Synthesizing Complementary Circuits Automatically

ShengYu Shen, JianMin Zhang, Ying Qin, and SiKun Li
School of Computer Science, National University of Defense Technology
410073,ChangSha,China
Email: {syshen,jmzhang,qy123,skli}@nudt.edu.cn

***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 ease 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*—Synthesis,Complementary circuit,All solution SAT, Discovering XOR gates, Extracting unsatisfiable core.**

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 blue ray [2]. In such highly competitive markets, designing correct chip 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, and so on.

In order to ease 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 there exists a valuation 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 **ALL-SAT**) based on discovering XOR gates and extracting unsatisfiable cores.

We implement our algorithm on zchaff [5], and run it on several complex encoder circuits from industrial projects, including PCIE and 10G Ethernet. **Some of these circuits have very wide datapath, up to 64 bits**. It has turned out that all these complementary circuits can be built within 1000 seconds. And all these experimental results and related programs can be downloaded from http://www.ssypub.org.

One issue need to be clarified before we start presentation of our idea. There are two classes of complementary circuit pair, each with its own character and design methodology that is totally different from the other:

1) **Standard datapath-intensive circuits: Such circuits often work as digital signal processing components, including : Fast Fourier Transform (FFT), Discrete Cosine Transform(DCT), and so on. These circuits and their complementary circuits usually have standard and highly optimized implementations from varies foundries and IP venders, such as Xilinx core generator [3] and Synopsys designware library [4]. So our algorithm aren't design for such circuits.**
2) **Non-standard and control-intensive citcuits: Such circuit are often used to handle communication protocol, and typically don't have standard implementations. Our algorithm are designed for such circuits.**

Compared with our previous ICCAD'09 version, the major contribution of this paper is twofold:

1) We give a more formal and accurate description of checking parameterized complementary condition. Please refer to subsection III-C.
2) More importantly, we integrate unsatisfiable core extraction into the algorithm that characterize the Boolean function f^{-1}, to improve its run time overhead. 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 [5] ,Berkmin [6] and MiniSAT [7].

Normally, a SAT solver requires formula F to be represented in **Conjunctive Normal Form(CNF)** or **And-Inverter Graph(AIG)** formats. In this paper we only discuss 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 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 **total 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:

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

Intuitively, $A|_W$ 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:

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

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

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

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

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.

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 [8] and Zhang [9].

Although our algorithm relies on unsatisfiable core extraction algorithms, the readers do not need to dive deeply into the details of it. The only thing that need to be understood about these well known algorithms is, the result of unsatisfiable core extraction algorithms is an unsatisfiable subset of the original formula.

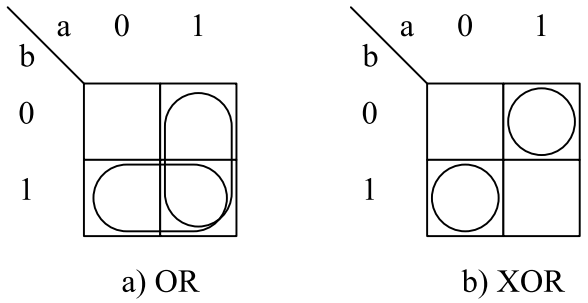

Fig. 1. Satisfying assignments for simple gates

B. All Solution SAT Solver

State-of-the-Art SAT solvers normally find only one total satisfying assignment. But many applications, such as two-level logic minimization [10], need to enumerate all satisfying assignments.

Such technologies that enumerate all satisfying assignments of a formula are called **all solution SAT(ALLSAT)**. Obviously, we can enumerate all satisfying assignments by repeatedly calling a SAT solver, and inserting the blocking clause $bclk_A$ 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** technology to remove irrelevant variables' assignments from satisfying assignment A, so that A can cover more total satisfying assignments. For example, for OR gate $z = a \vee b$ in figure 1a), its total satisfying assignments that can make $z \equiv 1$ are $\{a = 1, b = 0\}$,$\{a = 1, b = 1\}$ and $\{a = 0, b = 1\}$, which contain 6 assignments to individual variables. It's obvious that $\{a = 1, b = 0\}$ and $\{a = 1, b = 1\}$ can be merged into $\{a = 1\}$, in which b is removed. At the same time, $\{a = 1, b = 1\}$ and $\{a = 0, b = 1\}$ can also be merged into $\{b = 1\}$, in which a is removed. Therefore, these two newly-merged partial assignments contain only two assignments to individual variables, and are much more succinct than previous three total assignments.

Formally, assume that $\boldsymbol{F}$ is a formula over Boolean variable set V, $\boldsymbol{v} \in V$ 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 assignment we would like to minimize and enumerate. We can test whether $u \in U$ is irrelevant to forcing v to be 1, by testing unsatisfiability of $F \wedge \neg v \wedge A|_{U-\{u\}}$. If $F \wedge \neg v \wedge A|_{U-\{u\}}$ is unsatisfiable, then $A|_{U-\{u\}}$ can't make 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 [11]–[18] share this idea of satisfying assignments minimization. We will only present here one of them, BFL(brutal force lifting) algorithm [13]:

Algorithm 1: $\boldsymbol{ALLSAT}$ **based on** $\boldsymbol{BFL}$ **Algorithm**

```
1)  ALLSAT(F, v, U) {
2)      SA_v = {}
3)      while(F ∧ v has a satisfying assignment A) {
4)          A = BFL(F, v, U, A)
5)          SA_v = SA_v ∪ {A}
6)          F = F ∧ bcls_A
7)      }
8)      return SA_v
9)  }
10) BFL(F, v, U, A) {
11)     foreach u ∈ U
12)         if(F ∧ ¬v ∧ A|_{U−{u}} is unsatisfiable)
13)             A = A|_{U−{u}}
14)     return A
15) }
```

Line 12 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 total assignments.

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

An extreme example is a n-bits comparator that compares two n-bits variables. In this case, there are 2^n total 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)** [19], 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 $I \subseteq S$, transition relation between states $T \subseteq S \times S$, and the labeling of the states $L : S \to 2^A$ with atomic propositions set A.

