

Cover letter

The following paper is accepted by ICCAD'11: “Inferring Assertion for Complementary Synthesis”.

This TCAD paper extends that ICCAD11 paper in the following ways:

1. Proposing an automatic algorithm to discover all decoders' Boolean relations in Subsection IV-A.
2. Proposing an automatic algorithm to characterize precondition for each discovered decoder in Subsection IV-B.
3. Presenting the experimental results on dealing with multiple decoders in Subsection V-C.

Inferring Assertion for Complementary Synthesis

ShengYu Shen, *Member, IEEE*, Ying Qin, KeFei Wang, ZhengBin Pang, JianMin Zhang, and SiKun Li

Abstract—Complementary synthesis can automatically synthesize the decoder circuit of an encoder. However, its user needs to manually specify an assertion on some configuration pins to prevent the encoder from reaching the non-working states.

To avoid this tedious task, this paper propose an automatic approach to infer this assertion, by iteratively discovering and removing cases without decoders.

To discover all decoders that may exist simultaneously under this assertion, a second algorithm based on functional dependency is proposed to decompose \mathbb{R} , the Boolean relation that uniquely determines the encoder's input, into all possible decoders.

To help the user select the correct decoder, a third algorithm is proposed to infer each decoder's precondition formula, which represents those cases that lead to this decoder's existence.

Experimental results on several complex encoders indicate that our algorithm can always infer assertions and generate decoders for them. Moreover, when multiple decoders exist simultaneously, the user can easily select the correct one by inspecting their precondition formulas.

Index Terms—Complementary Synthesis, Inferring Assertion, Cofactoring, Craig Interpolation, Functional Dependency

I. INTRODUCTION

One of the most difficult tasks in designing communication chips is to design the complex circuit pair (E, E^{-1}) , in which the encoder E transforms information into a particular format, while its decoder E^{-1} recovers this information.

Thus, complementary synthesis [1] was proposed to automatically synthesize an encoder's decoder. As shown in Fig. 1a), it includes three steps: **i)** manually specifying an assertion that assigns constant value on the encoder's configuration pins to prevent it from reaching the non-working states without a decoder; **ii)** determining the decoder's existence by checking whether the encoder's input can be uniquely determined by its output; and **iii)** characterizing the decoder's Boolean function.

To manually specify an assertion, the user must read extensive documentation and often perform laborious trial-and-error process. For example, our most complex benchmark—the XFI encoder—has 120 configuration pins. Finding out their meaning and correct combinations was a very difficult and lengthy process, which had already been experienced by **the first author** in [1]. To avoid this tedious task, this paper propose the following three algorithms.

First, an algorithm is proposed to automatically infer this assertion [2], which corresponds to the first step of the new approach in Fig. 1b). This algorithm iteratively discovers the encoder's configuration value that leads to the decoder's nonexistence, and then uses Craig interpolation to infer a new formula that covers a larger set of such invalid configuration

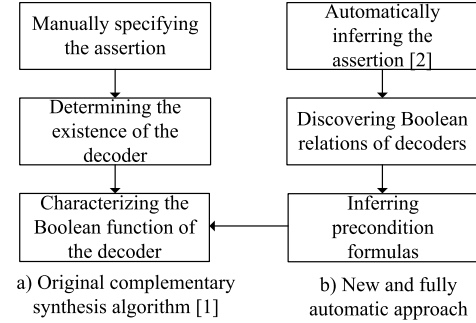


Fig. 1. The original and new flows of complementary synthesis

values. These inferred formulas will be ruled out until no more invalid configuration values can be found. The final assertion is obtained by anding the inverses of all these inferred formulas.

Second, under this inferred assertion, however, multiple decoders may exist simultaneously. Thus, as shown in the second step of Fig. 1b), a second algorithm is proposed to discover all these decoders by iteratively testing whether \mathbb{R} , the Boolean relation that uniquely determines the encoder's input, functionally depends [5] on these discovered decoders. For every configuration value c that fails this test, a new decoder's Boolean relation is discovered by asserting c into \mathbb{R} . This step is repeated until all decoders are discovered.

Third, to help the user select the correct decoder, a third algorithm is proposed to characterize a precondition formula for each decoder, which represents the configuration value set that leads to this decoder's existence. The user can easily select the correct decoder by inspecting these preconditions.

For example, for the two decoders of XFI discovered by our algorithm, their corresponding precondition formulas refer to only three pins, only two of which have different values. Therefore, to select the correct decoder, one only needs to find out the meaning of these two pin, instead of all 120 pins. More details can be found in the experimental results section, which indicates that our algorithm can significantly save the human effort in specifying assertion and selecting decoder. All the Experimental Results and programs can be downloaded from <http://www.ssypub.org>.

