SAT-Based Combinational Equivalence Checking with Don't Care

Myoung-Jin Nam[1] Chang-Hun Sung[2] Jin-Young Choi[1]
[1]Department of Computer Science and Engineering, Korea University,
Anam 5-street, Sungbuk-gu, Seoul, 136-701, Korea
Tel.: +82-2-929-8681, Fax.: +82-2-953-0771
[2]Department of Computer Science, Korea Military Academy,
Gongreung-dong, Nowon-gu, Seoul, 139-240, Korea
Tel.: +82-2-2197-2575
E-mail : mjnam@formal.korea.ac.kr, kucse@kma.ac.kr, choi@formal.korea.ac.kr

Abstract: In recent years, many approaches have been investigated for combinational equivalence checking problem. Despite the huge size of works, there is no work for combinational equivalence checking dealing with don't care. In this paper, we propose a method for combinational equivalence checking with don't cares. In particular, our technique proves the logical-consistency of circuits containing don't cares. In checking equivalence, we use SAT for equivalence checking problem instead of using BDDs or simulation. In verification, there can be false negatives which come from don't care space. False negatives can be eliminated using ternary simulation and model accumulation.

Keywords—**Equivalence Checking, Don't Care, Satisfiability**

1. Introduction

Combinational equivalence checking (CEC) is one of the most widely used technologies [2], [3], [4], [5], [10] in the verification of digital circuits. During synthesis process, a register-transfer level (RTL) description, given in e.g. VHDL or given as a set of boolean expression, is translated into a gate-level description. In this process, there is a need for checking equivalence of a specification and an implementation because we cannot guarantee to be error free. When F is an input into synthesis and G is a results of synthesis, we have to prove the equality F=G to check equivalence of a specification and an implementation. Most works have met with considerable successes in this area, however, there is no research for CEC with don't care (X) value. Most methods are suggested and used on the assumption that circuits to be verified do not contain X. In real world, Xs assignments are frequently used in RTL descriptions for the purpose of logic minimization, and this requires a need for efficient ways to deal with Xs during verification. Synthesized designs can be significantly smaller due to Xs, however, X values can cause verification to be more complicated.

Although approaches for CEC have not taken X value into account, several approaches have been suggested for CEC. These techniques are based on BDD, SAT or test generator. Among these, most widely used technologes are based on BDD. [4]. These approaches are to build BDDs representation for each function and to compare the BDDs for identity. But using BDDs exhibits state explosion problem which is an exponential growth of memory in the size of function. Therefore, many methods using other techniques instead of using BDDs have been suggested, and using SAT is one of these methods that do not explode in terms of memory. Actually, memory growth is more serious than time growth. SAT-based verification requires more time to solve the problem, than to allocate more memory. Furthermore, many researches have been suggested for efficient SAT solving in recent years. There have been significant advances in SAT algorithms [8], [9], [10], and many publicly available SAT solvers [11], [12], [13] are developed. As there have been remarkable improvements in SAT solving, SAT have been applied to many areas [14].

This paper suggests a SAT-based method for CEC. In particular, this technique proves the logical-consistency of circuits with X. We verify equivalence of a RTL design and a gate-level design. RTL design circuits contain X values, but gate-level design circuits do not, hence, it is hard to use exisiting approaches for hardware verification. We describe a new method that allows a technique for CEC to handle X values. In this process, we can encounter false negatives from X space. We use model accumulation and ternary simulation to block the false negatives.

The rest of the paper is organized as follows: section 2 addresses previous methods based on dual-rail encoding to verify equivalence of a RTL design and a gate-level design. In section 3, we describe the details of our SAT-based verification methodology. Section 4 describes how to eliminate false negative which comes from X-space. Finally, we present experimental results and conclude.

2. Related Work

There are two comparison modes in available formal equivalence checkers [7], [15]. They are based on how an equivalence checker compares Xs assignments.

