Synthesizing Complementary Circuits Automatically

ABSTRACT

One of the most difficult jobs in designing communication 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 E's complementary circuit E^{-1} recovers this information. In order to ease this job, we propose a novel two steps approach to synthesize complementary circuit E^{-1} from E fully automatically. First, we assume 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 the proper value of these parameters. Second, with parameters value and 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 communication 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 circuit 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

Synthesizing,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 format, even still in non-standardized draft status, are implemented in chips and pushed to market, to maximize the chance of being accepted by consumer 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 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 E's complementary circuit E^{-1} recovers this information. Many factors significantly complicate the job of designing and verifying such circuit pairs. For examples, 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 i can be uniquely determined by its output alphabet sequence o. We call this assumption **parameterized complementary assumption**. We use a SAT solver to check this assumption and find out proper value for some parameters that make this assumption hold.

2. In the second step, with the SAT instance and parameters value obtained in the first step, we build a circuit E^{-1} with an efficient satisfying assignment 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 circuit E's SAT instance.

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 value 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 SAT solver

For a Boolean formula F over variable set V, the **SAT problem** is to find an **assignment** $A : V \to \{0, 1\}$, such that F can be evaluated to 1. **Unsatisfiable formula** is a formula without any satisfying assignment .

For an assignment $A : U \to \{0, 1\}$,if $U \subset V$, then A is a **partial assignment**, if $U \equiv V$, then A is a **total assignment**.

For an assignment $A : U \to \{0, 1\}$, and $W \subset U$, $A|_W : W \to \{0, 1\}$ 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 variables $v \notin W$.

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

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

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

Normally, formula F is represented in **CNF** format, in which **formula** $F = \bigwedge_{cl \in CL} cl$ is a conjunction of its clauses set CL, and **clause** $cl = \bigvee_{l \in Lit} l$ is a disjunction of its literals set Lit, and **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 isn't satisfying assignment of $F \wedge bcls_A$. So $bcls_A$ can be inserted into SAT solver to prevent A from being 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 by repeatedly calling a SAT solver, and adding blocking clauses of satisfying assignments 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 assignment. 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 new merged partial assignments contain only two assignments to individual variables, and are much more succinct than previous three total assignments.

Formally, assume $\boldsymbol{F}$ is the 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 we can merged $A|_{V-\{v\}}|^{v\leftarrow 0}$ 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 to save space. 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 an 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 the circuit E satisfies parameterized complementary assumption, in other word, if its input sequence can be recovered from its output sequence.

3.1 Parameterized Complementary Assumption

Our algorithm care about the input and output sequence of circuits 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 alphabet I
4. A finite set of 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 1. This relationship is built by unfolding the transition function T and output function G by d times, as shown in formula (2).

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

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 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.

Thus, $f^{-1} : O^l \to I$ is now a boolean function that takes finite length sequence $< o_{n+d-l+1}, \ldots, o_{n+d} >$ as input, and computes i_n.

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

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

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

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

In formula (3), the first line is same as formula (2), the second line is a copy of formula (2), except that its variables are all renamed by appending a prime. The third line constraints their output sequences $< o_{n+d-l+1}, \ldots, o_{n+d} >$ and

