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, we 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 me in [1]. To avoid this tedious **task**, in this study, we propose the following three algorithms.

First, we propose an algorithm that automatically infers 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

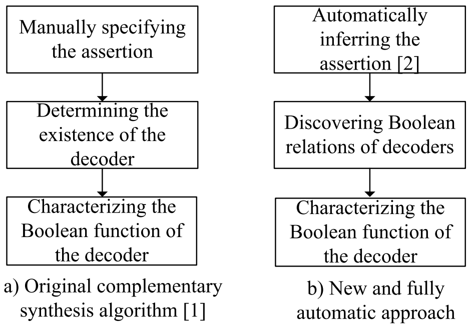

Manually specifying the assertion
Determining the existence of the decoder
Characterizing the Boolean function of the decoder
a) Original complementary synthesis algorithm [1]
Automatically inferring the assertion [2]
Discovering Boolean relations of decoders
Characterizing the Boolean function of the decoder
b) New and fully automatic approach

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), we propose a second algorithm to discover all these decoders by iteratively testing whether $\mathbb{R}$, the Boolean relation that uniquely determines the encoder's input, functionally depends on those discovered decoders. For every configuration value that fails this test, a new decoder's Boolean relation is discovered by asserting this configuration value into $\mathbb{R}$. This step is repeated until all decoders are discovered.

Third, to help the user select the correct decoder, we propose a third algorithm 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.

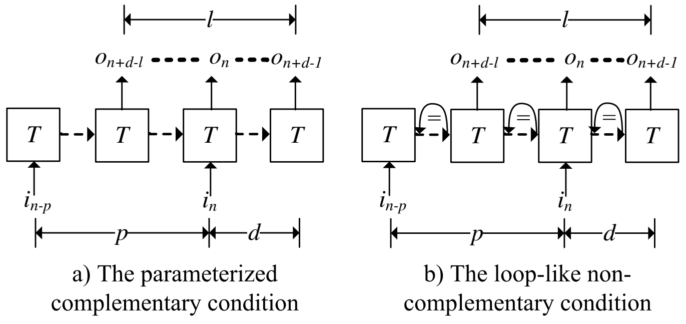
For example, for the two decoders of XFI discovered by our algorithm, their corresponding precondition formulas refer to only three pins, in which only two have different values. Therefore, to select the correct decoder, one only needs to find out the meaning of these TWO pin, instead of all 120 pins. 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: {syshen, yingqin, kfwang, zbpang, jmzhang, skli}@nudt.edu.cn, (see http://www.ssypub.org/).



l
o_{n+d-l} o_n o_{n+d-1}
T
i_{n-p} i_n
p d
a) The parameterized complementary condition
b) The loop-like non-complementary condition

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 $T : S \times I \times C \to S \times O$ 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. For a configuration letter c, $R(c)$ 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, $E \vDash PC(p, d, l, R)$ 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).

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

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 $< p, d, l >$ that makes $E \vDash PC(p, d, l, R)$ holds true. **Therefore**, intuitively, the decoder does not exist if for every parameter value tuple $< p, d, l >$, we can always find another tuple $< p', d', l' >$ with $p' > p$,$l' > l$ and $d' > d$, such that $E \vDash PC(p', d', l', R)$ 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, $E \vDash LN(p, d, l, R)$ 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).