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

We only present here how to check reachability in BMC, and more details can be found in [19]. 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:

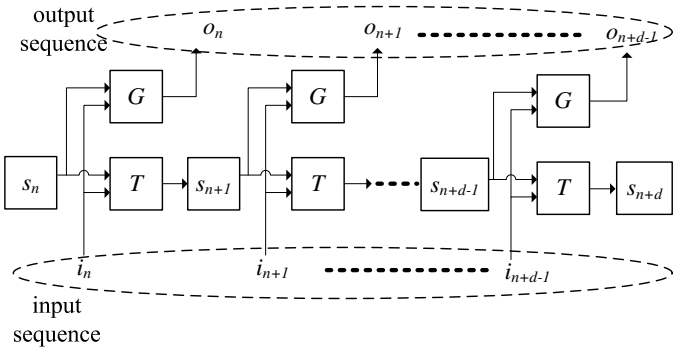

Fig. 2. Unfolding Transition Function of Mealy Finite State Machine

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

Reduce formula (2) into CNF format, and solve it with a SAT solver, then a counterexample of length b can be found if it exist.

III. Checking the Parameterized Complementary Condition

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

A. Parameterized Complementary Condition

Our algorithm cares about the input and output sequence of circuit E, so **Mealy finite state machine** [22] 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 state S
2) An initial state $s_0 \in S$
3) A finite set of letter called the input alphabet I
4) a finite set of letter called the output alphabet O
5) A state transition function $T : S \times I \to S$
6) An output function $G : S \times I \to O$

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 figure 2. This relationship can be built by unfolding the transition function T and output function G d times, as shown in formula (3).

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

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 can't take the infinite length sequence $o \in O^\omega$ as input to f^{-1}, we can only use a finite length sub-sequence of o. This

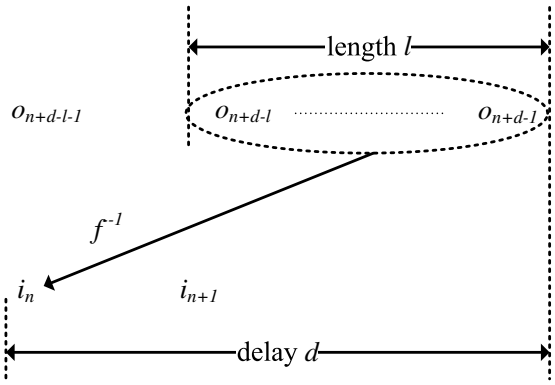

Fig. 3. f^{-1} and Parameters of o's Finite Length Subsequence

sub-sequence has two parameters, its length l and its delay d compared to i_n, as shown in figure 3.

Thus, $f^{-1} : O^l \rightarrow I$ should be a Boolean function that takes the finite length sequence $< o_{n+d-l}, \ldots, o_{n+d-1} >$ as 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 $< o_{n+d-l}, \ldots, o_{n+d-1} >$, there exists no more than one valuation of i_n that can make formula (3) satisfiable.

To test whether there exists another i_n that can make formula (3) satisfiable, we need to unfold function T and G another time:

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

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

Thus, parameterized complementary condition holds if and only if the following formula (5) is unsatisfiable.

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

In formula (5), the first line contains two unfolding of circuit E. The second line constrains their output sequences $< o_{n+d-l}, \ldots, o_{n+d-1} >$ and $< o'_{n+d-l}, \ldots, o'_{n+d-1} >$ 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 formula (5) may return two results:

1) **Satisfiable**. In this situation, for a $< o_{n+d-l}, \ldots, o_{n+d-1} >$, there exist two different i_n and i'_n that can both make formula (3) satisfiable. This means the input letter i_n can't be uniquely determined by $< o_{n+d-l}, \ldots, o_{n+d-1} >$. 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 $< o_{n+d-l}, \ldots, o_{n+d-1} >$, there exist no more than one valuation of i_n that can make formula (3) satisfiable. This means the input letter i_n can be uniquely determined by $< o_{n+d-l}, \ldots, o_{n+d-1} >$. So a f^{-1} exists for this pair of d and l. We will characterize f^{-1} with formula (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, formula (3),(4) and (5) should be rewritten as the following formula (6), (7) and (8), in which bold formulas are used to account for predicate R.

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

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

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

C. Overapproximating Reachable State Set

In the previous subsection, we have constrained the valid pattern of i_m. However, the s_n in figure 2 still hasn't 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 formula (10), and constraining that $s_n \in RS$:

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

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

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

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

$$RS^{S\to p} = \Big\{ s | \; s \equiv s_n \wedge \bigwedge_{m=n-p}^{n-1} \big\{ s_{m+1} \equiv T(s_m, i_m) \wedge R(i_m) \big\} \Big\} \tag{11}$$

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

$$RS^{S\to p} \subseteq \cdots \subseteq RS^{S\to q} \subseteq \ldots \text{ where } p > q$$

Unfortunately, RS is not a subset of any $RS^{S\to p}$, because there may exist 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.

$$0 \to 1 \to 2 \to 3 \to \underbrace{\overset{\curvearrowleft}{4}}_{RS^{S\to 4}}$$

In this case, number 0 to 3 are not in $RS^{S\to p}$, for $p > 3$. Thus, we can't overapproximate RS with $RS^{S\to p}$.

On the other hand, because circuit E and E^{-1} run in a never ending 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 $\bigcup_{q>p} RS^{s_0\to q}$ instead of RS. Obviously, $\bigcup_{q>p} RS^{s_0\to q}$ is a subset of $RS^{S\to p}$. Thus, we can use $RS^{S\to p}$ as an overapproximation of $\bigcup_{q>p} RS^{s_0\to q}$.

In order to account for $s_n \in RS^{S\to p}$, we prepend $\bigwedge_{m=n-p}^{n-1} \big\{ s_{m+1} \equiv T(s_m, i_m) \wedge R(i_m) \big\}$ to formula (6),(7) and (8), and obtain formula (12), (13) and (14). As a result, in addition to parameters d and l, we have a third parameter p to be searched.

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

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

$$\begin{array}{c} F_E \wedge F'_E \wedge \\ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ i_n \neq i'_n \end{array} \tag{14}$$

Now putting it all together, with formula (12), (13) and (14), we iterate through all valuations of d, l and p, from smaller one to larger one, until we find one valuation of d,l and p that makes formula (14) unsatisfiable. Then F_E in formula (12) will be used in section IV and V to build the complementary circuit E^{-1}.