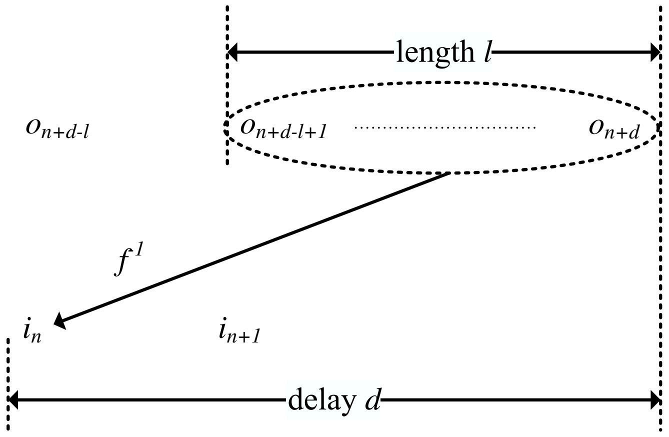


Figure 2: f^{-1} and its parameters

$< o'_{n+d-l+1}, \ldots, o'_{n+d} >$ 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**. This means, for a $< o_{n+d-l+1}, \ldots, o_{n+d} >$, 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**. This means parameterized complementary assumption is satisfied, a $f^{-1} : O^l \to I$ 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 this restriction is expressed as an assertion predicate $R : I \to \{0, 1\}$, 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 and $i'_n \in \{i_m | R(i_m) \equiv 0\}$, they will both be mapped to the same error output alphabet $e \in O$. This will prevent our approach from distinguish 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 changed into following formula (4) and (5), in which bold formulas are used to account for predicate R.

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

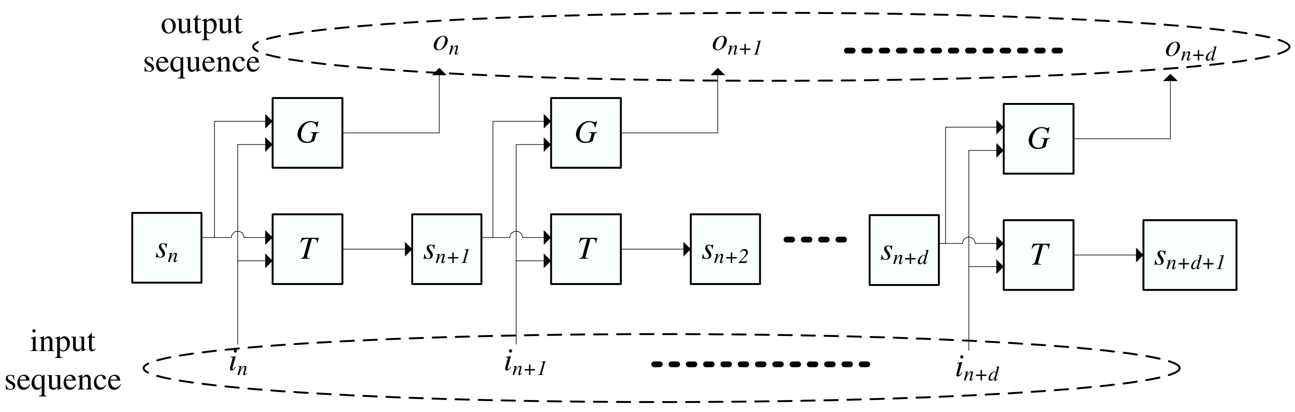


Figure 1: Unfolding transition function of mealy finite state machine

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

3.3 Approximating Reachable State Set with Prefix Sequence

In last subsection, we have constraint 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 circuit E can be modeled by mealy state machine $M_E=(S,s_0,I,O,T,G)$. Its reachable state set as:

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

Thus, to rule out unreachable s_n, we need to change formula (4) and (5) into formula (7) and (8) below:

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

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

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 doesn't affected by unreachable state set, but with very high computation complexity in computing reachable state set RS_E.

So can we find a tradeoff between this two extremes, a method that 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:

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

It is obvious that 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 :

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

So now, in addition to parameter d and l, we have the third parameter p to be searched. In order to account for RS_E^p, we need to change formula (4) and (5) to:

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

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

Now put it altogether, with formula (10) and (11), We iterate though 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.

4. BUILDING COMPLEMENTARY CIRCUIT WITH SATISFYING ASSIGNMENTS ENUMERATION ALGORITHM DESIGNED FOR XOR INTENSIVE CIRCUITS

If we find proper value for parameters d,l and p in section 3, we can then 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 3.2, $f^{-1} : O^l \rightarrow I$ can be built from formula (10) by enumerating satisfying assignments to $< o_{n+d-l+1}, \dots, o_{n+d} >$ and i_n.