The remainder of this paper is organized as follows. Section II introduces background materials. Section III presents the algorithm that infers assertions. Section IV discusses how to discover all decoders and characterize their precondition formulas. Sections V and VI present the experimental results and related works. Finally, Section VII provides the conclusion.

II. PRELIMINARIES

To save space, we assume the readers' familiarity with propositional satisfiability (SAT), cofactoring [7], Craig inter-

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

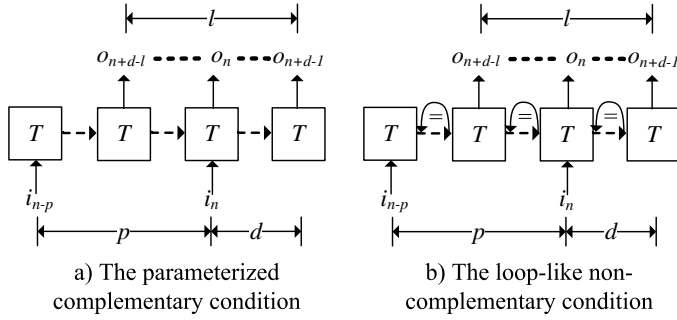


Fig. 2. The parameterized complementary condition and the loop-like non-complementary condition

polant generation [8], and functional dependency [5].

The encoder is modeled using *Mealy finite state machine with configuration* $M = (S, s_0, I, C, O, T)$, consisting of a finite state set S , an initial state $s_0 \in S$, a finite input letter set I , a finite configuration letter set C , a finite output letter set O , and a transition function $T : S \times I \times C \rightarrow S \times O$ that computes the next state and output letter from the current state, input letter, and configuration letter.

We denote the state, input letter, output letter, and configuration letter at the n -th cycle as s_n, i_n, o_n , and c_n , respectively. We further denote the sequence of state, input letter, output letter, and configuration letter from the n -th to the m -th cycle as s_n^m, i_n^m, o_n^m , and c_n^m , respectively.

An *assertion (or formula) on configuration pins* is defined as a configuration letter set R . $R(c)$ means $c \in R$. If $R(c)$ holds, we also say that R covers c .

The decoder exists if the encoder's input can be uniquely determined by its output. As shown in Fig. 2a), this can be formally defined as follows.

Definition 1: Parameterized complementary condition (PC): For encoder E , assertion R , and parameters p, d , and l , $E \models PC(p, d, l, R)$ holds if i_n can be uniquely determined by o_{n+d-l}^{n+d-1} , and R covers the constant configuration letter c . This amounts to the unsatisfiability of $F_{PC}(p, d, l, R)$ in Equation (1).

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

Lines 2 and 3 of Equation (1) correspond to two state sequences of E , respectively. Line 4 forces the output sequences of these two state sequences to be the same, while Line 5 forces their input letters to be different. The last line constrains c to be covered by R .

The algorithm based on checking PC [1] just enumerates all combinations of p, d , and l , from small to large, until $F_{PC}(p, d, l, R)$ becomes unsatisfiable, which means that the decoder exists.

III. A HALTING ALGORITHM TO INFER ASSERTION

In Subsection III-A, we first define how to determine the decoder's nonexistence for a particular configuration letter. And then, in Subsection III-B, we introduce the algorithm that infers assertion by iteratively detecting and removing all invalid configuration letters.

A. Determining the decoder's nonexistence

According to Definition 1, the decoder exists for a set of configuration letters R if there is a parameter value tuple $\langle p, d, l \rangle$ that makes $E \models PC(p, d, l, R)$ holds true. Therefore, intuitively, the decoder does not exist if for every parameter value tuple $\langle p, d, l \rangle$, we can always find another tuple $\langle p', d', l' \rangle$ with $p' > p, l' > l$ and $d' > d$, such that $E \models PC(p', d', l', R)$ does not hold.

This case can be detected by the SAT instance shown in Figure 2b), which is similar to that of Figure 2a), except that three new constraints are inserted to detect loops on $s_{n-p}^{n+d-l}, s_{n+d-l+1}^n$, and s_{n+1}^{n+d} .

Thus, the decoder's nonexistence can be determined by:

Definition 2: Loop-like Non-complementary Condition (LN): For encoder E , $E \models LN(p, d, l, R)$ holds if i_n can not be uniquely determined by o_{n+d-l}^{n+d-1} on s_{n-p}^{n+d-1} , and there are loops on $s_{n-p}^{n+d-l}, s_{n+d-l+1}^n$, and s_{n+1}^{n+d} . This amounts to the satisfiability of $F_{LN}(p, d, l, R)$ in Equation (2).

