Synthesizing Complementary Circuits Automatically

ShengYu Shen
School of Computer Science, National University of Defense Technology
410073,DeYa Avenue
ChangSha,China
syshen@nudt.edu.cn

JianMin Zhang
School of Computer Science, National University of Defense Technology
410073,DeYa Avenue
ChangSha,China
jmzhang@nudt.edu.cn

Ying Qin
School of Computer Science, National University of Defense Technology
410073,DeYa Avenue
ChangSha,China
qy123@nudt.edu.cn

SiKun Li
School of Computer Science, National University of Defense Technology
410073,DeYa Avenue
ChangSha,China
skli@nudt.edu.cn

ABSTRACT

One of the most difficult jobs in designing communication and multimedia chips, is to design and verify complex complementary circuit pair (E, E^{-1}), in which circuit E transforms information into a format that is suitable for transmission and storage, while E's complementary circuit E^{-1} recovers this information.

In order to ease this job, we propose a novel two-step approach to synthesize complementary circuit E^{-1} from E fully automatically. First, we assume that the circuit E satisfies parameterized complementary assumption, which means its input can be recovered from its output under some parameter setting. We check this assumption with SAT solver and find out proper values of these parameters. Second, with parameter values and the SAT instance obtained in the first step, we build the complementary circuit E^{-1} with an efficient satisfying assignments enumeration technique that is specially designed for circuits with lots of XOR gates.

To illustrate its usefulness and efficiency, we run our algorithm on several complex encoders from industrial projects, including PCIE and 10G ethernet, and successfully generate correct complementary circuits for them.

Categories and Subject Descriptors

B.5.2 [**REGISTER-TRANSFER-LEVEL IMPLEMENTATION**]: Design Aids—*Automatic synthesis*; B.4.4 [**INPUT/OUTPUT AND DATA COMMUNICATIONS**]: Performance Analysis and Design Aids—*Formal models*

General Terms

Algorithms, Design, Theory, Verification

Keywords

Synthesis,Complementary Circuit,Satisfying Assignments Enumeration

1. INTRODUCTION

Communication and multimedia electronic applications are major driving forces of semiconductor industry. Many leading edge communication protocols and media formats, even still in non-standardized draft status, are implemented in chips and pushed to market, to maximize the chance 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 complex complementary circuit pair (E, E^{-1}), in which circuit E transforms information into a format that is suitable for transmission and storage, while E's complementary circuit E^{-1} recovers this information. Many factors significantly complicate the job of designing and verifying such circuit pairs. For example, deep pipeline to achieve high frequency, 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 synthesize E^{-1} from E fully automatically in two steps.

1. In the first step, we assume that, for E under some parameters valuation, its input alphabet sequence can be uniquely determined by its output alphabet sequence. We call this assumption **parameterized complementary assumption**. We use a SAT solver to check this assumption and find out proper values for some parameters that make this assumption hold.

2. In the second step, with the SAT instance and parameter values obtained in the first step, we build a circuit E^{-1} with an efficient satisfying assignments enumeration technology(abbreviated as **ALLSAT**), which is specially designed for communication and arithmetic circuit with lots of XOR gates.

We implement our algorithm on zchaff [3], and run it on several complex encoder circuits from industrial projects, including PCIE and 10G Ethernet. We can build complementary circuits for all of them within 3000 seconds.

The contribution of this paper is twofold: 1) We propose the first approach to decide if it's possible to recover input sequence of an circuit E from its output sequence. 2) We propose an efficient ALLSAT algorithm for XOR intensive circuits, to build complementary circuit E^{-1} from the SAT instance of circuit E.

The remainder of this paper is organized as follows. Section 2 presents background material. Section 3 presents how to check parameterized complementary assumption, and how to find out proper values of its parameters. Section 4 presents how to build complementary circuit. Section 5 presents experimental results of our approach. Section 6 presents related works. Section 7 concludes with a note on future work.

2. PRELIMINARIES

2.1 Propositional Satisfiability Problem

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

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

A computer program that decides the existence of such satisfying assignment is called **SAT solver**.

For an assignment $A : U \rightarrow \{0,1\}$,if $U \subset V$, then A is a **partial assignment**, 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. Its definition is, for $v \in W$, $A|_W(v) = A(v)$. Intuitively, $A|_W$ is obtained from A by removing all variables $v \notin W$.

For an assignment $A : U \rightarrow \{0,1\}$, and $u \notin U$, and $b \equiv 0$ or 1, $A|^{u\leftarrow b}$ is the **extension** of A on u, its definition is:

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

Intuitively, $A|^{u\leftarrow b}$ is obtained from A by adding the assignment of u.

Normally, SAT solver requires formula F to be represented in **CNF** format, in which a **formula** $F = \bigwedge_{cl\in CL} cl$ is a conjunction of clauses set CL, and a **clause** $cl = \bigvee_{l\in Lit} l$ is a disjunction of literals set Lit, and a **literal** is a variable v or its negation $\neg v$. A formula in CNF format is also called **SAT instance**.