Assume the set of all total satisfying assignments of formula (10) is $\{A_1, \dots, A_t\}$, then f^{-1} can be defined as in following equation.

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

But this naive approach suffers from searching 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 exist many much more efficient approaches to enumerate satisfying assignments of SAT instance [4, 5, 6, 7, 8, 9, 10, 24]. The basic idea of these approaches has been introduced in subsection 2.2.

But they 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 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 $\boldsymbol{I_{var}}$ and $\boldsymbol{O_{var}}$ are respectively boolean variables set that represent $i_n$ and $< o_{n+d-l+1}, \dots, o_{n+d} >$.
    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.         $F_E \leftarrow F_E \wedge \bigwedge_{\{x \leftarrow XOR(x_1,x_2)\} \in G} x \equiv XOR(x_1, x_2)$
13.     }
14.     Building $f_v^{-1} : O^l \rightarrow \{0,1\}$
15. }
16. Building $f^{-1} : O^l \rightarrow I$
```

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

Line 3 will iterate through all input boolean variables $v \in I_{var}$, and build a function $f_v^{-1} : O^l \rightarrow \{0,1\}$ for it at **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.

At 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}.

At line 5, we will iterate through all satisfying assignment 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 XOR gate set G. This is one of our major contributions, and will be described in subsection 4.3.

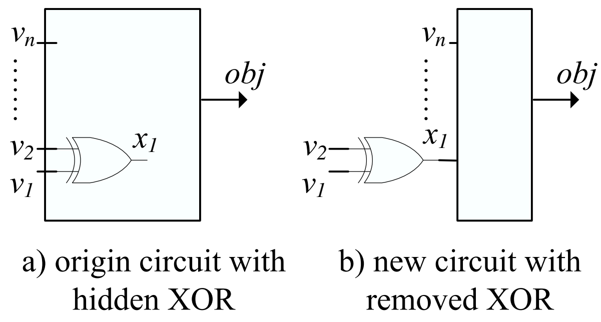
At line 11, we will rule out the set of enumerated satisfying assignments by adding blocking clauses into F_E.

At line 12, we will add clauses into F_E for newly discovered XOR.

4.2 Minimizing Satisfying Assignment 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)***



v_n
obj
x_1
v_2
v_1
a) origin circuit with hidden XOR
b) new circuit with removed XOR

Figure 3: Removing hidden XOR

```
1. Assume $\boldsymbol{O_{var}}$ is boolean variables set that represents
   $< o_{n+d-l+1}, \dots, o_{n+d} >$
2. foreach $u \in O_{var}$ {
3.     if($\boldsymbol{F_E \wedge \neg v \wedge A|_{O_{var}-\{u\}}}$ is unsatisfiable) {
4.         $A = A|_{O_{var}-\{u\}}$
5.         $O_{var} = O_{var} - \{u\}$
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 assignment of $O_{var} - \{u\}$ 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 it's satisfying assignment. Thus, algorithm 1 can't terminate in reasonable time for large circuits.

4.3 Minimizing Satisfying Assignment by Discovering XOR gates

Intuitively, assume circuit in figure 3a) has input variables set $V = \{v_1, v_2, v_3, \dots, v_n\}$, and it contains a XOR $x_1 = v_1 \oplus v_2$. We can avoid enumerate the assignments of v_1 and v_2 by moving this XOR out of the circuit, and enumerate assignments of $V \cup \{x_1\} - \{v_1, v_2\} = \{\boldsymbol{x_1}, v_3, ..., v_n\}$.

In order to remove this XOR, we must first make sure that there actually exists a XOR in this circuit.

For a satisfying assignment A_F of formula F, 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.

