Compatible Equivalence Checking of X-Valued Circuits

Yu-Neng Wang*§, Yun-Rong Luo*§, Po-Chun Chien†§, Ping-Lun Wang*, Hao-Ren Wang†, Wan-Hsuan Lin*, Jie-Hong Roland Jiang*† and Chung-Yang Ric Huang*†
* Department of Electrical Engineering, National Taiwan University
† Graduate Institute of Electronics Engineering, National Taiwan University

***Abstract*—The X-value arises in various contexts of system design. It often represents an unknown value or a don't-care value depending on the application. Verification of X-valued circuits is a crucial task but relatively unaddressed. The challenge of equivalence checking for X-valued circuits, named compatible equivalence checking, is posed in the 2020 ICCAD CAD Contest. In this paper, we present our winning method based on X-value preserving dual-rail encoding and incremental identification of compatible equivalence relation. Experimental results demonstrate the effectiveness of the proposed techniques and the outperformance of our approach in solving more cases than the commercial tool and the other teams among the top 3 of the contest.**

I. Introduction

Equivalence checking (EC) is an essential procedure in system design to ensure functional correctness. Conventional combinational equivalence checking (CEC) [1]–[4] performs reasoning over binary valued logic. However, there are verification tasks that require reasoning over multi-valued logic.

In register transfer level (RTL) design, the notion of X-value emerges naturally in various contexts. The X-value may represent an unknown value, e.g., due to uninitialized signals, due to power shut off in low-power design, due to a wire driven by multiple sources, and so on. It may also represent a don't care value, e.g., due to out of range in part-select addressing, due to an unconstrained condition in a case statement, due to don't care of flip-flop input under the reset condition, and so on. When X-values correspond to don't cares, synthesis tools may exploit them for circuit optimization. Equivalence verification in the presence of X-values is an important subject and is posed as a challenge, *compatible equivalence checking*, in the 2020 ICCAD CAD Contest.

CEC has long been an important research topic in electronic design automation (EDA). The approaches to CEC have evolved from binary decision diagram based methods, e.g., [1], [2], to more scalable and-inverter graph (AIG) and Boolean satisfiability (SAT) based ones, e.g., [3], [4]. In [4], several improvements to AIG-based checker are proposed using fast logic synthesis techniques, such as rewriting [5], to reduce the AIG size. These methods do not handle X-values.

For equivalence checking of X-valued circuits, the X-values serve as don't cares when the equivalence between two circuits is to be checked. The problem was studied in [6], where a methodology similar to standard CEC was proposed. As their formulation fails to model the compatible equivalence relation between X-value and Boolean values in SAT solving, false counterexamples have to be ruled out separately by simulation. This process leads to limited scalability. Besides EC, there are other verification tasks related to X-values. For example, in [7] symbolic trajectory evaluation (STE), an industrial-strength formal method based on symbolic simulation, is applied with a ternary system model to verify digital circuits. In [8], finding bugs caused by uninitialized registers in an RTL design is considered. Despite these related efforts, the X-valued CEC problem remains relatively unaddressed.

Binary encoding is an essential step to represent multi-valued logic with Boolean circuits. E.g., in [9], different encoding schemes for four-valued logic and their efficiency in stuck-at fault test pattern generation are discussed. In [10], symmetry encoding for symbolic multi-valued functions is proposed aiming at improving synthesis quality. Nevertheless, the effective encoding of X-valued (ternary-valued) logic for equivalence checking is not well-studied in the literature.

In this paper, we present our winning method tackling the compatible equivalence checking problem of the 2020 ICCAD CAD Contest. The techniques of X-value preserving dual-rail encoding and incremental identification of compatible equivalence relation are proposed, and integrated into the state-of-the-art CEC flow. Experimental evaluation is conducted to demonstrate the efficiency of our method compared to other winning teams in the contest, and the effectiveness of the proposed techniques.

The rest of this paper is organized as follows. With Section II providing the essential preliminaries, the problem of compatible equivalence checking is then formulated in Section III. Our proposed solutions are presented in Sections IV and V. Section VI evaluates the experimental results, and finally Section VII concludes this paper.

II. Preliminaries

In this paper, sets are denoted by capital letters, *e.g.*, S, and the elements in a set are denoted by minuscule letters, *e.g.*, $s \in S$. Boolean negation, conjunction, and disjunction are denoted by $\neg$, $\wedge$ and $\vee$, respectively. By convention, Boolean conjunction operators are sometimes omitted. A *literal* l is either a Boolean variable v or its negation $\neg v$. A *clause* is a disjunction of literals. A conjunction of a set of clauses is referred to as a *conjunctive normal form* (CNF) formula. An assignment to a Boolean variable x is to give x a Boolean value in $\mathbb{B} = \{0, 1\}$. A CNF formula F over variables $V = \{v_1, \ldots, v_n\}$ is called *satisfiable* if there exists an assignment to V making F valuate to true. Otherwise, F is *unsatisfiable*.

§ Equal contribution

Table I: Function tables of ternary valued primitive gates.

(a) $\hat{o} = \textsc{And}(\hat{a}, \hat{b})$

$\hat{a}$ \ $\hat{b}$	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

(b) $\hat{o} = \textsc{Not}(\hat{a})$

$\hat{a}$	$\hat{o}$
0	1
1	0
x	x

(c) $\hat{o} = \textsc{Dc}(\hat{c}, \hat{d})$

$\hat{d}$ \ $\hat{c}$	0	1	x
0	0	1	x
1	x	x	x
x	x	x	x

(d) $\hat{o} = \textsc{Mux}(\hat{s}, \hat{a}, \hat{b})$

$\hat{s}=0$				$\hat{s}=1$				$\hat{s}=\mathsf{x}$			
$\hat{a}$ \ $\hat{b}$	0	1	x	$\hat{a}$ \ $\hat{b}$	0	1	x	$\hat{a}$ \ $\hat{b}$	0	1	x
0	0	0	0	0	0	1	x	0	0	x	x
1	1	1	1	1	0	1	x	1	x	1	x
x	x	x	x	x	0	1	x	x	x	x	x

A. Ternary-Valued Logic

A signal/variable in an X-valued circuit is ternary and can take on a value in $\mathbb{T} = \{0, 1, \mathsf{x}\} = \mathbb{B} \cup \{\mathsf{x}\}$, where x represents an unknown or don't care value. In the sequel, the term *X-value* refers to value x. To distinguish a ternary variable from a Boolean variable, a hatted variable $\hat{v}$ denotes its ternary characteristics in contrast to its Boolean counterpart v. In the sequel, we reuse the symbols of Boolean connectives for ternary valued logic.

According to [11], ternary-valued primitive gates can be defined and are used in X-valued circuits. Among others, the operations of the AND and NOT gates are specified in Table I (a) and (b), respectively. Two additional gates DC and MUX, essential in modeling industrial designs, are specified in the 2020 ICCAD CAD Contest with their operations shown in Table I (c) and (d), respectively.