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

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

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

2.2 Satisfying Assignments Enumeration

Technologies that enumerate all satisfying assignments of a formula are called **ALLSAT**. It is obvious that we can enumerate all total satisfying assignments A by repeatedly calling a SAT solver, and adding blocking clause $bclk_A$ of satisfying assignment A into 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 large n.

In order to reduce the number of satisfying assignments to be enumerated, we need **satisfying assignments minimization** technology to remove irrelevant variable's assignments from satisfying assignment A, such that A can cover more total satisfying assignments. For example, for OR gate $z \leftarrow u \vee v$, its total satisfying assignments that can make $z \equiv 1$ are $\{u \leftarrow 1, v \leftarrow 0\}$,$\{u \leftarrow 1, v \leftarrow 1\}$ and $\{u \leftarrow 0, v \leftarrow 1\}$, they contain 6 assignments to individual variables. It's obvious that the first two assignments can be merged into $\{u \leftarrow 1\}$, in which assignment to v is removed, and the latter two assignments can be merged into $\{v \leftarrow 1\}$, in which assignment to u is removed. 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, and $\boldsymbol{obj}$ is one object variable that should always be 1, and A is a satisfying assignment of $F \wedge obj$. If $F \wedge \neg obj \wedge A|_{V-\{v\}}$ is unsatisfiable, then $A|_{V-\{v\}}$ can't make obj to be 0, so obj must still be 1. Thus, we can merge $A|_{V-\{v\}}|^{v\leftarrow 0}$ and $A|_{V-\{v\}}|^{v\leftarrow 1}$ to remove v from A, and obtain a succinct partial satisfying assignment $A|_{V-\{v\}}$.

All existing ALLSAT approaches [4, 5, 6, 7, 8, 9, 10, 24] share this idea of satisfying assignments minimization.

On the other hand, for XOR gate $z \leftarrow u \oplus v$, its total satisfying assignments that can make $z \equiv 1$ are $\{u \leftarrow 1, v \leftarrow 0\}$ and $\{u \leftarrow 0, v \leftarrow 1\}$, they 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. There are 2^n total satisfying assignments for this comparator, none of them can be merged with each other.

Thus, enumerating satisfying assignments for XOR intensive circuits is a major difficulty of state-of-the-art ALLSAT approaches, we will solve this problem in section 4.

3. CHECKING PARAMETERIZED COMPLEMENTARY ASSUMPTION

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

3.1 Parameterized Complementary Assumption

Our algorithm cares about the input and output sequence

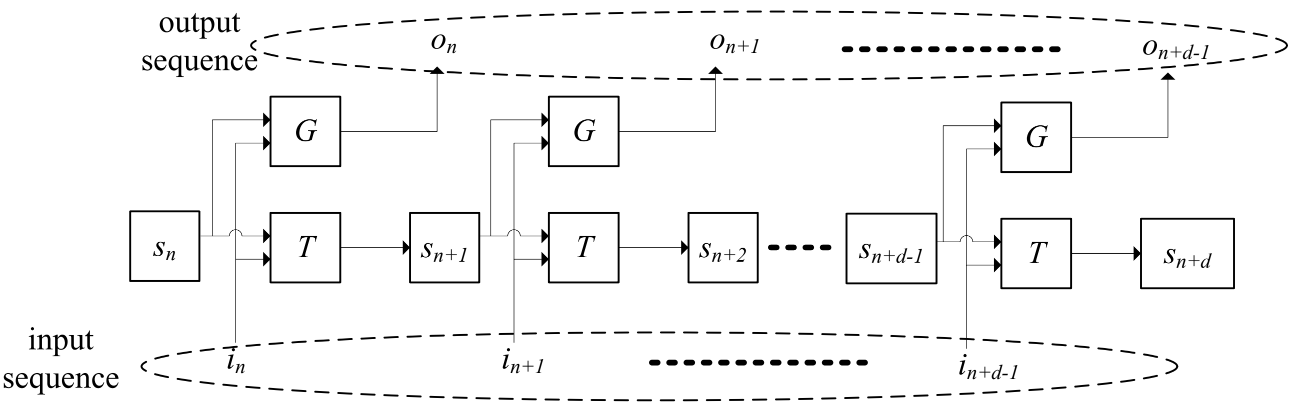

Figure 1: Unfolding Transition Function of Mealy Finite State Machine

of circuit E, so **Mealy finite state machine**[25] is a suitable model for us.

Definition 1. **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 input alphabets I
4. A finite set of output alphabets 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 figure 1. This relationship is built by unfolding the transition function T and output function G d times, as shown in formula (2).

$$\bigwedge_{m=n}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \right\} \qquad (2)$$

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 sub-sequence has two parameters, its length l and its delay d compared to i_n, as shown in following figure 2.