IV. Characterizing f^{-1} with ALLSAT Algorithm Designed for XOR Intensive Circuits

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

A. Partitioning f^{-1}

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

Assume that i_n is represented by Boolean variable set I_{var}, and $< o_{n+d-l}, \ldots, o_{n+d-1} >$ is represented by Boolean variable set O_{var}. Then, f^{-1} in Boolean domain is $f^{-1} : \{0,1\}^{O_{var}} \to \{0,1\}^{I_{var}}$, and can be defined as:

$$f^{-1} = \prod_{v \in I_{var}} f_v^{-1} \tag{15}$$

Thus, characterizing f^{-1} can be partitioned into multiple tasks, each task characterizing a Boolean function $f_v^{-1} : \{0,1\}^{O_{var}} \to \{0,1\}$ for a $v \in I_{var}$. The function f_v^{-1} will 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 $SA_v = \{A_1, \ldots, A_m\}$ is the set of satisfying assignments of $F_E \wedge v$, that is, the set of satisfying assignments that forces v to be 1. Then f_v^{-1} can be defined as :

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

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 exist some more efficient approaches to enumerate satisfying assignments of SAT instance [11]–[18]. According to subsection 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 they are still not efficient enough for our application. The reasons for their inefficiency and the improvements of our approach are:

1) XOR gates are used intensively in communication and arithmetic circuits. As explained in subsection II-B, satisfying assignments of XOR can't be merged by existing

approaches. We solve this problem by discovering XOR gates within $F_E \wedge v$ with $\boldsymbol{XORMIN}$ function.

2) There are lots of redundant clauses in F_E. We use the function $\boldsymbol{SIMPLIFY}$ 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.
3) The function BFL in algorithm 1 can remove at most 1 irrelevant variable with each SAT solving. Our improved version $\boldsymbol{BFL\_UNSAT}$ 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 $\boldsymbol{ALLSAT}$ in algorithm 1, with our improvements in boldface.

Algorithm 2: **Characterizing f^{-1}**

1) $\boldsymbol{F_E^v = SIMPLIFY(F_E, v)}$
2) $SA_v = \{\}$
3) while ($F_E^v \wedge v$ is satisfiable) {
4) Assume A is a satisfying assignment
5) $\boldsymbol{A_{BFL} = BFL\_UNSAT(F_E^v, A, v)}$
6) $\boldsymbol{A_{XOR} = XORMIN(F_E^v, A_{BFL}, v)}$
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 formula (16)

The details of function $SIMPLIFY$, BFL_UNSAT and $XORMIN$ are described in the following subsections.

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 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 $\boldsymbol{SIMPLIFY(F_E, v)}$ 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 the following formula (17) with depth first approach in Lintao Zhang et al. [9]:

$$\begin{array}{c} F_E \wedge F'_E \wedge \\ \bigwedge_{u \in O_{var}} u \equiv u' \wedge \\ v \neq v' \end{array} \tag{17}$$

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

$$F_E^v = F_E \cap F_E^{UNSAT} \tag{18}$$

We first need to prove that:

Theorem 1 (): **Formula (17) is unsatisfiable**

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

$$formula(14) \Rightarrow \begin{array}{c} F_E \wedge F'_E \wedge \\ \bigwedge_{u \in O_{var}} u \equiv u' \wedge \\ \bigvee_{v \in I_{var}} v \neq v' \end{array} \Rightarrow \begin{array}{c} \bigvee_{v \in I_{var}} \Big\{ \\ formula(17) \\ \Big\} \end{array}$$

According to this rewritten result, if for any v, formula (17) is satisfiable, then the unsatisfiable formula (14) will be satisfiable. It is a contradiction, so formula (17) must be unsatisfiable. ■

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

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 formula (17) and (18), the following formula is also unsatisfiable:

$$\begin{array}{c} F_E^v \wedge F'_E \wedge \\ \bigwedge_{u \in O_{var}} u \equiv u' \wedge \\ v \neq v' \end{array}$$

because it is a super set of unsatisfiable core F_E^{UNSAT} of formula (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 value 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 will significantly reduce F_E size and run time overhead.

D. Minimizing Satisfying Assignments by Extracting Unsatisfiable Core

In algorithm 1 line 4, $\boldsymbol{BFL}$ [13] is used to remove those variables irrelevant to forcing v to be 1. According to 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 run time overhead of BFL is to remove more than one u with every unsatisfiability test. We will achieve this goal by:

1) In the first step, computing unsatisfiable core F^{US} of $F \wedge \neg v \wedge A|_{U-\{u\}}$ with depth first approach in Lintao Zhang et al. [9]:
2) In the second step, computing a new A by intersecting the clause set of $A|_{U-\{u\}}$ and F^{US}

The implementation of the improved $\boldsymbol{BFL}$ is shown below:

Algorithm 3: **Improved $\boldsymbol{BFL}$ based on Extracting Unsatisfiable Core**

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

Its correctness is proven below:

Theorem 3 (): **After $\boldsymbol{BFL\_UNSAT}$ is finish,** $F \wedge \neg v \wedge A$ **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 $\boldsymbol{BFL\_UNSAT}$, we know that A is a satisfying assignment of $F_E^v \wedge v$. Again according to theorem 1 and 2, v can't 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 $\boldsymbol{BFL\_UNSAT}$ is also a set of necessary variable assignments that force v to be 1. Thus $\boldsymbol{BFL}$ can be replaced by $\boldsymbol{BFL\_UNSAT}$ safely.

We will also show in experimental results that the function $\boldsymbol{BFL\_UNSAT}$ 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 $\boldsymbol{BFL\_UNSAT}$ is a minimal assignment, which means removing any variable from A will make it unable to force v to be 1.

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

XOR gates are used intensively in communication and arithmetic circuits. According to subsection II-B and figure 1b), the two satisfying assignments of the XOR gate can't 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 $A|_{V-\{v_1,v_2\}}|^{v_1 \to \neg A(v_1)}|^{v_2 \to \neg A(v_2)}$ by replacing v_1 and v_2 with z in A.

Intuitively, for a satisfying assignment A that can force v to be 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:

$$A_{\overline{v}_1,\overline{v}_2} = A|_{V-\{v_1,v_2\}}|^{v_1 \to \neg A(v_1)}|^{v_2 \to \neg A(v_2)} \tag{19}$$

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

$$F_E \wedge \neg v \wedge A_{\overline{v}_1,\overline{v}_2} \tag{20}$$

the unsatisfiability of formula (20) means that $A_{\overline{v}_1,\overline{v}_2}$, just like A, can also force v to be 1.

