computation using formula manipulation, where the state sets and the transition relation are efficiently represented. We explain an image computation technique using the circuit structure in the following example. Consider the ISCAS89 circuit s27 shown in Figure 1. Gates $1,2,3,4$ are the primary inputs, $5,6,7$ are the present-state flip-flops and $5',6',7'$ are the next state flip-flops. The primary inputs (outputs) and present (next) state flip-flops are together called as inputs (outputs) to the circuit, when the meaning is clear from the context.

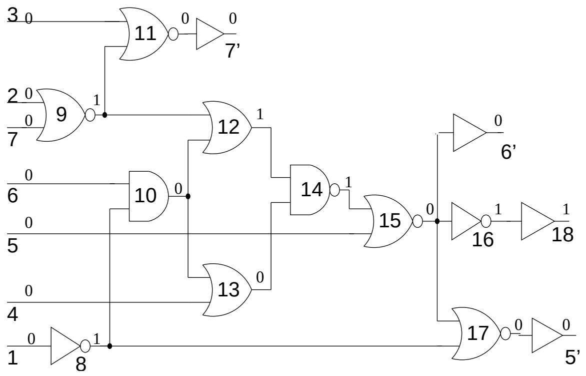

Figure 1: Image computation for s27 - 000 state

Let us compute the image for the initial state 000 using a **naive** technique. First, we assign the initial state values to the present state flip-flops $(5,6,7)$. Then, we have to assign all possible values at the primary inputs $(1,2,3,4)$. For each possible assignment at the primary inputs, we logic simulate the circuit and store the values obtained at the next-state flip flops $(5',6',7')$ as an image cube. For example, when we assign 0000 at the primary inputs, the next state values is 000. After enumerating the 16 possible values at $1,2,3,4$ in this naive technique, the following image cubes are obtained in the above example: $000, 001, 100, 101, 010$. It should be noted that we may obtain the same cube multiple times in this technique, albeit, the final solution set gives the complete image. We refer the reader to [10, 12, 22, 23] for existing BDD/SAT based image computation techniques.

2.2 Zero-suppressed BDD (ZBDD)

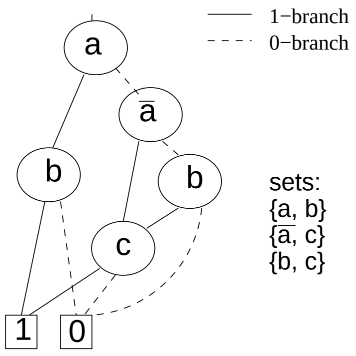

Figure 2: ZBDD

In [15], Minato introduced ZBDDs to represent *sets of combinations* compactly and perform set operations efficiently.

Sets of combinations $S = \{\{a,b\},\{\bar{a},c\},\{b,c\}\}$ are represented in a ZBDD as shown in Figure 2 (A). Each path from the root to TERMINAL-1 represents a set in the ZBDD. A 1-edge from a node denotes the presence of the element in the set and a 0-edge denotes its absence. If the variables in the ZBDD are linearly ordered, then it is called an ordered ZBDD. Otherwise, it is a general ZBDD. An excellent tutorial on ZBDDs is available at the website [21], and ZBDD algorithms have been implemented in publicly available CUDD [26] and Extra [20] packages.

ZBDDs are used in [18] to store the pre-image clauses. Each clause is added one by one to a ZBDD to compress its size. In this paper, we construct a general ZBDD incrementally during image computation to represent the image state cubes.

3. BASIC IMAGE COMPUTATION

In general, an ATPG engine attempts to generate an assignment that satisfies a given objective. This leads to a straight-forward application in pre-image computation as seen in [4, 5, 25], where the target next state-set is naturally set as the objective. In their setup, the primary inputs are chosen as decisions and circuit search-states are used for learning. The input assignments generated in the decision tree represent the complete pre-image set. This cannot be directly extended for image computation due to the following reasons:

1. Image computation is not symmetric to pre-image computation. In pre-image computation, exploring all possible assignments at the inputs is equivalent to exploring the complete search-space for all variables in the circuit. This is not true for image computation, if we explore all the possible assignments at the outputs.