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

B. Algorithm implementation

Algorithm 1 InferAssertion

```
1:  $NA = \{\}$
2:  for $x = 0 \to \infty$ do
3:      $< p, d, l >=< 2x, x, 2x >$
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 $< p, d, l >$ in Line 3. For every configuration letter $A(c)$ 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].**

An earlier study [2] had proved that this algorithm 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 decoder.

A. Constructing SAT instance to discover decoders

Assume the assertion inferred by Algorithm 1 is:

$$IA \overset{def}{=} \bigwedge_{na \in NA} \neg na \tag{3}$$

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

$$\begin{aligned} X &\overset{def}{=} o_{n+d-l}^{n+d-1} \\ Y &\overset{def}{=} i_n \end{aligned} \tag{4}$$

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

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

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

$$Y = f(c, X) \tag{6}$$

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

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

Two different c_i and c_j may share the same $\mathbb{R}_{c_i}$. **Therefore**, IA can be partitioned into $\{IA_1, \ldots, IA_n\}$, 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. **Thus, $\mathbb{R}_i$ defines a mapping from X to Y, which is actually a function $f_i : X \to Y$. Therefore, f can be rewritten as:**

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

Thus, our **task** here is to find out the set $\{\mathbb{R}_1, \ldots, \mathbb{R}_n\}$ step–by–step. Assume that we have already discovered a set of decoders' Boolean relations $\{\mathbb{R}_1, \ldots, \mathbb{R}_m\}$. To test whether it contains $\{\mathbb{R}_1, \ldots, \mathbb{R}_n\}$, that is, whether all decoders have already been discovered, we construct the following SAT instance, which is also shown in Fig. 3:

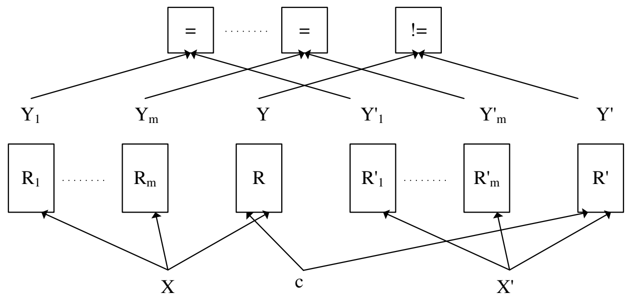


Fig. 3. The SAT instance that discovers decoders

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

Line 1 of Equation (9) represents the Boolean relations in $\{\mathbb{R}_1, \ldots, \mathbb{R}_m\}$ 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 $\{\mathbb{R}_1, \ldots, \mathbb{R}_m\}$ contains $\{\mathbb{R}_1, \ldots, \mathbb{R}_n\}$.

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

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

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 as follows, is a newly discovered decoder's Boolean relation:

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

To prove that our approach does not carry out redundant work, we need to prove that $\mathbb{R}_{m+1}$ has not 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 $0 \le i \le m$ 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 $Y \not\equiv Y'$ in the last line of Equation (9), which concludes the proof. ■

Based on all 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:

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

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:

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

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:

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

TABLE I
INFORMATION ON BENCHMARKS

	XGXS	XFI	scrambler	PCIE	T2 Ethernet
Line number of verilog source	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 $ITP : C \times \mathbb{B}^m \to \mathbb{B}$ 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:

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

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:

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

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.

The fourth row shows the total runtime of our new algorithm, while the fifth row shows the value of the discovered d, p, and l. The last row shows the number of discovered decoders.

By comparing the second and the fourth rows, it is obvious that our approach is much slower than that of [10],

TABLE II
EXPERIMENTAL RESULTS

		XG-XS	XFI	scra-mbler	PCI-E	T2 E-ther
[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	3.83	3841.34	18.73	8.51	1791.22
	d,p,l	1,5,1	0,5,2	0,4,2	2,5,1	4,5,1
	#decoders	1	2	2	1	1

which is caused by the much more complicated procedures, $InferCove\ ringFormula$ and $DiscoveringDecoders$.

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

B. Inferred assertions

For XGXS: `( ( !bad_code ) )`

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

For scrambler: `True`

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

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

C. Dealing with multiple decoders

The last row of Table II indicates that in most of the cases, our algorithm generates only one decoder. For other cases with multiple decoders, the user need to inspect $\{IA_1, \ldots, IA_m\}$ 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, and the dynamic simulation had confirmed its correctness.

For the two decoders of the most complex XFI encoder [4], their corresponding precondition formulas are $RESET\ \&!TEST\_MODE\ \&\ !DATA\_VALID$ and $!RESET\ \&\ !TEST\_MODE\ \&\ DATA\_VALID$. 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 decoder, and the dynamic simulation had also confirmed its correctness.