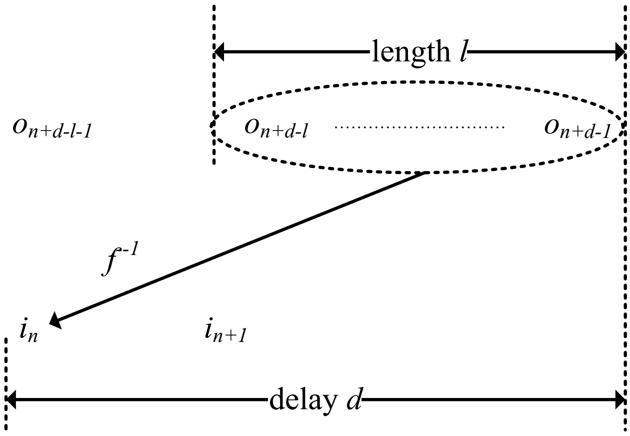

Figure 2: f^{-1} and its Parameters

Thus, $f^{-1} : O^l \rightarrow I$ is now 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 the following assumption holds:

Definition 2. **Parameterized Complementary Assumption**:For any valuation of sequence $< o_{n+d-l}, \ldots, o_{n+d-1} >$, assume there exists no more than one valuation of i_n, that can make formula (2) satisfiable.

This assumption holds if and only if the following formula (3) is unsatisfiable.

$$\begin{array}{c} \bigwedge_{m=n}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \right\} \wedge \\ \bigwedge_{m=n}^{n+d-1} \left\{ s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m) \right\} \wedge \\ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ i_n \neq i'_n \end{array} \qquad (3)$$

In formula (3), the first line is the same as formula (2), the second line is a copy of formula (2), except that its variables are all renamed by appending a prime. These two lines mean two individual unfolding of circuit E. The third line constraints 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 fourth line constraints that their input alphabet i_n and i'_n are different.

For a particular pair of d and l , checking formula (3) 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 (2) satisfiable. This violates definition 2, so no f^{-1} exists for this pair of d and l. We should continue searching further for larger d and l.

2. **Unsatisfiable**.In this situation, the parameterized complementary assumption is satisfied, so a $f^{-1} : O^l \rightarrow I$

exists for this pair of d and l. We will build f^{-1} with formula (2) in section 4.

3.2 Ruling out Invalid Input Alphabets with Assertion

Most communication protocols and systems have some restrictions on 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 alphabet. Invalid input alphabets will be mapped to some predefined error output alphabet, 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 alphabet $e \in O$. Hence, this will prevent our approach from distinguishing i_n from i'_n.

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

Thus, formula (2) and (3) should be rewritten as following formula (4) and (5), in which bold formulas are used to account for predicate R.

$$\bigwedge_{m=n}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge \boldsymbol{R(i_m)} \right\} \quad (4)$$

$$\begin{gathered} \bigwedge_{m=n}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge \boldsymbol{R(i_m)} \right\} \wedge \\ \bigwedge_{m=n}^{n+d-1} \left\{ s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m) \wedge \boldsymbol{R(i'_m)} \right\} \wedge \\ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ i_n \neq i'_n \end{gathered} \quad (5)$$

3.3 Approximating Reachable State Set with Prefix Sequence

In last subsection, we have constrained the valid pattern of i_m. But s_n in figure 1 still hasn't been constrained yet. This s_n may be outside of reachable state set of circuit E, which may make checking parameterized complementary assumption fail unnecessary.

Assume that circuit E can be modeled by Mealy state machine $M_E = (S, s_0, I, O, T, G)$. Its reachable state set with assertion predicate R is:

$$\begin{aligned} RS_E \leftarrow \Big\{ & s | \exists q \text{ such that} \\ & s \equiv s_q \wedge \bigwedge_{m=0}^{q-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge R(i_m) \right\} \Big\} \end{aligned} \quad (6)$$

Thus, to rule out unreachable s_n, we need to rewrite formula (4) and (5) as formula (7) and (8) :

$$\begin{gathered} s_n \in RS_E \wedge \\ \bigwedge_{\boldsymbol{m=n}}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge R(i_m) \right\} \end{gathered} \quad (7)$$

$$\begin{gathered} s_n \in RS_E \wedge s'_n \in RS_E \wedge \\ \bigwedge_{\boldsymbol{m=n}}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge R(i_m) \right\} \wedge \\ \bigwedge_{\boldsymbol{m=n}}^{n+d-1} \left\{ s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m) \wedge R(i'_m) \right\} \wedge \\ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ i_n \neq i'_n \end{gathered} \quad (8)$$

Now we have two extreme cases:

1. One extreme case is formula (4) and (5) with low computation complexity, but high risk of unnecessary fail in checking parameterized complementary assumption.

2. The other extreme case is formula (7) and (8), which are not affected by unreachable state set, but with very high computation complexity in computing reachable state set RS_E.

