



Synthesizing Complementary Circuits Automatically

ShengYu Shen, *Member, IEEE*, Ying Qin, KeFei Wang, LiQuan Xiao, JianMin Zhang, and SiKun Li

AQ:1 

AQ:2 


Abstract—One of the most difficult jobs in designing communication and multimedia chips is to design and verify the complex complementary circuit pair (E, E^{-1}) , in which circuit E transforms information into a format suitable for transmission and storage, and its complementary circuit E^{-1} recovers this information. In order to facilitate this job, we proposed a novel two-step approach to synthesize the complementary circuit E^{-1} from E automatically. First, a SAT solver was used to check whether the input sequence of E can be uniquely determined by its output sequence. Second, the complementary circuit E^{-1} was built by characterizing its Boolean function, with an efficient all-solution SAT solver based on discovering XOR gates and extracting unsatisfiable cores. To illustrate its usefulness and efficiency, we ran our algorithm on several complex encoders from industrial projects, including PCIE and 10G ethernet, and successfully built correct complementary circuits for them.

Index Terms—All-solution SAT, complementary circuit, discovering XOR gates, extracting unsatisfiable core, synthesis.

I. INTRODUCTION

COMMUNICATION and multimedia electronic applications are the major driving forces of the semiconductor industry. Many leading edge communication protocols and media formats, even still in their non-standardized draft status, are implemented in chips and pushed to market to maximize the chances of being accepted by consumers and becoming the de facto standards. Two such well-known stories are the 802.11n wireless standard competition [1], and the disk format war between HD and Blu-ray [2]. In such highly competitive markets, designing correct chips as fast as possible is the key to success.

One of the most difficult jobs in designing communication and multimedia chips is to design and verify the complex complementary circuit pair (E, E^{-1}) , in which circuit E transforms information into a format suitable for transmission and storage, and its complementary circuit E^{-1} recovers this information. Such difficulties are caused by many factors, such as deep pipelining to achieve high frequency, the complex encoding mechanism to achieve reliability and compression ratio.

Manuscript received September 18, 2009; revised January 10, 2010, March 10, 2010, and March 24, 2010. This work was supported by National Natural Science Foundation of China, under Project 60603088. This work was also supported by the Program for Changjiang Scholars and Innovative Research Team in University, under Grant IRT0614.  paper was recommended by Associate Editor St. Nowick.

The authors are with the School of Computer, National University of Defense Technology, Changsha, Hunan 410073, China (e-mail: syshen@nudt.edu.cn; qy123@nudt.edu.cn; lqxiao@nudt.edu.cn; kfwang@nudt.edu.cn; jmzhang@nudt.edu.cn; skli@nudt.edu.cn).

Digital Object Identifier 10.1109/TCAD.2010.2049152

In order to facilitate this job, we propose in this paper a novel approach to automatically synthesize E^{-1} from E in two steps.

- 1) In the first step, a SAT solver is used to check whether a valuation exists for some parameters, so that the input sequence of E can be uniquely determined by its output sequence. We call this the *parameterized complementary condition*.
- 2) In the second step, with the SAT instance and parameter values obtained in the first step, circuit E^{-1} is built by characterizing its Boolean function f^{-1} , with an efficient all-solution SAT solver (abbreviated as ALLSAT) based on discovering XOR gates and extracting unsatisfiable cores.

We implement our algorithm on zchaff [3], and run it on several complex encoder circuits from industrial projects, including PCIE and 10G Ethernet. Some of these circuits have datapath as wide as 64 bits. It has turned out that all these complementary circuits can be built within 1000 s. And all these experimental results and related programs can be downloaded from <http://www.ssypub.org>.

There is one issue that needs to be clarified before we start presenting our idea. There are two classes of complementary circuit pairs, each with its own character and design methodology.

- 1) *Standard datapath-intensive circuits*: Such circuits often work as digital signal processing components, including fast Fourier transform, discrete cosine transform, and so on. These circuits and their complementary circuits usually have standard and highly optimized implementations from various foundries and IP vendors, such as Xilinx core generator [4] and Synopsys DesignWare Library [5]. Therefore, our algorithm is not designed for these circuits.
- 2) *Non-standard and control-intensive circuits*: These circuits, such as PCIE and Ethernet, are often used to handle communication protocols, and typically do not have standard implementations. Our algorithm is designed for these circuits.

Compared with our previous work presented at ICCAD'09 [6], the major contributions of this paper are twofold.

- 1) We present a more formal and accurate description of checking the parameterized complementary condition. For more information, please refer to Section III-C.
- 2) More importantly, we integrate an unsatisfiable core extraction algorithm into the algorithm that characterizes the Boolean function f^{-1} to improve its run-time overhead. For more information, please refer to Section IV.

The remainder of this paper is organized as follows. Section II introduces the background material. In Section III, we discuss how to check the parameterized complementary condition, and how to find out proper values of its parameters. Section IV describes how to characterize the Boolean function of the complementary circuit. Section V describes how to build the complementary circuit from its Boolean function. Section VI presents experimental results. Section VII presents related works. Finally, we conclude with a note on future work in Section VIII.

II. PRELIMINARIES

A. Basic Notation of Propositional Satisfiability Problem

For a Boolean formula F over variable set V , the *Propositional Satisfiability Problem* (abbreviated as SAT) is to find a *satisfying assignment* $A : V \rightarrow \{0, 1\}$, so that F can be evaluated to 1.

If such a satisfying assignment exists, then F is a *satisfiable formula*; otherwise, it is an *unsatisfiable formula*.

A computer program that decides the existence of such a satisfying assignment is called a *SAT solver*. Some famous SAT solvers are zchaff [3], Berkmin [7] and MiniSAT [8].

Normally, a SAT solver requires formula F to be represented in the conjunctive normal form (CNF) or the and-inverter graph (AIG) formats. In this paper, we will only discuss the CNF format, in which a *formula* $F = \bigwedge_{cl \in CL} cl$ is a conjunction of its clause set CL , and a *clause* $cl = \bigvee_{l \in Lit} l$ is a disjunction of its literal set Lit , and a *literal* is a variable v or its negation $\neg v$. A formula in the CNF format is also called a *SAT instance*.

For an assignment $A : U \rightarrow \{0, 1\}$, if $U \subset V$, then A is a *partial assignment*; otherwise, if $U \equiv V$, then A is a *complete assignment*.

For an assignment $A : U \rightarrow \{0, 1\}$, and $W \subset U$, $A|_W : W \rightarrow \{0, 1\}$ is the *projection* of A on W , which can be defined as follows:

$$A|_W(v) = \begin{cases} A(v) & v \in W \\ \text{undefine} & \text{otherwise.} \end{cases}$$

Obviously, $A|_W$ is obtained from A by removing all variables $v \notin W$.

For an assignment $A : U \rightarrow \{0, 1\}$, $u \notin U$, and $b \in \{0, 1\}$, $A|^{u \rightarrow b}$ is the *extension* of A on u , which can be defined as follows:

$$A|^{u \rightarrow b}(v) = \begin{cases} A(v) & v \in U \\ b & v \equiv u. \end{cases}$$

Obviously, $A|^{u \rightarrow b}$ is obtained by inserting the assignment of u into A .

For a satisfying assignment A of formula F , its *blocking clause* is as follows:

$$bcls_A = \bigvee_{A(v) \equiv 1} \neg v \vee \bigvee_{A(v) \equiv 0} v. \quad (1)$$

It is obvious that A is not a satisfying assignment of $F \wedge bcls_A$. So $bcls_A$ can be inserted into the SAT solver to prevent A from becoming a satisfying assignment again.

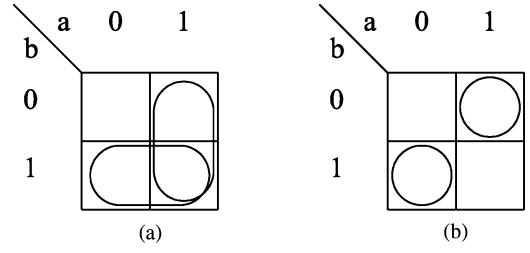


Fig. 1. Satisfying assignments for simple gates. (a) OR. (b) XOR.