In this process, the user only needs to inspect the meaning of two configuration pins, instead of all 120 configuration pins of the XFI encoder. In this way, the human effort in specifying assertion is significantly reduced.

VI. RELATED WORKS

Complementary synthesis was first proposed in [1]. Its major shortcomings are that it may not halt and its runtime overheads in building decoders are large. The runtime problem was addressed by simplifying the SAT instance with unsatisfiable core extraction [9] and Craig interpolation [11], while the halting problem was handled by detecting loops [10], [11].

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, J. Zhang, Y. Qin, and S. Li, “Inferring Assertion for Complementary Synthesis,”*accepted by ICCAD11.*.

[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.

05-Oct-2011

Dear Dr. Shen:

Manuscript ID TCAD-2011-0407 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) have requested that your paper be shortened to a transactions brief. 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=jGm4kt98NCSyhfsGhD3x

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 09-Nov-2011. 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 contribution makes sense, and the paper is structured well.
The authors should check the work in
Ed Clarke, Reconsidering Cegar: learning good abstractions without refinement
(and related publications) and discuss any connections that exist - conceptual
or technical -- to their work.

REPLY TO REVIEWER: Thank you for your advice. But our algorithm does not have any connection to abstraction, because it does not construct abstract transition relation or remove state variables.

The material added to the ICCAD paper seems significant.

The writing includes occasional glitches in terms of grammar and style.

! So, we propose
 Therefore, we propose
% Do not use "So" as "Therefore" in scholarly publications. It is too colloquial.

REPLY TO REVIEWER: Thank you for your advice. I have changed them.

! that reversely computes
 that inverts
% OR
 that computes an inverse of
% OR "the inverse", if it is unique

REPLY TO REVIEWER: Thank you for your advice. I have changed it to a more appropriate statement: "that uniquely determines".

> To help the user selects
  To help the user select
REPLY TO REVIEWER: Thank you for your advice. I have changed them.

On page 1 line 11, line 18, line 47 (left column) and page 5, line 3, there are missing ligatures "fi"in "configuration", "final" and "defined" -- this may be an issue with PDF processing or cut-and-paste, but needs to be resolved one way or another. Also, I am not giving a full list of these problems -- the authors must carefully check the paper. There's also a spurious spare character in "assertio n" in the abstract. If the authors can't write a clean abstract, this paper can definitely NOT be published.

REPLY TO REVIEWER: Thank you for your advice. I will try my best to avoid this. I use acrobat, evnince and xpdf to view my paper, but can not find such problem in the version downloaded from TCAD website. So if you find such problem again, can you forward it to me through the associate editor? Thank you again.

! One of the most difficult jobs in designing
  One of the most difficult tasks in designing

REPLY TO REVIEWER: Thank you for your advice. I have changed it.

! Manually specifying an assertion needs the user
! to read lots of documentations and try many combinations
 To manually specify an assertion, the user must read extensive documentation and often perform laborious trial-and-error process.
% note that "documentation" is uncountable in English, so there's no plural form

REPLY TO REVIEWER: Thank you for your advice. I have changed it.

% I am not going to proofread the entire paper - it's the authors' job.

REPLY TO REVIEWER: Thank you for your work. I will proofread it more carefully.

Reviewer: 2
Comments to the Author
My main objections are o the problem you are solving and to methods you have chosen.

1) First of all, reading a manual is not such a bad idea. Of course if you had an undocumented chip and wanted to find its working configuration the methods you describe make a perfect sense.
However, even if replacing a lost manual is a tough problem it hardly merits a scientific publication. (Safe-cracking is not easy, but nobody writes papers about it.)

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my motivation more clearly.
According to my working experiment on complementary synthesis, reading documentation and find out the correct combination is a very tedious work.
Please refer to page 1, right column paragraph 4, which starts with “For example”, and the subsection V.C. They describe my experience on XFI encoder with 120 configuration pins. Finding the meaning and correct combination of them take me a very long time. But under the guidance of the precondition formulas inferred by our new algorithm, I only need to find out the meaning of two pins instead of 120 pins. Obviously this can significantly save the human effort.

2) But let us assume that indeed the problem at hand is to save the user's time

I still have some doubts about the methods you employ. The largest assertion among the four examples you give in the experimental section has 6 variables. I assume that any reasonably good user should be able to find the correct configuration by making a sensible assignment to 6 variables that satisfies this assertion. If necessary this user may take a look at the manual to find out what these 6 variables specify.