Obviously, on the balance of these two extremes , we want a method with both acceptable computation complexity and low risk of unnecessary fail in checking parameterized complementary assumption.

To achieve this, we approximate RS_E with a prefix state transition sequence of length p:

$$\begin{aligned} RS_E^p \leftarrow \Big\{ & s | \\ & s \equiv s_n \wedge \bigwedge_{m=n-p}^{n-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge R(i_m) \right\} \Big\} \end{aligned} \quad (9)$$

Obviously, formula (9) is very similar to (6), except that (9) doesn't consider initial state s_0. So RS_E and all RS_E^p form a total order relation :

$$RS_E \subseteq \cdots \subseteq RS_E^p \subseteq \cdots \subseteq RS_E^q \subseteq \ldots \text{ where } p > q$$

So now, in addition to parameters d and l, we have the third parameter p to be searched. In order to account for RS_E^p, we need to rewrite formula (4) and (5) as following formula (10) and (11), by changing starting index of m from n to $n - p$:

$$\begin{aligned} & \boldsymbol{F_E} \leftarrow \\ & \bigwedge_{\boldsymbol{m=n-p}}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge R(i_m) \right\} \end{aligned} \quad (10)$$

$$\begin{gathered} \bigwedge_{\boldsymbol{m=n-p}}^{n+d-1} \left\{ s_{m+1} \equiv T(s_m, i_m) \wedge o_m \equiv G(s_m, i_m) \wedge R(i_m) \right\} \wedge \\ \bigwedge_{\boldsymbol{m=n-p}}^{n+d-1} \left\{ s'_{m+1} \equiv T(s'_m, i'_m) \wedge o'_m \equiv G(s'_m, i'_m) \wedge R(i'_m) \right\} \wedge \\ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \wedge \\ i_n \neq i'_n \end{gathered} \quad (11)$$

Now put it altogether, with formula (10) and (11), 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 (11) unsatisfiable, then that valuation and F_E in formula (10) will be used in section 4 to build complementary circuit E^{-1}.

4. BUILDING COMPLEMENTARY CIRCUIT WITH ALLSAT ALGORITHM DESIGNED FOR XOR INTENSIVE CIRCUITS

If we find a proper value for parameters d,l and p in section 3, we can now build the complementary circuit E^{-1}'s boolean function $f^{-1} : O^l \rightarrow I$ in this section.

4.1 Algorithm Framework

According to section 3.3, $f^{-1} : O^l \rightarrow I$ can be built from formula (10) by enumerating satisfying assignments of $< o_{n+d-l}, \ldots, o_{n+d-1} >$ and i_n.

Assume that the set of all total satisfying assignments of formula (10) is $\{A_1, \ldots, A_t\}$, then f^{-1} can be defined as:

$$f^{-1}(< o_{n+d-l}, \ldots, o_{n+d-1} >) =$$
$$\begin{cases} A_1(i_n) & if \ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv A_1(o_m) \\ \qquad \ldots & \\ A_t(i_n) & if \ \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv A_t(o_m) \end{cases}$$

But this naive approach suffers from the state space explosion problem. For O^l that contains m boolean variables, there may be 2^m satisfying assignments, which make it impossible to build f^{-1} for large m.

There exists some much more efficient approaches to enumerate satisfying assignments of SAT instance [4, 5, 6, 7, 8, 9, 10, 24]. Their basic idea has been introduced in subsection 2.2.

But these approaches are still not efficient enough for our application. The essential reason that leads to this inefficiency is the wide usage of XOR gates in communication and arithmetic circuits. As explained in subsection 2.2, satisfying assignments of XOR can't be minimized.

So we invent a novel approach that is specially designed for XOR intensive circuits, as shown below:

ALGORITHM 1. ***Synthesizing Complementary Circuit***

1. *assume that* $\boldsymbol{I_{var}}$ *and* $\boldsymbol{O_{var}}$ *are respectively boolean variable sets that represent* i_n *and* $< o_{n+d-l}, \ldots, o_{n+d-1} >$. *and* $\boldsymbol{F_E}$ *is defined in formula (10)*
2. $G_{XOR} \leftarrow \{\}$
3. *foreach* $v \in I_{var}$ {
4. $SA_v \leftarrow \{\}$
5.   *while* ($F_E \wedge v \equiv 1$ *is satisfiable*) {
6.   *Assume* A *is a satisfying assignment*
7. $\boldsymbol{A_{BFL} \leftarrow BFL(F_E, A, v)}$
8. $\boldsymbol{\{A_{XOR}, G\} \leftarrow XORMIN(F_E, A_{BFL}, v)}$
9. $SA_v \leftarrow SA_v \cup \{A_{XOR}\}$
10. $G_{XOR} \leftarrow G_{XOR} \cup G$
11. $F_E \leftarrow F_E \wedge bcls_{A_{XOR}}$
12. $F_E \leftarrow F_E \wedge \bigwedge_{(x_1 \leftarrow v_1 \oplus v_2) \in G} \{x_1 \equiv v_1 \oplus v_2\}$
13.   }
14.   *Building* $f_v^{-1} : \{0,1\}^{O_{var}} \rightarrow \{0,1\}$ *for* $v \in I_{var}$
15. }
16. *Building* $f^{-1} : \{0,1\}^{O_{var}} \rightarrow \{0,1\}^{I_{var}}$

