A Halting Algorithm to Determine the Existence of the Decoder

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

***Abstract*—Complementary synthesis automatically synthesizes the decoder circuit of an encoder. It determines the existence of the decoder by checking whether the encoder's input can be uniquely determined by its output. However, this algorithm will not halt if the decoder does not exist.**

To solve this problem, a novel halting algorithm is proposed. For every path of the encoder, this algorithm first checks whether the encoder's input can be uniquely determined by its output. If yes, the decoder exists; otherwise, this algorithm checks if this path contains loops, which can be further unfolded to prove the non-existence of the decoder for all those longer paths.

To illustrate its usefulness and efficiency, this algorithm has been run on several complex encoders, including PCI-E and Ethernet. Experimental results indicate that this algorithm always halts properly by distinguishing correct encoders from incorrect ones, and it is more than three times faster than the previous work.

***Index Terms*—Halting Algorithm, Complementary Synthesis**

I. Introduction

Among the most difficult tasks in designing communication and multimedia chips are the design and verification of complementary circuit pairs (E, E^{-1}), in which the encoder E transforms information into a format suitable for transmission and storage, while its complementary circuit(or decoder) E^{-1} recovers this information. **To accomplish this task**, the complementary synthesis **algorithm is proposed** [1], [2] to automatically synthesize the decoder circuit of an encoder, by checking the parameterized complementary condition(PC), that is, whether the encoder's input letter can be uniquely determined by its output sequence.

However, that algorithm will not halt if E^{-1} does not exist. Another **algorithm was proposed recently** [3] to solve this problem by first constructing a list of over-approximations of PC that is similar to onion rings, and then checking whether E is in all these rings. This algorithm is very slow and complicated. Therefore yet another halting algorithm is proposed in this paper, which is both faster and more straightforward. For every path of the encoder, this new algorithm checks two cases:

1) Just like the non-halting algorithm [1], [2], checking whether the encoder's input can be uniquely determined by its output. If yes, the decoder exists;
2) Otherwise, checking whether there are loops in the path mentioned above. As shown in Figure 1, the path in Figure 1a) contains three sub-sequences a, b and c. The sub-sequence b is a loop that can be further unfolded to obtain a longer path shown in Figure 1b). In this way, the non-existence of the decoder can be proved for all those longer paths.

This new algorithm has been implemented **in** the OCaml language. All generated SAT instances are solved with **the** Zchaff SAT solver [4]. The benchmark set includes several complex encoders from industrial projects (e.g., PCI-E [5] and Ethernet [6]), and their slightly modified variants without corresponding decoders. Experimental results indicate that this paper's algorithm always halts properly by distinguishing correct encoders from incorrect ones, and can be three times faster than the other halting algorithm [3] **proposed by us**. All these experimental results and programs can be downloaded from http://www.ssypub.org.

We distinguish two classes of complementary circuit pairs, which require different design methodologies:

1) Standard datapath-intensive circuits: These circuits often work as digital signal processing components, such as Fast Fourier Transform (FFT) and Discrete Cosine Transform(DCT). These circuits usually have standard and highly optimized implementations from various foundries and IP vendors, such as Xilinx core generator [7] and Synopsys DesignWare library [8]. **So their decoders do not need to be generated by our algorithm**.
2) Non-standard and control-intensive circuits: These circuits, such as PCI-E [5] and Ethernet [6], are often used to handle communication protocols, and typically **do not** have standard implementations. This paper's algorithm is designed for them.

The remainder of this paper is organized as follows. Background material is presented in Section II. The algorithm is introduced in Section III, while Section IV describes how to remove redundant output letters to

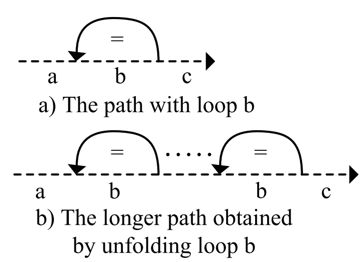

Fig. 1. Unfolding the loop to prove the non-existence of the decoder for longer path

The authors are with the School of Computer, National University of Defense Technology, ChangSha, HuNan, 410073 CHINA. e-mail: {syshen, qy123, lqxiao, kfwang, jmzhang, skli}@nudt.edu.cn, (see http://www.ssypub.org/).

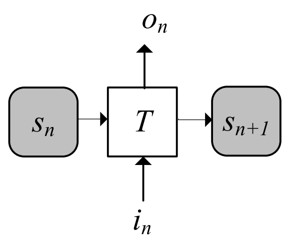

Fig. 2. Mealy finite state machine

minimize the circuit area. The experimental results are in section V and related work is discussed in section VI. Finally, Section VII concludes this paper.

II. PRELIMINARIES

A. Propositional satisfiability

The Boolean value set is denoted as $B = \{0, 1\}$. For a Boolean formula F over a variable set V, the propositional satisfiability problem(abbreviated as SAT) is to find a satisfying assignment $A : V \to B$, **so that** F **evaluates to 1**. If such a satisfying assignment exists, then F is satisfiable; otherwise, it is unsatisfiable.