An unsatisfiable formula often has many clause subsets that are also unsatisfiable, these subsets are called *unsatisfiable cores*. Some unsatisfiable core extraction algorithms are proposed by Goldberg and Novikov [9] and Zhang and Malik [10].

Although our algorithm is based on the unsatisfiable core extraction algorithms, the readers do not need to understand the details of it. The readers only need to know that the result of unsatisfiable core extraction algorithms is an unsatisfiable subset of the original formula.

B. All-Solution SAT Solver

State-of-the-art SAT solvers normally find only one complete satisfying assignment. But many applications, such as the two-level logic minimization [11], need to enumerate all satisfying assignments.

Such technologies that enumerate all satisfying assignments of a formula are called (ALLSAT). Obviously, we can enumerate all satisfying assignments by repeatedly calling a SAT solver, and inserting the blocking clause $bcls_A$ of satisfying assignment A into the SAT solver, until no more new satisfying assignments can be found.

But for a formula with n variables, there may be 2^n satisfying assignments to be enumerated. Thus, this approach is impractical for a large n .

In order to reduce the number of satisfying assignments to be enumerated, we need *satisfying assignments minimization* technique to remove irrelevant variables' assignments from satisfying assignment A , so that A can cover more complete satisfying assignments. For example, for OR gate $z = a \vee b$ in Fig. 1(a), its complete satisfying assignments that can make $z \equiv 1$ are $\{a = 1, b = 0\}$, $\{a = 1, b = 1\}$, and $\{a = 0, b = 1\}$, which contain six assignments to individual variables. It's obvious that $\{a = 1, b = 0\}$ and $\{a = 1, b = 1\}$ can be merged into $\{a = 1\}$ by removing b . At the same time, $\{a = 1, b = 1\}$ and $\{a = 0, b = 1\}$ can also be merged into $\{b = 1\}$ by removing a . Therefore, these two newly-merged partial assignments contain only two assignments to individual variables, and are much more succinct than the previous three complete assignments.

Formally, assume that F is a formula over Boolean variable set V , $v \in V$ is an object variable that should always be 1, A is a satisfying assignment of $F \wedge v$, and $U \subseteq V$ is a variable set whose assignments we would like to minimize and enumerate. We can test whether $u \in U$ is irrelevant to forcing v to 1, by testing the unsatisfiability of $F \wedge \neg v \wedge A|_{U-\{u\}}$. If $F \wedge \neg v \wedge A|_{U-\{u\}}$ is unsatisfiable, then $A|_{U-\{u\}}$ cannot make

Algorithm 1: *ALLSAT* based on *BFL* Algorithm.

```

1) ALLSAT( $F, v, U$ ) {
2)    $SA_v = \{\}$ 
3)   while ( $F \wedge v$  has a satisfying assignment  $A$ ) {
4)      $A = \mathbf{BFL}(F, v, U, A)$ 
5)      $SA_v = SA_v \cup \{A\}$ 
6)      $F = F \wedge bcl_{SA}$ 
7)   }
8)   return  $SA_v$ 
9) }
10) BFL( $F, v, U, A$ ) {
11)   foreach  $u \in U$ 
12)     if ( $(F \wedge \neg v \wedge A|_{U-\{u\}})$  is unsatisfiable)
13)        $A = A|_{U-\{u\}}$ 
14)   return  $A$ 
15) }
```

v to be 0, so v must still be 1. Thus, by removing u from A , we can merge A and $A|_{U-\{u\}}|^{u \rightarrow \neg A(u)}$, and obtain a succinct satisfying assignment $A|_{U-\{u\}}$.

All existing ALLSAT approaches [12]–[19] share this idea of satisfying assignments minimization. Here we will present only one of them, i.e., the brutal force lifting (BFL) algorithm [14].

Line T2 will test whether removing u from A can still make v to always take on value 1. If yes, then u will be removed from both A and V . In this way, A will become a partial assignment covering more complete assignments.

On the other hand, for XOR gate $z = a \oplus b$ in Fig. 1(b), its complete satisfying assignments that can make $z \equiv 1$ are $\{a = 1, b = 0\}$ and $\{a = 0, b = 1\}$, which cannot be merged. Unfortunately, XOR gates are widely used in almost all communication circuits, including but not limited to the scrambler and descrambler, the CRC generator and checker, the pseudo random test pattern generator and checker.

An extreme example is a n -bit comparator that compares two n -bit variables. In this case, there are 2^n complete satisfying assignments, none of which can be merged with each other.

Thus, enumerating satisfying assignments for XOR intensive circuits is a major difficulty of all existing ALLSAT approaches. We will solve this problem in Section IV.

C. Checking Reachability With Bounded Model Checking

The description of our algorithm will largely follow that of bounded model checking (BMC) [20], so we present here briefly how to check reachability in BMC.

Definition 1: A *Kripke structure* is a 5-tuple $M = (S, I, T, A, L)$, with a finite set of states S , the set of initial states $I \subseteq S$, the transition relation between states $T \subseteq S \times S$, and the labeling of the states $L : S \rightarrow 2^A$ with atomic propositions set A .

BMC is a model-checking technique that considers only limited-length paths. We call this length the bound of path. We denote the i th and the $i + 1$ th state as s_i and s_{i+1} , and the transition relation between them as $T(s_i, s_{i+1})$.

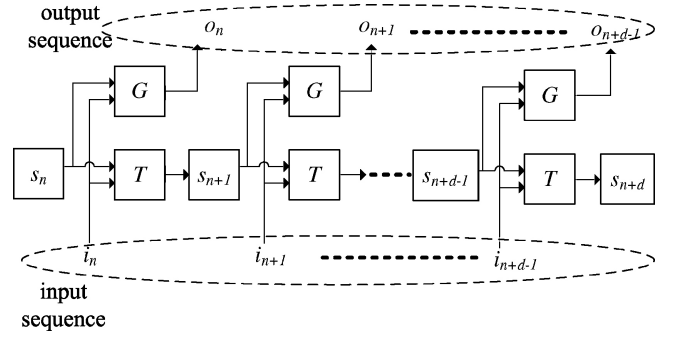


Fig. 2. Unfolding transition function of **mealy** finite state machine.

Let the safety property under verification be *ASSERT*, the goal of BMC is to find a state that violates *ASSERT*. Then the BMC problem with bound b can be expressed as follows:

$$I(s_0) \wedge \bigwedge_{i=1}^{b-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=1}^b \neg \text{ASSERT}(s_i). \quad (2)$$

We can search for a counterexample of length b by solving (2) with a SAT solver.

III. CHECKING THE PARAMETERIZED COMPLEMENTARY CONDITION

In this section, we will explain how to check whether the input sequence of circuit E can be recovered from its output sequence.

A. Parameterized Complementary Condition

Our algorithm is concerned about the input and output sequence of circuit E , so *Mealy finite state machine* [21] is a more suitable model for us than the Kripke structure.

Definition 2: *Mealy finite state machine* is a 6-tuple $M = (S, s_0, I, O, T, G)$, consisting of the following.

- 1) A finite set of states S .
- 2) An initial state $s_0 \in S$.
- 3) A finite set of letters called the input alphabet I .
- 4) A finite set of letters called the output alphabet O .
- 5) A state transition function $T : S \times I \rightarrow S$.
- 6) An output function $G : S \times I \rightarrow O$.

The circuit E can be modeled by such a Mealy finite state machine. The relationship between its output sequence $o \in O^\omega$ and input sequence $i \in I^\omega$ is shown in Fig. 2. This relationship can be built by unfolding the transition function T and the output function G for d times, as shown in (3)

$$F_E = \bigwedge_{m=n}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \right\}. \quad (3)$$

In order to recover $i \in I^\omega$ from $o \in O^\omega$, we must know how to compute i_n for every n , that is, to find a function f^{-1} that can compute i_n from $o \in O^\omega$.