In line 2, G_{XOR} is a set of XOR gates discovered by XORMIN on line 8. It will help to speedup the process of enumerating satisfying assignments of $F_E \wedge v \equiv 1$ on line 5. More details will be given in subsection 4.3.

Line 3 will iterate through all input boolean variables $v \in I_{var}$, and build a function f_v^{-1} for it in **line 14**, f^{-1} can be built from all such f_v^{-1} in **line 16**. The detail of building f_v^{-1} and f^{-1} will be given in subsection 4.4.

In line 4, SA_v is a set of satisfying assignments that can make $v \equiv 1$, this set will be used in line 14 to build f_v^{-1}.

In line 5, we will iterate through all satisfying assignments A of $F_E \wedge v \equiv 1$, and minimize them in two steps.

1. **BFL** in line 7 is a modified BFL[6] that will be described in subsection 4.2.

2. **XORMIN** in line 8 will further minimize the result of BFL by discovering hidden XOR gates, and return the minimized assignment A_{XOR} and discovered XOR gate set G. This is one of our major contributions, and will be described in subsection 4.3.

In line 11, we will rule out the enumerated satisfying assignment A_{XOR} by adding its blocking clauses into F_E.

In line 12, we will add clauses to F_E for newly discovered XOR.

4.2 Minimizing Satisfying Assignments with Modified BFL algorithm

The basic idea of BFL has been introduced in subsection 2.2, so we present the modified BFL here directly.

ALGORITHM 2. ***BFL(***F_E***,A,v)***

1. *Assume* $\boldsymbol{O_{var}}$ *is boolean variable set that represents* $< o_{n+d-l}, \ldots, o_{n+d-1} >$
2. *foreach* $u \in O_{var}$ {
3.   *if(*$\boldsymbol{F_E \wedge \neg v \wedge A|_{O_{var}-\{u\}}}$ ***is unsatisfiable****)* {
4. $A \leftarrow A|_{O_{var}-\{u\}}$
5. $O_{var} \leftarrow O_{var} - \{u\}$
6.   }
7. }
8. *return* A

In line 2, we iterate through all output variable $u \in O_{var}$.

In line 3, if that formula is unsatisfiable, then $A|_{O_{var}-\{u\}}$ is sufficient to lead to $v \equiv 1$. Thus, we can remove assignment of u from A.

According to subsection 2.2, for circuits with lots of XOR gates, BFL can't minimize its satisfying assignments. Thus, algorithm 1 can't terminate in reasonable time for large XOR intensive circuits.

4.3 Minimizing Satisfying Assignments by Discovering XOR gates

Intuitively, assume that the circuit in figure 3a) has an input variable set $V = \{v_1, v_2, v_3, \ldots, v_n\}$, and it **MAY** contain a XOR $v' \leftarrow v_1 \oplus v_2$. We can avoid enumerating the assignments of v_1 and v_2 by first checking the existence of this XOR, and then, as shown in figure 3b), if this XOR actual exists, we add another XOR $x_1 \leftarrow v_1 \oplus v_2$ into this circuit, and enumerate assignments on $V \cup \{x_1\} - \{v_1, v_2\}$ instead of V.

Formally, to check existence of XOR $v' \leftarrow v_1 \oplus v_2$, for a satisfying assignment A_F of formula F_E, we define a new assignment A_{x_1} in formula (12), by first removing assignments of v_1 and v_2 from A_F, and then adding assignment of x_1 as result of XORing v_1 and v_2 into A_F.

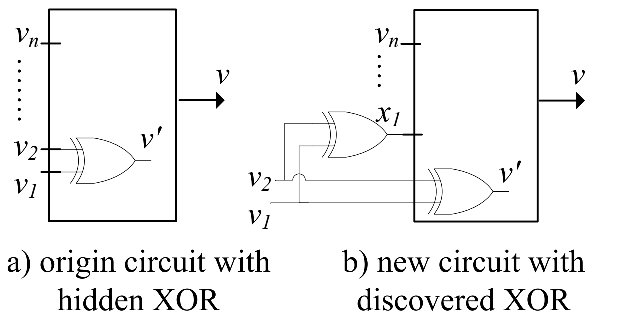

Figure 3: Discovering Hidden XOR

$$A_{x_1} \leftarrow A_F|_{O_{var}-\{v_1,v_2\}}|^{x_1 \leftarrow A_F(v_1) \oplus A_F(v_2)} \tag{12}$$

With this A_{x_1}, existence of $v' \leftarrow v_1 \oplus v_2$ can be decided by checking unsatisfiability of the following formula :

$$F_E \wedge \neg v \wedge A_{x_1} \wedge \{x_1 \equiv v_1 \oplus v_2\} \tag{13}$$

Unsatisfiable of formula (13) means that A_{x_1} can't make v to be 0, so A_{x_1} must be a satisfying assignment of $F_E \wedge v \wedge \{x_1 \equiv v_1 \oplus v_2\}$.