A computer program that decides the existence of such a satisfying assignment is called a SAT solver, such as Zchaff [4], Grasp [9], Berkmin [10], and MiniSAT [11]. A formula to be solved by a SAT solver is also called a SAT instance.

B. Recurrence diameter

The encoder E can be modeled by a Mealy finite state machine [12].

***Definition** 1:* **Mealy finite state machine** is a 5-tuple $M = (S, s_0, I, O, T)$, consisting of a finite state set S, an initial state $s_0 \in S$, a finite set of input letters I, a finite set of output letters O, a transition function $T : S \times I \to S \times O$ that computes the next state and the output letter from the current state and the input letter.

As shown in Figure 2, as well as in the remainder of this paper, the state is represented as a gray round corner box, and the transition function T is represented by a white rectangle. The state, the input letter and the output letter at the n-th cycle are denoted as s_n, i_n and o_n respectively. The sequence of state, input letter and output letter from the n-th to the m-th cycle are denoted as s_n^m, i_n^m and o_n^m respectively. **A path is a state sequence** s_n^m **with** $\exists i_j o_j (s_{j+1}, o_j) \equiv T(s_j, i_j)$ **for all** $n \le j < m$**. A loop is a path** s_n^m **with** $s_n \equiv s_m$**.**

Kroening et al. [13] defined the state variables recurrence diameter of **the Mealy machine** M, denoted by $rrd(M)$, as the longest path that starts from an initial state and does not contain a loop.

$$rrd(M) \stackrel{def}{=} \max\{i | \exists s_0 \dots s_i i_0 \dots i_i o_0 \dots o_i : I(s_0) \wedge \bigwedge_{j=0}^{i-1} (s_{j+1}, o_j) \equiv T(s_j, i_j) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k\} \tag{1}$$

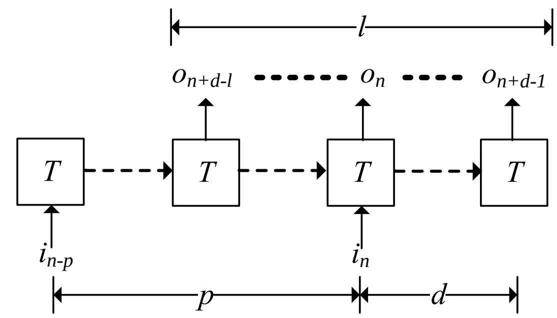

Fig. 3. The parameterized complementary condition

In this paper, a similar concept: the uninitialized state variables recurrence diameter of M, denoted by $uirrd(M)$, is defined as the longest path without loop.

$$uirrd(M) \stackrel{def}{=} \max\{i | \exists s_0 \dots s_i i_0 \dots i_i o_0 \dots o_i : \bigwedge_{j=0}^{i-1} (s_{j+1}, o_j) \equiv T(s_j, i_j) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k\} \tag{2}$$

The only difference between these two definitions is that $uirrd$ does not consider the initial state. These definitions are only used in proving the theorems below. This paper's algorithm does not need to compute these diameters.

C. The original non-halting algorithm to determine the existence of the decoder

The complementary synthesis algorithm [1] includes two steps: determining the existence of the decoder and characterizing its Boolean function. Only the first step is introduced here.

According to our previous research [1]–[3], many communication protocols are lossless, that is, every input letter can be recovered from its output sequence.

More formally, as shown in Figure 3, a sufficient condition for the existence of the decoder E^{-1} is, there exist three parameter values p, d and l, so that i_n of the encoder E can be uniquely determined by E's output sequence o_{n+d-l}^{n+d-1}. d is the relative delay between o_{n+d-l}^{n+d-1} and the input letter i_n, while l is the length of o_{n+d-l}^{n+d-1}, and p is the length of the prefix path used to rule out some unreachable states of the encoder. Thus, the parameterized complementary condition(PC) [1] can be formally defined as:

***Definition** 2:* **Parameterized complementary condition (*PC*) :** For encoder E, $E \vDash PC(p, d, l)$ holds if i_n can be uniquely determined by o_{n+d-l}^{n+d-1} in the path s_{n-p}^{n+d-1}. This equals the unsatisfiability of $F_{PC}(p, d, l)$ in Equation (3). $E \vDash PC$ is further defined as $\exists p, d, l : E \vDash PC(p, d, l)$.

$$F_{PC}(p, d, l) \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=n-p}^{n+d-1} \{(s_{m+1}, o_m) \equiv T(s_m, i_m)\} \\ \wedge & \bigwedge_{m=n-p}^{n+d-1} \{(s'_{m+1}, o'_m) \equiv T(s'_m, i'_m)\} \\ \wedge & \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \\ \wedge & i_n \neq i'_n \end{array} \right\} \tag{3}$$

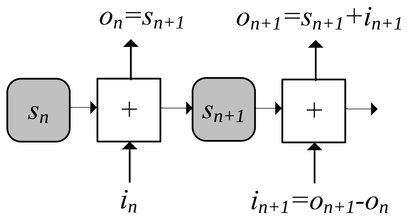

Fig. 4. The circuit that breaks the causal relation

The second and third lines of Equation (3) correspond respectively to two paths of the encoder. The only difference between them is that a prime is appended to every variable in the third line. The fourth line forces the output sequences of these two paths to be the same, while the fifth line forces their input letters to be different.