$$F_{LN}(p, d, l, R) \stackrel{def}{=} \left\{ \begin{array}{l} F_{PC}(p, d, l, R) \\ \bigwedge \bigvee_{x=n-p}^{n+d-l-1} \bigvee_{y=x+1}^{n+d-l} \{ s_x \equiv s_y \wedge s'_x \equiv s'_y \} \\ \bigwedge \bigvee_{x=n+d-l+1}^{n-1} \bigvee_{y=x+1}^n \{ s_x \equiv s_y \wedge s'_x \equiv s'_y \} \\ \bigwedge \bigvee_{x=n+1}^{n+d-1} \bigvee_{y=x+1}^{n+d} \{ s_x \equiv s_y \wedge s'_x \equiv s'_y \} \end{array} \right\} \quad (2)$$

B. Algorithm implementation

Algorithm 1 InferAssertion

```

1:  $NA = \{\}$ 
2: for  $x = 0 \rightarrow \infty$  do
3:    $\langle p, d, l \rangle = \langle 2x, x, 2x \rangle$ 
4:   if  $F_{PC}(p, d, l, \bigwedge_{na \in NA} \neg na)$  is unsatisfiable then
5:     Halt. The final assertion is  $\bigwedge_{na \in NA} \neg na$ , the decoder exists if it is satisfiable.
6:   else
7:     while  $F_{LN}(p, d, l, \bigwedge_{na \in NA} \neg na)$  is satisfiable do
8:       Assume  $A(c)$  is  $c$ 's satisfying assignment leading to the decoder's nonexistence.
9:        $na \leftarrow InferCoveringFormula(A(c))$ 
10:       $NA \leftarrow NA \cup \{na\}$ 
11:     end while
12:   end if
13: end for

```

Algorithm 1 is used to infer assertion on configuration pins that can lead to the decoder's existence. Intuitively, it iteratively tests all combinations of $\langle p, d, l \rangle$ in Line 3. For every $A(c)$ found in Line 8 that can lead to the decoder's

nonexistence, a larger set of such configuration letters are discovered by the procedure *InferCoveringFormula* in Line 9, and ruled out in Line 10. This loop is repeated until the formula F_{PC} in Line 4 is unsatisfiable.

InferCoveringFormula uses cofactoring [7] to assert the satisfying assignments of i_{n-p}^{n+d-1} , $(i')_{n-p}^{n+d-1}$, s_{n-p} , and $(s')_{n-p}$ back to F_{LN} , and then uses Craig interpolation [8] to characterize a larger set of invalid configuration letters. More details can be found in [2], which had proved that Algorithm 1 is a halting one.

IV. DISCOVERING MULTIPLE DECODERS

Subsection IV-A introduces how to discover decoders and its correctness proof, while Subsection IV-B presents how to characterize the precondition formula of each discovered decoder, to help the user select the correct one.

A. Constructing SAT instance to discover decoders

Assume the assertion inferred by Algorithm 1 is:

$$IA \stackrel{def}{=} \bigwedge_{na \in NA} \neg na \quad (3)$$

To simplify the presentation, we denote o_{n+d-l}^{n+d-1} and i_n as:

$$\begin{aligned} X &\stackrel{def}{=} o_{n+d-l}^{n+d-1} \\ Y &\stackrel{def}{=} i_n \end{aligned} \quad (4)$$

Thus, the Boolean relation that uniquely determines i_n from o_{n+d-l}^{n+d-1} and c can be denoted as:

$$\mathbb{R}(c, X, Y) \stackrel{def}{=} F_{PC}(p, d, l, IA) \quad (5)$$

$\mathbb{R}(c, X, Y)$ defines a mapping from c and X to Y , which is actually a function f :

$$Y = f(c, X) \quad (6)$$

For every configuration letter $c_i \in IA$, there is a \mathbb{R}_{c_i} :

$$\mathbb{R}_{c_i}(X, Y) \stackrel{def}{=} \mathbb{R}(c, X, Y) \wedge c \equiv c_i \quad (7)$$

Two different c_i and c_j may share the same \mathbb{R}_{c_i} . **Therefore, IA can be partitioned into $\{IA_1, \dots, IA_n\}$, such that all c in the same IA_i share the same \mathbb{R}_i , while two c and c' in two different IA_i and $IA_{i'}$ do not. So we call IA_i the precondition formula of \mathbb{R}_i , which means IA_i is the set of all configuration letters that leads to the decoder \mathbb{R}_i 's existence. When there exists multiple decoders, the user can select the correct one by inspecting $\{IA_1, \dots, IA_n\}$ characterized in Subsection IV-B.**

Thus, \mathbb{R}_i defines a mapping from X to Y , which is actually a function $f_i : X \rightarrow Y$. Therefore, f can be rewritten as:

$$f(c, X) = \bigvee_{i=1}^n \{IA_i(c) \wedge f_i(X)\} \quad (8)$$

Thus, our task here is to find out the set $\{\mathbb{R}_1, \dots, \mathbb{R}_n\}$ step-by-step. Assume that we have already discovered a set of