Thus, A_F and $A_F|_{V-\{v_1,v_2\}}|^{v_1 \leftarrow \neg A_F(v_1)}|^{v_2 \leftarrow \neg A_F(v_2)}$, that can't be merged by BFL, have now been merged into A_{x_1} with the help of a newly discovered XOR gate $x_1 \leftarrow v_1 \oplus v_2$. If we repeatedly merge assignments by checking unsatisfiability of formula (13), we can get a partial assignment of $F_E \wedge v \wedge \{x_1 \equiv \cdots \oplus \ldots\} \cdots \wedge \{x_n \equiv \cdots \oplus \ldots\}$ in formula (14), which contains 2^n total assignments, that can't be merged by BFL.

$$A_{x_1 \ldots x_n} \leftarrow A_F|_{O_{var}-\{v_1,\ldots\}}|^{x_1 \leftarrow \cdots \oplus \cdots} \ldots |^{x_n \leftarrow \cdots \oplus \cdots} \tag{14}$$

With above discussion in mind, we describe **XORMIN** below:

ALGORITHM 3. $\boldsymbol{XORMIN(F_E, A_F, v)}$

1. $G = \{\}$
2. *do* {
3. $\quad G_{new} = \{\}$ *// the set of newly discovered XOR*
4. $\quad$ *foreach* $v_1, v_2 \in O_{var}$ {
5. $\qquad$ *if(**formula (13) is unsatisfiable**)*{
6. $\qquad\quad G_{new} \leftarrow G_{new} \cup \{x_1 \leftarrow v_1 \oplus v_2\}$
7. $\qquad\quad A_F \leftarrow A_F|_{O_{var}-\{v_1,v_2\}}|^{x_1 \leftarrow A_F(v_1) \oplus A_F(v_2)}$
8. $\qquad\quad O_{var} \leftarrow O_{var} \cup \{x_1\} - \{v_1, v_2\}$
9. $\qquad\quad F_E \leftarrow F_E \wedge bcls_{A_F}$
10. $\qquad\quad F_E \leftarrow F_E \wedge \bigwedge_{\{x_1 \leftarrow v_1 \oplus v_2\} \in G_{new}} \{x_1 \equiv v_1 \oplus v_2\}$
11. $\qquad$ }
12. $\quad$ }
13. $\quad G = G \cup G_{new}$
14. } *while*$(G_{new} \neq \{\})$
15. *return* $\{A_F, G\}$

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 $v_1, v_2 \in V$, and line 5 will test if there is a XOR gate between v_1 and v_2.

Table 1: Information of Benchmarks

	XGXS	XFI	66 bits scrambler	PCIE	T2 ethernet
Line number of verilog source code	214	466	24	1139	1073
Number of registers	15	135	58	22	48
Data path width	8	64	66	10	10

Table 2: Result of Checking Parameterized Complementary Assumption

	XGXS	XFI	66 bit 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

4.4 Building f_v^{-1} and f^{-1}

The complementary function $f^{-1} : O^l \rightarrow I$ is the function that takes $< o_{n+d-l}, \ldots, o_{n+d-1} >$, and computes 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}.

Thus, f^{-1} in boolean domain is $f^{-1} : \{0,1\}^{O_{var}} \rightarrow \{0,1\}^{I_{var}}$. Then building f^{-1} can be partitioned into multiple tasks, each task builds a boolean function $f_v^{-1} : \{0,1\}^{O_{var}} \rightarrow \{0,1\}$ for a $v \in I_{var}$.

To build f_v^{-1}, let's assume that SA_v is the set of satisfying assignments that makes v to take on 1. Then f_v^{-1} can be defined as :

$$f_v^{-1}(x) = \begin{cases} 1 & \exists A \in SA_v.st.x \equiv A(x) \\ 0 & else \end{cases}$$

5. EXPERIMENTAL RESULTS

5.1 Benchmarks

Our approach is the first one that can synthesize complementary circuits automatically, so we can't compare it with other research results.

Temporal logic synthesis is a research topic that is somewhat close to us, but it can't be scaled to large circuits, and no commercial available IP cores are written in temporal logic. Thus, it's impossible to compare our result with temporal logic synthesis.

Table 1 shows some information of following benchmarks.