This non-halting algorithm [1] just **iterates** through all valuations of p,d and l, from small to large, until one valuation of p,d and l that makes Equation (3) unsatisfiable is found. Then the existence of the decoder is proved. However, if the decoder does not exist, this algorithm will never halt. This problem will be solved in the next section.

According to Figure 3, if $l > d$, then, to compute the value of input i_n, one may need to know the output o_m where $m < n$. This looks like breaking the causal relation. **Informally, in different states, the encoder may produce different outputs for the same input i_n. The knowledge of outputs o_m where $m < n$ may be necessary to identify the state of the encoder in which the input i_n has been processed.** This problem can be further explained with the circuit in Figure 4. Assume its transition function T is:

$$\begin{array}{c} s_{n+1} = i_n + s_n \\ o_n = i_n + s_n \end{array} \tag{4}$$

Intuitively, this circuit adds its input to its current state, and puts the result to its output and next state. So, to recover the input letter i_{n+1}, the output letter o_n must be subtracted from o_{n+1}. In this case, l is 1, and d is 0. This explains why l can be larger than d.

III. A Halting Algorithm to Determine the Existence of the Decoder

A. Determining the non-existence of the decoder

PC in Definition 2 only defines how to determine the existence of the decoder E^{-1}. But how to determine the non-existence of E^{-1} is left undefined. So the key to a halting algorithm is to find out a necessary and sufficient condition for the non-existence of E^{-1}.

According to Definition 2 and Figure 3, E^{-1} exists if there is a parameter value tuple $< p, d, l >$, such that $E \vDash PC(p, d, l)$ holds. So, intuitively, E^{-1} does not exist if for every parameter value tuple $< p, d, l >$, another tuple $< p', d', l' >$ with $p' > p$,$l' > l$ and $d' > d$ can always be found, such that $E \vDash PC(p', d', l')$ does not hold.

This case can be detected by the SAT instance in Figure 5, which is similar to Figure 3, except that three new constraints

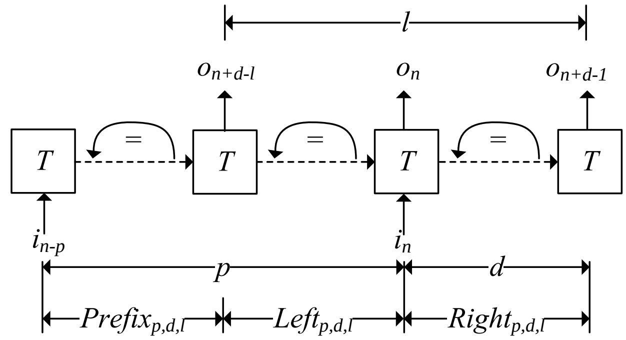

Fig. 5. The loop-like non-complementary condition

are inserted to detect loops in the paths s_{n-p}^{n+d-l},$s_{n+d-l+1}^{n}$ and s_{n+1}^{n+d}. If this SAT instance is satisfiable for the parameter value $< p, d, l >$, these three loops can be unfolded in the following way: Assume that the length of loops in s_{n-p}^{n+d-l},$s_{n+d-l+1}^{n}$ and s_{n+1}^{n+d} are l_1, l_2 and l_3 respectively, and that these loops are **unfolded q times**. Then, the SAT instance generated **from this unfolding is shown** in Figure 6. This unfolded SAT instance corresponds to $F_{LN}(p", d", l")$, where:

$$\begin{array}{rcl} p" & = & l_1 * q + (d - l + p - l_1) + l_2 * q + (l - d - l_2) \\ d" & = & l_3 * q + (d - l_3) \\ l" & = & l_2 * q + (l - d - l_2) + l_3 * q + (d - l_3) \end{array} \tag{5}$$

It is obvious that for every particular $< p, d, l >$ and $< p', d', l' >$, there always exists a q, such that $s_{n-p"}^{n+d"-l"}$,$s_{n+d"-l"+1}^{n}$ and $s_{n+1}^{n+d"}$ resulted from this **unfolding** are not shorter than $s_{n-p'}^{n+d'-l'}$,$s_{n+d'-l'+1}^{n}$ and $s_{n+1}^{n+d'}$ respectively. The satisfiability of this unfolded instance will be proved in Lemma 1 in next subsection. This means for every particular $< p', d', l' >$, we can always find another $< p", d", l" >$, such that $E \vDash PC(p", d", l")$ does not hold. So the decoder does not exist.

