

## Complementary Synthesis for Encoder with Flow Control Mechanism

YING QIN and SHENGYU SHEN and YAN JIA, School of Computer, National University of Defense Technology

Complementary synthesis automatically generates an encoder's decoder with the assumption that the encoder's input can always be uniquely determined by its output. However, many modern encoders employ flow control mechanism that inserts *invalid* data into the encoder's input sequence, which prevents some input signals from being uniquely determined. None of the current algorithms can handle such cases.

By studying several complex encoders, we found that all flow control mechanisms classify the encoder's input signals into two sets: the flow control signals that can always be uniquely determined, and the data signals whose validness is indicated by a predicate over the flow control signals. When the data signals are invalid, the decoder is supposed to recover only the flow control signals, while the data signals can be ignored.

Thus, we propose a three-step algorithm to handle such encoders with flow control mechanism. First, it finds out all signals that can be uniquely determined and take them as flow control signals. Second, it infers a predicate over these flow control signals that can make all other data signals to be uniquely determined. Third, it characterizes the decoder's Boolean function for each input signal, by first building a conjunction of two formulas that assign conflicting values to this input signal, and then generating a Craig interpolant from its unsatisfiability proof. But for the data signals, the inferred predicate must be enforced first.

Experimental results on several complex encoders indicate that our algorithm can always correctly find out the flow control signals, infer the predicates and generate the decoder's Boolean functions.

Categories and Subject Descriptors: B.5.2 [Design Aids]: Automatic synthesis; B.6.3 [Design Aids]: Automatic synthesis

General Terms: Algorithms, Logic synthesis, Verification

Additional Key Words and Phrases: Craig interpolation, decoder, encoder, finite-state transition system, satisfiability solving

### ACM Reference Format:

Ying Qin, Shengyu Shen, and Yan Jia, 2014. Complementary Synthesis for Encoder with Flow Control Mechanism. *ACM Trans. Des. Autom. Electron. Syst.* 9, 4, Article 39 (March 2010), 15 pages.  
DOI: <http://dx.doi.org/10.1145/0000000.0000000>

## 1. INTRODUCTION

One of the most difficult jobs in designing communication and multimedia chips, is to design and verify complex encoder and decoder pairs. The encoder maps its input signals  $\vec{i}$  to its output signals  $\vec{o}$  according to some predefined rules, while the decoder recovers  $\vec{i}$  from  $\vec{o}$ . Complementary synthesis [Shen et al. 2009] facilitates this job by automatically generating an encoders' decoder. As shown in Figure 1a), this algorithm assumes that the encoder eventually reaches and never leaves the unique state set,

---

This work was funded by projects 61070132 and 61133007 supported by National Natural Science Foundation of China.

Author's addresses: Ying Qin, School of Computer, National University of Defense Technology; Shengyu Shen, School of Computer, National University of Defense Technology; Yan Jia, School of Computer, National University of Defense Technology.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 2010 ACM 1084-4309/2010/03-ART39 \$15.00

DOI: <http://dx.doi.org/10.1145/0000000.0000000>

Fig. 1. An encoder with flow control mechanism

in which  $\vec{i}$  can always be uniquely determined by a bounded sequence of  $\vec{o}$ . The set of all other states are called the non-unique state set, because  $\vec{i}$  can not be uniquely determined in it by a bounded sequence of  $\vec{o}$ .

At the same time, high speed communication system, such as Ethernet [wikipedia 2013b] and PCIe [wikipedia 2013d], run at several Giga bits per second. It's impossible to distribute so fast a clock to both the transmitter and receiver. So they must be driven by clocks with slightly different frequencies. To prevent the faster transmitter from overwhelming the slower receiver, flow control mechanism [wikipedia 2013c] is widely employed to insert *invalid* data into the sequence of input signals  $\vec{i}$ , such that the decoder can recognize and discard them.

As shown in Figure 1b), an encoder with flow control mechanism partitions its input signals  $\vec{i}$  into two set: the data signals  $\vec{d}$  to be encoded, and the flow control signals  $\vec{f}$  indicating the validness of  $\vec{d}$  with a predicate  $valid(\vec{f})$ . When  $valid(\vec{f}) \equiv 1$ , the encoder maps  $\vec{d}$  with an encoding function  $enc$  to the output signals  $\vec{o} := enc(\vec{d})$ , in which case both  $\vec{d}$  and  $\vec{f}$  can be uniquely determined by a bounded sequence of  $\vec{o}$ .