1. The first benchmark is a XGXS encoder that is compliant to clause 48 of IEEE-802.3ae 2002 standard [11].

2. The second benchmark is a XFI encoder that is compliant to clause 49 of the same IEEE standard.

3. The third benchmark is a 66 bit scrambler that is used to make a data sequence to have enough transitions between 0 and 1, such that it can run through high speed noisy serial transmission channel.

Table 3: Result of Building Complementary Circuits

		XGXS	XFI	66 bit scrambler	PCIE	T2 ethernet
BFL only	run time(seconds)	32.67	$> 10,000$	8.56	$> 10,000$	$> 10,000$
	line number of generated verilog	2927	N/A	11882	N/A	N/A
BFL+ XORMIN	run time	1.52	2939.47	11.97	47.55	36.64
	line number of generated verilog	1525	48829	4723	11254	16616

4. The fourth benchmark is a PCIE physical coding module.
5. The fifth benchmark is Ethernet module of Sun's OpenSparc T2 processor.

5.2 Experimental Results

Table 2 shows the run time of checking parameterized complementary assumption on these circuits, and the discovered proper values of parameters.

Table 3 shows the results of generated verilog description of complementary circuits. With **BFL only**, complementary circuits of the three most complex circuits: XFI, PCIE and T2 ethernet, can't be built within 10,000 seconds. While with **BFL and XORMIN**, we can finally build all complementary circuits successfully within 3000 seconds.

These generated complementary circuits are all been verified by dynamic simulations.

6. RELATED WORKS

6.1 Temporal Logic Synthesis

Automatically synthesis of program from logic specification is first identified as Church's problem in 1962[12]. Some early researches [13, 14] solve this problem by reducing it to checking emptiness of tree automata.

With the invention of temporal logic in the early 1980s, this problem was considered again [15, 16]. But in 1989, A. Pnueli and R. Rosner[17] pointed out that the complexity of LTL synthesis was double exponent in the size of the formula.

This high complexity drives researchers turning their focus on finding smaller but still useful subset of temporal logic, such that synthesis problem can be solved with lower complexity.

One line of research [18, 19, 20] focused on the so-called generalized reactive formulas of the form: $(\Box\Diamond p_1 \wedge \cdots \Box\Diamond p_m) \rightarrow (\Box\Diamond q_1 \wedge \cdots \Box\Diamond q_n)$. Complexity of solving synthesis problem for such formula is $O(N^3)$.

The other line of research focused on finding efficient symbolic algorithm [23] for expensive safra determination algorithm [21] on a useful formula subset, or just avoiding it[22].

6.2 Satisfying Assignments Enumeration

ALLSAT algorithms all try to enlarge total satisfying assignment by removing irrelevant variables from total satisfying assignment, such that a large cube that contains more total satisfying assignment can be obtained.

The first such approach is proposed by K. L. McMillan [5]. He constructs an alternative implication graph in SAT solver, which records the reasoning relation that leads to the assignment of a particular object variable obj. All variables outside this graph can be ruled out from the total assignment. Kavita Ravi et al.[6] and P. P. Chauhan et al.[10] remove those variables that can make $obj \equiv 0$ satisfiable one by one. Shen et al.[8] and HoonSang Jin et al.[4, 7] use a conflict analysis based approach, to remove multiple irrelevant variables in one SAT run. Orna Grumberg et al.[9] separates the variable set into important subset and non-important subset. Variables in important subset have higher decision priority than non-important ones. Thus, the important subset form a search tree, each leaf is another search tree for non-important set. Cofactoring [24] qualifies out non-important variables by setting them to constant value returned by SAT solver.

6.3 AND-XOR Logic Synthesis

Classical logic synthesis works on AND-OR network. Its kernel is two-level logic minimization, which tries to find a small sum-of-products expression for boolean function f. It is obvious that such two-level logic minimization algorithms are very similar to satisfying assignments enumeration described in previous subsection, except that they don't work on SAT solvers.

Three most well known two-level logic minimization algorithms are Quine-McCluskey[26], Scherzo[27], and Espresso-II[28].

Just like state-of-the-art ALLSAT that can't deal with XOR-intensive circuits efficiently, classical logic synthesis also has the same problem. So many researchers propose synthesis algorithms that target XOR-intensive circuits.

One research direction focuses on extending classical two-level AND-OR minimization to two-level AND-XOR network [29, 30]. These works normally describe circuits with most general ESOP (exclusive sum of product) expressions. But very high computation complexity of these approaches prevents them from handling large circuits.

Another line of research relies on Reed-Muller expansion[31], one of its most used variant is Fixed Polarity Reed-Muller Form (FPRM) given by Davio and Deschamps[32], in which a variable can have either positive or negative polarity. Some related works that rely on FPRM are [33, 34, 35].

7. CONCLUSIONS AND FUTURE WORKS

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

One possible future work is to automatically generate assertions that rule out invalid input data patterns, such that the users can be free from the burden of inspecting documentation and writing assertions.