1. Logical-equality mode : In logical-equality mode, all X values must match exactly between two circuits.

2. Logical-consistency mode : X value in the golden design is considered a match to 0,1, or X in the implementation design, so two designs are logically-consistent in every case except the case which primary outputs (PO) of the two designs have different binary values.Checking in logical-consistency mode is appropriate in the verification of a RTL design and a gate-level design, because the gate-level design does not contain X value. Logical-consistency verification mode is a weaker concept than logical-equality mode, but logical-consistency verification can be more important in the real world.

To check logical-consistency, the concept of dual-rail encoding [1] can be used. Consider representing a domain 0,1,X. At least 2 bits should be used to describe ternary values. The value of free variable a can be encoded as 2-arity vector (a.1, a.0). Ternary value 0 is encoded as (0,1), 1 as (1,0), and X as (1,1). Note that (0,0) is not used. While nodes in RTL design can have three values 0,1,X, encoded nodes from using dual-rail have only binary values. One can use ternary extensions of boolean function as shown in Table 1. Circuits to be verified are transformed using this encoding, and then the logical-consistency of two circuits are proved. The method based on dual-rail encoding is very simple, but using this encoding to verify the logical-consistency can be expensive and time requirements may grow by 2 square.

3. Proposed Method

Two Circuits to be checked are combined into a miter to use SAT. Once the miter is constructed, we transform it into CNF representation. An SAT algorithm starts by asserting that two circuits to be verified are not logically-consistent. The SAT solver then searches through a series of implications and case splitting steps until a so-

operation f()	Encoded versioin f.1()	f.0()
a	a.1	a.0
¬a	a.0	a.1
a ∧ b	a.1 ∧ b.1	a.0 ∨ b.0
a ∨ b	a.0 ∨ b.0	a.1 ∧ b.1
a Xnor b	(a1 ∧ b1) ∨ (a0 ∧ b0) (a0 ∨ b1) ∧ (a1 ∨ b0)	(a0 ∨ b0) ∧ (a1 ∨ b1) (a1 ∧ b0) ∨ (a0 ∧ b1)
a Xor b	(a1 ∨ b1) ∧ (a0 ∨ b0) (a1 ∧ b0) ∨ (a0 ∧ b1)	(a0 ∧ b0) ∨ (a1 ∧ b1) (a0 ∨ b1) ∧ (a1 ∨ b0)

Table 1. Dual-rail Versions of Ternary Operations

lution is found. If all choices are exhausted and no satisfying solution is found, the algorithm proves the unsatisfiability of the CNF converted from the miter. Thus we can conclude that two designs are logically-consistent (or logically-equivalent if no X is involved.). On the other hand, if a satisfying solution is found, then logical-consistency cannot be established, and we can provide a witness as a counterexample. Using this concept as a basis, we propose a new method [16] for combinational equivalence checking with Xs using SAT.

3.1 CNF Transformation

Given an AND gate c = AND(a,b), we can trace as follows:

Circuit to CNF Translation
c = AND (a, b)
$a \wedge b \leftrightarrow c$
$\equiv (a \wedge b \leq c) \wedge (c \leq a \wedge b)$
$\equiv (a \wedge b \rightarrow c) \wedge (c \rightarrow a \wedge b)$
$\equiv (\neg a \vee \neg b \vee c) \wedge (\neg c \vee (a \wedge b))$
$\equiv (\neg a \vee \neg b \vee c) \wedge (\neg c \vee a) \wedge (\neg c \vee b)$

This is a well-known transformation, but it is impossible to convert circuits having X values to CNF using only this transformation. To transform gates with Xs into CNF, we propose the following way. For simplicity, we assume that circuits contain only AND or OR gates.