According to the second and third lines of Equation (3), there are actually two paths, so these loops must be detected in both of them, i.e., on the product machine M^2 defined below:

***Definition** 3:* **Product machine :** For Mealy machine $M = (S, s_0, I, O, T)$, its Product machine is $M^2 = (S^2, s_0^2, I^2, O^2, T^2)$, where

1) $S^2 = S \times S$
2) $s_0^2 = s_0 \times s_0$
3) $I^2 = I \times I$

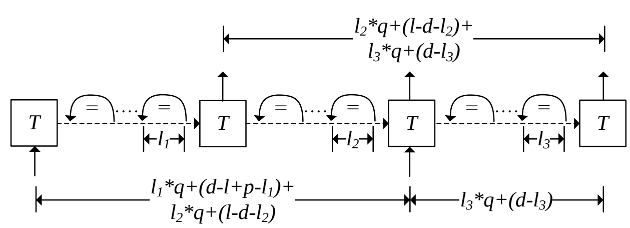

Fig. 6. The loop-like non-complementary condition unfolded q times

4) $O^2 = O \times O$
5) T^2 is defined as $(< s_{m+1}, s'_{m+1} >, < o_m, o'_m >) = T^2(< s_m, s'_m >, < i_m, i'_m >)$ with $(s_{m+1}, o_m) = T(s_m, i_m)$ and $(s'_{m+1}, o'_m) = T(s'_m, i'_m)$.

Thus, the loop-like non-complementary condition is formally defined below to determine the non-existence of E^{-1}:

***Definition** 4:* **Loop-like Non-complementary Condition (*LN*) :** For encoder E and its Mealy machine $M = (S, s_0, I, O, T)$, assume its product machine is $M^2 = (S^2, s_0^2, I^2, O^2, T^2)$, then $E \vDash LN(p,d,l)$ holds if i_n cannot be uniquely determined by o_{n+d-l}^{n+d-1} in the path s_{n-p}^{n+d-1}, and there are loops in $(s^2)_{n-p}^{n+d-l}$,$(s^2)_{n+d-l+1}^{n}$ and $(s^2)_{n+1}^{n+d}$. This equals the satisfiability of $F_{LN}(p,d,l)$ in Equation (6). $E \vDash LN$ is further defined as $\exists p,d,l : E \vDash LN(p,d,l)$.

$$F_{LN}(p,d,l) \overset{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=n-p}^{n+d-1}\{(s_{m+1},o_m) \equiv T(s_m,i_m)\} \\ \wedge & \bigwedge_{m=n-p}^{n+d-1}\{(s'_{m+1},o'_m) \equiv T(s'_m,i'_m)\} \\ \wedge & \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \\ \wedge & i_n \neq i'_n \\ \wedge & \bigvee_{x=n-p}^{n+d-l-1} \bigvee_{y=x+1}^{n+d-l}\{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \\ \wedge & \bigvee_{x=n+d-l+1}^{n-1} \bigvee_{y=x+1}^{n}\{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \\ \wedge & \bigvee_{x=n+1}^{n+d-1} \bigvee_{y=x+1}^{n+d}\{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \end{array} \right\} \quad (6)$$

By comparing Equations (3) and (6), it is obvious that **the** only difference is the last three newly inserted lines in (6), which will be used to detect loops in the following three paths:

$$\begin{array}{rcl} Prefix_{p,d,l} & = & (s^2)_{n-p}^{n+d-l} \\ Left_{p,d,l} & = & (s^2)_{n+d-l+1}^{n} \\ Right_{p,d,l} & = & (s^2)_{n+1}^{n+d} \end{array} \quad (7)$$

The correctness of this approach will be proved in the next subsection.

B. Proving correctness

Before proving correctness of this approach, some lemmas are needed.

***Lemma** 1 ():* For $F_{LN}(p'',d'',l'')$ in Figure 6, $E \vDash LN(p,d,l)$ implies $E \vDash LN(p'',d'',l'')$.

Proof: The formula $F_{LN}(p'',d'',l'')$ is:

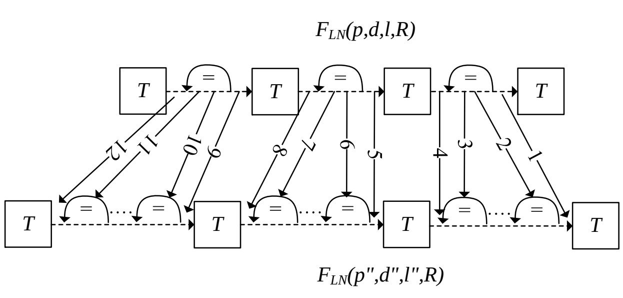

Fig. 7. Correspondence between $F_{LN}(p,d,l)$ and $F_{LN}(p'',d'',l'')$

$$F_{LN}(p'',d'',l'') \overset{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=n-p''}^{n+d''-1}\{(s_{m+1},o_m) \equiv T(s_m,i_m)\} \\ \wedge & \bigwedge_{m=n-p''}^{n+d''-1}\{(s'_{m+1},o'_m) \equiv T(s'_m,i'_m)\} \\ \wedge & \bigwedge_{m=n+d''-l''}^{n+d''-1} o_m \equiv o'_m \\ \wedge & i_n \neq i'_n \\ \wedge & \bigvee_{x=n-p''}^{n+d''-l''-1} \bigvee_{y=x+1}^{n+d''-l''}\{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \\ \wedge & \bigvee_{x=n+d''-l''+1}^{n-1} \bigvee_{y=x+1}^{n}\{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \\ \wedge & \bigvee_{x=n+1}^{n+d''-1} \bigvee_{y=x+1}^{n+d''}\{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \end{array} \right\} \quad (8)$$

$E \vDash LN(p,d,l)$ means that $F_{LN}(p,d,l)$ is satisfied. Assume its satisfying assignment is A. The directed arcs numbered from 1 to 12 in Figure 7 show the correspondence between $F_{LN}(p,d,l)$ and $F_{LN}(p'',d'',l'')$.