Thus, A and $A_{\overline{v}_1,\overline{v}_2}$, which can't be merged by BFL, can be merged into:

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

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

$$z = v_1 \oplus v_2 \tag{22}$$

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 unsatisfiability of formula (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 subsection V-B to build E^{-1}.

With the above discussion, we describe **XORMIN** as below:

Algorithm 4: $\boldsymbol{XORMIN(F_E, A, v)}$

1) $G = \{\}$
2) do {
3) $G_{new} = \{\}$ // the set of newly discovered XOR
4) foreach $v_1, v_2 \in O_{var}$ {
5) if(**formula (20) is unsatisfiable**){
6) $G_{new} = G_{new} \cup \{z = v_1 \oplus v_2\}$
7) $A = A|_{O_{var}-\{v_1,v_2\}}|^{z \to A(v_1) \oplus A(v_2)}$
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

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, 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}.

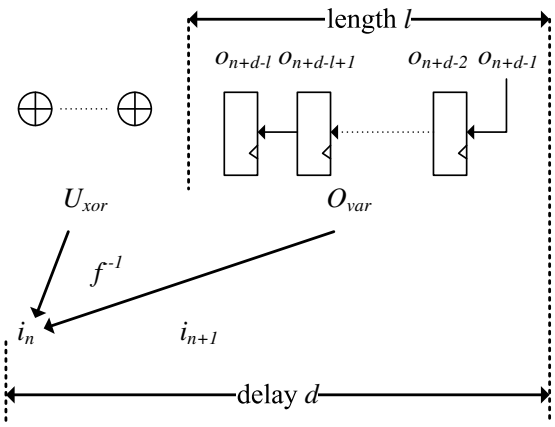

Fig. 4. Circuit structure of E^{-1}

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 $< o_{n+d-l}, \ldots, o_{n+d-1} >$ as input, and computes i_n.

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

B. Instantiating Discovered XOR Gates

According to subsection VII-C, assume the set of all XOR gates discovered by function $\boldsymbol{XORMIN}$ is G. Then the output variable set of these XOR gates is:

$$U_{xor} = \Big\{ z | \{ z = v_1 \oplus v_2 \} \in G \Big\} \tag{23}$$

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 figure 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 be 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($condition_1 || \ldots || condition_n$)
3) $v <= 1'b1$
4) else
5) $v <= 1'b0$
6) end

The $condition_1$ to $condition_n$ in line 2 correspond to every satisfying assignments in SA_v.

VI. Experimental Results

We implement our algorithm in zchaff [5], 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.ssypub.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 (seconds)	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 [20].
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 make a data sequence to have enough transitions between 0 and 1, 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 Assertion

To write assertion for ruling out invalid input letters, we refer to the following documentations, and find out the valid letter pattern easily:

1) For the XGXS and T2 ethernet encoders, table 48-2, 48-3 and 48-4 of IEEE-802.3ae 2002 standard [20] give the pattern of valid letters.
2) For the XFI encoder and scrambler, figure 49-7 and table 49-1 of IEEE-802.3ae 2002 standard [20] give the pattern of valid letters.
3) For the PCIE physical coding module, table 4-1 of PCI Express Base Specification [21] give 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 + XORMIN	time(s)	1.52	2939.47	11.97	47.55	36.64
	$\|F_E\|$	25470	5084496	499200	52209	459204
	#SAT	984	137216	8320	528	1032
BFL+ XORMIN +UNSAT	time(s)	1.08	752.83	1.84	0.82	27.08
	$\|F_E^v\|$	6694	188717	4807	6635	51204
	#SAT	480	16828	256	243	538

TABLE IV
COMPARING DECODER AREA

	XGXS	XFI	scrambler	PCIE	T2 ethernet
hand written decoder	913	4886	1514	952	2225
decoder built by Shen [33]	667	15269	1302	344	661
decoder built by our algorithm	652	16659	1302	345	569

D. Improvement on Run Time Overhead

Table III compares the following three statistics between the BFL algorithm [13], BFL+XORMIN proposed in our previous work [33], 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 [33].
2) $|\boldsymbol{F_E}|$ and $|\boldsymbol{F_E^v}|$ 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 $\boldsymbol{SIMPLIFY}$. Obviously, $|\boldsymbol{F_E^v}|$ is significantly smaller than $|\boldsymbol{F_E}|$.
3) The two **#SAT** rows compare the total number of SAT solving invoked by $\boldsymbol{BFL}$ and $\boldsymbol{BFL\_UNSAT}$. 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 [33], and our algorithm. We synthesize these decoders with LSI10K technology library coming from Synopsys DesignCompiler.

From table IV, we observe that:

1) Except the most complex XFI, our synthesis result is better than that of hand written decoders. However, this dose not mean that our algorithm is better than human designer. Actually, hand written decoders often include some other logic irrelevant to decoder functionality.
2) For the XFI case, our circuit area is about 3 times larger than hand written decoders. This means we need to improve circuit area in the future work.
3) There are no significant area difference between our algorithm and our previous work. [33].

VII. RELATED WORKS

A. Unsatisfiable Core Extraction

There are many classes of unsatisfiable core extraction algorithms.

The first class of algorithms extracted small but not necessary minimal unsatisfiable core. Such algorithms were often used to verify unsatisfiability of certain instance. Bruni and Sassano [36] proposed a constructive approach, which started from an initial clause set and iteratively added new clauses into it, until the clause set became unsatisfiable. Goldberg and Novikov [8] and Zhang and Malik [9] recorded the resolution graph that generated conflict clauses during SAT solving, and traced backward along this graph to find out those clauses that caused unsatisfiability. Gershman et al. [38] tried to find internal dominator node d in resolution tree that consumed large number of original clauses set M, and then tried to prove d without M, if it was ok, then M could be removed.

