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 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 use a SAT solver to check whether the input sequence of E can be uniquely determined by its output sequence. Second, we build the complementary circuit E^{-1} 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 ~~run~~ our algorithm on several complex encoders from industrial projects, including PCIE and 10G ethernet, and successfully generate 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 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 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 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 use a SAT solver to check ~~that,~~ whether there exists a valuation for some parameters, ~~such that~~, the input alphabet sequence of E can be uniquely determined by its output alphabet sequence. We call this **parameterized complementary condition**.
2) In the second step, with the SAT instance and parameter values obtained in the first step, we build the circuit E^{-1} by characterizing its boolean function, with an efficient all solution SAT solver(abbreviated as **ALLSAT**) based on discovering XOR gates and extracting unsatisfiable cores.

We implement our algorithm on zchaff [3], and run it on several complex encoder circuits from industrial projects, including PCIE and 10G Ethernet. We can build complementary circuits for all of them within 1000 seconds. All these experimental result and related programs can be downloaded from http://www.ssypub.org.

The contribution of this paper is twofold: 1) We propose the first approach to decide if it's possible to recover input sequence of a 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 II presents background material. Section III presents how to check parameterized complementary condition, and how to find out proper values of its parameters. Section IV presents how to characterize the boolean function of complementary circuit. Section V presents how to build the complementary circuit from its boolean function. Section VI presents experimental results. Section VII presents related works. Section VIII concludes with a note on future work.

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** $A : V \rightarrow \{0, 1\}$, ~~such 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 satisfying assignment is called **SAT solver**. Some famous SAT solvers are zchaff [3] ,Berkmin [4] and MiniSAT [5].

Normally, 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 clauses set CL, and a **clause** $cl = \bigvee_{l \in Lit} l$ is a disjunction

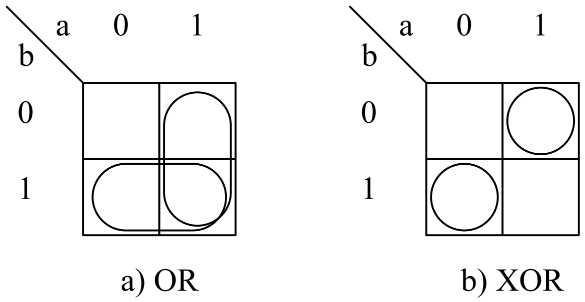

Fig. 1. Satisfying assignments for simple gates

of its 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 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,

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 $b \in \{0, 1\}$, $A|^{u \rightarrow b}$ is the **extension** of A on u, its definition is:

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

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

(1)

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

An unsatisfiable formula often has many clause subsets that are also unsatisfiable, these subsets are called **unsatisfiable core**. Some unsatisfiable core extraction algorithms are proposed by Goldberg [6] and Zhang [7].

B. All Solution SAT Solver

State-of-the-Art SAT solvers normally only find one total satisfying assignment. But many applications, such as two-level logic minimization [8], need to enumerate all satisfying assignments.

Such technologies that enumerate all satisfying assignments of a formula are called **all solution SAT(ALLSAT)**. It is obvious that we can enumerate all total satisfying assignments by repeatedly calling a SAT solver, and inserting 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 $z = a \vee b$ in figure 1a), its total satisfying assignments that can make $z \equiv 1$ are $\{a = 1, b = 0\}$,$\{a = 1, b = 1\}$ and $\{a = 0, b = 1\}$. They 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. $\{a = 1, b = 1\}$ and $\{a = 0, b = 1\}$ can also be merged into $\{b = 1\}$, in which a 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, $\boldsymbol{v} \in V$ 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 [9]–[16] share this idea of satisfying assignments minimization. ~~We will only present here one of them~~, BFL(brutal force lifting) algorithm [11]:

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

Line 12 will test ~~that if~~ 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 ~~that covers~~ 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 $\{a = 0, b = 1\}$, ~~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 ~~difficul~~ty of state-of-the-art ALLSAT approaches, we will solve this problem in section IV.

C. Checking Reachability with Bounded Model Checking

Description of our algorithm will largely follow that of **bounded model checking (BMC)** [17], so we present here briefly how to check reachability in BMC.

Definition 1: **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 $L : S \rightarrow 2^A$ with atomic propositions set A.