But due to the limited memory of realistic computers, we cannot take the infinite length sequence $o \in O^\omega$ as input to f^{-1} . Instead, we can only use a finite length sub-sequence of

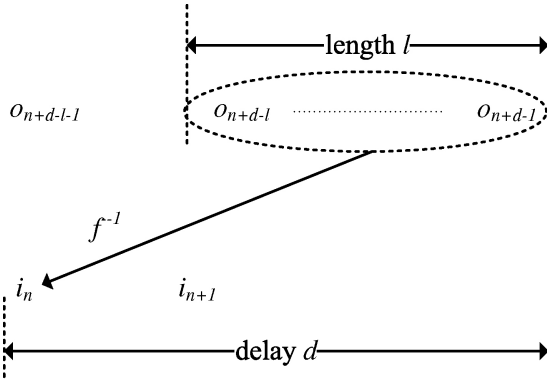


Fig. 3. f^{-1} and parameters of o 's finite length subsequence.

o , which has two parameters, length l and delay d compared to i_n , as shown in Fig. 3.

Thus, $f^{-1} : O^l \rightarrow I$ should be a Boolean function that takes the finite length sequence $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$ as its input, and computes i_n .

For a particular pair of d and l , f^{-1} exists if the following condition holds.

Definition 3: Parameterized complementary condition: For any valuation of the sequence $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$, there exists no more than one valuation of i_n that can make (3) satisfiable.

To test whether there exists another i'_n that can make (3) satisfiable, we need to unfold function T and G for another d times

$$F'_E = \bigwedge_{m=n}^{n+d-1} \{s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m)\}. \quad (4)$$

Obviously, (4) is just another copy of (3), except that its variables are all renamed by appending a prime.

Thus, the parameterized complementary condition holds if and only if the following is unsatisfiable:

$$\begin{aligned} & F_E \wedge F'_E \wedge \\ & \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ & i_n \neq i'_n. \end{aligned} \quad (5)$$

In (5), the first line contains two unfolded instances of circuit E . The second line constrains their output sequences $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$ and $\langle o'_{n+d-l}, \dots, o'_{n+d-1} \rangle$ to be the same, and the third line constrains that their input letter i_n and i'_n are different.

For a particular pair of d and l , checking (5) may return two results.

- 1) *Satisfiable*. In this situation, for a $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$, there exists two different i_n and i'_n that can both make (3) satisfiable. This means the input letter i_n cannot be uniquely determined by $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$. So no f^{-1} exists for this pair of d and l . We should continue searching for larger d and l .
- 2) *Unsatisfiable*. In this situation, for any valuation of $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$, there exists no more than one valuation of i_n that can make (3) satisfiable. This

means the input letter i_n can be uniquely determined by $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$. So a f^{-1} exists for this pair of d and l . We will characterize f^{-1} with (3) in Section IV.

B. Ruling out Invalid Input Letters With Assertion

Most communication protocols and systems have some restrictions on the valid pattern of input alphabet. Assume that this restriction is expressed as an assertion predicate $R : I \rightarrow \{0, 1\}$, in which $R(i_n) \equiv 0$ means that i_n is an invalid input letter. Invalid input letters will be mapped to some predefined error output letter, that is, for $i_n, i'_n \in \{i_m | R(i_m) \equiv 0\}$, they will both be mapped to the same error output letter $e \in O$. This will prevent our approach from distinguishing i_n from i'_n .

Such restrictions are often documented clearly in the specification of communication protocols. Thus, we choose to employ an assertion-based mechanism, so that the user can code these restrictions R into their script or source code.

Thus, (3)–(5) should be rewritten into the following (6)–(8), in which the bold formulas are used to account for predicate R :

$$F_E = \bigwedge_{m=n}^{n+d-1} \{s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge \mathbf{R}(i_m)\} \quad (6)$$

$$F'_E = \bigwedge_{m=n}^{n+d-1} \{s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m) \wedge \mathbf{R}(i'_m)\} \quad (7)$$

$$\begin{aligned} & F_E \wedge F'_E \wedge \\ & \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ & i_n \neq i'_n. \end{aligned} \quad (8)$$

C. Overapproximating Reachable State Set

In the previous section, we have constrained the valid pattern of i_m . However, the s_n in Fig. 2 still has not been constrained yet. It may be outside of the reachable state set RS of circuit E . Under some special circumstances, the parameterized complementary condition may hold on reachable state set RS , but not on unreachable state set.

We can solve this problem by computing the reachable state set RS in (10), and constraining that $s_n \in RS$

$$RS^{s_0 \rightarrow p} = \{s | s \equiv s_p \wedge \bigwedge_{m=0}^{p-1} \{s_{m+1} \equiv T(s_m, i_m) \wedge \mathbf{R}(i_m)\}\}. \quad (9)$$

$$RS = \bigcup_{p>0} RS^{s_0 \rightarrow p} \quad (10)$$

$RS^{s_0 \rightarrow p}$ in (9) is the set of states that can be reached from the initial state s_0 with exact p steps.

Since it is very expensive to compute RS , we want to overapproximate it with

$$RS^{S \rightarrow p} = \left\{ s \mid \bigwedge_{m=n-p}^{n-1} \{s_{m+1} \equiv T(s_m, i_m) \wedge R(i_m)\} \right\}. \quad (11)$$

$RS^{S \rightarrow p}$ is the set of states that can be reached within p steps from *any* state in S . It is obvious that all $RS^{S \rightarrow p}$ form a total order relation

$$RS^{S \rightarrow p} \subseteq \dots \subseteq RS^{S \rightarrow q} \subseteq \dots \text{ where } p > q.$$

Unfortunately, RS is not a subset of any $RS^{S \rightarrow p}$, because there may be some state which, when starting from the initial state, can only be reached with no more than p steps. For example, a counter shown below that counts from 0 to 4, and then stays at 4 forever

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \underbrace{4}_{RS^{S \rightarrow 4}}.$$

In this case, numbers 0–3 are not in $RS^{S \rightarrow p}$, for $p > 3$.

Thus, we cannot overapproximate RS with $RS^{S \rightarrow p}$.

On the other hand, because circuits E and E^{-1} run in a non-stop way, we can safely assume that there always exists a prefix state transition sequence with enough length before the current state s_n . Thus, for any particular p , we only need to consider those states in $\bigcup_{q>p} RS^{S \rightarrow q}$ instead of RS . Obviously, $\bigcup_{q>p} RS^{S \rightarrow q}$ is a subset of $RS^{S \rightarrow p}$.

Thus, we can use $RS^{S \rightarrow p}$ as an overapproximation of $\bigcup_{q>p} RS^{S \rightarrow q}$.

In order to account for $s_n \in RS^{S \rightarrow p}$, we prepend $\bigwedge_{m=n-p}^{n-1} \{s'_{m+1} \equiv T(s'_m, i'_m) \wedge R(i'_m)\}$ to (6)–(8), and obtain (12)–(14). Thus, in addition to parameters d and l , we have a third parameter p to be searched

$$F_E = \bigwedge_{m=n-p}^{n+d-1} \{s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge R(i_m)\} \quad (12)$$

$$F'_E = \bigwedge_{m=n-p}^{n+d-1} \{s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m) \wedge R(i'_m)\} \quad (13)$$

$$\bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge i_n \neq i'_n. \quad (14)$$

Now putting it all together, with (12)–(14), we iterate through all valuations of d , l , and p , from the smaller one to the larger one, until we find one valuation of d , l , and p that makes (14) unsatisfiable. Then F_E in (12) will be used in Sections IV and V to build the complementary circuit E^{-1} .

IV. CHARACTERIZING f^{-1} WITH ALLSAT ALGORITHM DESIGNED FOR **XORINTENSIVE** CIRCUITS

If we find the proper values for parameters d , l , and p in Section III, we can now characterize the Boolean function $f^{-1} : O^l \rightarrow I$ in this section.

A. Partitioning f^{-1}

According to Section III-C, the complementary function $f^{-1} : O^l \rightarrow I$ is the function that takes $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$, and computes i_n . It can be characterized from the SAT instance of (12) by enumerating satisfying assignments of $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$ and i_n .