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

8. ACKNOWLEDGMENTS

This work was supported in part by Chinese NSF contract 60603088, and Supported by the Program for Changjiang Scholars and Innovative Research Team in University No IRT0614. Thanks to anonymous reviewers for their effort in reviewing this paper.

9. REFERENCES

[1] Chris Kozup. Is 802.11n Right for You? Mobility blog. 2008.

[2] Stephen J. Dubner. What Are the Lessons of the Blu-Ray/HD-DVD Battle? A Freakonomics Quorum. The New York Times. 2008.

[3] M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In DAC'01, pp 530-535, 2001.

[4] HoonSang Jin , Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In DAC'05, pp 750-753, 2005.

[5] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In CAV'02, pp 250-264, 2002.

[6] Kavita Ravi, Fabio Somenzi. Minimal Assignments for Bounded Model Checking. In TACAS'04, pp 31-45, 2004.

[7] H. Jin, H. Han, and F. Somenzi. Efficient conflict analysis for finding all satisfying assignments of a Boolean circuit. In TACAS'05, pp 287-300, 2005.

[8] ShengYu Shen, Ying Qin, Sikun Li. Minimizing Counterexample with Unit Core Extraction and Incremental SAT. In VMCAI'05, pp 298-312, 2005.

[9] 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.

[10] P. P. Chauhan, E. M. Clarke, and D. Kroening. A SAT-based algorithm for reparameterization in symbolic simulation. In DAC'04, pp 524-529, 2004.

[11] IEEE Std. 802.3ae-2002. Amendment to IEEE Std 802.3-2002

[12] Alonzo Church. Logic,Arithmetic and Automata. International Congress of Mathematicians, pp 23-35, 1962

[13] J.R. Buchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. Transaction American Mathematic Society, Vol 138:295-311, 1969.

[14] M.O. Rabin. Automata on Infinite Objects and Church's Problem, volume 13 of Regional Conference Series in Mathematics. American Mathematic Society, 1972.

[15] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In IBM Workshop on Logics of Programs,LNCS 131, pp 52-71, 1981.

[16] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. ACM Trans. Prog. Lang. Sys., 6:68-93, 1984.

[17] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In Proc. 16th ACM Symp. Princ. of Prog. Lang.,pages 179-190, 1989.

[18] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In IFAC Symposium on System Structure and Control, pages 469-474. Elsevier, 1998.

[19] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. ACM Trans. Comput. Log., 5(1):1-25,2004.

[20] N. Piterman, A. Pnueli and Y. Saar, Synthesis of Reactive(1) Designs, in VMCAI'06, pp 364-380, 2006.

[21] S. Safra. Complexity of Automata on Infinite Objects. PhD thesis, The Weizmann Institute of Science, Rehovot, Israel, March 1989.

[22] Aidan Harding, Mark Ryan, and Pierre-Yves Schobbens. A New Algorithm for Strategy Synthesis in LTL Games. in TACAS'05, pp 477-492, 2005.

[23] Oded Maler, Dejan Nickovic and Amir Pnueli. On Synthesizing Controllers from Bounded-Response Properties. In CAV'07, pp 95-107, 2007.

[24] M. K. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In ICCAD'04, pp 510-517, 2004.

[25] Mealy, George H. A Method for Synthesizing Sequential Circuits. Bell Systems Technical Journal v 34, pp1045-1079, 1955.

[26] E.J. McCluskey. Logic Design Principles. Prentice-Hall, 1986.

[27] O. Coudert. On solving covering problems. In DAC'96, 1996.

[28] R. Rudell and A. Sangiovanni Vincentelli. Multiple valued minimization for PLA optimization. IEEE Transactions on CAD,6(5), pp 727-750, 1987.

[29] P.W. Besslich and M. Riege. An efficient program for logic synthesis of Mod-2 Sum Expressions. In Euro ASIC'91, pp 136-141, 1991.

[30] T. Sasao. AND-EXOR expressions and their optimization. Kluwer Academic Publishers, Editor, Logic Synthesis and Optimization, Boston, 1993.

[31] I. Reed. A class of multiple-error-correcting codes and their decoding scheme. IRETrans. on Inf. Theory, PGIT-4:48-49, 1954.

[32] M. Davio, Y. Deschamps, and A. Thayse. Discrete and switching Functions. George and McGraw-Hill, NY, 1978.

[33] A. Sarabi and M. Perkowski. Fast exact and quasi-minimal minimization of highly testable Fixed-Polarity AND/XOR canonical networks. In DAC'92, pp 30-35, 1992.

[34] Rolf Drechsler, Bernd Becker, Michael Theobald. Fast OFDD based minimization of fixed polarity Reed-Muller expressions. in EURO-DAC, 1994.

[35] Unni Narayanan and C. L. Liu. Low power logic synthesis for XOR based circuits. in ICCAD'97, pp 570-574, 1997.