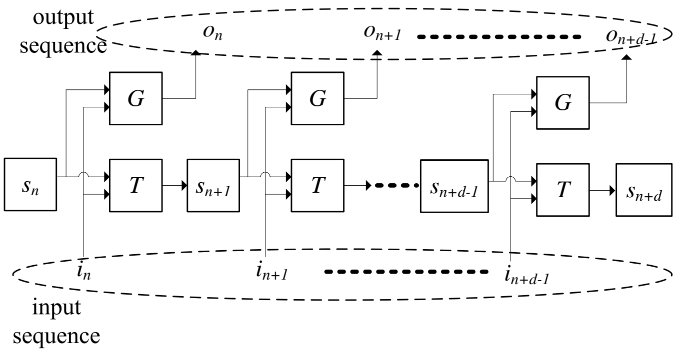

Fig. 2. Unfolding Transition Function of Mealy Finite State Machine

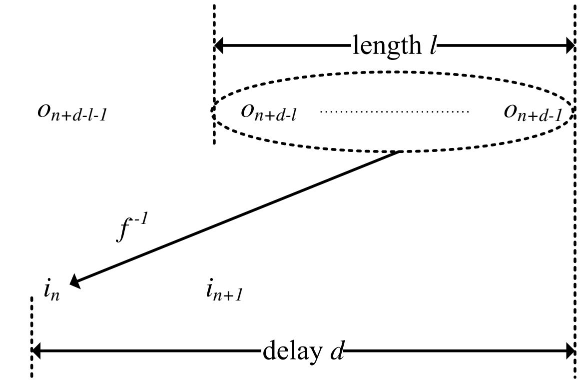

Fig. 3. f^{-1} and Parameters of o's Finite Length Subsequence

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

To save space, we only present here how to check reachability in BMC, more details can be found in [17]. Let the safety property under verification be $ASSERT$, the goal of BMC is to find a state that violates $ASSERT$. Then BMC problem with bound b can be expressed as:

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

(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 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:

(4)

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

Assume that I_{var} and O_{var} are respectively boolean variable sets that represent i_n and $< o_{n+d-l}, \ldots, o_{n+d-1} >$, then parameterized complementary condition holds if and only if the following formula (5) is unsatisfiable.

(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 alphabet 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. 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, a f^{-1} exists for this pair of d and l. We will characterize f^{-1} with formula (3) in **algorithm 2** of section IV.

B. 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 $e \in O$. 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 choose to employ an assertion based mechanism, such that the user can code these restrictions R into their script or source code.

Thus, formula (3),(4) and (5) should be rewritten as following formula (6), (7) and (8), in which bold formulas are used to account for predicate R.

(6)

(7)

(8)

C. Approximating Reachable State Set

In last subsection, we have constrained the valid pattern of i_m. But the s_n in figure 2 still hasn't been constrained yet. It may be outside of reachable state set of circuit E. This may make checking parameterized complementary condition fail unnecessary.

We can solve this problem by computing reachable state set RS as in formula (10), and constrain that $s_n \in RS$:

(9)

(10)

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

Computing RS is very expensive. In order to avoid computing RS, we want to approximate RS with:

(11)

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

But unfortunately, RS is not subset of any $RS^{S \rightarrow p}$, because there may exist some state $s \in RS$, that when starting from initial state s_0, can only be reached within p steps, and can't be reached with more than p steps. For example, a counter shown below that counts from 0 to 4, and then stay at 4 forever.

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

On the other hand, because circuit E and E^{-1} run in a never ending way, we can safely assume that there are always a prefix state transition sequence with enough length before the current state. Thus, for any particular p, we only need to consider $\bigcup_{q>p} RS^{s_0 \rightarrow q}$ instead of RS. Obviously, $\bigcup_{q>p} RS^{s_0 \rightarrow q}$ is a subset of $RS^{S \rightarrow p}$. Thus, we can use $RS^{S \rightarrow p}$ as an over approximation of $\bigcup_{q>p} RS^{s_0 \rightarrow q}$.

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

(12)

(13)

(14)

Now put it altogether, 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 complementary circuit E^{-1}.

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

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

A. Partitioning f^{-1}

According to section III-C, The complementary function $f^{-1} : O^l \rightarrow I$ is the function that takes $< o_{n+d-l}, \ldots, o_{n+d-1} >$, and computes i_n . It can be characterized from 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 $f^{-1} : \{0,1\}^{O_{var}} \rightarrow \{0,1\}^{I_{var}}$, and can be defined as:

(15)

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

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 :

(16)

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

There exists some much more efficient approaches to enumerate satisfying assignments of SAT instance [9]–[16]. According to subsection II-B, they all try to merge satisfying assignments in SA_v by removing irrelevant variables from each $A \in SA_v$, such that the size of SA_v can be reduced.