Arcs 2 and 3 mean applying the satisfying assignment of the loop in $Right_{p,d,l}$ to the unfolded loops in $Right_{p'',d'',l''}$. Arcs 1 and 4 mean applying the satisfying assignments of the two paths that **are** not in the loop, to $Right_{p'',d'',l''}$. With Arcs 1,2,3 and 4, the path $Right_{p'',d'',l''}$ is satisfied.

Similarly, the paths $Prefix_{p'',d'',l''}$ and $Left_{p'',d'',l''}$ can also be satisfied with A. Thus the **second** line of Equation (8) is satisfied with the assignment A.

Similarly, the **third to fifth** lines of Equation (8) are also satisfied with the assignment A.

At the same time, there are q loops in $Prefix_{p'',d'',l''}$, $Left_{p'',d'',l''}$ and $Right_{p'',d'',l''}$, which will make the last three lines in Equation (8) satisfied.

Thus,the satisfying assignment A of $F_{LN}(p,d,l)$ can also make $F_{LN}(p'',d'',l'')$ satisfied. This concludes the proof. ∎

***Lemma** 2 ():* For two tuples $< p,d,l >$ and $< p',d',l' >$, if $Prefix_{p',d',l'}$,$Left_{p',d',l'}$ and $Right_{p',d',l'}$ are **not shorter** than $Prefix_{p,d,l}$,$Left_{p,d,l}$ and $Right_{p,d,l}$ respectively, then $E \vDash PC(p,d,l) \rightarrow E \vDash PC(p',d',l')$.

Proof: It is obvious that $F_{PC}(p,d,l)$ is a sub-formula of $F_{PC}(p',d',l')$, **so the unsatisfiability of the former implies the unsatisfiability of the latter**. Thus, $E \vDash PC(p,d,l) \rightarrow E \vDash PC(p',d',l')$ holds. ∎

The following two theorems will prove that $E \vDash LN \leftrightarrow \neg\{E \vDash PC\}$.

***Theorem** 1 ():* $E \vDash LN \rightarrow \neg\{E \vDash PC\}$

Proof: **The proof is by contradiction**. Assume that $E \vDash LN$ and $E \vDash PC$ both hold. This means **that** there exist $< p,d,l >$ and $< p',d',l' >$, such that $E \vDash PC(p,d,l)$ and $E \vDash LN(p',d',l')$.

On the one hand, $E \vDash LN(p',d',l')$ implies that there are loops in the paths $Prefix_{p',d',l'}$,$Left_{p',d',l'}$ and $Right_{p',d',l'}$. By unfolding these loops, another tuple $< p'',d'',l'' >$ can be found, so that :

1) $Prefix_{p'',d'',l''}$,$Left_{p'',d'',l''}$ and $Right_{p'',d'',l''}$ are not shorter than $Prefix_{p,d,l}$,$Left_{p,d,l}$ and $Right_{p,d,l}$ respectively;
2) According to Lemma 1, $F_{LN}(p'',d'',l'')$ is satisfiable.

$F_{PC}(p'',d'',l'')$ is a sub-formula of $F_{LN}(p'',d'',l'')$, so $F_{PC}(p'',d'',l'')$ is also satisfiable, which means that $E \vDash PC(p'',d'',l'')$ does not hold.

On the other hand, according to Lemma 2, $E \vDash PC(p'',d'',l'')$ holds.

This contradiction concludes the proof. ■

***Theorem** 2 ():* $E \vDash LN \leftarrow \neg\{E \vDash PC\}$

Proof: **The proof is by contradiction**. Assume that neither $E \vDash LN$ nor $E \vDash PC$ holds. Then for every $< p, d, l >$ and $< p', d', l' >$, $F_{PC}(p, d, l)$ is satisfiable, while $F_{LN}(p', d', l')$ is unsatisfiable.

Thus, assume the $uirrd(M^2)$ is the uninitialized state variables recurrence diameter of E's **product machine. Define** $< p, d, l >$ as:

$$\begin{aligned} p &= uirrd(M^2) * 2 + 2 \\ d &= uirrd(M^2) + 1 \\ l &= uirrd(M^2) * 2 + 2 \end{aligned} \tag{9}$$

With this definition, it is obvious that $Prefix_{p,d,l}$,$Left_{p,d,l}$ and $Right_{p,d,l}$ are both longer than $uirrd(M^2)$. This means that there **exist** loops in all these three paths, which will make $F_{LN}(p, d, l)$ satisfiable. **This contradicts the fact that** $F_{LN}(p', d', l')$ is unsatisfiable for every $< p', d', l' >$.