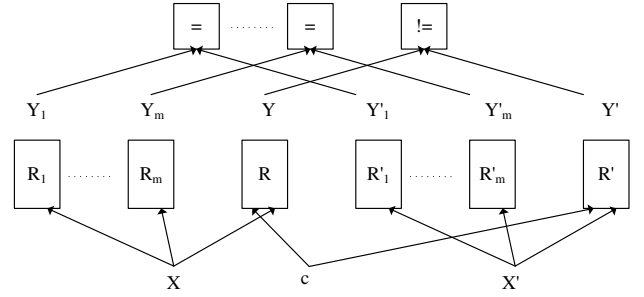


Fig. 3. The SAT instance that discovers decoders

decoders' Boolean relations $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$. To test whether it contains $\{\mathbb{R}_1, \dots, \mathbb{R}_n\}$, that is, whether all decoders have already been discovered, we construct the following SAT instance, which is also shown in Fig. 3:

$$\left\{ \begin{aligned} &\mathbb{R}(c, X, Y) \wedge \bigwedge_{i=1}^m \mathbb{R}_i(X, Y_i) \\ &\wedge \mathbb{R}'(c, X', Y') \wedge \bigwedge_{i=1}^m \mathbb{R}'_i(X', Y'_i) \\ &\wedge \bigwedge_{i=1}^m Y_i \equiv Y'_i \\ &\wedge Y \neq Y' \end{aligned} \right\} \quad (9)$$

Line 1 of Equation (9) represents the Boolean relations in $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$ and \mathbb{R} . Line 2 is a copy of Line 1. The only common variable shared by them is c . Line 3 forces all Y_i and Y'_i to take on the same values, while the last line forces Y and Y' to be different.

The following theorem proves that, if Equation (9) is unsatisfiable, then all decoders have been discovered.

Theorem 1: If Equation (9) is unsatisfiable, then $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$ contains $\{\mathbb{R}_1, \dots, \mathbb{R}_n\}$.

Proof: The proof is by contradiction. Assume $\mathbb{R}_n \notin \{\mathbb{R}_1, \dots, \mathbb{R}_m\}$, and IA_n is its corresponding set of configuration letters, and $c_n \in IA_n$.

We can construct an assignment A such that $A(c) \equiv c_n$. Thus, we have $\{\mathbb{R}(c, X, Y) \wedge A(c) \equiv c_n\} \equiv \mathbb{R}_n(X, Y)$, that is, we can change \mathbb{R} and \mathbb{R}' shown in Fig. 3 to \mathbb{R}_n with A .

As $\mathbb{R}_n \notin \{\mathbb{R}_1, \dots, \mathbb{R}_m\}$, there must be an assignment A' , such that when we assign $A'(X)$ to X and $A'(X')$ to X' , we can make both $\bigwedge_{i=1}^m Y_i \equiv Y'_i$ and $Y \neq Y'$ hold.

Therefore, by combining A and A' , Equation (9) becomes satisfied. This contradiction concludes the proof. ■

On the other hand, if Equation (9) is satisfiable, we need to prove the following theorem.

Theorem 2: If Equation (9) is satisfiable, then there must be at least one decoder that has not been discovered.

Proof: The proof is by contradiction. Assume that all decoders have already been discovered, that is, $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$ contains $\{\mathbb{R}_1, \dots, \mathbb{R}_n\}$. Thus, for any assignment A that makes the first three lines of Equation (9) satisfied, we have:

$$Y = \bigvee_{i=1}^m \{IA_i(c) \wedge Y_i\} = \bigvee_{i=1}^m \{IA_i(c) \wedge Y'_i\} = Y' \quad (10)$$

Thus, the last line of Equation (9) will never be satisfied. Therefore, Equation (9) is unsatisfiable. This contradiction concludes the proof. ■

With the satisfying assignment A , \mathbb{R}_{m+1} defined below is a newly discovered decoder's Boolean relation:

$$\mathbb{R}_{m+1} \stackrel{\text{def}}{=} \mathbb{R}(c, X, Y) \wedge c \equiv A(c) \quad (11)$$

Theorem 3 proves that \mathbb{R}_{m+1} hasn't been discovered before:

Theorem 3: $\mathbb{R}_{m+1} \notin \{\mathbb{R}_1, \dots, \mathbb{R}_m\}$

Proof: The proof is by contradiction. Assume that there is $0 \leq i \leq m$ such that $\mathbb{R}_i \equiv \mathbb{R}_{m+1}$.

As \mathbb{R}_i can uniquely determine Y from X , while \mathbb{R} can uniquely determine Y from X as well as c , and $\mathbb{R}_{m+1} \equiv \mathbb{R}_i$, it is obvious that we can make $Y \equiv Y_i$ by forcing c to be $A(c)$ in Equation (11). Similarly, we have $Y'_i \equiv Y'$.