However, when  $valid(\vec{f}) \equiv 0$ , the encoder drives some predefined patterns to  $\vec{o}$  independent of  $\vec{d}$ , such as the K28.5 character defined in clause 36 of IEEE 802.3 standard [IEEE 2012b], which can uniquely determine only  $\vec{f}$ , but not  $\vec{d}$ . In this case, the decoder is supposed to recognize only the validness of  $\vec{d}$  instead of their exact value, which means recovering  $\vec{f}$  is enough.

So just like the path 2 in Figure 1a), such an encoder may visit the non-unique state set infinitely often. Thus, none of the current complementary synthesis algorithms can handle such encoders, because they all assume that  $\vec{i}$  can always be uniquely determined by  $\vec{o}$ , and enforce this by manually specified assertion [Shen et al. 2009;2010; 2011], or automatically inferred assertions [Shen et al. 2012].

We propose in this paper a novel three-step algorithm to handle such encoders with flow control mechanism. **First**, we apply the classical halting complementary synthesis algorithm [Shen et al. 2011] to find out all those flow control signals  $\vec{f}$  that can be uniquely determined. **Second**, we infer a sufficient and necessary predicate  $valid(\vec{f})$  that make  $\vec{d}$  to be uniquely determined by a bounded sequence of the encoder's output  $\vec{o}$ . **Finally**, we characterize the decoder's Boolean function for each flow control signal  $f \in \vec{f}$  by building two copies of unrolled transition function sequence with common output sequences  $\vec{o}$ , but different value on  $f$ . The conjunction of these two copies are unsatisfiable, so a Craig interpolant [McMillan 2003] can be built and used as the decoder's Boolean function that computes  $f$ . On the other hand, for other input data signals  $\vec{d}$ , their values are only meaningful when  $valid(\vec{f}) \equiv 1$ . Thus, the decoder's Boolean functional computing each  $d \in \vec{d}$  can be built similarly, but with  $valid(\vec{f}) \equiv 1$  enforced first.

The second step of this algorithm seems somewhat similar to that of [Shen et al. 2012] in the sense that both algorithms infer assertions that can make  $\vec{d}$  or  $\vec{i}$  to be uniquely determined. But the essential difference between them is, the algorithm of [Shen et al. 2012] inferred a **global** assertion that must be enforced at all cycles, such that the encoder eventually reaches and never leaves the unique state set. While our algorithm inferred a **local** predicate that is enforced at the current cycle only when we need to recover the value of  $\vec{d}$ . So our algorithm is the first algorithm that allows the unique and non-unique states to be interleaved freely.

Experimental results indicate that, for several complex encoders from real projects (e.g., Ethernet [wikipedia 2013b] and PCIe [wikipedia 2013d]), our algorithms can always find out the flow control signals, infer the predicates and generate the decoders.

*The remainder of this paper is organized as follows.* Section 2 introduces background materials. Section 3 presents the algorithm that finds out the flow control signals, while Section 4 infers the predicate that makes  $\vec{d}$  to be uniquely determined by a bounded sequence of  $\vec{o}$ . Section 5 presents the algorithm to characterize the decoder's Boolean function. Sections 6 and 7 present the experimental results and related works. Finally, Section 8 provides the conclusion.

## 2. PRELIMINARIES

### 2.1. Propositional satisfiability

The Boolean value set is denoted as  $B = \{0, 1\}$ . A vector of variables is represented as  $\vec{v} = (v, \dots)$ . If a variable  $v$  is a member of  $\vec{v}$ , that is  $\vec{v} = (\dots, v, \dots)$ , then we say  $v \in \vec{v}$ . Otherwise we say  $v \notin \vec{v}$ . For a variable  $v$  and a vector  $\vec{v}$ , if  $v \notin \vec{v}$ , then the new vector that contains both  $v$  and all members of  $\vec{v}$  is denoted as  $v \cup \vec{v}$ . If  $v \in \vec{v}$ , then the new vector that contains all members of  $\vec{v}$  except  $v$  is denoted as  $\vec{v} - v$ . For two vectors  $\vec{a}$  and  $\vec{b}$ , the new vector with all members of  $\vec{a}$  and  $\vec{b}$  is denoted as  $\vec{a} \cup \vec{b}$ . The set of truth valuations of  $\vec{v}$  is denoted as  $\llbracket \vec{v} \rrbracket$ , for instance,  $\llbracket (v_1, v_2) \rrbracket = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ .

A Boolean formula  $F$  over a variable set  $V$  is constructed by connecting variables from  $V$  with symbols  $\neg$ ,  $\wedge$ ,  $\vee$  and  $\Rightarrow$ , which stand for logical connectives negation, conjunction, disjunction, and implication, respectively.