Assume that i_n is represented by a Boolean variable set I_{var} , and $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$ is represented by another Boolean variable set O_{var} , then, f^{-1} in Boolean domain can be represented as $f^{-1} : \{0, 1\}^{O_{var}} \rightarrow \{0, 1\}^{I_{var}}$, and can be defined as follows:

$$f^{-1} = \prod_{v \in I_{var}} f_v^{-1}. \quad (15)$$

Thus, characterizing f^{-1} can be partitioned into multiple tasks, with each task characterizing a Boolean function $f_v^{-1} : \{0, 1\}^{O_{var}} \rightarrow \{0, 1\}$ for a $v \in I_{var}$. The function f_v^{-1} will be used to compute the value of v .

Thus, in the remainder of this section, we will focus on characterizing f_v^{-1} instead of f^{-1} .

B. Algorithm Framework for Characterizing f_v^{-1}

Assume that $SA_v = \{A_1, \dots, A_m\}$ is the set of satisfying assignments of $F_E \wedge v$, that is, the set of satisfying assignments that forces v to 1. Then f_v^{-1} can be defined as follows:

$$f_v^{-1}(x) = \begin{cases} 1 & x \equiv A_1(x) \\ \dots & \\ 1 & x \equiv A_m(x) \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

but this naive approach suffers from the state space explosion problem. For O_{var} that contains k Boolean variables, there may be 2^k satisfying assignments in SA_v , which make it impossible to characterize f_v^{-1} for a large k .

There exists some much more efficient approaches to enumerate satisfying assignments of a SAT instance [12]–[19]. According to Section II-B, they all try to merge satisfying assignments in SA_v by removing irrelevant variables from each $A \in SA_v$, so that the size of SA_v can be reduced.

But these approaches are still not efficient enough for our application. The reasons for their inefficiency and the improvements of our approach are as follows.

- 1) The XOR gates are used extensively in communication and arithmetic circuits. As explained in Section II-B, the satisfying assignments of XOR cannot be merged by existing approaches. We solve this problem by discovering XOR gates within $F_E \wedge v$ with **XORMIN** function.
- 2) There are lots of redundant clauses in F_E . We use the function **SIMPLIFY** to simplify F_E to F_E^v before passing it to the main body of **ALLSAT**, by removing these redundant clauses with unsatisfiable core extraction.

Algorithm 2: Characterizing f_v^{-1} .

-
- 1) $F_E^v = \text{SIMPLIFY}(F_E, v)$
 - 2) $SA_v = \{\}$
 - 3) while ($F_E^v \wedge v$ is satisfiable) {
 - 4) Assume A is a satisfying assignment
 - 5) $A_{BFL} = \text{BFL_UNSAT}(F_E^v, A, v)$
 - 6) $A_{XOR} = \text{XORMIN}(F_E^v, A_{BFL}, v)$
 - 7) $SA_v = SA_v \cup \{A_{XOR}\}$
 - 8) $F_E^v = F_E^v \wedge bcls_{A_{XOR}}$
 - 9) }
 - 10) Characterizing f_v^{-1} as (16)
-

3) The function *BFL* in Algorithm 1 can remove at most 1 irrelevant variable with each SAT solving. Our improved version *BFL_UNSAT* can remove multiple irrelevant variables with every SAT solving. Thus, the number of unnecessary and expensive SAT solving is significantly reduced.

Our new algorithm to characterize f_v^{-1} is presented below. Its structure is very similar to the function *ALLSAT* in Algorithm 1, with our improvements in boldface.

The details of function *SIMPLIFY*, *BFL_UNSAT*, and *XORMIN* are described in the following sections.

C. Simplifying Formula by Extracting Unsatisfiable Core

Intuitively, F_E contains all the clauses necessary to uniquely determine the value of all variables in I_{var} . But when characterizing f_v^{-1} for a particular $v \in I_{var}$, we only need the set of clauses F_E^v that is necessary to uniquely determine the value of v . This clause set F_E^v must be a subset of F_E , and in most cases, it is much smaller than F_E , as shown in experimental results.

So we propose the function *SIMPLIFY*(F_E, v) to simplify F_E to F_E^v for every particular v .

- 1) The first step is to extract the unsatisfiable core F_E^{UNSAT} from (17) with the depth-first approach proposed by Zhang *et al.* [10]

$$F_E \wedge F_E' \wedge \bigwedge_{u \in O_{var}} u \equiv u' \wedge v \neq v'. \quad (17)$$

The unsatisfiability of this formula will be proven in Theorem 1 below.

- 2) The second step is to intersect the clause set of F_E and F_E^{UNSAT} to get formula F_E^v

$$F_E^v = F_E \cap F_E^{UNSAT}. \quad (18)$$

We first need to prove the following.

Theorem 1: Formula (17) is unsatisfiable

Proof: We can rewrite the unsatisfiable (14) by moving $\bigvee_{v \in I_{var}}$ to the outmost layer

$$\text{Formula}(14) \Rightarrow \bigwedge_{u \in O_{var}} u \equiv u' \wedge \bigvee_{v \in I_{var}} \left\{ \begin{array}{l} F_E \wedge F_E' \wedge \\ v \neq v' \end{array} \right\} \Rightarrow \text{Formula}(17)$$

According to this rewritten result, if for any v , (17) is satisfiable, then the unsatisfiable (14) will be satisfiable. This contradiction concludes the proof. ■

Furthermore, to replace F_E with F_E^v , we must make sure that $F_E \wedge v$ and $F_E^v \wedge v$ have the same set of satisfying assignments on the variable set O_{var} , which will be enumerated by Algorithm 2.

Theorem 2: $F_E \wedge v$ and $F_E^v \wedge v$ have the same set of satisfying assignments on O_{var}

Proof: On one hand, if A is a satisfying assignment of $F_E \wedge v$, then A is also a satisfying assignment of $F_E^v \wedge v$, because the clause set of $F_E^v \wedge v$ is a subset of $F_E \wedge v$.

On the other hand, assume that A is a satisfying assignment of $F_E^v \wedge v$. According to (17) and (18), the following formula is also unsatisfiable:

$$F_E^v \wedge F_E' \wedge \bigwedge_{u \in O_{var}} u \equiv u' \wedge v \neq v'$$

because it is a super set of the unsatisfiable core F_E^{UNSAT} of (17). This formula means that, no matter what value we assign to O_{var} , we can not make F_E^v and F_E to force different values on v . Thus, A is also a satisfying assignment of $F_E \wedge v$.

Thus, this theorem is proven. ■

So now, we can be sure that it is safe to replace F_E with F_E^v to characterize f_v^{-1} . And we will also show in experimental results that such replacing can significantly reduce the size of F_E and run-time overhead.

D. Minimizing Satisfying Assignments by Extracting Unsatisfiable Cores

In Algorithm 1, line 4, *BFL* [14] is used to remove those variables irrelevant to forcing v to 1. According to the implementation of *BFL* in line 11 of Algorithm 1, every $u \in U$ is tested one by one, and if the formula in line 12 is unsatisfiable, u will be removed from A .

That is to say, every unsatisfiability test can remove at most one u . The more u removed, the more difficult it is to test unsatisfiability.

So the key to reduce the run-time overhead of *BFL* is to remove more than one u with every unsatisfiability test. We will achieve this goal through two steps.

- 1) In the first step, computing unsatisfiable core F^{US} of $F \wedge \neg v \wedge A|_{U-\{u\}}$ with the depth-first approach proposed by Zhang *et al.* [10].
- 2) In the second step, computing a new A by intersecting the clause set of $A|_{U-\{u\}}$ and F^{US} .

The implementation of this improved *BFL* is shown in Algorithm 3.

Its correctness is proven below.

Theorem 3: After *BFL_UNSAT* is finished, $F \wedge \neg v \wedge A$ is unsatisfiable.

Proof: We need to prove by induction on the foreach statement in line 2 of Algorithm 3.