According to Line 5 of Equation (9), we have $Y \equiv Y_i \equiv Y'_i \equiv Y'$. This is in contradiction with $Y \neq Y'$ in the last line of Equation (9), which concludes the proof. ■

Based on these discussions, the following Algorithm 2 describes how to discover the Boolean relations of all decoders.

Algorithm 2 *DiscoveringDecoders*

- 1: **while** Equation (9) is satisfiable **do**
 - 2: Insert \mathbb{R}_{m+1} of Equation (11) into $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$
 - 3: **end while**
 - 4: The set of decoders' Boolean relations is $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$
-

The loop in Algorithm 2 monotonically increases the size of $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$. As the number of decoders that compute Y from X is finite, Algorithm 2 will eventually halt.

B. Characterizing $\{IA_1, \dots, IA_m\}$

Assume $\{\mathbb{R}_1, \dots, \mathbb{R}_m\}$ is the set of all decoders' Boolean relations discovered by Algorithm 2. To help the user select the correct decoder, we need to characterize their precondition formulas $\{IA_1, \dots, IA_m\}$. According to Fig. 3, the relation between Y and all Y_i can be written as:

$$Y = \bigvee_{i=1}^m \{IA_i(c) \wedge Y_i\} \quad (12)$$

Assume Y and all Y_i are vectors of the same length v , whose j -th bit are y^j and y_i^j , respectively. Therefore, we can rewrite Equation (12) by splitting it into bits, and the relation between the j -th bits of Y and all Y_i can be written as:

$$y^j = \bigvee_{i=1}^m \{IA_i^j(c) \wedge y_i^j\} \quad (13)$$

According to Lee et al. [5], it is obvious that Equation (13) represents a functional dependency problem. We can characterize $IA_i^j(c)$ with the functional dependency algorithm proposed by Lee et al. [5] with the following two formulas:

$$\begin{aligned} \phi_A &\stackrel{\text{def}}{=} \left\{ \begin{array}{l} \wedge \\ \mathbb{R}(c, X, Y) \wedge \bigwedge_{i=1}^m \mathbb{R}_i(X, Y_i) \\ y^j \equiv 1 \end{array} \right\} \\ \phi_B &\stackrel{\text{def}}{=} \left\{ \begin{array}{l} \wedge \\ \mathbb{R}'(c, X', Y') \wedge \bigwedge_{i=1}^m \mathbb{R}'_i(X', Y'_i) \\ \bigwedge_{i=1}^m y_i^j \equiv y_i'^j \\ y'^j \equiv 0 \end{array} \right\} \end{aligned} \quad (14)$$

TABLE I
INFORMATION ON BENCHMARKS

	XGXS	XFI	scrambler	PCIE	T2 Ethernet
#line of verilog	214	466	24	1139	1073
#regs	15	135	58	22	48
Data path width	8	64	66	10	10
#Config pin	3	120	1	16	26

It is obvious that $\phi_A \wedge \phi_B$ is very similar to Equation (9), except that only the j -th bits are constrained to be the same, and the y^j and y'^j are constrained to be different constants.

Therefore, $\phi_A \wedge \phi_B$ is unsatisfiable. The support set of its interpolant $ITP : C \times \mathbb{B}^m \rightarrow \mathbb{B}$ is $\{c, y_1^j, \dots, y_m^j\}$. According to Equation (13), ITP is the over-approximation of $\bigvee_{i=1}^m \{IA_i^j(c) \wedge y_i^j\}$. Thus, an over-approximation of $IA_i^j(c)$ can be obtained by setting y_i^j to 1, and all other y_k^j to 0:

$$ITP \wedge \bigwedge_{k \neq i} y_k^j \equiv 0 \wedge y_i^j \equiv 1 \quad (15)$$

As $\phi_A \wedge \phi_B$ is unsatisfiable, this over-approximation of $IA_i^j(c)$ can make $y^j \equiv 1$. Therefore, we can take it as $IA_i^j(c)$. Thus, $IA_i(c)$ can be defined as:

$$IA_i(c) = \bigwedge_{j=0}^{v-1} IA_i^j(c) \quad (16)$$

In Subsection V-C, we will show that by inspecting these IA_i , the user can easily select the correct decoder.

V. EXPERIMENTAL RESULTS

We have implemented this algorithm and solved the generated SAT instances with Minisat [6]. All experiments have been run on a PC with a 2.4GHz Intel Core 2 Q6600 processor, 8 GB memory, and Ubuntu 10.04 Linux.

Table I shows information on the following benchmarks: the XGXS and XFI encoders compliant to clause 48 and 49 of IEEE-802.3ae 2002 standard [4]; a 66-bit scrambler used to ensure that a data sequence has sufficient 0-1 transitions; a PCI-E physical coding module [3]; and the Ethernet module of Sun's OpenSparc T2 processor.