2. A circuit represents a many-to-one mapping from input assignments to output assignments. Hence, it is not possible to set the target initial state as the objective and make outputs as the only decisions.

3. In order to benefit from search-state based learning similar to [4, 5, 25], it is necessary to choose the inputs as decisions and logic simulate the circuit i.e., we need a PODEM-like ATPG.

In this section, we explain a new procedure to perform PODEM like ATPG-based image computation. The basic procedure is outlined in Algorithm 1, where variables are quantified and a ZBDD is incrementally built that represents the complete image set.

Initially, a multi-level Boolean Circuit that represents the initial state set is constructed. This circuit is appended to the circuit-under-verification that represents the transition relation. The unspecified present-state flip-flops and primary inputs are chosen as decisions for the ATPG engine. After choosing each decision, the initial state circuit is also logic simulated to verify if we are still in the initial state space (care space). This is analogous to the use of don't-care space to constrain the search as in [14], but we use the care space instead of the don't care space. To choose a decision, we backtrace from an *unspecified* next state flip-flop through an X-path and pick a circuit input. Then, similar to PODEM, we logic simulate starting from the decision using an event driven mechanism. The next-state flip-flops

Algorithm 1: Image Computation
```
 1 computeImage(){
 2 if (!belongsToInitState()) then
     /*present state assignment ∉ initial state */
 3   return TERMINAL0 ;
   end
 4 if (allNxtStatesSpecified()) then
     /*found an image cube */
 5   return TERMINAL1 ;
   end
   /*choose an unspecified input */
 6 <decision, value> := backtraceToInput ( ) ;
 7 logicSimulate (decision, val) ;
 8 nxt_states0 := getSpecNxtStates ( ) ;
 9 z_node0 := computeImage ( ) ;
10 logicSimulate (decision, !val) ;
11 nxt_states1 := getSpecNxtStates ( ) ;
12 z_node1 := computeImage ( ) ;
   /*reset the input */
13 logicSimulate (decision, X) ;
14 z_node := buildPartialZbdd (nxt_states0, z_node0,
                               nxt_states1, z_node1) ;
15 return z_node ;
 }
```

that are specified in that decision level are noted. This is done recursively until all the next state flip-flops are specified, i.e., a fully-specified image cube is obtained. Then, the decision is flipped (enforce a backtrack) and the other branch of the decision tree is explored similarly. Finally, a partial ZBDD is stored for the next state flip-flops specified at each decision level.

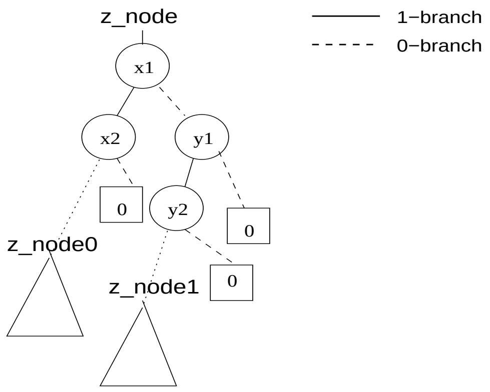

Figure 3: Partial Image ZBDD

A simple mechanism to construct the partial ZBDD at one recursive call of Algorithm 1 is as follows: Let next-state values $\{x1, x2, ...\}$ be specified for $decision := val$ and $\{y1, y2, ...\}$ be specified for $decision := !val$. The partial ZBDDs z_node0 and z_node1 are constructed earlier in $computeImage()$ during previous recursive calls to the algorithm. Since we construct the ZBDD in a bottom-up fashion, the resulting partial image ZBDD can be constructed as shown in Figure 3. Basically, the image cubes are added as sets to the ZBDD. After exploring all inputs assignments in such a fashion, this will incrementally lead to a ZBDD that represents the complete image at the end of ATPG.

4. EFFICIENT IMAGE COMPUTATION

In this section, we propose optimization techniques to improve the efficiency of basic image computation algorithm.