The second class of algorithm tried to find minimal unsatisfiable core, whose subset are all satisfiable. Oh et al. [37] added a clause-selector variable to each clause of the CNF formula, and used a DPLL-like procedure to find a clause subset with minimal number of clause-selector. Huang [40] also used clause-selector variable approach, but it converted the problem of finding minimal unsatisfiable core to model counting problem and then used BDD to perform model counting. Dershowitz et al. [44] first constructed a resolution graph from the unsatisfiable core, and then tested every initial clause in this graph to make sure if it was in a minimal core. Gregoire et al. [46] used a local search approach to search for minimal core. This approach used a scoring heuristic that recorded the common literal between different clauses, and removed those clause that weren't likely in a minimal core. Liffiton and Sakallah [45] proposed a preprocessing step to search autarky, which was a partial assignment to the variables of a Boolean CNF formula that satisfied every clause containing an assigned variable.

The third class of algorithm tried to find the smallest or minimum core. Lynce and Silva [39] used a clause-selector approach that was similar to AMUSE [37] mentioned above. It either searched for satisfy and then backtracked to the most recent clause selector variable set to true, or searched for conflict and then backtracked to a clause selector variables, which meant an unsatisfiable core was found. At this point, it recorded the size of core and continued searching for smaller core. Mneimneh et al. [41] and Liffiton et al. [42] used a branch-and-bound algorithm that utilized iterative MAXSAT solutions to generate lower and upper bounds on the size of the smallest unsatisfiable core, and branched on specific sub-formulas to find it. Zhang et al. [43] used a greedy genetic algorithm to find smallest unsatisfiable core.

The fourth class of algorithm tried to find out all minimal cores, such that the complete information about infeasibility could be obtained. This technology was particular useful in refining abstract model of model checking.

Bailey and Stuckey [48] and Liffiton and Sakallah [47] proposed to use the relation between minimal correction subsets and minimal unsatisfiable cores, first computed minimal correction subsets, and then computed minimal unsatisfiable cores.

B. Satisfying Assignments Enumeration

Existing ALLSAT algorithms all tried to enlarge the total satisfying assignments, so that a large state set that contains more total satisfying assignments can be obtained.

The first such approach was proposed by K. L. McMillan [12]. He constructed an alternative implication graph in SAT solver, which recorded the reasoning relation that led to the assignment of a particular object variable. All variables outside this graph could be ruled out from the total assignment. Kavita Ravi et al. [13] and P. P. Chauhan et al. [17] removed those variables whose absence could not make $obj \equiv 0$ satisfiable one by one. Shen et al. [15] and HoonSang Jin et al. [11], [14] used a conflict analysis based approach to remove multiple irrelevant variables in one SAT run. Orna Grumberg et al. [16] separated the variable set into important subset and non-important subset. Variables in important subset had higher decision priority than non-important ones. Thus, the important subset formed a search tree, with each leaf being another search tree for non-important set. Cofactoring [18] qualified out non-important variables by setting them to constant value returned by SAT solver.

C. AND-XOR Logic Synthesis

Classical logic synthesis worked on AND-OR network. Its kernel was two-level logic minimization, which tried to find a smaller sum-of-products expression for Boolean function f.

Three most well known two-level logic minimization algorithms were Quine-McCluskey [23], Scherzo [24], and Espresso-II [25].

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

One research direction focused on extending classical two-level AND-OR minimization to two-level AND-XOR network [26], [27]. These work normally described circuits with the most general ESOP (exclusive sum of product) expressions.

Another line of research relied on Reed-Muller expansion [28], and one of its most used variant was Fixed Polarity Reed-Muller Form (FPRM) given by Davio and Deschamps [29], in which a variable could have either positive or negative polarity. Some related works that relied on FPRM are [30]–[32].

A recent paper [35] tried to decompose a function into result 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 improve by applying this approach to character the complementary function f^{-1}.

VIII. Conclusions and Future Works

In this paper, we propose 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 possible future work is to improve the circuit area of generated complementary circuit E^{-1}. **According to section IV, the algorithm XORMIN discover 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 [35], then we can avoid redundant XOR gates and reduce the circuit area.**

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

Yet another 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 in reasonable time bound.

One issue that wasn't addressed by this paper is timing. The generated complementary circuit is 2-level logic structure. It seems that the well know retiming algorithm [34] can't be applied to such circuits to improve timing. But we must notice here that, the 2-level structure in our generated complementary circuits are built by AND and OR gates with large fanin, which aren't available in realistic target cell libraries. When these complementary circuits are mapped to target cell library with logic synthesizer, such large fanin gates will be mapped to a chain of real gates with smaller fanin. In such mapped circuits, the retiming algorithm can be applied easily to improve timing.

References

[1] Chris Kozup. Is 802.11n Right for You? Mobility blog. Retrieved April 27, 2009, from http://blogs.cisco.com/wireless/comments/is_80211n_right_for_you/, 2008.

[2] Stephen J. Dubner. What Are the Lessons of the Blu-Ray/HD-DVD Battle? A Freakonomics Quorum. The New York Times. Retrieved April 27, 2009, from http://freakonomics.blogs.nytimes.com/2008/03/04/what-are-the-lessons-of-the-blu-rayhd-dvd-battle-a-freakonomics-quorum/, 2008.

[3] Xilinx Core Generator System. Retrieved January 06, 2010, from http://www.xilinx.com/tools/coregen.htm, 2008.

[4] DesignWare Library . Retrieved January 06, 2010, from http://www.synopsys.com/IP/DESIGNWARE/Pages/default.aspx, 2008.

[5] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In DAC'01, pp 530-535, 2001.

[6] Evguenii I. Goldberg, Yakov Novikov. BerkMin: A Fast and Robust Sat-Solver. in DATE'02, pp 142-149, 2002.

[7] Niklas Een, Niklas Sorensson. Extensible SAT-solver. in SAT'03, pp 502-518, 2003.

[8] Evguenii I. Goldberg, Yakov Novikov. Verification of Proofs of Unsatisfiability for CNF Formulas. in DATE'03, 10886-10891, 2003.

[9] Lintao Zhang, Sharad Malik. Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. in DATE'03, 10880-10885, 2003.

[10] Samir Sapra, Michael Theobald, Edmund M. Clarke. SAT-Based Algorithms for Logic Minimization. in ICCD'03, pp 510-519, 2003.