A. Comparing the results with previous work

Table II compares the halting algorithm proposed in [10] and this paper's approach. The algorithm in [10] needs a manually specified assertion, while our approach does not.

The second row of Table II shows the runtime of the halting algorithm proposed in [10], which checks whether the decoders exist, and the third row shows the value of d , p , and l discovered by that algorithm.

Our algorithm includes three steps: inferring assertions, discovering decoders, and inferring preconditions. Their runtimes are shown in the fourth to the sixth rows of Table II. The seventh row shows the value of the discovered d , p , and l .

By comparing the second and the fourth rows, it is obvious that our approach is much slower than that of [10], which is caused by the much more complicated procedure *InferCoveringFormula*.

TABLE II
EXPERIMENTAL RESULTS

		XG-XS	XFI	scrambler	PCI-E	T2 Ethernet
[10]	Runtime(sec)	0.07	17.84	2.70	0.47	30.59
	d, p, l	1,2,1	0,3,2	0,2,2	2,2,1	4,2,1
this paper	Runtime 1	4.53	264.19	13.03	10.39	426.12
	Runtime 2	0.11	12.11	1.26	0.27	3.07
	Runtime 3	0.13	13.69	1.49	0.23	2.86
	d, p, l	1,5,1	0,5,2	0,5,2	2,5,1	4,5,1

The third and the seventh rows indicate that there are some minor differences in those parameter values, which is caused by the difference between the embedded assertions and inferred assertions. The latter contain much more configuration letters than the former.

B. Inferred assertions

For XGXS: ((!bad_code))

For XFI: ((RESET&!TEST_MODE)|(!RESET&DATA_VALID&!TEST_MODE))

For scrambler: True

For PCI-E: ((CNTL_RESETN_P0&!TXELECIDLE&!CNTL_Loopback_P0&CNTL_TXEnable_P0))

For T2 ethernet: ((tx_enc_conf_sel[1]&tx_enc_conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0]&!jitter_study_pci[1]&!reset_tx)|(tx_enc_conf_sel[1]&!tx_enc_conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0]&!jitter_study_pci[1]&!reset_tx&tx_enc_conf_sel[0]&tx_enc_conf_sel[2]&link_up_loc)|(tx_enc_conf_sel[1]&!tx_enc_conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0]&!jitter_study_pci[1]&!reset_tx&tx_enc_conf_sel[2])|(tx_enc_conf_sel[1]&!tx_enc_conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0]&!jitter_study_pci[1]&!reset_tx&tx_enc_conf_sel[2])|(!tx_enc_conf_sel[1]&tx_enc_conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0]&!jitter_study_pci[1]&!reset_tx&tx_enc_conf_sel[2]&link_up_loc)|(!tx_enc_conf_sel[1]&tx_enc_conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0]&!jitter_study_pci[1]&!reset_tx&link_up_loc))

C. Dealing with multiple decoders

Of the five benchmarks, only scrambler and XFI have more than one decoders, the user needs to inspect their corresponding precondition formulas to select the correct one.

For the two decoders of the scrambler, their corresponding precondition formulas are *reset* and *!reset*. By inspecting the Verilog source code of the scrambler, we found that the *reset* is used to reset the scrambler when it is *True*. Thus, the

scrambler will work in normal mode when *reset* is *False*. Therefore, the second decoder is the correct one.

For the two decoders of the most complex XFI encoder [4], their corresponding precondition formulas are *!TEST_MODE&RESET* and *!TEST_MODE&!RESET&DATA_VALID*. The only differences between them are the values of *RESET* and *DATA_VALID*. By inspecting the Verilog source code of XFI, we found that the *RESET* is used to reset the XFI encoder when it is *True*, and *DATA_VALID* means that the input data is valid when it is *True*. Thus, the XFI encoder will work in normal mode when *RESET* is *False* and *DATA_VALID* is *True*. Therefore, the second decoder is the correct one.

VI. RELATED WORKS

Complementary synthesis was first proposed in [1]. Shen et al. [9] and Liu et al. [11] improved its runtime with unsatisfiable core extraction and Craig interpolation, respectively. Shen et al. [10] and Liu et al. [11] proposed halting algorithms to determine the existence of the decoders by detecting loops.

The protocol converter synthesis [12] is the problem that automatically generates a translator between two different communication protocols, by computing a greatest fixed point for the update function of the buffer's control states.

VII. CONCLUSIONS

This paper has proposed a fully automatic approach to infer assertion for complementary synthesis and generate all decoders. Experimental results indicate that our approach can significantly reduce the human effort in specifying assertion.