This contradiction concludes the proof. ■

Theorems 1 and 2 illustrate that, a halting algorithm can be implemented by enumerating all combinations of $< p, d, l >$ from small to large, and checking $E \vDash PC(p, d, l)$ and $E \vDash LN(p, d, l)$ in every iteration. This process will eventually terminate with one and only one answer between $E \vDash PC$ and $E \vDash LN$. The implementation of this algorithm will be presented in the next subsection.

C. Algorithm implementation

Algorithm 1 $check\_PCLN$

1: **for** $x = 1 \rightarrow \infty$ **do**
2: $p = 2x$
3: $d = x$
4: $l = 2x$
5: **if** $F_{PC}(p, d, l)$ is unsatisfiable **then**
6: **print** "E^{-1} exists with $< p, d, l >$"
7: **halt**;
8: **else if** $F_{LN}(p, d, l)$ is satisfiable **then**
9: **print** "E^{-1} does not exist"
10: **halt**;
11: **end if**
12: **end for**

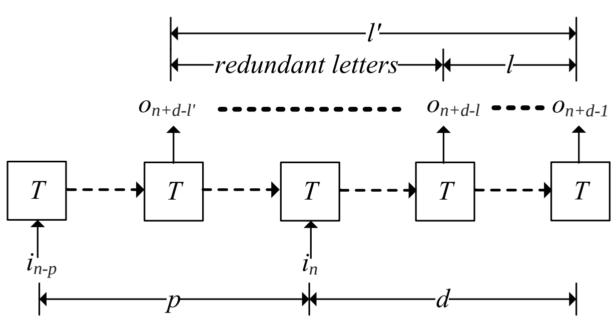

Fig. 8. The redundant output letters

Instead of enumerating **all combinations of values for** p**,** d **and** l **as in earlier work** [1], [2], Line 2,3 and 4 of Algorithm 1 ensure that the lengths of $Prefix_{p,d,l}$,$Left_{p,d,l}$ and $Right_{p,d,l}$ are **all set to** x **(Line 1)**. **Hence, the runtime overhead** is further reduced because many redundant combinations do not need to be tested any more.

According to Theorems 1 and 2, Algorithm 1 will eventually terminate at line 6 or 9.

IV. Removing Redundant Output Letters

Algorithm 2 $RemoveRedundancy(p, d, l)$

for $p' = p \rightarrow 0$ **do**
  if $F_{PC}(p' - 1, d, l)$ is satisfiable **then**
    break
  end if
end for
for $d' = d \rightarrow 0$ **do**
  if $F_{PC}(p', d' - 1, l)$ is satisfiable **then**
    break
  end if
end for
for $l' = 1 \rightarrow l - (d - d')$ **do**
  if $F_{PC}(p', d', l')$ is unsatisfiable **then**
    break
  end if
end for
print "final result is $< p', d', l' >$"

Although Algorithm 1 is sufficient to determine the existence of E^{-1}, **the values found** by line 6 of Algorithm 1 contain some redundancy, which will cause unnecessarily large overheads on the circuit area and the run time of characterizing.

For example, as shown in Figure 8, assume that l is the **smallest value** that leads to $E \vDash PC(p, d, l)$, and $l < d$, which means that i_n is uniquely determined by some output letters o_k with $k > n$. Further assume that line 6 of Algorithm 1 **proves that** $E \vDash PC(p, d, l')$. It is obvious that $l' > d$, which makes i_n **dependent** on some redundant o_k with $k \le n$. So $o_{n+d-l'}^{n+d-l-1}$ is the sequence of redundant output letters, which should be removed to prevent them from being instantiated as latches in circuit E^{-1}. At the same time, also according to Figure 8, larger p and d lead to larger SAT instances for the characterization algorithm in the **second** step of complementary synthesis.

So, Algorithm 2 is used to minimize $< p, d, l >$ before passing it to the characterization **algorithm.**

V. Experimental Results

This algorithm has been implemented **in** the OCaml language. The generated SAT instances are solved with **the** Zchaff SAT solver [4]. All experiments are run on a PC with a 2.4GHz Intel Core 2 Q6600 processor, 8GB memory and CentOS 5.2 linux. All these experimental results and programs can be downloaded from http://www.ssypub.org.

TABLE I
INFORMATION OF BENCHMARKS

	XGXS	XFI	scrambler	PCI-E	T2 ethernet
Lines number of Verilog source code	214	466	24	1139	1073
#regs	15	135	58	22	48
Data path width	8	64	66	10	10

A. Benchmarks

Table I shows the information of the following benchmarks.

1) A XGXS encoder that is compliant to clause 48 of IEEE-802.3ae 2002 standard [6].
2) A XFI encoder that is compliant to clause 49 of the same IEEE standard.
3) A 66-bit scrambler that is used to ensure that a data sequence has sufficiently many 0-1 **transitions, so** that it can run through a high-speed noisy serial transmission channel.
4) A PCI-E physical coding module [5].
5) The Ethernet module of Sun's OpenSparc T2 processor.