4.1 Decision selection heuristics

Several heuristics have been proposed in ATPG literature in choosing better decisions. These techniques aimed at detecting a fault and hence were mostly dependent on the testability measures of the circuit. Some of the popular techniques include [3, 6, 9, 24], where the difficulty of justifying/propagating a gate value is measured as controllability/observability measure. A recent decision selection heuristic for an all-solutions ATPG was proposed in [5] that considers the connectivity of variables to choose a decision. However, their technique specifically targets pre-image computation, where both solutions and conflicts can occur depending on the objective chosen.

In the case of image computation, our main idea is to efficiently explore all possible assignments at the inputs, and we backtrack only when the next state flip-flops are completely specified. Note that there is no specific objective in the case of image computation. We consider the following two factors to derive a decision selection heuristic:

1. If we always choose a decision that restricts the input assignment to the initial state space, then there will never be conflict-terminals, and we need not backtrack unnecessarily during ATPG.

2. If a small set of inputs can simply *specify* all the next-state flip-flops, then we can reach an image cube quicker in the decision tree. This can potentially reduce the depth of the decision tree.

Based on these two observations, we propose the following decision selection heuristic:

- If the output of the initial-state circuit is unspecified, we backtrace through the initial state circuit and choose a decision. This will help, albeit does not guarantee, to choose the input assignments that are within the initial state space.

- If the output of the initial-state circuit is specified, we backtrace in the original circuit using the *observability measures* of each gate. Since we want to specify the next-state flip-flops using minimum number of input assignments, it is desirable to choose an input that can easily propagate its value to the next state flip-flop.

In our method, we use the observability measures proposed in SCOAP [9] with a slight modification. For the sake of completeness, we explain how the observability measures are derived. For image computation, we are only interested at observing the input values at next-state flip-flops. For the next-state flip-flops, we assign the observability values as 0. For the primary outputs, we assign very high values (theoretically infinity) as their observability values. Then the observability values at the input of each gate is derived level by level based on equations proposed in [9]. For example, the observability at the input of an AND gate depends on the sum of the observability of its output and controlability of its other inputs to 1. Based on these measures, we backtrace through a gate with least observability measures and finally select an input as a decision.

4.2 Search-state based learning

ATPG based techniques largely depend on learning techniques that help to accelerate their performance. We show that the search-state based learning techniques, initially proposed in [8, 25] and later analyzed and enhanced in [4, 5], can be construed for image computation as well.

Let the logic decomposition of the circuit after each decision be called a search state for the ATPG. It should be noted that we are exhausting all 2^n input assignments in a decision tree fashion. Each branch in the decision tree corresponds to a search state in the circuit. This search state can be uniquely represented by a cutset of gate values in the circuit as shown in [5, 25]. These cutsets were used to identify equivalent search-states and the corresponding sub-tree was shared in the final pre-image.

In the case of the image computation discussed in the previous section, we store a ZBDD as the resulting image. We do not have a decision tree as such. However, it is not hard to see that we can share sub-ZBDDs and take advantage of search-state based learning. Similar to [5, 25], we store the cutsets, but instead of decision tree nodes we store the ZBDD nodes that correspond to the cutset. When the same cutset/search-state occurs again during ATPG, we simply link that ZBDD node instead of re-searching the search-space corresponding to that logic decomposition of the circuit. In order to formally define this technique we propose the following theorem:

THEOREM 1. *If two equivalent cutsets are obtained using Algorithm 1, then the corresponding partial ZBDDs generated thereafter will be isomorphic, for a given ATPG.*

PROOF. If two equivalent cutsets are obtained, then the logical decomposition of the circuit will be the same for both the cutsets. This will lead to a sub-circuit. Thereafter, the ATPG engine targets to compute the complete image of that sub-circuit. This image will be stored as a partial ZBDD. Since the sub-circuits are identical, the partial ZBDDs will be isomorphic. □

4.3 On the fly state-set minimization

All image/pre-image computation techniques generally suffer from solution explosion problem, which in turn leads to temporal explosion to obtain all the solutions and memory explosion to store them. The search-state based learning in ATPG helps to avoid the solution explosion problem by reusing previously explored search spaces and sharing partial ZBDDs. In order to reduce the size of the ZBDD further, we take advantage of two simple minimization techniques:

1. Subsumption:
$v_1 ... v_{i-1}.v_i.v_{i+1} ... v_n + v_1 ... v_{i-1}.v_{i+1} ... v_n = v_1 ... v_{i-1}.v_{i+1} ... v_n$

2. Cube minimization:
$v_1 ... v_{i-1}.v_i.v_{i+1} ... v_n + v_1 ... v_{i-1}.\overline{v_i}.v_{i+1} ... v_n = v_1 ... v_{i-1}.v_{i+1} ... v_n$

Although these are simple rules, they are very effective in reducing the state set since a lot of such structures occur during ATPG-based image computation. For illustration, we show the cube minimization technique in the ZBDD in Figure 4. In our implementation, we check for these structures while constructing the ZBDD in a bottom up fashion. For the Figure 4, ZA and ZB are considered to be minimized. It should be noted that we check only two levels to perform this minimization. Hence, the method is not complete, but

Table 1: Image computation for ISCAS '89 & ITC '99 circuits

Circuit	#FF	#gates	SAT		BDD		Basic ATPG			Efficient ATPG		
			#enum	Time(s)	#nodes	Time(s)	#enum	#nodes	Time(s)	#matches	#nodes	Time(s)
s1196	18	575	823	0.13	404	0.02	3187	7776	0.13	805	2326	0.08
s1269	37	634	4339	0.3	8423	0.1	9501	55K	0.36	1309	8071	0.11
s1512	57	887	6144	0.57	74	0.13	27724	56154	0.23	34	87	0.03
s9234	228	5866	24	0.14	6179	1.8	63	934	0.03	14	362	0.03
s3271	116	1728	-	T_O	1380	0.8	-	M_O	-	196	866	0.06
s3384	183	1937	-	T_O	322	0.5	-	M_O	-	60	144	0.03
s5378	179	3042	-	T_O	1M	54.7	-	M_O	-	1528	6495	0.34
s38417	1636	24K	-	T_O	3893	17.6	-	M_O	-	24	1692	0.1
s38584	1452	20K	66	0.51	11K	10.1	66	2131	0.06	8	1637	0.06
b11	31	770	64	0.01	55	0.02	64	153	0.02	7	27	0.02
b14	245	11K	-	T_O	-	T_O	-	M_O	-	32	215	0.04
b15	448	9K	1	0.23	734	11.8	1	451	0.03	1	451	0.03
b17	1415	32K	1	0.86	2735	263.6	1	1417	0.06	1	1417	0.06
b20	490	20K	-	T_O	-	T_O	-	M_O	-	34	460	0.06
b22	735	30K	-	T_O	-	T_O	-	M_O	-	34	805	0.07

Note: a) T_O - Time Out (1800s) *b) M_O - Memory Out (1M nodes)*

it is sound. In other words, our minimization is correct but all the cubes that can be minimized are not essentially minimized in the state-set ZBDD. It is easier to detect all such cubes if we have an ordered ZBDD. It should be noted that we can convert the general ZBDD into an ordered ZBDD and detect all the cubes that can be minimized. But, this will lead to an extra overhead that will generate a ZBDD with lesser cubes and not necessarily smaller size.

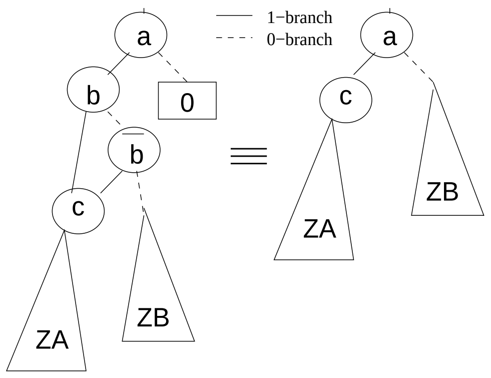

Figure 4: Minimization of ZBDD nodes

5. EXPERIMENTAL EVALUATION

The suite of techniques discussed in Sections 3 and 4 were implemented in C++ and integrated into a PODEM-like ATPG engine. We aimed at computing the one-cycle image for the all-zero initial state of large ISCAS89 and ITC99 circuits. Note that different initial states could easily be used instead of the all-zero state. We conducted the experiments on a Pentium 4, 3GB RAM machine, running the Linux Operating System. We compare the basic ATPG and the efficient ATPG with SAT and BDD techniques. The time limit was set to 1800 seconds and memory limit was set to 1 million nodes. For comparison with a SAT-based technique, we modified MINISAT [7] to compute the image for sequential circuits. After each solution is obtained, an enlarged blocking clause is added to constrain the solution state space until the complete search space is explored. Then we use the inbuilt *compute_reach* command of VIS [10] with dynamic variable ordering option to compute the one-cycle image for sequential circuits using BDDs.