REFERENCES

- [1] S. Shen, J. Zhang, Y. Qin, and S. Li, "Synthesizing complementary circuits automatically," in *ICCAD09*. IEEE, Nov. 2009, pp. 381–388.
- [2] S. Shen, Y. Qin, J. Zhang, and S. Li, "Inferring Assertion for Complementary Synthesis," in *ICCAD11*. IEEE, Nov. 2011, pp. 404–411.
- [3] "PCI Express Base Specification Revision 1.0". [Online]. Available: <http://www.pcisig.com>
- [4] "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.
- [5] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental SAT solving," in *ICCAD07*. IEEE, Nov. 2007, pp. 227–233.
- [6] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT03*, pages 502–518. Springer, May 2003.
- [7] M. K. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *ICCAD04*. IEEE, Nov. 2004, pp. 510–517.
- [8] K. L. McMillan, "Interpolation and SAT-based model checking," in *CAV03*. Springer, July 2003, pp. 1–13.
- [9] S. Shen, Y. Qin, K. Wang, L. Xiao, J. Zhang, and S. Li, "Synthesizing complementary circuits automatically," *IEEE trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 8, pp. 1191–1202, Aug. 2010.
- [10] S. Shen, Y. Qin, L. Xiao, K. Wang, J. Zhang, and S. Li, "A halting algorithm to determine the existence of the decoder," *IEEE trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 10, pp. 1556–1563, Oct. 2011.
- [11] S. Liu, Y. Chou, C. Lin, J. Jiang : Completely Automatic Decoder Synthesis. *accepted by ICCAD11*.
- [12] K. Avnit, V. D'Silva, A. Sowmya, S. Ramesh, and S. Parameswaran, "A formal approach to the protocol converter problem," in *DATE08*. IEEE, Mar. 2008, pp. 294–299.

25-Dec-2011

Dear Dr. Shen: Manuscript ID TCAD-2011-0407.R1 entitled "Inferring Assertion for Complementary Synthesis" which you submitted to the Transactions on Computer-Aided Design of Integrated Circuits and Systems, has been reviewed. The comments of the reviewer(s) are included at the bottom of this letter.

The reviewer(s) suggest some minor revisions to your manuscript. I invite you to respond to their comments and revise your manuscript.

To upload your revised manuscript, log into <http://mc.manuscriptcentral.com/tcad> and enter your Author Center, where you will find your manuscript title listed under "Manuscripts with Decisions." Under "Actions," click on "Create a Revision." Your manuscript number has been appended to denote a revision.

You may also click the link below to start the revision process (or continue the process if you have already started your revision) for your manuscript. If you use this link you will not be required to login to ScholarOne Manuscripts.

http://mc.manuscriptcentral.com/tcad?URL_MASK=Py8G8YXf22ZBZjHSqMdM

You will be unable to make your revisions on the originally submitted version of the manuscript. Instead, revise your manuscript using a word processing program and save it on your computer. Please also highlight the changes to your manuscript within the document by using bold, underlined, or colored text.

Once the revised manuscript is prepared, you can upload it and submit it through your Author Center.

When submitting your revised manuscript, you will be able to respond to the comments made by the reviewer(s) in the space provided. You can use this space to document any changes you make to the original manuscript. In order to expedite the processing of the revised manuscript, please be as specific as possible in your response to the reviewer(s).

IMPORTANT: Your original files are available to you when you upload your revised manuscript. Please delete any redundant files before completing the submission.

Because we are trying to facilitate timely publication of manuscripts submitted to the Transactions on Computer-Aided Design of Integrated Circuits and Systems, your revised manuscript should be submitted by 29-Jan-2012. If it is not possible for you to submit your revision by this date, we may have to consider your paper as a new submission.

Once again, thank you for submitting your manuscript to the Transactions on Computer-Aided Design of Integrated Circuits and Systems and I look forward to receiving your revision.

Sincerely,
Prof. Sachin Sapatnekar
Editor-in-Chief, Transactions on Computer-Aided Design of Integrated Circuits and Systems
tcad@umn.edu

Reviewer(s)' Comments to Author:

Reviewer: 1
Comments to the Author
The authors addressed my comments.

REPLY TO REVIEWER:

Thank you very much for your effort in polishing this paper.

Reviewer: 2

Comments to the Author

You have not answered my objection about enumeration of reversible encoders. (The most complex assertion you get consists of 6 variables. Finding the right assignment should be trivial.) There are three ways to address this objection.

1) You just admit that for the examples you tried enumeration of reversible encoders does not have much practical value but you believe that it may come useful in the future. You also give a plausible scenario where such enumeration becomes practical.

2) You give an example (that you have not described in the current manuscript) where enumeration of reversible encoders satisfying your assertion is really helpful

3) You remove the part about enumeration of encoders from the paper.

REPLY TO REVIEWER:

According to the last row of table I, these benchmarks have 16, 26 or even 120 configuration pins. It is definitely non-trivial to find the right assignments for so many pins. Therefore, the ability to infer an assertion with small number of variables from large number of configuration pins indicates exactly the power of our algorithm.