B. Determining the existence of the decoder for properly designed encoders

The **first and third** rows of Table II compare the run time of checking $E \models PC$ between [3] and this paper. **The fourth row shows our approach's percentage of improvements over [3]**. It is obvious that this paper's approach is much faster than that of [3].

The **second and fifth** rows compare the discovered parameter values, and some minor differences are found on parameter value p. This is caused by the **different order of checking various parameter value combinations**.

C. Comparing decoder area

Table III compares the circuit area of the decoders built manually, and the decoders built by this paper's algorithm. These decoders are synthesized with LSI10K technology library from Synopsys DesignCompiler.

Table III suggests that, except for the most complex XFI, synthesis results of this paper's algorithm are more compact than those decoders built manually. However, this comparison is unfair because those decoders **built manually also include** additional functionality, such as testing logic.

TABLE II
EXPERIMENTAL RESULTS ON PROPERLY DESIGNED ENCODERS

		XGXS	XFI	scrambler	PCI-E	T2 ethernet
[3]	Time to check PC(sec)	1.06	70.52	5.74	2.40	66.37
	d, p, l	1,1,1	0,3,2	0,2,2	2,1,1	4,1,1
This paper	Time to check PC(sec)	0.29	17.86	2.67	0.47	29.64
	improve %	72.64	74.67	53.48	80.42	55.34
	d, p, l	1,2,1	0,3,2	0,2,2	2,2,1	4,4,1

TABLE III
COMPARING DECODER AREA

	XGXS	XFI	scrambler	PCI-E	T2 ethernet
The decoders built manually	921	6002	1629	852	1446
The decoders built by this paper's algorithm	700	12754	1455	455	552

On the other hand, for XFI, the circuit area of this paper's algorithm is about 2 times larger. This means the circuit area must be **improved in future work**.

D. Comparing decoder timing

Table IV compares the **critical-path latencies** of the decoders built manually and the decoders built by this paper's algorithm. Their synthesis settings are the same as Subsection V-C. For all those circuits, the **critical-path latencies** of the decoders built by this paper's algorithm are all better.

E. Determining the non-existence of the decoder for improperly designed encoders

To further show the usefulness of this paper's algorithm, some improperly designed encoders without corresponding decoders are needed. These improperly designed encoders are obtained by modifying each benchmark's output statements, so that they can explicitly output the same letter for two different input letters. In this way, input letter i_n can never be uniquely determined by E's output sequence.

The **first** row of Table V shows the run time of [3] on checking these improperly designed encoders, while the **second** row shows the run time of this paper's algorithm. **The third row shows our approach's percentage of improvements over [3]**. These results indicate that this paper's algorithm always terminate correctly, and is **much faster than previously reported methods** [3].

VI. RELATED PUBLICATIONS

A. Complementary synthesis

The concept of complementary synthesis was first **proposed in 2009** [1]. Its major shortcomings **are that it may not halt**, and its **runtime** overheads **while building complementary circuit is large**.

The halting problem was handled by building a set of over-approximations that are similar to onion rings [3], while the **runtime** overhead problem was addressed by simplifying the SAT instance with unsatisfiable core extraction [2].

TABLE IV
COMPARING CRITICAL-PATH LATENCIES IN NANOSECOND

	XGXS	XFI	scrambler	PCI-E	T2 ethernet
The decoders built manually	12.33	46.65	6.54	19.03	23.36
The decoders built by this paper's algorithm	11.96	28.13	6.54	9.09	12.69

TABLE V
COMPARING RUN TIME OF IMPROPERLY DESIGNED ENCODERS

	XGXS	XFI	scra-mbler	PCI-E	T2 et-hernet
The algorithm of [3](sec)	0.98	35.08	2.54	1.36	17.39
This paper's algorithm(sec)	0.16	7.59	1.17	0.33	2.19
improve %	83.67	78.36	53.94	75.74	87.41

B. Program inversion

According to Gulwani [16], **Program Inversion** is the problem that derives a program P^{-1} that negates the computation of a given program P. So the definition of **Program Inversion** is very similar to complementary synthesis.

The initial work on deriving program inversion used proof-based approaches [17], but it could only handle very small programs and very simple syntax structures.

Glück et.al [18] **inverted** the first-order functional programs by eliminating nondeterminism with LR-based parsing methods. **But the use of functional languages in that work is incompatible with our complementary synthesis**.

Srivastava et.al [19] assumed that an inverse program was typically related to the original program, so the space of possible inversions can be inferred by automatically mining the original program for expressions, predicates, and control flow. This algorithm inductively rules out invalid paths that **cannot** fulfill the requirement of inversion, to narrow down the space of candidate programs until only the valid ones remain. So it can only guarantee the existence of a solution, but not the correctness of this solution if its assumptions do not hold.

C. The completeness of bounded model checking

Bounded model checking(BMC) [14] is a model checking technology **that considers only paths of** limited length. So it is an incomplete algorithm. **Many researchers have tried to find** complete approaches for BMC.

One line of research [13], [14] tried to find out a bound b, which can guarantee the correctness of a specification, if the specification is correct on all paths that are shorter than b. Line 8 of Algorithm 1 finds out the value of p,d and l that can prove the non-existence of the decoder, **which is similar to [13], [14]**.