But they are still not efficient enough for our application. The essential reasons that lead to this inefficiency, and our improvements 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. ~~The function~~ $\boldsymbol{SIMPLIFY}$ simplify F_E to F_E^v before passing it to main body of $ALLSAT$ ~~and being used repeatedly,~~ by removing these redundant clauses with unsatisfiable core extraction.
3) The function BFL in algorithm 1 can remove at most 1 irrelevant variables within each SAT solving. Its improved version $\boldsymbol{BFL\_UNSAT}$ will 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} REMOVED

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

C. Simplifying Formula by Extracting Unsatisfiable Core

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

So we propose the function $\boldsymbol{SIMPLIFY(F_E, v)}$ to simplify F_E to F_E^v for every particular v:

1) In the first step, extracting unsatisfiable core F_E^{UNSAT} from following formula (17) with depth first approach in Lintao Zhang et al. [7]:

(17)

Unsatisfiability of this formula will be proven in Theorem 1 below.

2) In the second step, ~~intersecting~~ the clauses set of F_E and F_E^{UNSAT} to get formula F_E^v

(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 outmost layer.

If for any v, formula (17) is satisfiable, then 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 variables 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 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 definition of F_E in formula (12), (13) and (14), $A|_{O_{var}}$ can force a unique value on v:

1) If that value is 1, then $F_E \wedge v$ is satisfied. This means A is a satisfying assignment of $F_E \wedge v$.
2) if that value is 0, then $A|_{O_{var}}$ can force v to different value on F_E and F_E^v. This is contradictive to formula (14), so value on v can't be 0.

Thus, this theorem is proven. ■

So now, we can be sure that replacing F_E with F_E^v to characterize f_v^{-1} is safe. 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, $\boldsymbol{BFL}$ [11] is used to remove those variables ~~that are~~ irrelevant to forcing v to be 1. The implementation of $\boldsymbol{BFL}$ has been shown in algorithm 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. [7]:

2) ~~In the second step,~~ computing new A by intersecting 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** REMOVED

Its correctness is proven below:

Theorem 3 (): **After $\boldsymbol{BFL\_UNSAT}$ 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 the 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 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 result 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 return by $\boldsymbol{BFL\_UNSAT}$ is a minimal assignment. That means, removing any variable from A will make it ~~no longer being able~~ to force v to be 1.

To make A to 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 function such as f_v^{-1} that **MAY** contains XOR gate $z = v_1 \oplus v_2$, we can first check whether this XOR gate actually exists, and then merge these two satisfying assignments by replace 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:

(19)

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

(20)

Unsatisfiable 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 $A_{\overline{v}_1,\overline{v}_2}$, that can't be merged by BFL, can be merged into:

(21)

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

(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 checking 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 above discussion ~~in mind~~, we describe **XORMIN** below:

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

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.

V. Building Circuit E^{-1} from f^{-1}

A. Instancing 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 the circuit E^{-1}, as shown in right-top side of the figure 4, we need to instance $l-1$ 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. Instancing Discovered XOR Gates

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

(23)

Then the support set of boolean function f^{-1} will be changed from O_{var} to $O_{var} \cup U_{xor}$. As shown in left-top side of the figure 4.

So we need to instance all XOR gates discovered by the $XORMIN$ function in the generated netlist.

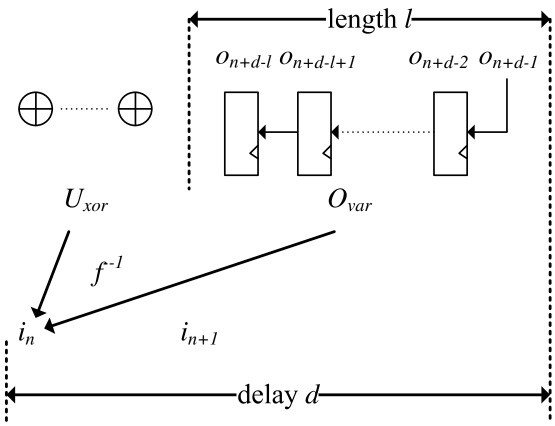

Fig. 4. Circuit structure of E^{-1}

TABLE I
INFORMATION OF BENCHMARKS

	XGXS	XFI	scrambler	PCIE	T2 ethernet
Line number of verilog source code					
#regs					
Data path width					