CNF tanslation with don't care
C = AND(a, X)
$0 \leq c \leq a$
$\equiv (0 \rightarrow c) \wedge (c \rightarrow a)$
$\equiv (\neg c \vee a)$

By making assignments to input variables and performing boolean constraints propagation rule (BCP), we can get an output value of a gate. The output variable c equals X when the input variable a has 1. The variable c becomes 0 when a has 0. As above, we generate CNF from OR-gate with X as follows:

CNF translation with with don't care
c = OR(a , X)
$a \leq c \leq 1$
$\equiv (a \rightarrow c)(c \rightarrow 1)$
$\equiv (\neg a \vee c)$

For example, the circuits shown in Figure 1 is represented as (¬c∨a) ∧ (d↔c∧b) ∧ (¬e∨d). This formula is translated into CNF (¬c∨a) ∧ (¬c∨¬b∨d)∧(¬d∨c)∧(¬d∨b)∧(¬e∨d)

3.2 Logical-Consistency Checking

In equivalence checking, two circuits are not logically-consistent if the output of XNOR-gate has 0 in the miter. If the formula in CNF

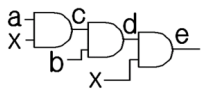

Figure 1. Circuit with Don't Care

which is converted from the miter is unsatisfiable, then the two designs are equivalent. In other words, the formula is satisfiable means that we have the condition which two designs are not equivalent. To verify the equivalence of a specification (a golden design) and an implementation (an implementation design), let the specification be CNF S, the implementation be CNF I, and be formula $F \equiv S \wedge I \wedge (H \leftrightarrow S \text{ XNOR } I) \wedge (\neg H)$. The formula F is unsatisfiable means that S and I are equivalent, and F is satisfiable means that S and I are not equivalent. However, the satisfiability of the formula always does not mean that S and I are not equivalent because of X value. Consider the following example shown in Figure 2:

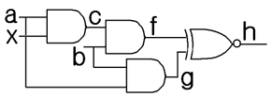

Figure 2. Circuit with Don't Care

It is clear that f and g are logically-consistent. Consider the following Table 2 which represents the values of outputs by make assignments to PIs, a and b. Note that PIs can have only binary values.

a	b	f	g	h
0	0	0	0	1
0	1	0	0	1
1	0	0	0	1
1	1	X	1	X

Table 2. Truth Table for Figure 2

Table 2 shows that g is an implementation of f since g and f are logically-consistent. That is, if h cannot have the value 0 (h has only 1 or X), then we can conclude that g is an implementation of f. To verify the logical-consistency, we must prove the unsatisfiability of the formula Φ

$$\Phi \equiv (c \leq a) \wedge (f \leftrightarrow c \wedge b) \wedge (h \leftrightarrow (f \leftrightarrow g)) \wedge (g \leftrightarrow a \wedge b) \wedge (\neg h)$$

But, we can find following two satisfying assignments.
(a ∧ b ∧ c ∧ f ∧ ¬g ∧ ¬h)
(a ∧ b ∧ ¬c ∧ ¬f ∧ g ∧ ¬h)
Having satisfying assignments means that the variable h equals 0 when making assignment 1 to PIs a and b. Since Φ is assigned 1 when inputs a and b have 1, so Φ is satisfiable. We might think that g is not an implementation of f because Φ is satisfiable. However, it doesn't matter which binary value g has, because f equals X. That is, a solution may come from X space and the solution cause a false negative which wrongly demonstrates that two circuits are not logically-consistent. This condition which two designs are clearly logically-consistent, nevertheless the formula converted from two circuits is proved to be satisfiable, is called *fake SAT*. Fake SAT is considered as a false negative.

3.3 Blocking Fake SAT

We cannot consider the formula satisfiable when caused by X. To prevent the false negative, learned clauses must be added to the

original CNF when we encounter the fake SAT condition. *Learned clause* is a conflict clause [10] which is a negation of a satisfying assignment. This clause consists of literals which correspond to PIs. In the case of Figure 2, the clause ($\neg a \vee \neg b$) is added to Φ. Consequently, when the SAT solver assigns 1 to the variable a and b , the learned clause conflicts. Therefore, SAT solver backtracks to the previous decision level and continues searching for the next choices.