The compatibility between two values in $\mathbb{T}$ is defined as follows.

Definition 1. *Given two values $\hat{a}$, $\hat{b} \in \mathbb{T}$, $\hat{a}$ is* compatible equivalent *to $\hat{b}$ if $(\hat{a}, \hat{b}) \in \{(0,0), (1,1), (\mathsf{x},0), (\mathsf{x},1), (\mathsf{x},\mathsf{x})\}$. Otherwise, $\hat{a}$ is* not compatible equivalent *to $\hat{b}$, i.e., $(\hat{a}, \hat{b}) \in \{(0,1), (1,0), (0,\mathsf{x}), (1,\mathsf{x})\}$.*

B. Combinational Equivalence Checking

In a modern CEC flow, two circuits G and R under checking are combined into a single *miter circuit* [12] by XORing their corresponding outputs O_G and O_R, and ORing these XOR gates as the primary output. Thereby, G and R are equivalent if and only if their miter circuit has output equals 0 for all input assignments.

The state-of-the-art CEC engine use the and-inverter graph (AIG) [3] to represent the logic function, and perform structural hashing to detect structural similarities in the AIG. Moreover, functionally equivalent AIG nodes are detected and merged to reduce the AIG size. Simulation is employed to partition the set of AIG nodes into potentially functional equivalent classes, and the pairwise equivalence within a class is verified by SAT solving. This step is known as *SAT sweeping*. The equivalence of the two circuits is finally verified by SAT solving at the output of miter. Prior work [4] has proposed heuristics to interleave SAT sweeping with low-effort logic syntheses. Figure 1 shows the CEC flow.

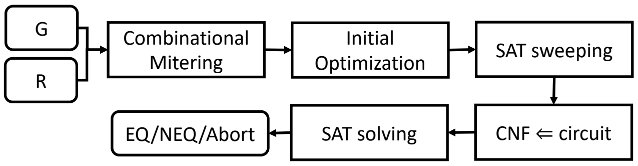


Figure 1: CEC flow.

III. Problem Formulation

Following [13], the compatible equivalence checking problem can be formulated as follows.

Problem Statement 1 (Compatible Equivalence Checking)**.** *Given two X-valued combinational circuits, the golden circuit $\hat{G}$ and the revised circuit $\hat{R}$, consisting of primitive gates,* DC, MUX, *and/or constant values in $\mathbb{T}$ and having identical primary inputs $I = \{p_i \mid i = 1 \dots n\}$, let the primary outputs of $\hat{G}$ and $\hat{R}$ be $\hat{O}_G = \{\hat{o}_{g,i} \mid i = 1 \dots m\}$ and $\hat{O}_R = \{\hat{o}_{r,i} \mid i = 1 \dots m\}$, respectively. We are asked to determine whether $\hat{G}$ is compatible equivalent to $\hat{R}$, that is, $\hat{o}_{g,i}$ is compatible equivalent to $\hat{o}_{r,i}$, for $i = 1, \dots, m$ and for any assignment in $\mathbb{B}^n$ to I. If so, report* EQ*. Else, report* NEQ *with an assignment to I making some $\hat{o}_{g,i}$ not compatible equivalent to $\hat{o}_{r,i}$.*

We note that the relation of compatible equivalence is not symmetric, that is, $\hat{G}$ being compatible equivalent to $\hat{R}$ is not the same as $\hat{R}$ compatible equivalent to $\hat{G}$. Note also that the above formulation assumes the input variables I are Boolean. Hence the X-values of an X-valued circuit can only originate from the DC gates.

IV. SAT Encoding

To achieve CEC-based compatible equivalence checking, CEC has to take x-value into account and requires two modifications: dual-rail encoding and constructing miter for compatible equivalence.

A. Dual-Rail Encoding

To encode the compatible equivalence checking problem for SAT solving, we employ dual-rail encoding to transform ternary valued logic over $\mathbb{T}$ into Boolean logic over $\mathbb{B}$. Essentially a 2-bit Boolean value (o^0, o^1) to encode symbols $\{0, 1, \mathsf{x}\}$. Among other possibilities[1], we mainly explore two encoding schemes: the x*-preserving encoding*, denoted E_{xp}, and the *symmetric encoding*, denoted $\mathsf{E}_{\mathsf{sym}}$, defined in Table II (a) and (b), respectively. Given a ternary variable $\hat{o} \in \mathbb{T}$, we denote its E_{xp}-encoded and $\mathsf{E}_{\mathsf{sym}}$-encoded signal (o^0, o^1) by $\mathsf{E}_{\mathsf{xp}}(\hat{o})$ and $\mathsf{E}_{\mathsf{sym}}(\hat{o})$, respectively. In E_{xp}, o^0 corresponds to the original Boolean bit and o^1 is the x-bit, for $o^1 = 1$ if

[1] We also studied one-hot encoding using 3 bits. As the encoding is not as good as the considered dual-rail encoding, it is excluded from our discussion.

Table II: Dual-rail encoding schemes.

(a) x-preserving encoding E_{xp}

$\mathbb{T}$	0	1	x
o^0, o^1	00	10	01, 11

(b) symmetry encoding $\mathsf{E}_{\mathsf{sym}}$

$\mathbb{T}$	0	1	x
o^0, o^1	10	01	00

and only if $\hat{o} = \mathsf{x}$. In $\mathsf{E}_{\mathsf{sym}}$, the values of (o^0, o^1) for $0, 1 \in \mathbb{T}$ are bitwise complement. An in-depth comparison of the two encoding schemes is to be made in Section VI-A.

To model the behavior of logic gates specified in Section II-A, we derive the logic formulas for each gate under E_{xp} and $\mathsf{E}_{\mathsf{sym}}$ as listed in Table III. We note that $\mathsf{E}_{\mathsf{sym}}$ is identical to the dual-rail encoding in [6] except that x is encoded as (11) in [6]. However, the derived logic formulas for primitive gates in $\mathsf{E}_{\mathsf{sym}}$ are identical to those in [6].

B. Miter Construction and SAT Solving

To construct a miter circuit for compatible equivalence checking under $\mathsf{E}_{\mathsf{sym}}$ and E_{xp}, we rely on the following proposition.