The propositional satisfiability problem (abbreviated as SAT) for a Boolean formula  $F$  over a variable set  $V$  is to find a satisfying assignment  $A : V \rightarrow B$ , so that  $F$  can be evaluated to 1. If such a satisfying assignment exists, then  $F$  is satisfiable; otherwise, it is unsatisfiable.

According to [Ganai et al. 2004], the positive and negative cofactors of  $f(v_1 \dots v_n)$  with respect to variable  $v$  are  $f_v = f(v_1 \dots 1 \dots v_n)$  and  $f_{\bar{v}} = f(v_1 \dots 0 \dots v_n)$ , respectively. **Cofactoring** is the action that applies 1 or 0 to  $v$  to get  $f_v$  or  $f_{\bar{v}}$ .

Given two first order logic formulas  $\phi_A$  and  $\phi_B$ , with  $\phi_A \wedge \phi_B$  unsatisfiable, there exists a formula  $\phi_I$  referring only to the common variables of  $\phi_A$  and  $\phi_B$  such that  $\phi_A \Rightarrow \phi_I$  and  $\phi_I \wedge \phi_B$  is unsatisfiable. We call  $\phi_I$  the **interpolant** [Craig 1957] of  $\phi_A$  with respect to  $\phi_B$ , and used McMillan's algorithm [McMillan 2003] to generate it.

### 2.2. Finite state machine

The encoder is modeled by a finite state machine  $M = (\vec{s}, \vec{i}, \vec{o}, T)$ , consisting of a state variable vector  $\vec{s}$ , an input variable vector  $\vec{i}$ , an output variable vector  $\vec{o}$ , and a transition function  $T : \llbracket \vec{s} \rrbracket \times \llbracket \vec{i} \rrbracket \rightarrow \llbracket \vec{s} \rrbracket \times \llbracket \vec{o} \rrbracket$  that computes the next state and output vector from the current state and input vector.

We denote the state variable  $s \in \vec{s}$ , input variable  $i \in \vec{i}$  and output variable  $o \in \vec{o}$  at the  $n$ -th cycle respectively as  $s_n$ ,  $i_n$  and  $o_n$ . We further denote the state, the input and the output vector at the  $n$ -th cycle respectively as  $\vec{s}_n$ ,  $\vec{i}_n$  and  $\vec{o}_n$ . A **path** is a state sequence  $\langle \vec{s}_n, \dots, \vec{s}_m \rangle$  with  $\exists \vec{i}_j \vec{o}_j (\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$  for all  $n \leq j < m$ . A **loop** is a path  $\langle \vec{s}_n, \dots, \vec{s}_m \rangle$  with  $\vec{s}_n \equiv \vec{s}_m$ .

### 2.3. The halting algorithm to determine whether an input signal can be uniquely determined by a bounded sequence of output vector

State-of-the-art complementary synthesis algorithms [Shen et al. 2009; Liu et al. 2011; Tu and Jiang 2013] all assume that  $\vec{i}$  can be uniquely determined, so they always take  $\vec{i}$  as a whole, and never talk about individual variables  $i \in \vec{i}$ . But in this paper, we

Fig. 2. Checking whether the input can and can NOT be uniquely determined by the output

---

**ALGORITHM 1:** *CheckUniqueness*( $T, i$ ): The halting algorithm to determine whether  $i$  can be uniquely determined by a bounded sequence of output vector  $\vec{o}$

---

**Input:** The encoder's transition relation  $T$ , and the input bit  $i$ .

**Output:** whether  $i$  can be uniquely determined by  $\vec{o}$ , and the value of  $p$ ,  $l$  and  $r$ .

---

```

1  $p := 1$ ;  $l := 1$ ;  $r := 1$ ;
2 while 1 do
3    $p++$ ;  $l++$ ;  $r++$ ;
4   if  $F_{PC}(p, l, r)$  is unsatisfiable then
5     return  $(1, p, l, r)$ ;
6   else if  $F_{LN}(p, l, r)$  is satisfiable then
7     return  $(0, p, l, r)$ ;
8
```

---

need to check each  $i \in \vec{i}$  one by one, so there may be minor difference between our presentation and that of [Shen et al. 2009; Liu et al. 2011; Tu and Jiang 2013].

As shown in Figure 2a),  $i \in \vec{i}$  can be uniquely determined, if there exist  $p$ ,  $l$  and  $r$ , such that for any particular valuation of the output sequence  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$ ,  $i_{p+l}$  can not take on two different values. This can be checked by solving  $F_{PC}(p, l, r)$  in Equation (1).