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my motivation more clearly.
Yes, you are right, these assertions has very small number of variables. But if you refer to the last row of table 1, you can find that these benchmarks have 16, 26 or even 120 configuration 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.
Under the guidance of these inferred formulas, the user can focus on those variables mentioned by these formulas, which can significantly save their time.

Instead, you generate all possible encoders (that have decoders) for the configurations satisfying this simple assertion and then let the user pick the right encoder based on the assertion specifying the configurations corresponding to this encoder. Note that the user still has to be able to distinguish between wrong assignments to those 6 variables.
It beats me, why it cannot be done without generating all possible encoders.

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my motivation more clearly.
Yes, you are right, reading the documentation can not be totally avoided. But with the formulas inferred by our algorithm, the user can focus on only 6, instead of 120 variables. That is a significantly saving.
Actually, although they mention up to 6 variables, the user don' t need to inspect them all. They only need to inspect those with different values. According to subsection V.C, no more than two pins are needed to be inspected.

3) Of course, hypothetically, one can have a very complex assertion eliminating wrong configurations that involves many variables. But then, it will be hard for the user to manually identify the correct configuration due to shear complexity of this communication chip. Besides, in this case the number of encoders may be very large.

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my idea more clearly.
Yes, you are right, so many configuration pins (up to 120) may corresponds to many many many encoders, but most of them do not have corresponding decoders, which will be ruled out by the first step of our algorithm.

4) One more technical comment. You build an assertion excluding wrong configurations iteratively repeating two steps:
a) Find an C counter-example proving that under particular assignment of configuration variables no decoder exists.

b) Then you build an interpolant to increase the set of excluded configurations.

What you do at step b) is frequently used in algorithms of quantifier elimination based on enumerating satisfying assignments.
There are more than a few papers on this topic, for example "M.Ganai, A.Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring", in Proc. ICCAD-2004, pp.510-517.

This paper describes a trick (circuit cofactoring) with the same objective you have. Namely, one satisfying assignment is used to remove many more.

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my idea more clearly.
Yes, you are right, that paper describes exactly what we do in our paper, except that we use craig interpolant to characterize the formula.
So we remove the detail description and refer the reader to our ICCAD' 11 paper.
At the same time we add a citation to the cofactoring paper to show the connection between it and ours.

Reviewer: 3
Comments to the Author
This paper addresses the problem of computing assertions in complementary circuit synthesis. The computation consists of three phases: 1) computing assertion configurations, 2) computing the set of Boolean relations, and 3) computing the assertion configuration of each Boolean relation. The problem formulation and its corresponding computation have practical applications.

The presentation in several ways is not clear and should be improved.

1. Boolean relation and its defined function(s) are never explicitly defined. Please provide the definitions in Section III with proper references.

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my idea more clearly.
Yes, you are right. As my paper has been shorten to 5 pages, I don' t have the space to define it separately, so I change the statement a little in place. Please refer to page 3 left column, section IV.A, the two paragraphs in bold font are changed according to your advice.

2. As Theorem 2 is just a slight extension of [16], the appended proof can be completely removed by simply referring the reader to [16].

REPLY TO REVIEWER: Thank you for your advice. I have changed it.

3. In the discussion of Section IV.C, the o variables seem to be incorrectly omitted. For example, f depends on o, o should be asserted in Equation (7), etc. Please correct related formulations. Otherwise the interpolation may be incorrect.
REPLY TO REVIEWER: Thank you for your advice.
Yes, you are right, the o variables are omitted intentionally. Because these o variables are the output of the transition relations, their values are uniquely determined by the

input and configuration variables. So they do not need to be asserted.
On the other hand, asserting these o variables may significantly narrow down the size of space covered by the interpolation, and then slow down our algorithm.
So we chose to omit o variables. Of course, our paper has be shorten to 5 pages, so we can only remove this subsection and refer the reader to our iccad 11 paper.