Proposition 1. *Given two X-valued combinational circuits $\hat{G}$ and $\hat{R}$ with identical primary inputs $I = \{p_i \mid i = 1 \ldots n\}$ and corresponding primary outputs $\hat{O}_G = \{\hat{o}_{g,i} \mid i = 1 \ldots m\}$ and $\hat{O}_R = \{\hat{o}_{r,i} \mid i = 1 \ldots m\}$, respectively, $\hat{G}$ and $\hat{R}$ are compatible equivalent if and only if the Boolean function of miter*

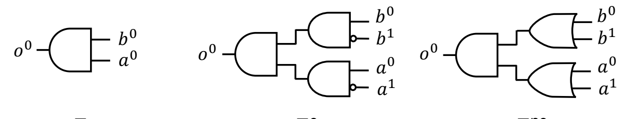
$$M = \begin{cases} \bigvee_{i=1}^{n}(o^0_{g,i}\neg o^0_{r,i} \vee o^1_{g,i}\neg o^1_{r,i}), & \textit{for } \mathsf{E}_{\mathsf{sym}} \\ \bigvee_{i=1}^{n}(o^0_{g,i}\neg o^0_{r,i} \vee \neg o^0_{g,i} o^0_{r,i} \vee o^1_{r,i})\neg o^1_{g,i}, & \textit{for } \mathsf{E}_{\mathsf{xp}} \end{cases}$$

outputs 0 under all assignments in $\mathbb{B}^n$ to I, where $(o^0_{g,i}, o^1_{g,i})$ and $(o^0_{r,i}, o^1_{r,i})$ are the $\mathsf{E}_{\mathsf{sym}}$/$\mathsf{E}_{\mathsf{xp}}$-encoded variables of $\hat{o}_{g,i}$ and $\hat{o}_{r,i}$, respectively.

Thereby we can determine whether two circuits are compatible equivalent by checking the Boolean satisfiability of miter M, given that the corresponding CNF formula can be obtained from the miter circuit by Tseitin transformation [14]. Below we exploit special properties of $\mathsf{E}_{\mathsf{sym}}$ and E_{xp} to improve SAT solving efficiency.

From Table III, we observe that the formulas for operators under $\mathsf{E}_{\mathsf{sym}}$ are more succinct than those under E_{xp} except for XOR. Therefore, the CNF formula transformed from an $\mathsf{E}_{\mathsf{sym}}$-encoded circuit can be shorter than that from E_{xp}. Moreover, because all gate functions in $\mathsf{E}_{\mathsf{sym}}$ are unate, Plaisted-Greenbaum encoding [15] can be applied to replace Tseitin transformation to further reduce the CNF formula size.

For a signal (o^0, o^1) of a circuit encoded under E_{xp}, observe that when $o^1 = 1$, the value of o^0 is a don't care for the primary outputs of the circuit because the value of $\hat{o} = \mathsf{x}$ is determined by $o^1 = 1$ regardless of the value of o^0. Let o^0 be the output of some gate (node) g with function $f(x_1, \ldots, x_k)$ over the fanin variables $x_1, \ldots, x_k$ of g under E_{xp}. Let $C = \{C_i \mid i = 1, \ldots, j\}$ be the clauses converted from the node with output o^0 in the circuit. When $o^1 = 1$, the clauses C impose no constraint on the function of primary outputs. Therefore, we can replace every clause $C_i \in C$ by



(a) E_{xp} (b) $\mathsf{E}^{\mathsf{c}}_{\mathsf{xp}}$ (c) $\mathsf{E}^{\mathsf{nc}}_{\mathsf{xp}}$

Figure 2: Example of x-bit fanin insertion of AND gate.

clause $(C_i \vee o^1)$, i.e., inserting the x-bit literal o^1 into C_i. Such an x-bit literal insertion technique, referred to as $\mathsf{E}^{\mathsf{xi}}_{\mathsf{xp}}$, may conditionally disable clauses and potentially speed up SAT solving.

C. X-Valued CEC Flow

The circuit-based CEC algorithm [4] has its advantage over pure SAT solving due to the additional utilization of circuit properties. Similarly, we exploit circuit similarities in our CEC-based compatible equivalence checking, referred to as xcec, flow.

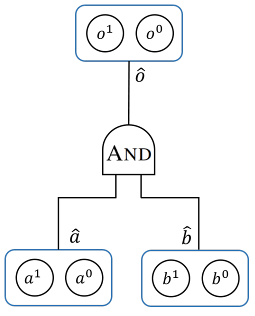
In theory, we can also do x-bit literal insertion in SAT solving under xcec flow. However, in practical implementation, optimizing a circuit M before SAT solving may destroy the pairwise relationship between the x-bit node and original-bit node that together encode some node of the ternary-valued circuit $\hat{M}$, making x-bit literal insertion infeasible. Therefore, we propose another approach *x-bit fanin insertion* in xcec flow that also utilizes the information of x-bit in the original-bit circuit, i.e., the subcircuit consists of purely original-bit signals in an E_{xp}-encoded circuit, before circuit optimization. Specifically, let $\hat{a}$ be a fanin to $\hat{o}$ and let $\mathsf{E}_{\mathsf{xp}}(\hat{o}) = (o^0, o^1)$, $\mathsf{E}_{\mathsf{xp}}(\hat{a}) = (a^0, a^1)$. Recall that when the x-bit $a^1 = 1$, the original-bit a^0 becomes a don't care to the primary output functions of the underlying circuit. There are different strategies to utilize the flexibility. In particular, we can replace the fanin a^0 of o^0 with constant 0, with constant 1, with the controlling (denoted $\mathsf{E}^{\mathsf{c}}_{\mathsf{xp}}$), or with the non-controlling value (denoted $\mathsf{E}^{\mathsf{nc}}_{\mathsf{xp}}$) of the gate of o^0 when $a^1 = 1$. Figure 2 shows an example of o^0=AND(a^0, b^0) with $(o^0, o^1) = \mathsf{E}_{\mathsf{xp}}(\hat{o})$, $(a^0, a^1) = \mathsf{E}_{\mathsf{xp}}(\hat{a})$, and $(b^0, b^1) = \mathsf{E}_{\mathsf{xp}}(\hat{b})$, where (a) shows the original implementation of AND, (b) corresponds to the $\mathsf{E}^{\mathsf{c}}_{\mathsf{xp}}$ strategy, and (c) corresponds to the $\mathsf{E}^{\mathsf{nc}}_{\mathsf{xp}}$ strategy applied on both fannins a^0 and b^0. Note that for $\mathsf{E}^{\mathsf{c}}_{\mathsf{xp}}$, when a^1 is assigned to value 1, o^0 is implied to be 0. Essentially, $\mathsf{E}^{\mathsf{c}}_{\mathsf{xp}}$ allows more implication propagation on gate variables topologically from primary inputs to primary outputs. For $\mathsf{E}^{\mathsf{nc}}_{\mathsf{xp}}$, when a^1 is assigned to value 1, o^0 equals $b^1 \vee b^0$ and is independent of fanin a^0. Essentially, $\mathsf{E}^{\mathsf{nc}}_{\mathsf{xp}}$ allows conditional blocking of some fanins.