For the base case, according to line 5 of Algorithm 2, which call *BFL_UNSAT*, we know that A is a satisfying assignment of $F_E^v \wedge v$. Again according to Theorem 1 and

Algorithm 3 : Improved *BFL* based on extracting unsatisfiable cores.

```

1) BFL_UNSAT( $F, v, U, A$ ) {
2)   foreach  $u \in U$ 
3)     if( $F \wedge \neg v \wedge A|_{U-\{u\}}$  is unsatisfiable) {
4)       Assume that  $F^{US}$  is unsatisfiable core of  $F \wedge \neg v \wedge A|_{U-\{u\}}$ 
5)        $A = A|_{U-\{u\}} \cap F^{US}$ 
6)     }
7)   return  $A$ 
8)}
```

2, v cannot be 0 under assignment A . So $F \wedge \neg v \wedge A$ is unsatisfiable when Algorithm 3 reaches the foreach statement in line 2 for the first time.

For the induction step, assume that when Algorithm 3 reaches the foreach statement in line 2, $F \wedge \neg v \wedge A$ is unsatisfiable. Then the if condition in line 3 may be:

```

1) False: in this situation,  $A$  will not be changed, thus  $F \wedge \neg v \wedge A$  is still unsatisfiable;
2) True: in this situation,  $F^{US}$  is an unsatisfiable core of  $F \wedge \neg v \wedge A|_{U-\{u\}}$ , then  $F \wedge \neg v \wedge (A|_{U-\{u\}} \cap F^{US})$  is also unsatisfiable, because its clause set is a super set of  $F^{US}$ . By assigning  $A|_{U-\{u\}} \cap F^{US}$  back to  $A$  in line 5 of Algorithm 3, we again get unsatisfiable formula  $F \wedge \neg v \wedge A$ .
```

Thus, this theorem is proven. ■

According to theorem 3, A returned by *BFL_UNSAT* is also a set of necessary variable assignments that forces v to 1. Thus *BFL* can be replaced by *BFL_UNSAT* safely.

We will also show in experimental results that the function *BFL-UNSAT* will significantly reduce the number of SAT solving.

E. Minimizing Satisfying Assignments by Discovering XOR Gates

According to Algorithm 3, the assignment A returned by *BFL_UNSAT* is a minimal assignment, which means removing any variable from A will make it unable to force v to 1.

For A to cover more satisfying assignments, we need to find a more efficient approach to merge satisfying assignments.

XOR gates are used extensively in communication and arithmetic circuits. According to Section II-B and Fig. 1(b), the two satisfying assignments of the XOR gate cannot be merged by removing input variables.

But for a larger Boolean function such as f_v^{-1} that MAY contain XOR gate $z = v_1 \oplus v_2$, we can first check whether this XOR gate actually exists, and then merge A and $A|_{V-\{v_1, v_2\}}|^{v_1 \rightarrow \neg A(v_1)}|^{v_2 \rightarrow \neg A(v_2)}$ by replacing v_1 and v_2 with z in A .

Intuitively, for a satisfying assignment A that can force v to 1, assume its domain is $U \subseteq O_{var}$, for certain $v_1, v_2 \in U$, we can invert the value of v_1 and v_2 in A

$$A_{\bar{v}_1, \bar{v}_2} = A|_{V-\{v_1, v_2\}}|^{v_1 \rightarrow \neg A(v_1)}|^{v_2 \rightarrow \neg A(v_2)}. \quad (19)$$

Algorithm 4 : *XORMIN*(F_E, A, v).

```

1)  $G = \{\}$ 
2) do {
3)    $G_{new} = \{\}$  // the set of newly discovered XOR
4)   foreach  $v_1, v_2 \in O_{var}$  {
5)     if((20) is unsatisfiable){
6)        $G_{new} = G_{new} \cup \{z = v_1 \oplus v_2\}$ 
7)        $A = A|_{O_{var}-\{v_1, v_2\}}|^{z \rightarrow A(v_1) \oplus A(v_2)}$ 
8)        $O_{var} = O_{var} \cup \{z\} - \{v_1, v_2\}$ 
9)        $F_E = F_E \wedge bcls_A$ 
10)       $F_E = F_E \wedge \bigwedge_{\{z=v_1 \oplus v_2\} \in G_{new}} \{z \equiv v_1 \oplus v_2\}$ 
11)    }
12)  }
13)   $G = G \cup G_{new}$ 
14)} while( $G_{new} \neq \{\}$ )
15) return  $A$ 
```

We then test whether $A_{\bar{v}_1, \bar{v}_2}$ can also force v to 1, by checking the unsatisfiability of the following formula:

$$F_E \wedge \neg v \wedge A_{\bar{v}_1, \bar{v}_2}. \quad (20)$$

The unsatisfiability of (20) means that $A_{\bar{v}_1, \bar{v}_2}$ can also force v to 1.

Thus, A and $A_{\bar{v}_1, \bar{v}_2}$, which cannot be merged by BFL, can be merged into

$$A_z = A|_{O_{var}-\{v_1, v_2\}}|^{z \rightarrow A(v_1) \oplus A(v_2)} \quad (21)$$

with the help of a newly discovered XOR gate that takes v_1 and v_2 as input, and output z

$$z = v_1 \oplus v_2. \quad (22)$$

Now, the support set of f_v^{-1} and f^{-1} will change from O_{var} to $O_{var} \cup \{z\}$.

If we repeatedly check the unsatisfiability of (20) for other pairs of v_1 and v_2 , we can discover all hidden XOR gates and merge their satisfying assignments. All such XOR gates will be used in Section V-B to build E^{-1} .

With the above discussion, we describe *XORMIN* in Algorithm 4.

In line 1, G is an empty set that will be used to hold all XOR gates discovered by this algorithm.

In line 2, the do-while statement will repeatedly discover new XOR gates, until no more XOR gates can be discovered.

In line 4, the foreach statement will enumerate each pair of $v_1, v_2 \in O_{var}$, and line 5 will test if there is a XOR gate between v_1 and v_2 .

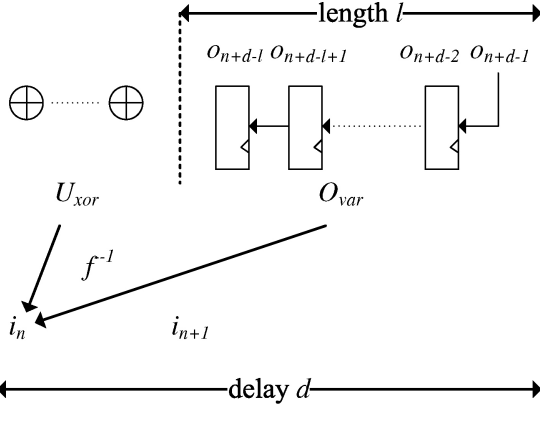
Line 6 will record the newly discovered XOR gate.

Line 7 will compute the newly merged satisfying assignment A , and line 8 will modify the support set of f_v^{-1} .

V. BUILDING CIRCUIT E^{-1} FROM f^{-1}

A. Instantiating Register Bank

The function $f^{-1} : O^l \rightarrow I$ is a Boolean function that takes the finite length sequence $\langle o_{n+d-l}, \dots, o_{n+d-1} \rangle$ as input, and computes i_n .

Fig. 4. Circuit structure of E^{-1} .

So while building circuit E^{-1} , as shown in the right-top side of Fig. 4, we need to instantiate $l - 1$ banks of registers to store the subsequence $\langle o_{n+d-l}, \dots, o_{n+d-2} \rangle$, and connect the output of o_i to the input of o_{i-1} .

B. Instantiating Discovered XOR Gates

According to Section IV-E, assume the set of all XOR gates discovered by function $XORMIN$ is G . Then the output variable set of these XOR gates is as follows:

$$U_{xor} = \{z | \{z = v_1 \oplus v_2\} \in G\}. \quad (23)$$

Then the support set of Boolean function f^{-1} will be changed from O_{var} to $O_{var} \cup U_{xor}$.

So we need to instantiate all XOR gates discovered by the $XORMIN$ function in the generated netlist, as shown in the left-top side of Fig. 4.