[11] HoonSang Jin, Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In DAC'05, pp 750-753, 2005.
[12] Kenneth L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In CAV'02, pp 250-264, 2002.
[13] Kavita Ravi, Fabio Somenzi. Minimal Assignments for Bounded Model Checking. In TACAS'04, pp 31-45, 2004.
[14] HoonSang Jin, HyoJung Han, Fabio Somenzi. Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit. In TACAS'05, pp 287-300, 2005.
[15] ShengYu Shen, Ying Qin, Sikun Li. Minimizing Counterexample with Unit Core Extraction and Incremental SAT. In VMCAI'05, pp 298-312, 2005.
[16] Orna Grumberg, Assaf Schuster, Avi Yadgar. Memory Efficient All-Solutions SAT Solver and Its Application for Reachability Analysis. In FMCAD'04, pp 275-289, 2004.
[17] Pankaj Chauhan, Edmund M. Clarke, Daniel Kroening. A SAT-based algorithm for reparameterization in symbolic simulation. In DAC'04, pp 524-529, 2004.
[18] Malay K. Ganai, Aarti Gupta, Pranav Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In ICCAD'04, pp 510-517, 2004.
[19] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, Yunshan Zhu. Symbolic Model Checking Using SAT Procedures instead of BDDs. In DAC'99, pp 317-320, 1999.
[20] 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 Download from http://people.freebsd.org/~wpaul/802_3ae_2002.pdf
[21] PCI Express Base Specification Revision 1.0. Download from http://www.pcisig.com
[22] Mealy, George H. A Method for Synthesizing Sequential Circuits. Bell Systems Technical Journal v 34, pp1045-1079, 1955.
[23] Edward J. McCluskey. Logic Design Principles. Prentice-Hall, 1986.
[24] Olivier Coudert. On Solving Covering Problems. In DAC'96, pp 197-202, 1996.
[25] Richard L. Rudell, Alberto L. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. IEEE Transactions on CAD,6(5), pp 727-750, 1987.
[26] Philipp W. Besslich and M. Riege. An efficient program for logic synthesis of Mod-2 Sum Expressions. In Euro ASIC'91, pp 136-141, 1991.
[27] Tsutomu Sasao. AND-EXOR expressions and their optimization. Kluwer Academic Publishers, Editor, Logic Synthesis and Optimization, Boston, 1993.
[28] Irving S. Reed. A class of multiple-error-correcting codes and their decoding scheme. IRE Trans. on Inf. Theory, PGIT-4:48-49, 1954.
[29] Marc Davio. Discrete and switching Functions. George and McGraw-Hill, NY, 1978.
[30] Andisheh Sarabi, Marek A. Perkowski. Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks. In DAC'92, pp 30-35, 1992.
[31] Rolf Drechsler, Bernd Becker, Michael Theobald. Fast OFDD based minimization of fixed polarity Reed-Muller expressions. in EURO-DAC'94, pp 2-7, 1994.
[32] Unni Narayanan, C. L. Liu. Low power logic synthesis for XOR based circuits. in ICCAD'97, pp 570-574, 1997.
[33] ShengYu Shen, JianMin Zhang, Ying Qin and SiKun Li. *Synthesizing Complementary Circuits Automatically*. accepted by ICCAD'09,http://www.ssypub.org/pub/iccad09_ssy.pdf
[34] Charles E. Leiserson, James B. Saxe. Optimizing Synchronous Systems. in FOCS'81, pp 23-36, 1981.
[35] Tomasz S. Czajkowski, Stephen Dean Brown. Functionally linear decomposition and synthesis of logic circuits for FPGAs. in DAC'08, pp 18-23, 2008.
[36] Renato Bruni, Antonio Sassano. Restoring Satisfiability or Maintaining Unsatisfiability by finding small Unsatisfiable Subformulae. In LICS Workshop on Theory and Applications of Satisfiability Testing, 2001.
[37] Yoonna Oh, Maher N. Mneimneh, Zaher S. Andraus, Karem A. Sakallah, Igor L. Markov. AMUSE: a minimally-unsatisfiable subformula extractor. In DAC'04, pp 518-523, 2004.
[38] Roman Gershman, Maya Koifman, Ofer Strichman. Deriving Small Unsatisfiable Cores with Dominators. In CAV'06, pp 109-122, 2006.
[39] Ines. Lynce and Joao P. Marques Silva. On Computing Minimum Unsatisfiable Cores. In SAT'04, pp 305-310, 2004.
[40] Jinbo Huang. MUP: a minimal unsatisfiability prover. In ASPDAC'05, pp 432-437, 2005.
[41] Maher N. Mneimneh, Ines. Lynce, Zaher S. Andraus, Joao P. Marques Silva, Karem A. Sakallah. A Branch-and-Bound Algorithm for Extracting Smallest Minimal Unsatisfiable Formulas. In SAT'05, pp 467-474, 2005.
[42] Mark H. Liffiton, Maher N. Mneimneh, Ines. Lynce, Zaher S. Andraus, Joao P. Marques Silva, Karem A. Sakallah. A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas. Constraints 14(4): 415-442, 2009.
[43] JianMin Zhang, SiKun Li, and ShengYu Shen. Extracting minimum unsatisfiable cores with a greedy genetic algorithm. In AI'06, pp 847-856, 2006.
[44] Nachum Dershowitz, Ziyad Hanna, Alexander Nadel. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In SAT'06, pp 36-41, 2008.
[45] Mark H. Liffiton, Karem A. Sakallah. Searching for Autarkies to Trim Unsatisfiable Clause Sets. In SAT'08, pp 182-195, 2008.
[46] Eric Gregoire, Bertrand Mazure, Cedric Piette. Local-search Extraction of MUSes. Constraints 12(3), pp 325-344, 2007.
[47] Mark H. Liffiton, Karem A. Sakallah. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. J. Autom. Reasoning 40(1), pp 1-33, 2008.
[48] James Bailey, Peter J. Stuckey. Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. in PADL'05, pp 174-186, 2005.

IX. Responses to the reviewers

The Associate Editor require us to explicitly highlight the 30 percent added material beyond the original ICCAD submission.

We rewrite the 10th paragraph in section I, which starts with "Compared with our previous ICCAD'09 version". Please refer to it.

At the same time, compared to ICCAD version, some more added material:

1) we add section V to describe how to construct the complementary circuits verilog source code.
2) we also add more experimental result to compare new run time overhead and area of the complementary circuits.
3) we also add a subsection VII-A to describe the related work on unsatisfiable core extraction.