V. CE Pair Identification

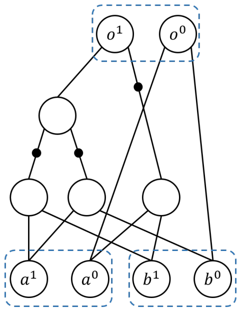
The above xcec flow does not make full use of the underlying information of the X-valued circuits. In fact, we can identify *compatible equivalent (CE) pairs of signals* within a circuit, in a way similar to identify equivalent signals in the conventional CEC flow. However, unlike the equivalence relation in CEC, the CE relation is not symmetric, and thus

Table III: Dual-rail encoding of primitive gate operations.

		$\hat{o}$ =NOT($\hat{a}$)	$\hat{o}$ =AND($\hat{a},\hat{b}$)	$\hat{o}$ =OR($\hat{a},\hat{b}$)	$\hat{o}$ =XOR($\hat{a},\hat{b}$)	$\hat{o}$ =DC($\hat{c},\hat{d}$)	$\hat{o}$ =MUX($\hat{a},\hat{b},\hat{s}$)
$\mathsf{E}_{\mathsf{sym}}$	o^0	a^1	$a^0 \vee b^0$	a^0b^0	$a^0b^0 \vee a^1b^1$	c^0d^0	$a^0b^0 \vee a^0S^0 \vee b^0S^1$
	o^1	a^0	a^1b^1	$a^1 \vee b^1$	$a^0b^1 \vee a^1b^0$	c^1d^0	$a^1b^1 \vee a^1S^0 \vee b^1S^1$
E_{xp}	o^0	$\neg a^0$	a^0b^0	$a^0 \vee b^0$	$a^0\neg b^0 \vee \neg a^0b^0$	$c^0\neg d^0$	$a^0\neg S^0 \vee b^0S^0$
	o^1	a^1	$a^1b^1 \vee a^1b^0 \vee a^0b^1$	$a^1b^1 \vee a^1\neg b^0 \vee \neg a^0b^1$	$a^1 \vee b^1$	$c^1 \vee d^0 \vee d^1$	$(a^1\neg S^0 \vee b^1S^0 \vee S^1)(\neg a^0b^0 \vee a^0\neg b^0 \vee a^1 \vee b^1)$



(a) High-level x-circuit.



(b) Low-level AIG.

Figure 3: Circuit representation of an AND gate.

does not form an equivalence relation. Consequently, a CE pair $(\hat{a},\hat{b})$ cannot be merged, as performed in conventional CEC, unless the pair $(\hat{b},\hat{a})$ is also compatible equivalent. Although not all CE pairs can be merged, their presence can be expressed and imposed as additional learned clauses to strengthen SAT solving of the miter output. Instead of working on the AIG representation, which loses the high-level information of the original X-valued circuits, we modify the circuit representation to exploit the internal CE relations.

A. Circuit Representation

We maintain two levels of circuit representation: the high-level X-valued circuit and the low-level AIG. The high-level representation corresponds to the original ternary-valued circuit, which consists of the primitive gates AND, OR, NAND, NOR, XOR, XNOR, NOT, BUF, special gates DC, MUX, and constants 0, 1, x.

In fact, all the above gates and constants can be re-expressed in terms of only gates AND, NOT and constants $1, \mathrm{x}$. For example, the special gates MUX$(\hat{s},\hat{\imath},\hat{\jmath})$ and DC$(\hat{c},\hat{d})$ can be re-expressed as follows.

$$\mathrm{MUX}(\hat{s},\hat{a},\hat{b}) = \neg(\neg(\neg\hat{s}\wedge\hat{a})\wedge\neg(\hat{s}\wedge\hat{b})\wedge\neg(\hat{a}\wedge\hat{b})) \quad (1)$$

$$\mathrm{DC}(\hat{c},\hat{d}) = \mathrm{MUX}(\hat{d},\hat{c},\mathrm{x}) = \neg(\neg(\neg\hat{d}\wedge\hat{c})\wedge\neg(\hat{d}\wedge\mathrm{x})\wedge\neg(\hat{c}\wedge\mathrm{x})) \quad (2)$$

Therefore, we can re-express the high-level X-valued circuit in terms of AND, NOT, 1, and x. After the re-expression, the X-valued circuit can be further encoded with E_{xp} (or $\mathsf{E}_{\mathsf{sym}}$) into an AIG, which we refer to as the low-level circuit representation. Figure 3 illustrates the high and low-level circuit representations of an AND gate.

Note that each signal in the high-level X-valued circuit can be associated with exactly two signals (nodes) in its low-level AIG. However, a low-level AIG node may not be associated with any high-level signals. Although there are various techniques to reduce AIG size, *e.g.*, rewriting [5], the only optimization we can perform on the low-level AIG is equivalent node merging. Each AIG node is replaced by the representative node in its equivalence class after equivalent node merging, such replacement can be easily updated on the high-level circuit. However, after rewriting the circuit, we may fail to find two associating AIG nodes for each X-valued signal.

B. Proving and Learning Internal CE Information

After the two representation levels of the miter circuit has been constructed, we then proceed to identify the CE pairs in the high-level X-valued circuit, with the low-level AIG being used for CNF formula translation for SAT solving. In contrast to the standard CEC procedure, we cannot simply merge signals $\hat{a}$ and $\hat{b}$ in the circuit even if $\hat{a}$ is compatible equivalent to $\hat{b}$. Therefore, we propose an alternative way to store the CE relations of internal signals. From the definition of compatible equivalence, we can derive the representing CNF formula of an X-valued signal $\hat{a}$ being compatible equivalent to another signal $\hat{b}$ as

$$(a^1 \vee \neg b^1) \wedge (a^1 \vee a^0 \vee \neg b^0) \wedge (a^1 \vee \neg a^0 \vee b^0), \quad (3)$$

where $\mathsf{E}_{\mathsf{xp}}(\hat{a}) = (a^0, a^1)$ and $\mathsf{E}_{\mathsf{xp}}(\hat{b}) = (b^0, b^1)$. This formula valuates to true if $\hat{a}$ is compatible equivalent to $\hat{b}$. By adding these three CE-clauses to the CNF formula converted from the miter circuit as the additional constraints, the satisfiability of the miter remains unchanged. That is, these three CE-clauses can be viewed as the learned information that may potentially guide the SAT oracle for more efficient solving.

To assist CE pair identification, the following proposition allows us to use transitivity for more efficient reasoning.

Proposition 2. *For a pair of X-valued signals $\hat{o}_1$ and $\hat{o}_2$ with $\hat{o}_1 = \mathrm{AND}(\hat{a}_1,\hat{b}_1)$ and $\hat{o}_2 = \mathrm{AND}(\hat{a}_2,\hat{b}_2)$, if $\hat{a}_1$ is compatible equivalent to $\hat{a}_2$ and $\hat{b}_1$ is compatible equivalent to $\hat{b}_2$, then $\hat{o}_1$ is compatible equivalent to $\hat{o}_2$.*

The proof of Proposition 2 can be done by enumerating all combinations of $\hat{a}_0$, $\hat{b}_0$, $\hat{a}_1$ and $\hat{b}_1$, and is omitted due to space limitations. Proposition 2 can be helpful when we are identifying the CE pairs in the high-level X-valued circuit. To determine the compatible equivalence of the outputs of two AND gates, we can first check whether their corresponding fanins are in CE relation, if so, we can conclude their equivalence and the costly SAT-proving effort can be saved.