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 RESULT

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

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

A. Benchmarks

Table I 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 [18].
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 II
RESULTS OF CHECKING PARAMETERIZED COMPLEMENTARY CONDITION

	XGXS	XFI	scrambler	PCIE	T2 ethernet
run time (seconds)					
d					
p					
l					

TABLE III
RUN TIME OF BUILDING COMPLEMENTARY CIRCUITS

		XGXS	XFI	scrambler	PCIE	T2 ethernet
BFL only	time(s)		time out		time out	time out
BFL + XORMIN	time(s)					
	$\|F_E\|$					
	#SAT					
BFL+ XORMIN +UNSAT	time(s)					
	$\|F_E^v\|$					
	#SAT					

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

B. Writing Assertion

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

1) For the XGXS and T2 ethernet encoders, table 48-2, 48-3 and 48-4 of IEEE-802.3ae 2002 standard [18] give the pattern of valid alphabets.
2) For the XFI encoder and scrambler, figure 49-7 and table 49-1 of IEEE-802.3ae 2002 standard [18] give the pattern of valid alphabets.
3) For the PCIE physical coding module, table 4-1 of PCI Express Base Specification [19] give the pattern of valid alphabets.

C. Result of Checking Parameterized Complementary Condition

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

D. Improvement on Run Time Overhead

Table III compares the following three statistics between the BFL algorithm [11], BFL+XORMIN proposed in our previous work [31], and BFL+XORMIN+UNSAT proposed by this paper.

1) The three **time** rows compare the run time overhead of building complementary circuit. Obviously, our approach ~~can be~~ more than one order of magnitude faster than BFL only approach, and three times faster than our previous work [31].

TABLE IV
COMPARING DECODER AREA

	XGXS	XFI	scrambler	PCIE	T2 ethernet
hand written decoder					
decoder built by Shen [31]					
decoder built by our algorithm					

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 simplify 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 [31], 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 decoder. However, this dose not mean that our algorithm is better than human designer. Actually, hand written decoder often include some other logic that is irrelevant to decoder functionality.
2) For the XFI case, our circuit area is about 3 time large than hand written decoder. This means we need to improve area in future work.
3) There are no significant area difference between our algorithm and our previous work. [31].

VII. Related Works

A. Satisfying Assignments Enumeration

Existing ALLSAT algorithms all tried to enlarge total satisfying assignments by removing irrelevant variables, so that a large cube that contains more total satisfying assignment can be obtained.

The first such approach was proposed by K. L. McMillan [10]. He constructed an alternative implication graph in SAT solver, which recorded the reasoning relation that leaded 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. [11] and P. P. Chauhan et al. [15] removed those variables whose absence can't make $obj \equiv 0$ satisfiable one by one. Shen et al. [13] and HoonSang Jin et al. [9], [12] used a conflict analysis based approach to remove multiple irrelevant variables in one SAT run. Orna Grumberg et al. [14] separated 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 formed a search tree, with each leaf being another search tree for non-important set. Cofactoring [16] qualified out non-important variables by setting them to constant value returned by SAT solver.

B. AND-XOR Logic Synthesis

Classical logic synthesis worked on AND-OR network. It's 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 [21], Scherzo [22], and Espresso-II [23].

Just like state-of-the-art 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 [24], [25]. These works normally described circuits with the most general ESOP (exclusive sum of product) expressions. But very high computation complexity of these approaches prevented them from handling large circuits.

Another line of research relied on Reed-Muller expansion [26], one of its most used variant was Fixed Polarity Reed-Muller Form (FPRM) given by Davio and Deschamps [27], in which a variable can have either positive or negative polarity. Some related works that relied on FPRM are [28]–[30].

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 very complex circuits, including but not limited to PCIE and Ethernet.

One possible future work is to improve the circuit area of generated E^{-1}.

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.

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] 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] Evgueni Goldberg, Yakov Novikov. BerkMin: a fast and robust sat-solver. in DATE'02, pp 142-149, 2002.

[5] Niklas En, Niklas Srensson. An Extensible SAT-solver. in SAT'03 pp 502-518, 2003.

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

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

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

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

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu . Symbolic Model Checking using SAT procedures instead of BDDs. In DAC'99), pp 317-320, 1999.

[18] 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

[19] PCI Express Base Specification Revision 1.0. Download from http://www.pcisig.com

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

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

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

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

[24] 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.

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

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

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

[28] 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.

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

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

[31] 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