The results are reported in Table 1. The circuit name, number of flip-flops and number of gates are reported in Columns 1, 2 and 3, respectively. Columns 4 and 5 report the number of solutions and time taken by the SAT-based technique. Columns 6 and 7 report the size of the solution set and time taken by BDD based technique. The number of solutions, size of solution set and time taken by the basic ATPG are reported in Columns 8, 9 and 10. The same results are reported for our proposed ATPG in Columns 11, 12 and 13. We count each search-state match as an enumeration, since we backtrack at that point. It can be seen that the proposed ATPG with enhanced heuristics consistently outperforms all other techniques. The basic ATPG aborts in many cases, such as s5378, s38417, b20 and b22, since the number of ZBDD nodes exceeds the pre-set limit. It is seen in Column 12 that the size of solution-set is reduced by several orders of magnitude because of the optimization techniques proposed in Section 4. On the other hand, the SAT based technique aborts for these large circuits because they lack the circuit knowledge and try to specify all the variables in the CNF. Due to the lack of structure in the CNF, it is difficult to learn from equivalent search-states in SAT. For small circuits, such as s1196, s1269 and s1512, the time taken by SAT and ATPG are almost the same. But for larger circuits, ATPG with optimization techniques clearly outperforms SAT. The BDD based techniques perform very well for medium sized circuits and are able to complete for a few large ISCAS circuits such as s5378, s38417, s38584, due to partitioning and dynamic variable ordering. However, in these cases, the solution-set is represented in a more compact manner by the general ZBDD. This is mainly because the BDD in VIS follows a linear order, whereas our general ZBDD trades-off space for canonicity. The BDD based technique spends a lot of time in finding a suitable variable order for these large circuits and this leads to Time-Out for circuits such as b14, b20 and b22. In fact, VIS could not finish constructing the transition relation for b14 which has 11,000 gates. The ATPG technique, on the other hand, directly computes the image on the Boolean circuit structure and does not need a separate representation for the transition relation.

6. CONCLUSION AND FUTURE WORK

We have proposed a novel image computation technique using an ATPG engine which constructs the image set on-the-fly as a general ZBDD. We used observability based decision selection heuristics to choose an input decision that can propagate quickly to the next state flip-flop. In addition, search-state-based learning is incorporated into the ATPG engine to avoided searching repeated spaces. A proof for correctness has also been provided. Finally, we use simple minimization techniques to reduce the size of the image ZBDD. Experimental results show that we can achieve several orders of magnitude improvement in both space and time over SAT and BDD based techniques for a number of large ISCAS 89 and ITC 99 circuits. Possible directions for future work include converting the image ZBDD into a Reduced Boolean Circuit (RBC) and developing an iterative framework for fixed point computation. Such a framework would help in promoting the proposed ATPG-based technique to check properties in Symbolic Model Checking.

7. REFERENCES

[1] P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic Reachability Analysis based on SAT-solvers. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, pages 411–425, 2000.

[2] Bin Li, M. S. Hsiao and S. Sheng. A Novel SAT All-solutions Solver for Efficient Preimage Computation. In *Proceedings of Design, Automation and Test in Europe*, pages 272–277, 2004.

[3] F. Brglez. On Testability of Combinational Networks. In *Proceedings of International Symposium on Circuits and Systems*, pages 221–225, 1984.

[4] K. Chandrasekar and M. S. Hsiao. ATPG-based Preimage Computation: Efficient search space pruning with ZBDD. In *Proceedings of High-Level Design Validation and Test Workshop*, pages 117–122, 2003.