The procedure of proving and learning internal CE pair information is briefly sketched as follows. When the CE relation of $(\hat{a},\hat{b})$ is to be checked, we first check whether the fanins of $\hat{a}$ and $\hat{b}$ fulfill the condition of Proposition 2.

If so, we can conclude that their compatible equivalence and return. Otherwise, for $\hat{a}$ being compatible equivalent to $\hat{b}$, the condition imposed by the three clauses in Formula 3 have to be satisfied under all circumstances, that is, the negated condition of each of the three clauses should not be satisfied. By incremental SAT solving, we can determine the compatible equivalence of $(\hat{a}, \hat{b})$ by checking the satisfiability of the CNF formula derived from the miter of the two sub-circuits with $\hat{a}$ and $\hat{b}$ as their respective primary outputs imposed with the negation of each of the three clauses. If a satisfying assignment can be found by the SAT oracle, then the Formula (3) is evaluated to false under such assignment, meaning that the two signals are not compatible equivalent. Otherwise, a CE pair is found and the three clauses in Formula (3) are treated as the additional learned CE-clauses and added to the CNF formula translated from the miter.

C. Overall CE Checking Flow

With internal CE pairs identified, the corresponding learned clauses can be incorporated into the miter CNF formula to strengthen final SAT solving. The overall CE checking procedure is sketched in Algorithm 1. In line 1, the set of clauses C is initialized as empty. Random pattern simulation is performed in line 2. The patterns serve as signatures of signals showing their (in)equivalences, the same as in SAT sweeping. In line 3-4, the procedure *FindCexInSim* checks if any counterexample can be found at the primary output end by simulation. If so, we can return `NEQ` at an early stage. The loop starting in line 5 iterates through each primary output pair ($\hat{o}_g$, $\hat{o}_r$) and acquires the learned CE-clauses. In line 6, procedure *GetCone* collects the signals in the transitive fanin cone of the input signal. The inner loop in line 7 iterates each pair of signals in the extracted cones that are potentially compatible equivalent by the simulation patterns. In line 8, procedure *CheckPairCE* returns the status of compatible equivalence checking on the two given signals and the corresponding set of learned CE-clauses L, which is then added to the clause set C in line 9. If $\hat{s}_g$ and $\hat{s}_r$ are not compatible equivalent, a witness *cex* is also returned by *CheckPairCE*. Then in lines 10-11 the corresponding witness *cex* is added to the set of simulation patterns and the circuits $\hat{G}$ and $\hat{R}$ are re-simulated. In lines 12-13, procedure *FindCexInSim* is called again to check the patterns at the primary output end. Lastly, in lines 14-15, we incorporate the CNF formula of the miter of $\hat{G}$ and $\hat{R}$ with the clauses acquired by internal CE proving into clause set C, and invoke a SAT call to determine the satisfiability of C. If C is unsatisfiable, then circuits $\hat{G}$ and $\hat{R}$ are compatible equivalent. Otherwise, they are inequivalent. Note that in line 7 the signals are traversed in a topological order from inputs to outputs, and the SAT solving of internal CE identification is done incrementally. As in the standard SAT sweeping procedure, the solver can accumulate circuit information in each round of incremental solving, and may gradually become more effective in decision making.

Algorithm 1 CEProve

```
Input: two x-valued topological-ordered circuits $\hat{G}$ and $\hat{R}$
Output: compatible equivalence of $\hat{G}$ and $\hat{R}$
 1: $C \leftarrow \emptyset$;
 2: RandomSim($\hat{G}$, $\hat{R}$);
 3: if FindCexInSim($\hat{G}$, $\hat{R}$) then
 4:     return NEQ;
 5: foreach ($\hat{o}_g$, $\hat{o}_r$) in GetPoPair($\hat{G}$, $\hat{R}$) do
 6:     $\hat{N}_g, \hat{N}_r \leftarrow$ GetCone($\hat{o}_g$), GetCone($\hat{o}_r$);
 7:     foreach ($\hat{s}_g$, $\hat{s}_r$) in GetCECandidatePair($\hat{N}_g$, $\hat{N}_r$) do
 8:         $stat, L \leftarrow$ CheckPairCE($\hat{s}_g$, $\hat{s}_r$);
 9:         $C \leftarrow C \cup L$;
10:         if stat = NEQ[cex] then
11:             add cex to simulation patterns;
12:             if FindCexInSim($\hat{G}$, $\hat{R}$) then
13:                 return NEQ;
14: $C \leftarrow C \cup$ ToCNF(BuildMiter($\hat{G}$, $\hat{R}$));
15: return IsSAT($C$) ? NEQ : EQ;
```

VI. Experimental Results

The proposed methods were implemented in C++ within the `ABC` system [16]. All the experiments were conducted on a Linux server with Intel Xeon CPU E5-2620 v4 of 2.10 GHz and 126 GB RAM. A timeout limit of 1800 seconds is imposed on each run of the experiment. The statistics of the 28 industrial benchmarks[2] of the 2020 ICCAD CAD Contest are shown in Table IV, where Column 2 lists the application source of the X-value of each benchmark including index-out-of-range (denoted IoR), casex (XC), X-bit (XB), power model (PM), and don't care space (DCS); Columns 3-7 list the numbers of primary inputs(PIs), primary outputs(POs), and logic gates of the golden and revised RTL circuits, respectively; Column 8 lists the answer to the compatible equivalence checking of the golden and revised circuits for each benchmark.

In the following tables and figures, we compared various methods, include dcec (the best performing CEC engine of `ABC` on the benchmarks), dsat (direct SAT solving of miter circuit)[3], xcec (the xcec flow in Section IV), and cepr (the procedure of Algorithm 1 with preprocessing using SAT sweeping). Each of the methods is further annotated by the applied encoding techniques. For a fair comparison, `kissat` [17], the champion in the main track of the 2020 SAT competition, was used as the SAT oracle in the final miter solving stage of all the methods.[4] On the other hand, since `kissat` does not support incremental solving, we adopted `glucose` [18] as the incremental SAT solver in our clause learning procedure in cepr. The studied methods were also compared with the commercial tool Cadence Conformal

[2] We excluded case25 and case30 from the table as they do not contain any X-value and can be solved by conventional CEC. [3] In dsat, the dual-rail encoded circuit was directly converted to a CNF formula by Tseitin transformation [14], without further simplification. [4] `kissat` is configured to target unsatisfiable instances for better performance in the number of solved cases among contest benchmarks.