(1)

Here,  $p$  is the length of the prefix state transition sequence that leads the encoder into the unique state set, in which  $i$  can be uniquely determined.  $l$  and  $r$  are the length of the two output sequences  $\langle \vec{o}_{p+1}, \dots, \vec{o}_{p+l} \rangle$  and  $\langle \vec{o}_{p+l+1}, \dots, \vec{o}_{p+l+r} \rangle$  that are on the left-hand and right-hand side of  $i_{p+l}$  used to determine  $i_{p+l}$ . Line 1 of Equation (1) corresponds to the path in Figure 2a), while Line 2 is a copy of it. Line 3 forces these two paths' output sequences to be the same, while Line 4 forces their  $i_{p+l}$  to be different.

Thus, we have the following proposition:

**PROPOSITION 2.1.** *If  $F_{PC}(p, l, r)$  is unsatisfiable, then  $i_{p+l}$  can be uniquely determined by  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$ .*

**On the other hand**, if  $F_{PC}(p, l, r)$  is satisfiable, then  $i_{p+l}$  can not be uniquely determined by  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$  for this particular  $p$ ,  $l$  and  $r$ . It may be uniquely determined for larger  $p$ ,  $l$  and  $r$ , or not uniquely determined for any  $p$ ,  $l$  and  $r$  at all. If it is the first case, then we can iteratively increase the value of  $p$ ,  $l$  and  $r$ , and eventually make  $F_{PC}(p, l, r)$  unsatisfiable. But if it is the second case, then this iterative algorithm will never terminate.

So, to obtain a halting algorithm, we need to distinguish these two cases. One such solution is shown in Figure 2b), which is similar to Figure 2a) but with three additional constraints to detect loops on the three state sequences  $\langle \vec{s}_0, \dots, \vec{s}_p \rangle$ ,  $\langle \vec{s}_{p+1}, \dots, \vec{s}_{p+l} \rangle$  and  $\langle \vec{s}_{p+l+1}, \dots, \vec{s}_{p+l+r} \rangle$ . It is formally defined in Equation (2) with the last three lines correspond to the three constraints used to detect loops.

(2)

**ALGORITHM 2:** *FindFlowControl*( $T, \vec{i}$ ): Finding out the flow control signals**Input:** The encoder's transition relation  $T$ , and its input vector  $\vec{i}$ .**Output:**  $\vec{f} \subset \vec{i}$  is the vector of the encoder's input bits that can be uniquely determined by a bounded sequence of output vector  $\vec{o}$ , and the maximal value of  $p, l$  and  $r$ .

```

1  $\vec{f} := \langle \rangle$ ;  $p_{max} := 0$ ;  $l_{max} := 0$ ;  $r_{max} := 0$ ;
2 foreach  $i \in \vec{i}$  do
3    $(uniq, p, l, r) := \text{CheckUniqueness}(T, i)$ ;
4   if  $uniq \equiv 1$  then
5      $\vec{f} := i \cup \vec{f}$ ;
6      $p_{max} := \max(p_{max}, p)$ ;
7      $l_{max} := \max(l_{max}, l)$ ;
8      $r_{max} := \max(r_{max}, r)$ ;
9   else
10    Assume  $A$  is the satisfying assignment of  $F_{LN}(p, l, r)$  in Line 6 of Algorithm 1;
11    foreach  $i \in \vec{i}$  do
12      if  $A(i_{p+l}) \neq A(i'_{p+l})$  then  $\vec{i} := \vec{i} - \{i\}$ 
13 return  $(\vec{f}, p_{max}, l_{max}, r_{max})$ 

```

When  $F_{LN}(p, l, r)$  is satisfiable, then  $i_{p+l}$  can not be uniquely determined by  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$ . More importantly, by unrolling these three loops, we can further prove that:

**PROPOSITION 2.2.** *If  $F_{LN}(p, l, r)$  is satisfiable, then  $i_{p'+l'}$  can not be uniquely determined by  $\langle \vec{o}_{p'}, \dots, \vec{o}_{p'+l'+r'} \rangle$  for any larger  $p' \geq p$ ,  $l' \geq l$  and  $r' \geq r$ .*

Thus, with Proposition 2.1 and 2.2, Algorithm 1 is a halting algorithm that determines if there exists  $p, l$  and  $r$  that make an input bit  $i_{p+l}$  to be uniquely determined by the encoder's output sequence  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$ . On one hand, if there actually exists such  $p, l$  and  $r$ , then eventually  $F_{PC}(p, l, r)$  will become unsatisfiable in Line 4; On the other hand, if there does not exist such  $p, l$  and  $r$ , then eventually  $p, l$  and  $r$  will be larger than the longest path without loop, which means that there will be three loops in  $\langle \vec{s}_0, \dots, \vec{s}_p \rangle, \langle \vec{s}_{p+1}, \dots, \vec{s}_{p+l} \rangle$  and  $\langle \vec{s}_{p+l+1}, \dots, \vec{s}_{p+l+r} \rangle$ . This will make  $F_{LN}(p, l, r)$  satisfiable in Line 5. Both cases will lead to this Algorithm's termination. Please refer to [Shen et al. 2011] for more detail.