[5] K. Chandrasekar and M. S. Hsiao. Decision Selection and Learning for an 'all-solutions ATPG engine'. In *Proceedings of International Test Conference*, pages 607–616, 2004.

[6] S. Chang, W. Jone, and S. Chang. TAIR: Testability Analysis by Implication Reasoning. *IEEE Transactions on Computer Aided Design*, 19(1):152–160, January 2000.

[7] N. Eén and N. Sörensson. An Extensible SAT-solver. In *Proceedings of SAT*, pages 502–518, 2003.

[8] J. Giraldi and M. L. Bushnell. Search state equivalence for redundancy identification and test generation. In *Proceedings of International Test Conference*, pages 184–193, 1991.

[9] L. H. Goldstein. Controllability/Observability Analysis of Digital Circuits. *IEEE Transactions on Circuits and Systems*, 26:685–693, September 1979.

[10] The VIS Group. VIS: A System for Verification and Synthesis. In *Proceedings of Computer Aided Verification*, pages 428–432, 1996.

[11] A. Gupta, A. Gupta, Z. Yang, and P. Ashar. Dynamic Detection and Removal of Inactive Clauses in SAT with Application in Image Computation. In *Proceedings of Design Automation Conference*, pages 536–541, 2001.

[12] A. Gupta, Z. Yang, P. Ashar, and A. Gupta. SAT-based Image Computation with Application in Reachability analysis. In *Proceedings of Formal Methods in Computer-Aided Design*, pages 354–371, 2000.

[13] A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik. Partition-based Decision Heuristics for Image Computation using SAT and BDDs. In *Proceedings of International Conference on Computer Aided Design*, pages 286–292, 2001.

[14] S.-Y. Huang, K.-T. Cheng, K.-C. Chen, C.-Y. Huang, and F. Brewer. AQUILA: An Equivalence Checking System for Large Sequential Designs. *IEEE Transactions on Computers*, 49(5):443–464, May 2000.

[15] S. Minato. Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proceedings of Design Automation Conference*, pages 272–277, 1993.

[16] H.-J. Kang and I.-C. Park. SAT-based Unbounded Symbolic Model Checking. In *Proceedings of Design Automation Conference*, pages 840–843, 2003.

[17] A. Gupta, M. K. Ganai and P. Ashar. Efficient SAT-based Unbounded Symbolic Model Checking using Circuit Cofactoring'. In *Proceedings of International Conference on Computer Aided Design*, pages 510–517, 2004.

[18] K. L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In *Proceedings of Computer Aided Verification*, pages 250–264, 2002.

[19] K. L. McMillan. Interpolation and SAT-based Model Checking. In *Proceedings of Computer Aided Verification*, pages 1–13, 2003.

[20] A. Mishchenko. Extra v 2.0: Software library extending CUDD. In *http://www.ee.pdx.edu/ alanmi/research/extra.htm.*

[21] A. Mishchenko. An Introduction to Zero-suppressed Binary Decision Diagrams. In *http://www.ee.pdx.edu/ alanmi/research.*

[22] I.-H. Moon, G. D. Hachtel, and F. Somenzi. Border block Triangular Form and Conjunction Schedule in Image Computation. In *Proceedings of Formal Methods in Computer-Aided Design*, pages 73–90, 2000.

[23] I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To Split or to Conjoin: The Question in Image Computation. In *Proceedings of Design Automation Conference*, pages 23–28, 2000.

[24] S. Seth, L. Pan, and V. D. Agrawal. PREDICT: Probabilistic Estimation of Digital Circuit Testability. In *Proceedings of Fault Tolerant Computing Symposium*, pages 220–225, 1985.

[25] S. Sheng and M. Hsiao. Efficient Preimage Computation using a Novel Success-driven ATPG. In *Proceedings of Design, Automation and Test in Europe*, pages 822–827, 2003.

[26] F. Somenzi. CUDD: CU Decision Diagram Package 2.4.0. In *http://vlsi.colorado.edu/ fabio/CUDD/*, 2004.

[27] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient symbolic model checking. In *Proceedings of International Conference on Computer Aided Verification*, pages 124–138, 2000.