Table IV: Benchmark statistics.

	X-source	#PIs	#POs	#Gates			CE
				Golden	Revised	Total	
case1	IoR	48	20	646	1019	1665	EQ
case2	IoR	48	20	646	1024	1670	NEQ
case3	XC	64	32	4407	7150	11557	EQ
case4	XC	64	32	4407	7442	11849	NEQ
case5	PM	160	32	3817	7376	11193	EQ
case6	PM	160	32	3817	7376	11193	EQ
case7	PM	96	59	7188	7157	14345	NEQ
case8	DCS	8214	93	38945	87216	126161	EQ
case9	XB	96	58	14931	14931	29862	NEQ
case10	IoR	256	119	44595	43231	87826	EQ
case11	IoR	256	85	43817	42282	86099	NEQ
case12	IoR	614	54	15170	20075	35245	EQ
case13	IoR	614	68	15320	15058	30378	EQ
case14	PM	128	1	21813	21806	43619	NEQ
case15	XB	193	120	44546	46487	91033	EQ
case16	XC, IoR	1903	1382	47443	38951	86394	EQ
case17	XB	128	115	22488	76352	98840	EQ
case18	IoR, PM	217	2	459	655	1114	EQ
case20	XC	64	27	4419	7752	12171	EQ
case21	DCS	8214	93	38944	57008	95952	EQ
case22	IoR	256	119	66268	46528	112796	EQ
case23	IoR	256	85	65155	45234	110389	NEQ
case24	XB	128	115	43018	39249	82267	EQ
case26	IoR	614	54	14520	15163	29683	EQ
case27	IoR	614	54	14424	21449	35873	EQ
case28	PM	48	20	1208	1363	2571	EQ
case29	XB	786	27	2991	10158	13149	EQ
case31	IoR	48	12	1157	1312	2469	EQ

Smart Logic Equivalence Checker (LEC), version 20.10-p100. Furthermore, we compared our methods with those of the other top contestants, the 2nd and 3rd place winners of the 2020 ICCAD CAD Contest (under the same machine execution and using the same SAT engine `kissat` as ours). We note that `ABC` provides a recent command "*xec*" in response to the contest. However, its performance is inferior to other contest winners in terms of both runtime and the number of solved cases, and thus it is excluded from the comparison.

The overall results are summarized in Table V, where Column 1 lists the method, Columns 2-4 list the number of solved cases, and Column 5 lists the total computation time (in seconds) of the solved cases. The methods in this table are sorted by the number of solved cases, with ties being broken by total runtime. From the table, we can see that our methods xcec-E_{xp}^{c} and xcec-E_{xp}^{nc} solved the most cases (20 out of 28), with xcec-E_{xp}^{nc} having a slight edge on the total computation time. (We note that the 3rd place winner of the CAD Contest solved more cases than the 2nd place winner due to the fact that the evaluation of the contest was conducted with respect to a selected subset of the benchmarks.) In addition, the quantile plot of eight selected methods is shown in Figure 4, where a data point (x, y) indicates that are x cases solvable by the respective approach within y seconds. Evidently all our and the 2nd and 3rd place contestants' methods were more effective than the baseline method dcec and the commercial tool LEC.

Table VI shows a more detailed runtime information of each case, where Columns 5-6 list the best achieved runtime and the corresponding approach on each benchmark, and Columns 2-4 list the ratio of the runtime of the top 3 contest approaches to the best runtime. An entry "-" in the table indicates that the case cannot be solved within the timeout limit and the unsolvable cases by all methods are excluded. As can be seen, the studied methods/configurations have their own strengths in solving different benchmarks.

Table V: Performance comparison in the number of solved cases and total runtime.

method	# solved cases			total time
	EQ	NEQ	total	
xcec-E_{xp}^{nc}	13	7	20	5625.25
xcec-E_{xp}^{c}	13	7	20	6305.45
dsat-E_{xp}^{xi}	12	7	19	4691.05
dsat-E_{xp}^{nc}	12	7	19	5279.84
cepr-E_{xp}	12	7	19	6645.13
xcec-E_{xp}	11	7	18	2600.02
3rd place	11	7	18	2727.24
xcec-E_{sym}	11	7	18	4021.63
dsat-E_{xp}^{c}	10	7	17	2671.09
dsat-E_{xp}	11	6	17	3380.51
2nd place	9	7	16	1568.63
dsat-E_{sym}	9	6	15	2157.75
dcec-E_{xp}	8	7	15	4910.72
LEC	6	5	11	2344.78

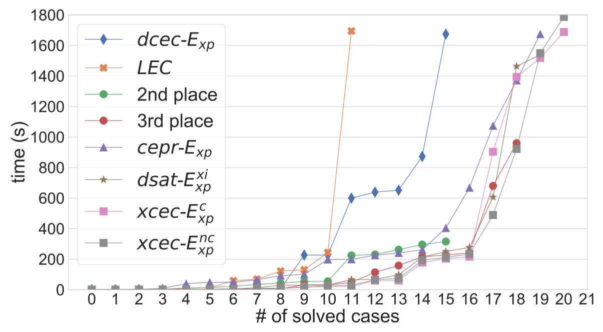


Figure 4: Performance comparison in the number of solved cases w.r.t. time bound.

A. Evaluation on Encoding Schemes and Synthesis Effects

The xcec flow was modified from dcec flow in `ABC` with the following changes: 1) `kissat` was used for the final SAT solving, 2) SAT sweeping was turned off, and 3) the number of `rewrite`, `refactor`, and `balance` in initial optimization stage were fine-tuned.

Table V shows that under xcec (resp. dsat) flow, E_{xp} encoding solved 18 (resp. 17) cases while E_{sym} solved 18 (resp. 15). Figure 5 further details the runtime comparison in a logarithmic scale between E_{xp} and E_{sym} of individual cases (if solvable by any of the two encodings). In the plot, the runtime of a timeout case is counted as 1800 seconds. As can be seen, the equivalent (EQ) cases in general took more time to solve than the non-equivalent (NEQ) cases. By considering the data points $(t_{\mathrm{xp}}, t_{\mathrm{sym}})$ for $10 \le t_{\mathrm{xp}}, t_{\mathrm{sym}} < 1800$ in Figure 5 to exclude relatively easy instances, it is evident that E_{xp} is superior to E_{sym}. Specifically, the average of $(t_{\mathrm{sym}} - t_{\mathrm{xp}})/t_{\mathrm{sym}} \times 100\%$ of all these points of xcec (resp. dsat) is 25.35% (resp. 25.92%). The results reveal that the superiority E_{xp} to E_{sym} for both xcec and dsat flows holds regardless of the effect of synthesis tools.