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.

REPLY TO REVIEWER: Thank you for your advice.
Yes, you are right, the formula obtained after cofactoring is already the set of enlarged invalid configuration letter. But it contains the whole relation F_{LN}, which is often very large. So we use interpolant to characterize it and then use BDD to minimize it. After that, their sizes are significantly reduced. You can find out this by looking at the inferred assertion shown in subsection V.B.
I am sorry that I do not have enough space to compare the computation with and without interpolation, because our paper has been shorten to 5 pages.

5. The presentation of Section V.A seems to hint that the function defined by R is unique. Please comment on this.

REPLY TO REVIEWER: Thank you for your advice.
Yes, you are right, the function defined by R is unique. Because R can uniquely determine Y from c and X, which means it defines a mapping from c and X to Y, which is actually a function. Please refer to subsection IV.A, the 4th paragraph in bold font.

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.

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my idea more clearly.
Actually the paper [2] only includes the phase 1, while the phase 2 and 3 are newly added in this paper.

7. Please compare the computed assertions of [2] and this work. It seems to the reviewer that the assertions of XGXS, PCI-E, scrambler, and T2 ethernet in [2] and this work are functionally the same, whereas those of XFI are functionally different. Please explain why the computed assertions can differ.
REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not checked it carefully.
For the assertion inferred for XFI, I compare the following three versions
1) ICCAD11

2) TCAD initial submitted version
3) The assertion inferred by current tool implementation
These three are all different from each other in syntax, but the 1) and 3) are equal, while the 2) is not. I think this is caused by preparing the TCAD initial submitted version with latex \texttt, which requires me to replace _ with _, & with \&, and also need to insert space in proper position to break too long symbol to two lines. In this process, it is very likely that I delete an ! unintentionally. I am really sorry about it.
But in this revised version, I update all experimental result, including these assertions, with the result generated by new tool implementation, which correct some bugs in previous implementation. So if you find any further inconsistency, it may be caused by the bug in the old implementation. But I still welcome your comment, because it may give me another opportunity to fix further bugs.
Please refer to www.ssypub.org to download the source code of new implementation, and try it to verify my experimental result.

8. XFI and scrambler have 2 decoders each. Please list their respective two assertions, rather than just one each.
REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my idea clearly.
Actually, each encoder has one inferred assertion, which represents the cases with decoders.
But for each decoder, they only have a precondition formula, which has been shown in subsection V.C.

9. How can scrambler have 2 decoders with its assertion equals true ? It seems that the assertions of different decoders must be disjoint.
REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not checked it carefully.
Actually this is a bug in my initial implementation of my algorithm. Now it is corrected. Please refer to subsection V.C.
Please refer to www.ssypub.org to download the source code of new implementation, and try it to verify my experimental result.

10. In introduction, it was said that for other benchmarks with multiple decoders, the user can easily select the correct one, by inspecting the precondition formulas and finding out the meaning of no more than one pin, instead of up to 120 pins like the original approach [1].?Which benchmark circuit and which pin do the authors refer to in the experiment?

REPLY TO REVIEWER: Thank you for your advice. It is my fault that has not described my idea clearly.
This benchmark is XFI, I have rewritten the 4 th paragraph on page 1 right column.

11. It was said that the user can easily select the correct decoder by inspecting these preconditions. ?Please elaborate how the selection can be done. It may seem to the reviewer that the selection cannot be done unless the user already know the right configuration, but in this case there is no need to compute inferred assertions.
REPLY TO REVIEWER: Thank you for your advice.
Yes, you are right. The user still needs to read the documentation or the source code, but under the guidance of the inferred precondition formula, they only need to find out

the meaning of those variables with different value in IA_0 and IA_1.
For XFI, the two inferred precondition formula refer to only 3 instead of all 120 configuration pins, and only 2 pins have different values. Therefore, the user only needs to figure out the meaning of these two pins, instead of 120.
Please refer to subsection V.C for more details.

Associate Editor: Kunz, Wolfgang
Comments to the Author:
The paper is considered to make a worthwhile contribution although there are doubts about its practical importance. In the revision as Transaction Brief please give a better motivation for the practical relevance of your work.