### 3. FINDING OUT FLOW CONTROL SIGNALS

To facilitate the presentation of our algorithm, we partition the input vector  $\vec{i}$  into two vectors: the vector of flow control signals  $\vec{f}$  and the vector of input data signals  $\vec{d}$ .

The flow control signals  $\vec{f}$  are intended for representing the validness of  $\vec{d}$ . So for a properly designed encoder,  $\vec{f}$  should always be uniquely determined by a bounded sequence of the encoder's output  $\vec{o}$ , or else the decoder can not recognize those encoded invalid data.

Thus, we propose Algorithm 2 to find out  $\vec{f}$ .

At Line 3, it simply apply Algorithm 1 to each input bit  $i \in \vec{i}$  of the encoder, to find out whether  $i_{p+l}$  can be uniquely determined by  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$ . If yes, the value of  $p, l$  and  $r$  are also computed.

At Line 5, the input bit  $i$  that can be uniquely determined will be added to the vector  $\vec{f}$ . And at the following three lines, the maximal value of  $p, l$  and  $r$  are computed.

On the other hand, when  $\vec{i}$  is very long, the run time overhead of testing each  $i \in \vec{i}$  one by one would also be very large. To speedup this testing procedure, when the result of *CheckUniqueness* is 0 at Line 3, every  $i \in \vec{i}$  that have different values for  $i_{p+l}$  and  $i'_{p+l}$  in solving  $F_{LN}(p, l, r)$  can also be ruled out at Line 12, because they also satisfy their own  $F_{LN}(p, l, r)$ .

#### 4. INFERRING PREDICATE THAT MAKES THE ENCODER'S INPUT DATA TO BE UNIQUELY DETERMINED

We propose in subsection 4.1 an algorithm to characterize a formula over a particular set of variables that makes a SAT instance satisfiable. And then in subsection 4.2, we apply this algorithm to infer  $valid(\vec{f})$ .

##### 4.1. ALLSAT algorithm based on Craig Interpolant and Cofactoring

Assume  $R(\vec{a}, \vec{b}, t)$  is a Boolean relation with  $R(\vec{a}, \vec{b}, 0) \wedge R(\vec{a}, \vec{b}, 1)$  unsatisfiable.  $\vec{a}$  and  $\vec{b}$  are respectively called the important and the non-important variable vector, while  $t$  is the target variable.

We need to characterize a formula  $FSAT(\vec{a})$  that covers and only covers all the valuations of  $\vec{a}$  that can make  $R(\vec{a}, \vec{b}, 1)$  satisfiable. It is formally defined below:

(3)

Thus, an naive method of computing  $FSAT(\vec{a})$  is to enumerate all valuations of  $\vec{a}$ , and collect all those valuations that make  $R(\vec{a}, \vec{b}, 1)$  satisfiable. But this naive method would run in exponential time with respect to the length of  $\vec{a}$ . For a long  $\vec{a}$ , it may not be possible to terminate in reasonable time.

We can speedup this naive algorithm by expanding each valuation of  $\vec{a}$  to a larger set with Craig interpolant [McMillan 2003]. Intuitively, assume  $R(\vec{a}, \vec{b}, 1)$  is satisfiable with an satisfying assignment  $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0, 1\}$ . Then we can construct the following new formula by cofactoring.

(4)

Because  $R(\vec{a}, A(\vec{b}), 0) \wedge R(\vec{a}, A(\vec{b}), 1)$  is unsatisfiable, so an Craig interpolant  $ITP(\vec{a})$  of  $R(\vec{a}, A(\vec{b}), 1)$  with respect to  $R(\vec{a}, A(\vec{b}), 0)$  can be computed and used as an over-approximation of  $R(\vec{a}, A(\vec{b}), 1)$ . At the same time,  $ITP(\vec{a}) \wedge R(\vec{a}, A(\vec{b}), 0)$  is unsatisfiable, so  $ITP(\vec{a})$  covers nothing that can make  $R(\vec{a}, A(\vec{b}), 0)$  satisfiable. Thus,  $ITP(\vec{a})$  covers exactly the set of valuations of  $\vec{a}$  that can make  $R(\vec{a}, A(\vec{b}), 1)$  satisfiable.