The advantage of E_{xp} over E_{sym}, especially for EQ cases, can be explained through the analysis of Table VII, where

Table VI: Runtime comparison of top solving methods.

	1st (ratio)	2nd (ratio)	3rd (ratio)	Our Best	
	(xcec-E_{xp}^{nc})			time (s)	method
case1	1.19	1.27	1.36	177.44	xcec-E_{xp}^{c}
case2	4.97	5.05	4.17	0.33	xcec-E_{sym}
case4	4.68	3.66	0.61	0.98	xcec-E_{sym}
case5	1.51	2.89	1.72	18.49	xcec-E_{xp}^{c}
case6	1.52	2.98	1.72	18.48	xcec-E_{xp}^{c}
case7	15.73	15.74	4.93	0.23	dsat-E_{sym}
case8	1.13	-	-	1371.15	cepr-E_{xp}
case9	1.29	15.35	2.30	0.51	dsat-E_{xp}^{xi}
case11	1.66	6.70	1.31	5.19	dsat-E_{xp}^{xi}
case12	2.03	-	2.12	453.92	cepr-E_{xp}[1]
case13	1.30	5.26	2.02	56.24	xcec-E_{xp}
case14	1.01	8.07	0.93	1.81	xcec-E_{xp}
case16	1.00	3.02	3.39	14.18	xcec-E_{xp}^{nc}
case18	1.20	5.59	1.06	0.07	xcec-E_{sym}
case21	2.97	-	-	602.00	cepr-E_{xp}
case23	1.00	2.41	0.72	9.21	xcec-E_{xp}^{nc}
case26	1.27	6.13	3.07	51.47	dsat-E_{xp}
case27	1.13	-	1.57	433.59	cepr-E_{xp}[2]
case28	1.00	1.20	1.12	192.39	xcec-E_{xp}^{nc}
case31	1.23	1.38	1.19	190.62	xcec-E_{xp}

[1] w/o sweep + incremental solver `minisat` [19] [2] w/o sweep

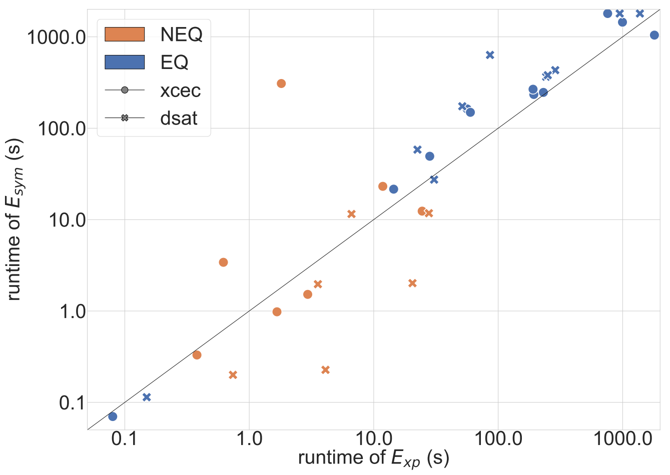


Figure 5: Runtime comparison of encoding schemes E_{sym} and E_{xp} under the dsat and xcec flows.

the ternary values and their binary coded values of two corresponding primary output signals $(\hat{o}_g, \hat{o}_r)$ between circuits $\hat{G}$ and $\hat{R}$ are shown. The symbol "–" denotes a don't-care. From the table, we see that if $o_g^1 = 1$ under a primary input assignment, then E_{xp} allows direct conclusion of EQ under the assignment. In contrast, E_{sym} cannot conclude EQ from a single bit observation. While E_{sym} can be more succinct resulting in smaller CNF formulas, E_{xp} exhibits stronger implication capability and is advantageous over E_{sym} in the EQ cases.

When the techniques of x-bit literal insertion and x-bit fanin insertion as mentioned in Sections IV-B and IV-C are applied, the performance of compatible equivalence checking can be further enhanced. In dsat flow, with x-bit literal insertion, dsat-E_{xp}^{xi} solved four more cases compared to dsat-E_{sym}, and two more cases compared to dsat-E_{xp} as observed in Table V. As discussed previously, dsat-E_{xp}^{xi} may conditionally disable some clauses under different assignments during SAT

Table VII: Valuations of E_{sym} and E_{xp} under EQ and NEQ conditions.

	Ternary Logic	E_{xp}	E_{sym}
	$(\hat{o}_g, \hat{o}_r)$	$(o_g^0 o_g^1, o_r^0 o_r^1)$	$(o_g^0 o_g^1, o_r^0 o_r^1)$
EQ	(x,0), (x,1), (x,x)	(–1,– –)	(00,– –)
	(0,0)	(00,00)	(10,10)
	(1,1)	(10,10)	(01,01)
NEQ	(1,0)	(10,00)	(01,–0)
	(1,x)	(–0,–1)	
	(0,x)		(10,0–)
	(0,1)	(00,10)	

solving. The speedup from dsat-E_{xp} to dsat-E_{xp}^{xi} indicates that conditionally disabling clauses indeed improves SAT solving.

A similar effect of performance improvement can be observed between E_{xp} with or without x-bit fanin insertion, given that dsat-E_{xp}^{nc} solved two more cases than dsat-E_{xp} and in xcec flow, with x-bit fanin insertion, xcec-E_{xp}^{c} and xcec-E_{xp}^{nc} dominate the number of solved cases as they solve the most number of cases among all of our methods. To further compare the effect between E_{xp}^{c} and E_{xp}^{nc} under both xcec and dsat flows, E_{xp}^{nc} either solved more cases or took less runtime than E_{xp}^{c} as seen from Table V. As discussed in Section IV-C, E_{xp}^{c} allows more implication propagation on gate variables topologically from primary inputs to primary outputs, while E_{xp}^{nc} conditionally disables some fanins. The results might suggest that conditionally disabling fanins could lead to more significant speedup than propagating implications.

We note that the validities of x-bit literal insertion and x-bit fanin insertion rely on the special characteristics of E_{xp}, that is, the original bit o^0 of a signal (o^0, o^1) becomes don't care, i.e., not affecting the miter output, when $o^1 = 1$. Essentially, x-preserving encoding E_{xp} allows these specialized techniques for performance enhancement. These results show that our x-preserving encoding E_{xp} is more suitable to compatible equivalence checking in the contest benchmarks compared to other dual-rail encoding proposed in [6].

Our observation that SAT sweeping was not helpful in compatible equivalence checking may seem counter-intuitive. Table VIII shows the two cases that was originally solvable by xcec but unsolvable if SAT sweeping is performed prior to xcec (denoted as swp-xcec), where the "#node" columns list the circuit size before final SAT solving, and the "final SAT" and "sweep" column list the runtime (in seconds) spent on them. To make sure this phenomenon is caused by sweeping, we let `kissat` solve the reduced circuits for another 1800 seconds. As shown in the table, although SAT sweeping effectively reduces circuit sizes, the resulting circuits become harder to prove and cannot be solved in timelimit. We therefore excluded SAT sweeping in our xcec flow. The fact that smaller circuit and CNF formula sizes may not be always helpful is consistent with our observation of E_{xp} being more efficient than E_{sym}. Both E_{sym} and SAT sweeping may reduce the formula size for SAT solving, but not the runtime.