C. Generating Verilog Source Code for E^{-1}

Assume SA_v is the set of all satisfying assignments that can force $v \in I_{var}$ to 1. Then the always statement that assigns value to v is shown below:

- ```

1) always@(list of all variables in $O_{var} \cup U_{xor}$) begin;
2) if(condition1 || ... || conditionn);
3) v <= 1'b1;
4) else;
5) v <= 1'b0;
6) end.
```

The condition<sub>1</sub> to condition<sub>n</sub> in line 2 corresponds to every satisfying assignments in  $SA_v$ .

## VI. EXPERIMENTAL RESULTS

We implement our algorithm in zchaff [3], and run it on a PC with a 2.4GHz AMD Athlon 64 X2 dual core processor, 6GB memory and CentOS 5.2 linux operating system.

All related programs and data files can be downloaded from <http://www.ssympub.org>.

TABLE I  
INFORMATION OF BENCHMARKS

|                                    | XGXS | XFI | Scrambler | PCIE | T2 Ethernet |
|------------------------------------|------|-----|-----------|------|-------------|
| Line Number of Verilog Source Code | 214  | 466 | 24        | 1139 | 1073        |
| #Regs                              | 15   | 135 | 58        | 22   | 48          |
| Data Path Width                    | 8    | 64  | 66        | 10   | 10          |

TABLE II  
RESULTS OF CHECKING THE PARAMETERIZED COMPLEMENTARY CONDITION

|              | XGXS | XFI   | Scrambler | PCIE  | T2 Ethernet |
|--------------|------|-------|-----------|-------|-------------|
| Run-time (s) | 0.51 | 71.60 | 2.51      | 32.74 | 44.48       |
| $d$          | 1    | 0     | 0         | 2     | 4           |
| $p$          | 0    | 3     | 1         | 1     | 0           |
| $l$          | 1    | 2     | 2         | 1     | 1           |

#### A. Benchmarks

Table I shows some information of the following benchmarks.

- 1) The first benchmark is a XGXS encoder compliant to clause 48 of IEEE-802.3ae 2002 standard [22].
- 2) The second benchmark is a XFI encoder compliant to clause 49 of the same IEEE standard.
- 3) The third benchmark is a 66-bit scrambler used to ensure that a data sequence has sufficiently many 0-1 transitions, so that it can run through high-speed noisy serial transmission channel.
- 4) The fourth benchmark is a PCIE physical coding module.
- 5) The fifth benchmark is the Ethernet module of Sun's OpenSparc T2 processor.

#### B. Writing Assertions

To write assertions for ruling out invalid input letters, we refer to the following documentations, and find out the valid letter patterns easily.

- 1) For the XGXS and T2 ethernet encoders, Tables 48-2, 48-3 and 48-4 of IEEE-802.3ae 2002 standard [22] describe the pattern of valid letters.
- 2) For the XFI encoder and scrambler, Fig. 49-7 and Table 49-1 of IEEE-802.3ae 2002 standard [22] describe the pattern of valid letters.
- 3) For the PCIE physical coding module, Table 4-1 of PCI Express Base Specification [23] describe the pattern of valid letters.

#### C. Result of Checking the Parameterized Complementary Condition

Table II shows the run-time of checking the parameterized complementary condition on these circuits, and the discovered proper values of parameters.



TABLE III  
RUN-TIME OF BUILDING COMPLEMENTARY CIRCUITS

|          |           | XGXS  | XFI      | Scrambler | PCIE     | T2 Ethernet |
|----------|-----------|-------|----------|-----------|----------|-------------|
| BFL Only | Time (s)  | 32.67 | Time Out | 8.56      | Time Out | Time Out    |
| BFL +    | Time (s)  | 1.52  | 2939.47  | 11.97     | 47.55    | 36.64       |
| XORMIN   | $ F_E $   | 25470 | 5084496  | 499200    | 52209    | 459204      |
|          | #SAT      | 984   | 137216   | 8320      | 528      | 1032        |
| BFL+     | Time (s)  | 1.08  | 752.83   | 1.84      | 0.82     | 27.08       |
| XORMIN   | $ F_E^v $ | 6694  | 188717   | 4807      | 6635     | 51204       |
| +UNSAT   | #SAT      | 480   | 16828    | 256       | 243      | 538         |

TABLE IV  
COMPARING DECODER AREA

|                                 | XGXS | XFI   | Scrambler | PCIE | T2 Ethernet |
|---------------------------------|------|-------|-----------|------|-------------|
| Hand-written decoders           | 913  | 4886  | 1514      | 952  | 2225        |
| Decoders built by Shen [6]      | 667  | 15269 | 1302      | 344  | 661         |
| Decoders built by our algorithm | 652  | 16659 | 1302      | 345  | 569         |

#### D. Improvement in Run-Time Overhead

Table III compares the following three statistics between the BFL algorithm [14], BFL+XORMIN proposed in our previous work [6], and BFL+XORMIN+UNSAT proposed by this paper.

- 1) The three *time* rows compare the run-time overhead of building complementary circuits. Obviously, our approach is more than one order of magnitude faster than BFL only approach, and three times faster than our previous work [6].
- 2)  $|F_E|$  and  $|F_E^v|$  compare the total size of  $F_E$  and  $F_E^v$  passed to ALLSAT algorithm, in which  $F_E^v$  is the result of simplifying  $F_E$  with *SIMPLIFY*. Obviously,  $|F_E^v|$  is significantly smaller than  $|F_E|$ .
- 3) The two #SAT rows compare the total number of SAT solving invoked by *BFL* and *BFL\_UNSAT*. Obviously, the number of SAT solving is reduced significantly.

#### E. Comparing Decoder Area

Table IV compares the circuit area of hand-written decoders, decoders built by our previous work [6], and decoders built by our algorithm. We synthesize these decoders with LSI10K technology library from Synopsys DesignCompiler.

From Table IV, we observe the following.

- 1) Except for the most complex XFI, our synthesis results are more compact than hand-written decoders. However, this comparison is unfair when hand-written decoders include error-detection and reporting functions, or other additions not required by the main functionality.
- 2) For the XFI case, our circuit area is about three times larger than that of hand-written decoders. This means we need to improve the circuit area in the future work.
- 3) There is no significant difference in area between our algorithm and our previous work [6].

TABLE V  
COMPARING CRITICAL PATH LATENCIES IN NANOSECOND

|                                 | XGXS  | XFI   | Scrambler | PCIE  | T2 Ethernet |
|---------------------------------|-------|-------|-----------|-------|-------------|
| Hand-written decoders           | 12.33 | 46.65 | 6.54      | 19.03 | 23.36       |
| Decoders built by our algorithm | 13.23 | 58.73 | 6.54      | 8.07  | 17.07       |

#### F. Comparing Decoder Timing

Table V compares the critical path latencies of hand-written decoders, and decoders built by our algorithm. Their synthesis settings are same as Section VI-E.

According to Table V, for most circuits, the critical path latencies of the decoders built by our algorithm are better than, or at least close to that of those hand-written decoders.

The only exception is XFI, the latency of the decoder built by our algorithm is significantly larger than that of hand-written decoder.

One interesting issue that is not shown in Table V is, all these critical paths start from registers, and end at output ports.

## VII. RELATED WORKS

### A. Unsatisfiable Core Extraction

There are many classes of unsatisfiable core extraction algorithms.

The first class of algorithms extract small but not necessary minimal unsatisfiable cores. Such algorithms are often used to verify the unsatisfiability of certain instance. Bruni and Sassano [24] proposes a constructive approach, which starts from an initial clause set and iteratively adds new clauses into it, until the clause set becomes unsatisfiable. Goldberg and Novikov [9] and Zhang and Malik [10] record the resolution graph that generates conflict clauses during a SAT solving, and trace backward along this graph to find out those clauses that cause the unsatisfiability. Gershman *et al.* [25] tries to find the internal dominator node  $d$  in the resolution tree that consumes large number of original clauses, and then tries to prove  $d$  without these clauses. When a proof is completed, those clauses can be removed.