With all discussion above, Algorithm 3 is proposed to characterize  $FSAT(\vec{a})$ . Line 1 initializes  $FSAT$  to an empty formula. Line 2 checks whether there is still some new valuation of  $\vec{a}$  that can make  $R(\vec{a}, \vec{b}, 1)$  satisfiable, but has not been covered by  $FSAT$ . Line 4 and 5 assign the value of  $\vec{b}$  from the satisfying assignment into  $R(\vec{a}, \vec{b}, 1)$  and  $R(\vec{a}, \vec{b}, 0)$  respectively, to remove  $\vec{b}$  from them.

Thus,  $\phi_A \wedge \phi_B$  in Line 6 is unsatisfiable, and the common variables vector of  $\phi_A$  and  $\phi_B$  is  $\vec{a}$ , the important variables over which the formula to be characterized. So a Craig interpolant  $ITP(\vec{a})$  can be generated with the Mcmillian's algorithm [McMillan 2003].

$ITP(\vec{a})$  is added to  $FSAT$  in Line 7 and ruled out in Line 2.

Each iteration of the while loop in Algorithm 3 adds at least a valuation of  $\vec{a}$  to  $FSAT$ , which means that  $FSAT$  is a formula that covers a bounded and strictly increasing set of valuations of  $\vec{a}$ . So Algorithm 3 is a halting one.

---

**ALGORITHM 3:** *CharacterizingFormulaSAT*( $R, \vec{a}, \vec{b}, t$ ): Characterizing a formula over  $\vec{a}$  that can make  $R(\vec{a}, \vec{b}, 1)$  satisfiable

---

**Input:** The Boolean relation  $R(\vec{a}, \vec{b}, t)$ , its important variable vector  $\vec{a}$ , its non-important variable vector  $\vec{b}$ , and its target variable  $t$ .

**Output:**  $FSAT(\vec{a})$  that makes  $R(\vec{a}, \vec{b}, 1)$  satisfiable.

```

1  $FSAT := \perp$  ;
2 while  $R(\vec{a}, \vec{b}, t) \wedge \neg FSAT$  is satisfiable do
3   assume  $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0, 1\}$  is the satisfying assignment ;
4    $\phi_A(\vec{a}) := R(\vec{a}, A(\vec{b}), 1)$  ;
5    $\phi_B(\vec{a}) := R(\vec{a}, A(\vec{b}), 0)$  ;
6   assume  $ITP(\vec{a})$  is the Craig interpolant of  $\phi_A$  with respect to  $\phi_B$  ;
7    $FSAT := ITP \vee FSAT$  ;
8 return ( $FSAT$ )
```

---

#### 4.2. Inferring $valid(\vec{f})$ that makes $\vec{d}$ to be uniquely determined

By replacing  $i$  in Equation (1) with  $\vec{d}$ , we have:

(5)

If  $F_{PC}^d(p, l, r)$  is satisfiable, then  $\vec{d}$  can not be uniquely determined by  $< \vec{o}_p, \dots, \vec{o}_{p+l+r} >$ . Let's define a new formula  $T_{PC}(p, l, r)$  by collecting the 3rd line of Equation (5):

(6)

Then  $\vec{d}$  can not be uniquely determined for a particular  $p, l$  and  $r$  if  $F'_{PC}(p, l, r, 1)$  is satisfiable, where  $F'_{PC}(p, l, r, t)$  is defined below:

(7)

Let's define:

(8)

(9)

$\vec{a} \cup \vec{b}$  is the vector that contains all the input vector  $< \vec{i}_0, \dots, \vec{i}_{p+l+r} >$  and  $< \vec{i}'_0, \dots, \vec{i}'_{p+l+r} >$  at all cycles for the two unrolled transition functions sequences. It also contains the two initial states  $\vec{s}_0$  and  $\vec{s}'_0$ . And  $T$  is a function that computes the next state and the output vector from the current state and input vector. So  $\vec{a}$  and  $\vec{b}$  can uniquely determine the value of  $t$  in  $F'_{PC}(p, l, r, t)$ . Thus, for a particular combination of  $p, l$  and  $r$ , the formula over  $\vec{f}_{p+l}$  that makes  $F'_{PC}(p, l, r, 1)$  satisfiable can be computed by calling Algorithm 3 with  $F'_{PC}(p, l, r, t)$ ,  $\vec{a}$  and  $\vec{b}$  defined above:

(10)

Thus, we have the following proposition:

PROPOSITION 4.1.