The remainder of this section will describe our responses to reviewer. Each subsection deal with one major question. The title of each subsection will summarize the question. And then there will be multiple bolded sentence **comment from reviewer x** in each subsection, these are the original comments from a particular reviewer x. For each such comments, there is a bolded sentence **response from author**. These are our response.

A. issue 1: Is this algorithm appropriate for datapaths with width>8 ?

==========

comment from reviewer 1.

==========

The authors point out that many transforms used in communication and multimedia are reversible in nature, often rich in XOR gates, with which I agree.

However, many such circuits are datapaths. For example, FFT/IFFT and used in wireless communications and sound processing, DCT/IDCT are used in image compression, 3x3 perspective transforms are used in computer graphics. It's not clear that bit-oriented synthesis and verification are appropriate for datapaths with width>8. The authors do not acknowledge this issue, but go on and develop a SAT-based technique that takes an existing circuit and reverses it, i.e., constructs a circuit that implements the inverse.

==========

response from author.

==========

You are right, I hasn't clarified this problem in section I.

To solve this problem, I add a bolded sentence in the 6th paragraph of section I, **Some of these circuits have very wide datapath, up to 64 bits** to highlight that our algorithm actually work on complex circuits with very wide datapath.

B. issue 2:If heavy-handed SAT-based techniques necessary? Why not simply use BMC-like techniques based on frame unrolling?

==========

comment from reviewer 1.

==========

There's also a requisite check for reversibility. The paper does not make a strong case for using heavy-handed SAT-based techniques, such as AllSAT-solver, unsatisfiable cores, BMC, etc – they seem far-fetched. Note that circuits used in communication and multimedia usually correspond to clear mathematical formulas, and it's not that difficult to guess whether they are reversible or not.

==========

response from author.

==========

Thank you for your comment. I add a bolded paragraph in section I as the 7th paragraph, which start with "One issue need to ...". Please refer to it.

==========

comment from reviewer 1.

==========

It is also fairly straightforward to construct a SAT formulation to check if two different inputs can map to one output, over some number of frames. Multimedia and communication circuits are usually not very deep sequentially, so known BMC-like techniques based on frame unrolling can be applied without much thought.

==========

response from author.

==========

You are right, actually our algorithm is almost the same as your comment above.

According to the 3rd paragraph of section 1, our algorithm includes two steps, the first step is used to check reversibility, while the second step is used to construct the complementary circuit.

In the first step, only SAT solver is used, no AllSAT-solver, no unsatisfiable cores extraction.

And the usage of SAT solver in the first step is almost the same as reviewer's comment above. Of course, some tricks are used to deal with nasty corner case, such as approximating reachable state set and ruling out invalid coding space.

Please refer to section III for more detail.

C. issue 3:Encoder often come with appropriate decoder, no need to develop algorithm to generate decoder from encoder

==========

comment from reviewer 1.

==========

A major problem with the proposed approach is that the authors assume that an encoder is given, but a decoder is not. This seems odd. If an encoder was synthesized or implemented by hand, then a decoder can be handled using the same procedure. In particular, XOR-intensive circuits often include linear transformations (over the 2-element field), which can be inverted mathematically – sometimes such a transformation is its own inverse, so the same circuit can be reused.

==========

response from author.

==========

Thank you for your comment. I add a bolded paragraph in section I as the 7th paragraph, which start with "One issue need to ...". Please refer to it.

D. issue 4:Is the proposed algorithm combinational or sequential?

==========
comment from reviewer 1.
==========

The introduction is vague and does not clearly discuss the difference between combinational and sequential behaviors, which would be critical to this work. Some sequential behaviors are mentioned in the introduction, and the body of the paper covers this subject to some depth. However, the main computational technique based on SAT is an essentially combinational procedure adapted to handle some number of sequential frames.

==========
response from author.
==========

You are right. According to section V, just like bounded model checking, we unfold the transition relation of circuit, then the sequential problem can be transformed to combinational problem, and processed by SAT solver.

E. issue 5:Had timing problem been considered?

==========
comment from reviewer 1.
==========

Comparisons to Synopsys Design Compiler are useful, but area is not the only/main criterion for high-speed circuits like PCI-E and Ethernet. I don't see how SAT-based approaches can optimize circuit timing.

==========
response from author.
==========

You are right, our algorithm doesn't consider timing optimization at current stage.

I add a bolded paragraph at the end of section VIII to discuss this problem, which start with "One issue that ...".

Basically, we can rely on retiming algorithm to improve the frequency of generated circuits.

F. issue 6:Some related works are missing

==========
comment from reviewer 1.
==========

Section VII is not particularly comprehensive. For example, the authors do not mention a recent technique for XOR-based circuits developed in Tomasz S. Czajkowski, Stephen Dean Brown: Functionally linear decomposition and synthesis of logic circuits for FPGAs. DAC 2008: 18-23 This approach seems particularly well-suited to the type of benchmarks studies in this work.

A number of papers on unsat cores and all-sat were published by Karem Sakallah and his students, see http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/s/Sakallah:Karem_A=.html

==========
response from author.
==========

You are right, writing of section VII actually is not very comprehensive,

I rewrite it by :

1) Adding DAC'08 paper to reference and the end of subsection VII-C.
2) Adding a new subsection in section VII, to discuss unsatisfiable core extraction.

G. issue 7:Writing has a number of problems

==========
comment from reviewer 1.
==========

Writing has a number of problems. One is that "Boolean" spells with a capital B. References are listed in an inconsistent style (e.g., sometimes fist names are abbreviated, and sometimes they are spelled out; titles are sometimes italicized and sometimes not) and include spurious space characters, etc. Page numbers sometimes are not given.

==========
response from author.
==========

Now all have been fixed. Thank you very much.

H. issue 8:Had timing problem been considered?

==========
comment from reviewer 2.
==========

In this work, an approach is presented to automatically synthesize complementary circuits for communication encoding functions. Using SAT solving techniques, it is first discovered under which constraints an injective mapping from the inputs to the outputs is performed by the circuit. Then, the inverse function is constructed using an ALLSAT algorithm. In the latter step, an optimization for XOR-intensive circuits is used, that speeds up the ALLSAT enumeration. The approach is applied to real-world communication circuits. In the experiments, data on area of the resulting circuits is given.