The other line of research [15] tried **to find a bound** for induction, such that the correctness of a specification within any bound b implies the correctness on bound $b+1$. Our algorithm proves the non-existence of the decoder by unfolding loops. This is similar to finding induction **patterns** in **[15]**.

D. Protocol converter synthesis

The protocol converter synthesis is the problem that automatically generates a translator between two different communication protocols. This is **related to our work because both focus** on synthesizing communication circuits.

Avnit et.al [20], [21] first defined a general model for describing the different protocols. **Then they provided** an algorithm to decide whether there are some functionality of a protocol that **cannot** be translated into another. Finally, **they synthesized a translator** by computing a greatest fixed point for the update function of the buffer's control states. Avnit et.al [22] improved the algorithm mentioned above with a more efficient design space exploration algorithm.

VII. CONCLUSIONS

This paper proposes a **faster and simpler halting** algorithm that checks whether a particular encoder has **a corresponding** decoder. The theoretical analysis shows that this paper's approach always distinguishes correct encoders from their incorrect variants and halts properly. Experimental results show that this paper's approach is **much faster than previous methods** [3].

ACKNOWLEDGMENT

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

This work was funded by projects 60603088 and 61070132 supported by National Natural Science Foundation of China.

REFERENCES

[1] S. Shen, J. Zhang, Y. Qin, and S. Li, "Synthesizing complementary circuits automatically," in Proc. IEEE ICCAD, Nov. 2009, pp. 381-388.
[2] S. Shen, Y. Qin, K. Wang, L. Xiao, J. Zhang, and S. Li, "Synthesizing Complementary Circuits Automatically," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 29, no. 8, pp. 1191-1202, Aug. 2010.
[3] S. Shen, Y. Qin, J. Zhang, and S. Li, "A Halting Algorithm to Determine the Existence of Decoder," in Proc. IEEE FMCAD, Oct. 2010, pp. 91-100.
[4] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in Proc. IEEE Int. DAC, Jun. 2001, pp. 530-535.
[5] "PCI Express Base Specification Revision 1.0". [Online]. Available: http://www.pcisig.com
[6] "IEEE Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation", IEEE Std. 802.3, 2002.
[7] Xilinx. (2008) Xilinx core generator system. [Online]. Available: http://www.xilinx.com/tools/coregen.htm
[8] Synopsys. (2008) Designware library. [Online]. Available: http://www.synopsys.com/IP/DESIGNWARE/Pages/default.aspx
[9] J. P. M. Silva and K. A. Sakallah, "GRASP - a new search algorithm for satisfiability," in Proc. IEEE ICCAD, Nov. 1996, pp. 220-227.
[10] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust Sat-solver," Discrete Applied Mathematics, vol. 155, no. 12, pp. 1549-1561, Dec. 2007.
[11] N. Eén and N. Sörensson., "Extensible SAT-solver," in Proc. Int. Conf. SAT, May 2003, pp. 502-518.
[12] G. H. Mealy , "A method for synthesizing sequential circuits," Bell Syst. Tech. J., vol. 34, pp. 1045-1079, Jan. 1955.
[13] D. Kroening and O. Strichman, "Efficient Computation of Recurrence Diameters," in Proc. Int. Conf. VMCAI, Jan. 2003, pp. 298-309.
[14] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving," Formal Methods in System Design, vol. 19, no. 1, pp. 7-34, Jan. 2001.
[15] M. Sheeran, S. Singh, and G. Stalmarck, "Checking Safety Properties Using Induction and a SAT-Solver," in Proc. IEEE FMCAD, Oct. 2000, pp. 108-125.
[16] S. Gulwani, "Dimensions in program synthesis," in Proc. ACM PPDP, Jul. 2010, pp. 13-24.
[17] E. W. Dijkstra, "Program Inversion," in Program Construction, 1978, pp. 54 - 57.

[18] R. Glück and M. Kawabe, ”A method for automatic program inversion based on LR(0) parsing,” Fundam. Inf., vol. 66, no. 4, pp. 367-395, Dec. 2005.

[19] S. Srivastava, S. Gulwani, S. Chaudhuri, and J. Foster, ”Program inversion revisited,” Technical Report MSR-TR-2010-34, Microsoft Research, 2010.

[20] K. Avnit, V. D’Silva, A. Sowmya, S. Ramesh, and S. Parameswaran, ”A Formal Approach To The Protocol Converter Problem,” in Proc. IEEE DATE Conf. Exposit., Mar. 2008, pp. 294-299.

[21] K. Avnit, V. D’Silva, A. Sowmya, S. Ramesh, and S. Parameswaran, ”Provably correct on-chip communication: A formal approach to automatic protocol converter synthesis,” ACM Trans. Design Autom. Electr. Syst., vol. 14, no. 2, pp. 1-41, Apr. 2009.

[22] K. Avnit, and A. Sowmya, ”A formal approach to design space exploration of protocol converters,” in Proc. IEEE DATE Conf. Exposit., Mar. 2009, pp. 129-134.