If a fake SAT is deduced, then we continue searching after adding learned clause. If all clauses are satisfied which is not fake, we can conclude that two designs are not logically-consistent. In other words, when all clauses in the formula are satisfied, extract the values of variables which correspond to PIs from the satisfying assignment, and assign selected variable values to the PIs of the RTL design. If the PO value has a binary value, then we can conclude that the two circuits are not logically-consistent. Or if the PO value equals X, then a learned clause must be added to the formula because this condition is fake. Clearly, two designs are logically-consistent if the formula is proved to unsatisfiability.

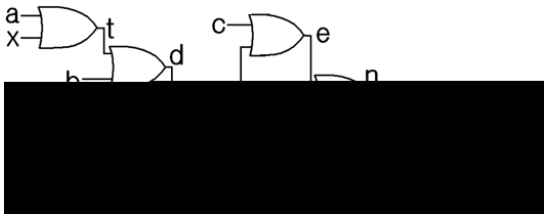

Figure 3. Circuit with Don't Care

Consider the circuits shown in Figure 3. The problem is of checking whether m is an implementation of f or not. We need to check that h cannot be 0. Table 3 shows that the values of nodes by assigning binary values to PIs. (h equals 1 in other cases which are omitted from the Table 3.)

a	b	c	m	d	g	e	n	p	f	m	h
0	0	0	1	x	x	x	x	x	x	1	x
0	0	1	1	x	x	1	1	x	x	1	x

Table 3. Truth Table for Figure 3

Since the output of the miter has 1 or X in all combinations of PI values, g and f are logically-consistent. Consider the formula Φ which corresponds to the circuit in Figure 3.

$$\Phi \equiv (t \geq a) \wedge (d \leftrightarrow t \vee b) \wedge (g \leq d) \wedge (p \leftrightarrow g \wedge m) \wedge (n \leftrightarrow e \vee g) \wedge (f \leftrightarrow n \wedge p) \wedge (h \leftrightarrow (f \leftrightarrow m)) \wedge (\neg h)$$

Φ becomes 1 when a, b and c equal 0 and m equals to 1, therefore Φ is satisfiable. However, this condition is a fake SAT because f has the value X in this case. Hence, we add the learned clause ($a \vee b \vee c \vee \neg m$) to the CNF converted from Φ.

To check if a solution comes from X space or not, we use a ternary simulation by making assignment to PIs. If PO value equals X in RTL design, then we can conclude that the solution comes from X space. Most SAT solvers provide satisfying assignments, thus PI values can be extracted from solutions (satisfying assignments). During the simulation, RTL design circuit is simulated to check whether the PO values of the design have X. If Xs are assigned to PO variables, then we must add a learned clause which are extracted from a satisfyign assginment and search another branches. If POs have binary values, then searching is terminated because this means

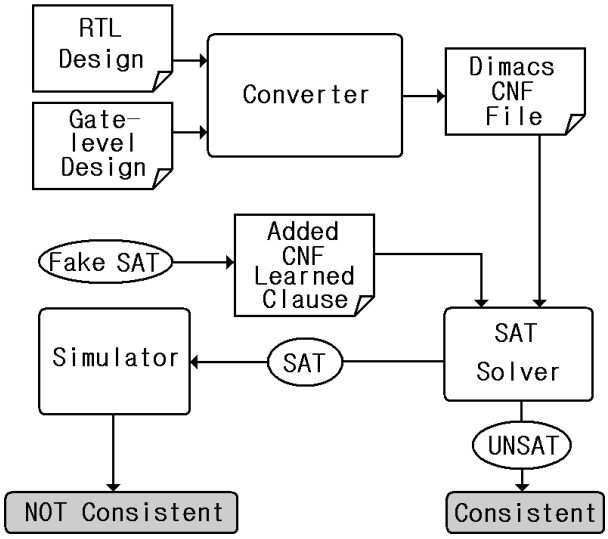

Figure 4. General Architecture of the System

that we find a witness which demonstrates that two designs are not logically-consistent. The unsatisfiability of the formula means that two designs are logically-consistent. General architecture of our system is shown in Figure 4.