B. *Evaluation on CE Relation Identification*

From our experimental evaluation, of all the different tested configurations of cepr, the one with SAT sweeping opti-

Table VIII: Results of xcec with and without SAT sweeping.

	xcec-E_{xp}^{nc}		swp-xcec-E_{xp}^{nc}		
	#node	Final SAT	#node	Final SAT	Sweep
case8	171410	1549.82	114936	>1800	940.94
case12	59451	922.25	30097	>1800	156.05

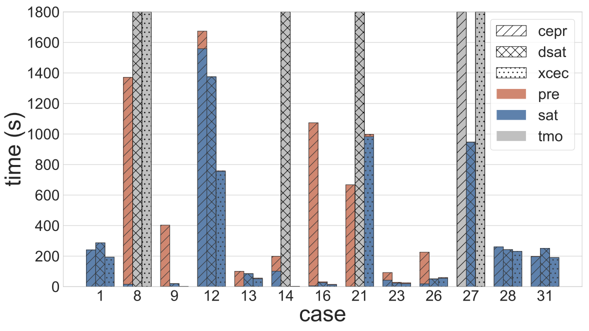


Figure 6: Runtime comparison of methods cepr, dsat and xcec.

mization and `glucose` as the incremental solver performed the best. Different from the previous observation in xcec, SAT sweeping helps in cepr due to its circuit simplification capability and thus reducing the effort of internal CE pair identification. On average, the number of CE-clauses acquired via Proposition 2 accounted for 9.56% of the total learned CE-clauses over the cases solved by cepr . When compared to other approaches under E_{xp}, method cepr was able to solve 19 cases over the 18, 17 and 15 cases solved by xcec, dsat and dcec, respectively. Figure 6 shows the runtime comparison of methods cepr dsat, and xcec under E_{xp}, each marked with a distinct screentone. A runtime is divided into two portions colored in orange and blue, which corresponds to CNF preparation time (pre) (including the time spent on circuit optimization, internal CE relation identification, and CNF translation from AIG), and miter satisfiability solving time (sat), respectively. Additionally, timeout (tmo) cases are marked in light gray. For simplicity, easily-solved cases, those solved within 50 seconds by all the three methods, are excluded from the chart. We note that case 8 was only solvable by cepr, where the majority of runtime was spent on internal CE relation identification (pre) while miter satisfiability solving (sat) only took a small fraction, indicating that the strengthened CNF formula with additional learned clauses can be solved more easily. In fact, case 8 is one of the hardest solvable cases in our experiments and cepr-$E_{xp}$was the best among the three successful methods cepr-E_{xp}, xcec-E_{xp}^{c}, and xcec-E_{xp}^{nc}. However, there were also cases, such as case 12, that even with an abundant learned information, the solving took longer than the other two methods. When evaluated by the number of solved cases, cepr is clearly the best among the four methods in Table V under E_{xp}, namely, xcec-E_{xp}, dsat-E_{xp}, cepr-E_{xp}, and dcec-E_{xp}. However, not all cases experienced speedup with such approach and a large number of learned clauses did not always imply the improved efficiency of the final miter solving.

In summary, our proposed encoding schemes and flows outperformed other methods in the CAD Contest. Additionally, E_{xp} is preferred over E_{sym} in compatible equivalence checking. Exploiting the don't care property and internal CE relation of E_{xp} can further improve the verification ability. Especially, xcec-E_{xp}^{c} and xcec-E_{xp}^{nc} solved the most cases and cepr-E_{xp} significantly reduced the runtime on two hard `EQ` cases.

VII. Conclusions

We have presented our method to compatible equivalence checking based on the X-value preserving dual-rail encoding and internal CE relation identification. Experiments on industrial designs have demonstrated the superiority of our method to other competitive methods in the ICCAD CAD Contest and the effectiveness of the proposed techniques. For future work, motivated by [20], we would like to devise new SAT solving strategies exploiting circuit don't cares.

References

[1] C. L. Berman and L. H. Trevillyan, "Functional comparison of logic designs for VLSI circuits," in *Proc. ICCAD*, pp. 456–459, 1989.

[2] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. DAC*, p. 263–268, 1997.

[3] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *IEEE TCAD*, vol. 21, no. 12, pp. 1377–1394, 2002.

[4] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proc. ICCAD*, p. 836–843, 2006.

[5] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *Proc. DAC*, pp. 532–535, 2006.

[6] M.-J. Nam, C.-H. Sung, and J. Choi, "Sat-based combinational equivalence checking with don't care," 2004.

[7] T. Melham, "Symbolic trajectory evaluation," in *Handbook of Model Checking*, pp. 831–870, 2018.

[8] H.-Z. Chou, H. Yu, K.-H. Chang, D. Dobbyn, and S.-Y. Kuo, "Finding reset nondeterminism in RTL designs-scalable x-analysis methodology and case study," in *Proc. DATE*, pp. 1494–1499, 2010.

[9] R. Drechsler, S. Eggersglüß, G. Fey, and D. Tille, *Test pattern generation using Boolean proof engines*. Springer, 2009.

[10] K. Yuan, C. Kuo, J. R. Jiang, and M. Li, "Encoding multi-valued functions for symmetry," in *Proc. ICCAD*, pp. 771–778, 2013.

[11] "IEEE standard for Verilog hardware description language," *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pp. 1–590, 2006.

[12] D. Brand, "Verification of large synthesized designs," in *Proc. ICCAD*, p. 534–537, 1993.

[13] C.-J. Hsu, C.-A. Wu, C.-Y. Huang, and C.-H. Chou, *Problem A: X-value Equivalence Checking*, 2020. http:/iccad-contest.org/2020/.

[14] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of reasoning*, pp. 466–483, 1983.

[15] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *J. Symb. Comput.*, vol. 2, no. 3, pp. 293–304, 1986.

[16] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. CAV*, pp. 24–40, 2010.

[17] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. SAT Competition 2020 – Solver and Benchmark Descriptions*, vol. B-2020-1, pp. 51–53, 2020.

[18] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proc. IJCAI*, p. 399–404, 2009.

[19] N. Sörensson and N. Eén, "Minisat v1.13 - a sat solver with conflict-clause minimization," in *Proc. SAT Competition 2005 – Solver Descriptions*, 2005.

[20] Z. Fu, Y. Yu, and S. Malik, "Considering circuit observability don't cares in CNF satisfiability," in *Proc. DATE*, pp. 1108–1113, 2005.