The work touches an interesting topic and gives the appropriate solutions. The algorithm is presented in enough details to for a deep understanding, thereby giving the essential background and references on the topic. The technique applied for the ALLSAT solution of XOR intensive instances is an important contribution. In comparison to the previous work in [31], techniques based on unsatisfiable core extraction are used to improve the algorithm.

Comments:

In the introduction, you state that the targeted communication circuits are difficult to design because of deep pipelining, which is necessary to meet high frequency demands. Now, the circuits resulting from your algorithm, are more or less 2-level logic circuits. Does this organization allow for high frequencies? And, if not, is it suitable for the automatic

introduction of pipelining? This is an issue you need to address since you claim to solve an engineering problem.

==========

response from author.

==========

You are right, our algorithm doesn't consider timing optimization at current stage.

I add a bolded paragraph at the end of section VIII to discuss this problem, which start with "One issue that ...".

Basically, we can rely on retiming algorithm to improve the frequency of generated circuits.

I. issue 9: Some writing problems

==========

comment from reviewer 2.

==========

Detailed comments / corrections:

Sec. I:

"the deep pipeline" -> deep pipelining

"to build complementary circuit" -> to build a complementary circuit

"how to check parameterized..." -> how to check the parameterized...

"the boolean function of complementary circuit" -> of the complementary circuit

"we concludes" -> we conclude

Sec. II A

"need to be understand" -> understood

Sec. II C

"Definition 1: Kripke structure is..." -> A Kripke structure is

"limited length path" -> paths

"We call this length as the bound... -> We call this length the bound...

"To save space,...": Why? You have as much space as you want in this article. If it is not important, you can leave it out.

"Then BMC problem" -> Then the BMC problem

Sec. III A

"A finite set of input alphabets": I assume you mean a finite input alphabet? Same for output alphabet.

==========

response from author.

==========

All are fixed. Thank you.

And I also change all **alphabets** to **letters**, according to your last suggestion.

==========

comment from reviewer 2.

==========

Sec. III C

"... and make checking parameterized complementary condition fail unnecessary on unreachable states.": I do not understand this sentence.

==========

response from author.

==========

I rewrite this sentence, please refer to the bolded sentence in the first paragraph of subsection III-C.

==========

comment from reviewer 2.

==========

"we only need to consider those states in ANDEXOR...": what is this ANDEXOR or is it a formatting error?

==========

response from author.

==========

It is a typing error, ANDEXOR is actually a bibliography index of latex, I have delete it. Thank you.

==========

comment from reviewer 2.

==========

Sec. V A

"Instancing" -> Instantiating?

"To instance" -> to instantiate?

==========

response from author.

==========

All are fixed, Thank you.

J. issue 10:

==========

comment from reviewer 3.

==========

The paper describes a way of synthesizing complementary circuits using a SAT based approach.

Overall, the paper makes a very interesting contribution which is of theoretical and practical interest. The paper is quite well-written. I enjoyed reading the paper. Good work!

I have found four areas that could be improved. The first one is the most important one and should be addressed before the paper is published.

i) Section III.C.

I found it hard to get through this key section and there seem to be some points that are not quite clear.

a.) it may be good to use "overapproximate" instead of "approximate" (I assume that is what you mean.)

==========

response from author.

==========

you are right, we has correct it.

==========

comment from reviewer 3.

==========

b.) "Thus, we can't approximate RS with RS (S-¿p)". I see where your argument is going, but at this point you have only shown that it doesn't work for p¿3, not for p¡=3, right?

In fact, the entire previous paragraph (starting with 'unfortunately') is quite misleading, as it seems RS is included in RS (S-¿1) Maybe I'm missing something here?

==========

response from author.

==========

You are right, this subsection is very hard to be understood, but it is the price payed for accuracy. Actually, the basic idea is very simple, we just prepend a state transition sequence of

length p at the head of formula (6) and (7), then the s_n will be in $RS^{S \to p}$.

In my ICCAD'09 version, this subsection is more straightforward. It just says that $RS \subset RS^{S \to p}$, and then use $RS^{S \to p}$ as overapproximation of RS. ICCAD'09 version is easier to be understood, but it is not accurate. One ICCAD reviewer find this inaccuracy, so I modify it to current version.

I think you can simply ignore the paragraph starting with "unfortunately", and jump directly to paragraph starting with "On the other hand".

Or should I just delete these ignored paragraph? can you tell me which way is your favor?

==========

comment from reviewer 3.

==========

c.) ANDEXOR (is needs explanation, is it a typo?)

==========

response from author.

==========

you are right, it is a typo, I has remove it.

==========

comment from reviewer 3.

==========

d.) Most importantly, you never seem to mention how the overapproximation is later dealt with. Doesn't the fact that you have only an overapproximation (in contrast to an exact reachable state set) mean that you need to check the results you get in some way? (This is probably the key argument I found missing in the paper)

==========

response from author.

==========

Just as I mention above, we account for this overapproximation by prepending a state transition sequence of length p at the head of formula (6) and (7)

==========

comment from reviewer 3.

==========

ii) Does your overall approach need a verification step at the end? (i.e. do you need to compose E and E-1 to see it is the identity fct?)

==========

response from author.

==========

Actually the correctness is guaranteed by theory. So we don't need such step.

==========

comment from reviewer 3.

==========

iii) (minor) In the introduction, there seems to be a big jump from the first to second paragraph. It seems that there are references missing (in the second paragraph) to justify some of the claims.

==========

response from author.

==========

Actually, no reference to prove this claim.

But I am a designer of router and interface chips for high performance super-computers. The motivation of complementary synthesis come from my engineering experiments. I spent almost one and a half year to develop and verify a physical coding sub-layer module that can achieve both leading edge bandwidth and latency performance. Those busy day make me to invent complementary synthesis. After I post ICCAD paper to review, I found many people think it is a very interesting idea.

I think that may be enough to prove my claim in the second paragraph of section I.

==========

comment from reviewer 3.

==========

iv) As for determining d, p, l. As it is not guaranteed to find those parameters. Do you have any termination criteria when to start searching? In other words, if no complementary circuit exists, would your approach be able to detect this?

==========

response from author.

==========

You are right, the algorithm described in this paper don't guarantee termination. We just run it until time out.

I add this topic as one on my future work in conclusion section VIII. Please refer to it.