4. Implementation and Experimental Results

We present experimental results based on the preliminary implementation of our method for SAT-based CEC with X. It was written in gnu g++ under Unix. The results are reported on a workstation with UntraSPARC III 750 MHz and 4 GB of RAM. We used Zchaff [11] for SAT solver and modified this engine to eliminate false negatives. Our system consists of three modules, which are translator, SAT solver (Zchaff) and simulator. The translator converts a verilog description into Dimacs CNF file.

After transformation of circuits into CNF, a SAT solver proves the satisfiability of CNF. A 32-bit parallel vector is used in SAT solving to improve the performance. Learned clauses are added to CNF each time when a solution is found. Just as the moment 16 solutions are gathered, Modified Zchaff calls the simulator. Similarly to the SAT solver, a bit-parallel operation is implemented in the simulator. 16 cases are simulated at the same time. If all the PO values have binary values, then the simulator is terminated. As a consequence, we proves the non-logical-consistency. On the other hand, if PO values have X, we can conclude this satisfiability is a fake. Therefore, the operation is transferred to the SAT solver, and keeps searching.

In the following we present experimental results on some representative circuits to check the logical-consistency regarding our proposed techniques proposed. We considered instances from IWLS benchmark [17]. Each example has 3 versions - original, swept, and optimized. In our experiments, the original versions are considered as RTL design circuits, and the optimized as gate-level design. Actually, two versions are logically-consistent. Several PIs of the original version were selected, and X values are assigned to the selected PIs. We made assignments different number of Xs to PIs for each instance, and we compared the count of fake SAT.

The results of the experiments are shown in Table 4. Names of instances are given in the column "Name". The column "PI" gives the number of PIs, and "Gates" describes the number of gates of the RTL design (original version) and the gate-level design (optimized version). The column "DC" gives the number of PIs which

Name	PI	Gates (RTL/Gate level)	DC	Proposed Method		Dual-rail
				Solusion	CPU TIME	Time
9symml	9	235/217	1	130	0.11	0
			2	110	0.07	0
			5	16	0.01	0
			7	4	0.002	0
C17	5	8/9	1	5	0	0
			2	3	0	0
			3	4	0	0
			4	2	0	0
C880	60	435/414	6	0	0	0
			15	0	0	0
			30	0	0	0
			45	512	0.754	0
apex6	135	765/771	14	0	0	0
			34	0	0	0
			68	0	0	0.004
			101	0	0	0.002
apex7	49	330/238	5	0	0	0
			12	0	0	0
			25	0	0	0
			37	0	0	0.002
count	35	143/144	4		>3600	0
			9		>3600	0
			18		>3600	0.002
			26	512	0.27	0
i6	138	834/448	14		>3600	0.004
			35		>3600	0.004
			69		>3600	0.004
			104		>3600	0.002
x4	94	981/388	9	0	0	0.002
			24	0	0	0.002
			47	0	0	0.002
			71	0	0	0.006
z4ml	7	252/42	1	8	0.002	0
			2	12	0.004	0
			4	8	0.002	0
			5	4	0.002	0

Figure 5. Experimental Results

are assigned Xs. The column ”Solution” describes the number of false negatives. The column ”Time” describes the average execution time. Since we experimented within the timeout limit of 3600 seconds, ”i6” and ”count” were suspended.

The maximum of solutions equals to 2^n (n : the number of PIs), but X can be blocked in the middle of circuits easily. Since Xs can be blocked in the middle of circuits, the actual number of fake SAT was pretty small in many cases.