---

**ALGORITHM 4:** *InferringUniqueFormula*: inferring the predicate  $valid(\vec{f}_{p+l})$  that makes  $\vec{d}_{p+l}$  to be uniquely determined

---

```

1  $p := p_{max}; l := l_{max}; r := r_{max};$ 
2 while  $\neg FSAT_{LN}(p, l, r) \wedge FSAT_{PC}(p, l, r)$  is satisfiable do
3    $p ++; l ++; r ++;$ 
4 end
5 return  $\neg FSAT_{LN}(p, l, r)$ 

```

---

Similarly, By replacing  $i$  in Equation (2) with  $\vec{d}$ , we have:

(11)

If  $F'_{LN}(p, l, r)$  is satisfiable, then  $\vec{d}$  can not be uniquely determined by  $< \vec{o}_p, \dots, \vec{o}_{p+l+r} >$ . Furthermore, by unrolling those three loops, we can prove that  $\vec{d}$  can not be uniquely determined for any larger  $p' \geq p, l' \geq l$  and  $r' \geq r$ . Let's define a new formula  $T_{PC}(p, l, r)$  by collecting the 3rd line and last three lines of Equation (11):

(12)

Then  $\vec{d}$  can not be uniquely determined for any larger  $p' \geq p, l' \geq l$  and  $r' \geq r$  if  $F'_{LN}(p, l, r, 1)$  is satisfiable, where  $F'_{LN}(p, l, r, t)$  is defined below:

(13)

Thus, for a particular combination of  $p, l$  and  $r$ , the formula over  $\vec{f}_{p+l}$  that makes  $F'_{LN}(p, l, r, 1)$  satisfiable can be computed by

(14)

Thus we have the following proposition:

**PROPOSITION 4.2.**  $FSAT_{LN}(p, l, r)$  is the formula over  $\vec{f}_{p+l}$  that make  $\vec{d}$  to be not uniquely determined for every  $p' \geq p, l' \geq l$  and  $r' \geq r$ .

With Proposition 4.1 and 4.2, the algorithm that infers the predicate  $valid(\vec{f}_{p+l})$  is shown in Algorithm 4. The proofs of its termination and correctness are given in the next subsection.

#### 4.3. Proofs of termination and correctness

First we need to prove the following three lemmas:

**LEMMA 4.3.**  $FSAT_{PC}(p, l, r)$  in Algorithm 4 monotonically decreases.

**PROOF.** According to the definition of  $F'_{PC}(p, l, r, t)$  in Equation (7), for any  $p' > p, l' > l$  and  $r' > r$ , we have  $F'_{PC}(p', l', r', 1) \Rightarrow F'_{PC}(p, l, r, 1)$ . so  $FSAT_{PC}(p', l', r') \Rightarrow FSAT_{PC}(p, l, r)$ . Thus,  $FSAT_{PC}(p, l, r)$  monotonically decreases.  $\square$

**LEMMA 4.4.**  $FSAT_{LN}(p, l, r)$  in Algorithm 4 monotonically increases.

**PROOF.** According to the definition of  $F'_{LN}(p, l, r, t)$  in Equation (13), with any satisfying assignment of  $F'_{LN}(p, l, r, 1)$ , those three loops in  $T_{LN}(p, l, r)$  can be unrolled to make  $F'_{LN}(p', l', r', 1)$  with larger  $p', l'$  and  $r'$  satisfiable. So we have  $FSAT_{LN}(p, l, r) \Rightarrow FSAT_{LN}(p', l', r')$ , that is,  $FSAT_{LN}(p, l, r)$  monotonically increases.  $\square$



Fig. 3. The monotonicity of  $\neg FSAT_{LN}(p, l, r) \wedge FSAT_{PC}(p, l, r)$ 

LEMMA 4.5.  $FSAT_{LN}(p, l, r) \Rightarrow FSAT_{PC}(p, l, r)$

PROOF. It is obvious that  $F'_{LN}(p, l, r, 1) \Rightarrow F'_{PC}(p, l, r, 1)$ , so  $FSAT_{LN}(p, l, r) \Rightarrow FSAT_{PC}(p, l, r)$  holds.  $\square$

These three lemmas are depicted intuitively in Figure 3, which make it obvious that  $\neg FSAT_{LN}(p, l, r) \wedge FSAT_{PC}(p, l, r)$  monotonically decreases in Algorithm 4. With these lemmas, let's first prove that Algorithm 4 is a halting one.