At the same time, for the most complex XFI encoder with 120 configuration pins, the user can easily select the correct decoder by inspecting only 2 pins, instead of all 120 pins.

Please also refer to page 1, right column paragraph 4, which starts with “For example”, and also the subsection V.C.

Reviewer: 3

Comments to the Author

Thanks for the clarification. However the following comments remain to be further clarified.

1. (Prior Comment 4: In Section IV.C, computing f using Craig interpolation may seem unnecessary as cofactoring f' already yields a desired solution. Having Craig interpolation in the computation may seem just the matter of speedup as stated in the last sentence of Section IV. If this is the case, please provide experimental evaluation comparing the computation with and without Craig interpolation.)

Please make room based on the suggestions below for a brief comparison on computations with and without Craig interpolation. Selecting one representative circuit to discuss is sufficient to let the reader gain some insights.

REPLY TO REVIEWER:

Actually, the Craig interpolation is used not only to speed up the computation, it is also used to get the formula na in step 9 of Algorithm 1. These na are used to compute the final assertion in step 5.

So Craig interpolation cannot be avoided in our algorithm.

2. (Prior Comment 6: For Section VI, there are no additional experiments compared to [2]. Essentially only Phase 1 is experimented. Please experiments on Phase 2 and Phase 3 computations as well. Showing statistics of the computed Boolean relations and their assertions (in terms of formula size, CPU time, etc.) provides the reader a better idea how the proposed algorithms work in practice.)

Please report the runtimes of Phases 1-3 separately.

REPLY TO REVIEWER:

I have changed it according to your suggestions. Please refer to table II and the third paragraph of Subsection V.A.

3. (Prior Comment 8: XFI and scrambler have 2 decoders each. Please list their respective two assertions, rather than just one each. (Prior Comment 9) How can scrambler have 2 decoders with its assertion equals “True”? It seems that the assertions of different decoders must be disjoint.)

What is the difference between “precondition formulas” and “inferred assertions”? While they seem to be the same, the shown formulas of XFI and scrambler in Section V.B and Section V.C in this revision are not well related. Please discuss their connections.

REPLY TO REVIEWER:

It is my fault that I had not presented it clearly.

Please refer to the paragraph under the Equation (7), which starts with “Two different”.

Actually, a decoder R_i ’s precondition formula IA_i is the set of configuration letters that leads to the existence of R_i . So the union of all such IA_i should be equal to the assertion IA inferred by Algorithm 1. This is also indicated by the experimental result in Subsection V.C.

In the last submission (the 2nd submission) of this paper, the union of IA_1 and IA_2 of XFI is not equal to IA , this was caused by a bug in the implementation of my AIG package.

This bug causes two problems in the last submission:

- 1 The Inferred assertion of XFI includes an additional sub-formula TEST_MODE. This problem does not exist in current submission(the 3rd submission). Please refer to Subsection V.B.**
- 2 The first precondition formula IA_1 of XFI was RESET &!TEST_MODE & !DATA_VALID in the last submission. It is not correct, and now it is corrected and is RESET &!TEST_MODE in this submission. Please refer to the third paragraph of Subsection V.C.**

The correction of this bug also leads to the change of the inferred assertion of T2_ether benchmark in Subsection V.B . But the new assertion is logically equivalent to that of the last submission.

4. Please explain the runtime differences (especially the substantial slow-downs of XFI and T2 Ether) between this new version and prior submitted version. Is it due to bug fixing or other reasons?

REPLY TO REVIEWER:

Yes, there is a deeply hidden bug in the program of the first submission. The encoding space of two totally unrelated SAT instances are overlapped with each other, so the result of merging these two SAT instances was not equal to their conjunction.

I correct this bug in the program of the second submission, which had led to significantly larger runtime overhead.

In the last two month, I had further improved the runtime overhead of my program, so it is much faster now in current submission. Please refer to Table II.

Other suggestions:

P1. “i) Manually specifying ...” -> “i) manually specifying ...”

P1. “me” -> “the first author”

P1. “... pins, in which only two have ...” -> “... pins, only two of which have ...”

P1. “these TWO pin” -> “these two pins”

REPLY TO REVIEWER: All are changed. Thank you for your suggestions.

P4. Table I can be removed by referring to [2] (with "#Config pin" info moved back to Table II) to make room for further clarification.

REPLY TO REVIEWER: According to our explanation about Craig interpolation, we don't need to make room any more.

P5. The last paragraph of Section V.C (as it repeats what was said in Section I) can be removed to make room for further clarification.

REPLY TO REVIEWER: I remove this paragraph to make room for explaining the issue 3.

Associate Editor: Kunz, Wolfgang

Comments to the Author:

Your paper has improved but requires another revision. The reviewers give specific suggestions. In case you should decide not to follow any of these suggestions in your revision please explain this carefully in your rebuttal.