While the number of generated variables using dual-rail encoding increases by square of the number of variables in the circuit, the new method generates the same number of variables as the original circuit. Furthermore, if circuits are transformed into CNF using this method, less clauses are generated, hence, this method requires the smaller search space. Actually, the performance of the proposed method depends on the frequency of the fake SAT. The false negatives are affected by the function of circuits and by the variables which are assigned X. In worst cases, fake SAT can be deduced in all truth assignments. Of couse, it is possible that all X values can be blocked in the middle of circuits. To improve the performance of this method, it is essential to analyze the pattern of fake SATs.

5. Conclusion and Future Work

Combinational equivalence checkers are widely deployed in industry. We presented a simple framework for SAT-based verification. Especially, the proposed method is to verify a RTL design and an gate-level design. Traditional approaches have been implemented on the assumption that circuits have only binary values. Therefore, this technique can be a new approach.

There are several approaches to improve the current method and the implementation. As previously mentioned, if the pattern of false negatives can be analyzed, the performance of the proposed method will be improved. This will be the key to the solution. There exist other approaches to promote efficiency. Current program use a ternary simulation to check fake SATs. However, if the X propagation condition can be stored and represented efficiently, we can improve the performance of our technique. This includes BDDs, CNF or Graphs. A more thorough investigation on the learned clauses would be an interesting future subject.

References

[1] Janusz A. Brzozowski and Carl-Johan H. Seger : Asynchronous Circuits. ISBN 0-387-94420-6.

[2] D. Brand : Verification of Large Synthesized Designs. In International conference on Computer -Aided Design, pages 534-537, 1993.

[3] E. Goldberg, M. Prasad and R. Brayton : Using SAT for Combinational Equivalence Checking. International Workshop on Logic Synthesis, pages 185-191, 2000.

[4] R. E. Bryant : Graph Based Algorithms for Boolean Function Manipulation. In IEEE Transactions on Computers, pages 677-691, 1986.

[5] Hee Hwan Kwak, In-Ho Moon, James H. Kukula, and Thomas R. Shiple : Combinational Equivalence Checking through Function Transformation. In International Conference on Computer-Aided Design, 2002.

[6] R. K. Brayton, G. D. Hachtel, C. McMullen, and, A. L. Sangiovanni Vincentelli : Logic Minimization Algorithms for VLSI Synthesis. ISBN 0898381649, 1984.

[7] Mike Turpin : The Dangers of Living with an X. SNUG Boston, 2003.

[8] M. Davis, G. Logemann, and D. Loveland : A Machine Program for Theorem Proving. Communications of the ACM, pages 394-397, 1962.

[9] Mary Sheeran, and Gunnar Stalmarck : A Tutorial on Stalmarcks’s Proof Procedures for Propositoinal Logic. In International Conference on Formal Methods in Computer-Aided Design, Springer LNCS, pages 82-100, 1998.

[10] J. Marques-Silva and L. G. e Silva : Algorithms for Satisfiability in Combinational Circuits Based on Backtrack Search and Recursive Learning. In Workshop Notes of The Internatioinal Workshop on Logic Synthesis, pages 227-241, 1999.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik : Chaff: Engineering an Efficient SAT Solver. In Design Automation Conference, pages 530-535, 2001.

[12] E. Goldberg, Ya. Novikov : BerkMin : a Fast and Robust SAT-Solver In Design, Automation and Test in Europe, pages 142-147, 2002.

[13] J. Marques-Silva and K. A. Sakallah : GRASP-A New Search Algorithm for Satisfiability. In IEEE/ACM International Conference on Computer-Aided Design, pages 220-227, 1996.

[14] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita and Y. Zhu : Symbolic Model Checking using SAT procedures instead of BDDs. In Design Automation Conference, pages 317-320, 1999.

[15] Formality User Guide. Version U-2003.06. Synopsys, 2002.

[16] Myoung Jin Nam : Combinational Equivalence Checking with Don’t Cares. Master’s thesis, Korea University, Dept. of Computer Science and Engineering, Seoul, Korea, 2003.

[17] http://www-cad.eecs.berkeley.edu/~kuehl/courses/219b/Homeworks/iwls.tgz.