The second class of algorithms try to find minimal unsatisfiable cores, whose subsets are all satisfiable. Oh *et al.* [26] adds a clause-selector variable to each clause of the CNF formula, and uses a DPLL-like procedure to find a clause subset with minimal number of clause-selector. Huang [27] also uses the clause-selector variable approach, but he converts the problem of finding minimal unsatisfiable cores to model counting problem and then uses BDD to solve it. Dershowitz *et al.* [28] first constructs a resolution graph from the unsatisfiable core, and then tests every initial clause in this graph to make sure if it is in a minimal core. Grégoire *et al.* [29] uses a local search approach to search for minimal cores. This approach uses a scoring heuristic that records the common literals between different clauses, and removes those clauses that are not likely in a minimal core. Liffiton and Sakallah [30] proposes a preprocessing step to search for autarkies, which are partial assignments to the variables of a

Boolean CNF formula that satisfy every clause containing an assigned variable.

The third class of algorithms try to find the smallest or minimum cores. Lynce and Silva [31] uses a clause-selector approach similar to AMUSE [26] mentioned above. It either searches for satisfy and then backtracks to the most recent clause selector variable set to true, or searches for conflicts and then backtracks to a clause selector variable, which means an unsatisfiable core is found. At this point, it records the size of core and continues searching for smaller cores. Mneimneh *et al.* [32] and Liffiton *et al.* [33] use a branch-and-bound algorithm that utilizes iterative MAXSAT solutions to generate lower and upper bounds on the size of the smallest unsatisfiable core, and branches on specific sub-formulas to find it. Zhang *et al.* [34] uses a greedy genetic algorithm to find the smallest unsatisfiable core.

The fourth class of algorithms try to find out all minimal cores, so that the complete information about infeasibility can be obtained. This technique is particularly useful in refining abstract model of model checking. Bailey and Stuckey [35] and Liffiton and Sakallah [36] propose to compute minimal unsatisfiable cores by first computing minimal correction sub-sets.

### B. Satisfying Assignments Enumeration

The existing ALLSAT algorithms all try to enlarge the complete satisfying assignments, so that a large state set that contains more complete satisfying assignments can be obtained.

The first approach of this kind is proposed by K. L. McMillan [13]. He constructs an alternative implication graph in SAT solver, which records the reasoning relation that leads to the assignment of a particular object variable. All variables outside this graph can be ruled out from the complete assignment. Ravi *et al.* [14] and Chauhan *et al.* [18] remove those variables whose absence can not make  $obj \equiv 0$  satisfiable one by one. Shen *et al.* [16] and Jin *et al.* [12], [15] use a conflict analysis based approach to remove multiple irrelevant variables in one SAT run. Grumberg *et al.* [17] divides the variable set into an important subset and an unimportant subset. Variables in the important subset have higher decision priority than those unimportant ones. Thus, the important subset forms a search tree, with each leaf being another search tree for the unimportant set. Cofactoring [19] qualifies out unimportant variables by setting them to constant value returned by the SAT solver.

### C. AND–XOR Logic Synthesis

The classical logic synthesis works on AND–OR network. Its core is the two-level logic minimization algorithm, which aims to find a smaller sum-of-products expression for Boolean function  $f$ .

Three best-known two-level logic minimization algorithms are Quine-McCluskey [37], Scherzo [38], and Espresso-II [39].

Just like the current ALLSAT that can not deal with XOR-intensive circuits efficiently, the classical logic synthesis also has the same problem. Thus, many researchers propose synthesis algorithms that target XOR-intensive circuits.

One research direction focuses on extending the classical two-level AND–OR minimization to two-level AND–XOR network [40], [41]. These works normally describe circuits with the most general exclusive sum of product expressions.

Another line of research relies on Reed–Muller expansion [42], and one of its most used variant is fixed polarity Reed–Muller form (FPRM) given by Davio and Deschamps [43], in which a variable can have either positive or negative polarity. Some related works that rely on FPRM are [44]–[46].

A recent paper [47] aims to decompose a function into results of XORing several sub-functions. So, this approach can be seen as a generalized form of our XORMIN algorithm, which discovers XOR between several variables instead of functions. It is possible that the circuit area of our generated complementary circuits can be improved by applying this approach to characterize the complementary function  $f^{-1}$ .

## VIII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a fully automatic approach that synthesizes complementary circuits for communication applications. According to experimental results, our approach can synthesize correct complementary circuits for many complex circuits, including but not limited to PCIE and Ethernet.

One direction for future work is to improve the circuit area of generated complementary circuit  $E^{-1}$ . According to Section IV, the algorithm XORMIN discovers XOR gates between variables. It is possible that the same XOR gates can be discovered many times in different iterations. So if we can decompose the logic formula into multiple functions before applying the XORMIN algorithm, just like [47], then we can avoid redundant XOR gates and reduce the circuit area.

A recent ICCAD'09 paper by Jiang [48] proposed to use Craig interpolation to characterize a Boolean function from a Boolean relation. We will integrate this algorithm into our framework in the future to reduce the run-time overhead.

Another direction for future work is to deal with circuits with memory array and multiple clocks, so that more complex communication mechanisms, such as data link layer and transaction layer, can be dealt with by our approach.

Yet another direction for future work is to find the termination point for checking the parameterized complementary condition. We are also considering a debug approach that can find the reason for not terminating within a reasonable time bound.

One issue that was not addressed by this paper is timing. The generated complementary circuit has a two-level logic structure. It seems that the well-known retiming algorithm [49] cannot be applied to such circuits to improve timing. But we must notice that, the two-level structure in our generated complementary circuits are built by AND and OR gates with large fanin, which are not available in realistic target cell libraries. When these complementary circuits are mapped to a target cell library with a logic synthesizer, these large fanin gates will be mapped to a chain of real gates with smaller fanin. In these mapped circuits, the retiming algorithm can be applied easily to improve timing.

## ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers for their hard work.

## REFERENCES

- [1] C. Kozup. (2008). Is 802.11n right for you? [Online]. Available: [http://blogs.cisco.com/wireless/comments/is\\_80211n\\_right\\_for\\_you/](http://blogs.cisco.com/wireless/comments/is_80211n_right_for_you/)
- [2] S. J. Dubner. (2008). What are the lessons of the blu-ray/hd-dvd battle? A freakonomics quorum [Online]. Available: <http://freakonomics.blogs.nytimes.com/2008/03/04/what-are-the-lessons-of-the-blu-rayhd-dvd-battle-a-freakonomics-quorum>
- [3] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. IEEE Int. DAC*, Jun. 2001, pp. 530–535.
- [4] *Xilinx Core Generator System*. (2008). Xilinx, San Jose, CA [Online]. Available: <http://www.xilinx.com/tools/coregen.htm>
- [5] *Designware Library*. (2008). Synopsys, Inc., Mountain View, CA [Online]. Available: <http://www.synopsys.com/IP/DESIGNWARE/Pages/default.aspx>
- [6] S. Shen, J. Zhang, Y. Qin, and S. Li, "Synthesizing complementary circuits automatically," in *Proc. IEEE ICCAD*, Nov. 2009, pp. 381–388.
- [7] E. I. Goldberg and Y. Novikov, "Berkmin: A fast and robust SAT-solver," in *Proc. IEEE DATE Conf. Exposit.*, Mar. 2002, pp. 142–149.
- [8] N. Eén and N. Sörensson, "Extensible SAT-solver," in *Proc. Int. Conf. SAT*, May 2003, pp. 502–518.
- [9] E. I. Goldberg and Y. Novikov, "Verification of proofs of unsatisfiability for CNF formulas," in *Proc. IEEE DATE Conf. Exposit.*, Mar. 2003, pp. 10886–10891.
- [10] L. Zhang and S. Malik, "Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications," in *Proc. IEEE DATE Conf. Exposit.*, Mar. 2003, pp. 10880–10885.
- [11] S. Sapra, M. Theobald, and E. M. Clarke, "SAT-based algorithms for logic minimization," in *Proc. IEEE ICCD*, Oct. 2003, pp. 510–519.
- [12] H. Jin and F. Somenzi, "Prime clauses for fast enumeration of satisfying assignments to boolean circuits," in *Proc. IEEE Int. DAC*, Jun. 2005, pp. 750–753.
- [13] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking," in *Proc. Int. CAV*, Jul. 2002, pp. 250–264.
- [14] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking," in *Proc. Int. Conf. TACAS*, Mar. 2004, pp. 31–45.
- [15] H. Jin, H. Han, and F. Somenzi, "Efficient conflict analysis for finding all satisfying assignments of a Boolean circuit," in *Proc. Int. Conf. TACAS*, Apr. 2005, pp. 287–300.
- [16] S. Shen, Y. Qin, and S. Li, "Minimizing counterexample with unit core extraction and incremental SAT," in *Proc. Int. Conf. VMAI*, Jan. 2005, pp. 298–312.
- [17] O. Grumberg, A. Schuster, and A. Yadgar, "Memory efficient all-solutions SAT solver and its application for reachability analysis," in *Proc. Int. Conf. FMCAD*, Nov. 2004, pp. 275–289.
- [18] P. Chauhan, E. M. Clarke, and D. Kroening, "A SAT-based algorithm for reparameterization in symbolic simulation," in *Proc. IEEE Int. DAC*, Jun. 2004, pp. 524–529.
- [19] M. K. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *Proc. IEEE ICCAD*, Nov. 2004, pp. 510–517.
- [20] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. IEEE Int. DAC*, Jun. 1999, pp. 317–320.
- [21] M. G. H., "A method for synthesizing sequential circuits," *Bell Syst. Tech. J.*, vol. 34, pp. 1045–1079, Jan. 1955.
- [22] *IEEE Standard for Information Technology Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks Specific Requirements. Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation*, IEEE Std. 802.3, 2002.
- [23] *PCI Express Base Specification Revision 2.1*, PCI Special Interest Group Std., 2010.
- [24] R. Bruni and A. Sassano, "Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae," *Electron. Notes Discrete Math.*, vol. 9, pp. 162–173, Jun. 2001.
- [25] R. Gershman, M. Koifman, and O. Strichman, "Deriving small unsatisfiable cores with dominators," in *Proc. Int. Conf. CAV*, Aug. 2006, pp. 109–122.
- [26] Y. Oh, M. N. Mneimneh, Z. S. Andraus, K. A. Sakallah, and I. L. Markov, "Amuse: A minimally-unsatisfiable subformula extractor," in *Proc. IEEE Int. DAC*, Jun. 2004, pp. 518–523.
- [27] J. Huang, "Mup: A minimal unsatisfiability prover," in *Proc. IEEE ASPDAC*, Jan. 2005, pp. 432–437.
- [28] N. Dershowitz, Z. Hanna, and A. Nadel, "A scalable algorithm for minimal unsatisfiable core extraction," in *Proc. Int. Conf. SAT*, Aug. 2006, pp. 36–41.
- [29] É. Grégoire and B. Mazure and C. Piette, "Local-search extraction of muses," *Constraints*, vol. 12, no. 3, pp. 325–344, Sep. 2007.
- [30] M. H. Liffiton and K. A. Sakallah, "Searching for autarkies to trim unsatisfiable clause sets," in *Proc. Int. Conf. SAT*, May 2008, pp. 182–195.
- [31] I. Lynce and J. P. Marques Silva, "On computing minimum unsatisfiable cores," in *Proc. Int. Conf. SAT*, May 2004, pp. 305–310.
- [32] M. N. Mneimneh and I. Lynce and Z. S. Andraus and J. P. M. Silva and K. A. Sakallah, "A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas," in *Proc. Int. Conf. SAT*, Jun. 2005, pp. 467–474.
- [33] M. N. Mneimneh and M. N. Mneimneh and I. Lynce and Z. S. Andraus and J. P. Marques Silva and K. A. Sakallah, "A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas," *Constraints*, vol. 14, no. 4, pp. 415–442, Dec. 2009.
- [34] J. Zhang, S. Li, and S. Shen, "Extracting minimum unsatisfiable cores with a greedy genetic algorithm," in *Proc. Aust. Joint Conf. Artif. Intell. Adv. Artif. Intell.*, Dec. 2006, pp. 847–856.
- [35] J. Bailey and P. J. Stuckey, "Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization," in *Proc. Int. Symp. PADL*, Jan. 2005, pp. 174–186.
- [36] M. H. Liffiton and K. A. Sakallah, "Algorithms for computing minimal unsatisfiable subsets of constraints," *J. Automat. Reason.*, vol. 40, no. 1, pp. 1–33, Jan. 2008.
- [37] E. J. McCluskey, *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
- [38] O. Coudert, "On solving covering problems," in *Proc. IEEE Int. Design Automat. Conf.*, Jun. 1996, pp. 197–202.
- [39] R. L. Rudell and A. L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 6, no. 5, pp. 727–750, Sep. 1987.
- [40] P. W. Besslich and M. Riege, "An efficient program for logic synthesis of mod-2 sum expressions," in *Proc. Euro ASIC*, 1991.
- [41] T. Sasao, "And-exor expressions and their optimization," in *Logic Synthesis and Optimization*, T. Sasao, Ed. Boston, MA: Kluwer, 1993.
- [42] I. S. Reed, "A class of multiple-error-correcting codes and their decoding scheme," *IRE Trans. Info. Theory*, vol. 6, pp. 48–49, Apr. 1954.
- [43] M. Davio, *Discrete and switching Functions*. New York: George and McGraw-Hill, 1978.
- [44] A. Sarabi and M. A. Perkowski, "Fast exact and quasi-minimal minimization of highly testable fixed-polarity and XOR canonical networks," in *Proc. IEEE Int. DAC*, Jun. 1992, pp. 30–35.
- [45] R. Drechsler, B. Becker, and M. Theobald, "Fast ofdd based minimization of fixed polarity Reed–Muller expressions," in *Proc. IEEE EURO-DAC*, Sep. 1994, pp. 2–7.
- [46] U. Narayanan and C. L. Liu, "Low power logic synthesis for xor based circuits," in *Proc. IEEE ICCAD*, Nov. 1997, pp. 570–574.
- [47] T. S. Czajkowski and S. D. Brown, "Functionally linear decomposition and synthesis of logic circuits for fpgas," in *Proc. IEEE Int. DAC*, Jun. 2008, pp. 18–23.
- [48] J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung, "Interpolating functions from large Boolean relations," in *Proc. IEEE ICCAD*, Jun. 2009, pp. 779–784.
- [49] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," in *Proc. IEEE Annu. FOCS*, Oct. 1981, pp. 23–36.



**ShengYu Shen** (M'10) received the B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, Hunan, China, in 1997, 2000, and 2005, respectively. He is an Assistant Professor with the School of Computer, National University of Defense Technology. His current research interests include formal method with emphasis on counterexample explanation and logic synthesis.



**Ying Qin** received the B.S. and M.S. degrees in computer science from the National University of Defense Technology, Changsha, Hunan, China, in 1997 and 2000, respectively.

She is an Assistant Professor with the School of Computer, National University of Defense Technology. Her current research interests include software engineering with emphasis on operating system implementation.



**JianMin Zhang** received the B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, Hunan, China, in 2001, 2003, and 2009, respectively.

He is a Lecturer with the School of Computer, National University of Defense Technology. His current research interests include formal method with emphasis on unsatisfiable core extraction.

**KeFei Wang** received the B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, Hunan, China, in 1994, 1997, and 2001, respectively.

He is an Assistant Professor with the School of Computer, National University of Defense Technology. His current research interests include high-performance computing with emphasis on high speed interconnection.



**SiKun Li** received the B.S. degree in computer science from the National University of Defense Technology, Changsha, Hunan, China, in 1968.

He is a Professor with the School of Computer, National University of Defense Technology. His current research interests include computer-aided design methods for very large scale integrated chips.

**LiQuan Xiao** received the B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, Hunan, China, in 1991, 1994, and 1997, respectively.

He is a Professor with the School of Computer, National University of Defense Technology. His current research interests include high-performance computing with emphasis on high speed interconnection.



## AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

987  
988 AQ:1= Please verify first name of all the authors.

989 AQ:2= Please provide expansion of "PCIE."

990 AQ:3= Please provide expansion of "CRC."

991 AQ:4= Please provide page range in Ref. [46]

992 END OF ALL QUERIES