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

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

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

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

Thus, A_F and $A_F|_{V-\{v_1,v_2\}}|^{v_1 \leftarrow \neg A_F(v_1)}|^{v_2 \leftarrow \neg A_F(v_2)}$ have been merged into A_{x_1} with the help of a newly discovered XOR gate $x_1 = v_1 \oplus v_2$. If we repeatedly merge assignments by checking unsatisfiability of formula (13), we can get a partial assignment of $F \wedge v \wedge \{x_1 \equiv \dots \oplus \dots\} \dots \wedge \{x_n \equiv \dots \oplus \dots\}$ in formula (14), which contains 2^n total assignments.

$$A_{x_1 \dots x_n} \leftarrow A_F|_{V-\{v_1,\dots\}}|^{x_1 \leftarrow XOR(\dots)} \dots |^{x_n \leftarrow XOR(\dots)} \tag{14}$$

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

Algorithm 3. $\boldsymbol{XORMIN(F, A_F, v)}$

```
1.  $G = \{\}$
2.  do {
3.      $G_{new} = \{\}$ // the set of newly discovered XOR
4.      foreach $v_1, v_2 \in V$ {
5.          if(formula (13) is unsatisfiable){
6.              $G_{new} \leftarrow G_{new} \cup \{x_1 \leftarrow XOR(v_1, v_2)\}$
7.              $A_F \leftarrow A_F|_{V-\{v_1,v_2\}}|^{x_1 \leftarrow XOR(A_F(v_1), A_F(v_2))}$
8.              $V \leftarrow V \cup \{x_1\} - \{v_1, v_2\}$
9.          }
10.     }
11.     $G = G \cup G_{new}$
12. } while($G_{new} \neq \{\}$)
13. 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.

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

The complementary function $f^{-1} : O^l \to I$ is the function that takes $< o_{n+d-l+1}, \dots, o_{n+d} >$, and computes i_n . Assume i_n is represented by boolean variables set I_{var}, and $< o_{n+d-l+1}, \dots, o_{n+d} >$ is represented by boolean variables set O_{var}.

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

To build f_v^{-1}, let's assume that SA_v is the set of satisfying assignments that make 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's 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 XGXSPCS encoder that is compliant to clause 48 of IEEE-802.3ae 2002 [11].
2. The second benchmark is a XFIPCS 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.
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 value of parameters.

The Table 3 shows the results of generated verilog description of complementary circuits. With **BFL only**, complementary circuits of the three most complex circuits: XFICPS, PCIE and T2 ethernet, can't be generated 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 at 1962[12]. Some early researches [13, 14] solve this problem by reducing it to checking emptiness of tree automata.

After invention of temporal logic at early 1980s, this problem has been consider again [15, 16]. But at 1989, A. Pnueli and R. Rosner[17] points out that the complexity of LTL synthesis is double exponent in the size of the formula. This high complexity drives researchers turning their focus to 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] focus 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 focus 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

There are two research directions for satisfying assignments enumeration. One is bottom up, which removes irrelevant variables from total assignment. The other is top down, which adds relevant variable's assignment into an empty set, to form a smaller and smaller assignment set.

The first bottom up 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 satisfiable one by one. Shen et al.[8] improve [6] with UNSAT core extraction, which can remove multiple irrelevant variables with only one SAT solving. 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] separate the variables 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] is the only top down approach, which starts from an empty set of assignment, and add relevant variable's assignments one by one.

7. CONCLUSIONS AND FUTURE WORKS

In this paper, we propose the first fully automatic approach that synthesizes complementary circuits for communication applications. According to the 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 the documentation and writing assertions.

Another possible future work is how 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 deal with by our approach.

8. REFERENCES

Table 1: Information of Benchmarks

	XGXSPCS	XFIPCS	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: Results of Checking parameterized complementary assumption

	XGXSPCS	XFIPCS	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

[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.

Table 3: Results of Generating Complementary Circuits

		XGXSPCS	XFIPCS	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