THEOREM 4.6. *Algorithm 4 is a halting algorithm.*

PROOF. Since the encoder is represented by a finite state machine, the length of the longest path without loop is finite. If Algorithm 4 does not halt, then eventually the value of  $p$ ,  $l$  and  $r$  in Algorithm 4 will be larger than the length of the longest path without loop, which means there will be loops in these three state sequences  $\langle \vec{s}_0, \dots, \vec{s}_p \rangle$ ,  $\langle \vec{s}_{p+1}, \dots, \vec{s}_{p+l} \rangle$  and  $\langle \vec{s}_{p+l+1}, \dots, \vec{s}_{p+l+r} \rangle$ . Thus, every satisfying assignment of  $F'_{PC}(p, l, r, 1)$  also satisfies  $F'_{LN}(p, l, r, 1)$ , which means  $\neg FSAT_{LN}(p, l, r) \wedge FSAT_{PC}(p, l, r)$  is unsatisfiable. This will lead to the termination of Algorithm 4. So it is a halting algorithm.  $\square$

We will then prove the correctness of Algorithm 4.

THEOREM 4.7.  $\neg FSAT_{LN}(p, l, r)$  returned by Algorithm 4 covers and only covers all valuations of  $\vec{f}$  that can make  $\vec{d}$  to be uniquely determined by a bounded sequence of  $\vec{o}$ .

PROOF. Let's first prove the covering case.  $FSAT_{LN}(p, l, r)$  covers a set of valuations of  $\vec{f}$  that make  $\vec{d}$  to be not uniquely determined for some particular  $p$ ,  $l$  and  $r$ . So  $\neg FSAT_{LN}(p, l, r)$  rules them out and covers all valuations of  $\vec{f}$  that can make  $\vec{d}$  to be uniquely determined.

We then prove the only covering case. If  $\neg FSAT_{LN}(p, l, r)$  covers a valuation of  $\vec{f}$  that make  $\vec{d}$  to be **NOT** uniquely determined for some particular  $p'$ ,  $l'$  and  $r'$ , then  $FSAT_{LN}(p', l', r')$  also covers this valuation but  $FSAT_{LN}(p, l, r)$  does not. But according to Lemma 4.3, 4.4 and 4.5, this is impossible, because  $FSAT_{LN}(p, l, r)$  is the maximal  $FSAT_{LN}(p', l', r')$  for all possible  $p'$ ,  $l'$  and  $r'$ . So  $\neg FSAT_{LN}(p, l, r)$  covers no valuation of  $\vec{f}$  that make  $\vec{d}$  to be **NOT** uniquely determined. This proves the only covering case.  $\square$

## 5. CHARACTERIZING THE DECODER'S BOOLEAN FUNCTION

In Section 3, the encoder's input  $\vec{i}$  has been partitioned into two vectors: the flow control vector  $\vec{f}$  and the data vector  $\vec{d}$ . The algorithms to characterize their Boolean functions are different, so they are discussed in the following two subsections respectively.

### 5.1. Characterizing the decoder's Boolean function of $\vec{f}$

Each bit  $f \in \vec{f}$  can be uniquely determined by the encoder's output, so it is straightforward to simply define two formulas  $\phi_A$  and  $\phi_B$  as:

$$(15)$$

$$(16)$$

It is obvious that  $\phi_A \wedge \phi_B$  equals to  $F_{PC}(p, l, r)$  in Equation (1), so it is unsatisfiable. The common variable set of  $\phi_A$  and  $\phi_B$  is  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$ . So a Craig interpolant

$ITP$  can be derived by Mcmillian's algorithm[McMillan 2003] from the unsatisfiability proof of  $\phi_A \wedge \phi_B$ , which covers all values of  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$  that make  $f_{p+l} \equiv 1$ . At the same time,  $ITP \wedge \phi_B$  is unsatisfiable, so  $ITP$  covers nothing that can make  $f_{p+l}$  to be 0. Thus,  $ITP$  is the decoder's Boolean function that computes  $f \in \vec{f}$ .

## 5.2. Characterizing the decoder's Boolean function of $\vec{d}$

Assume the predicate over  $\vec{f}_{p+l}$  that inferred by Algorithm 4 is  $valid(\vec{f}_{p+l})$ . Let's define the following two formulas for each data bit  $d \in \vec{d}$ :

$$(17)$$

$$(18)$$

Each bit  $d \in \vec{d}$  can be uniquely determined by the encoder's output only when  $valid(\vec{f}_{p+l})$  holds. So  $\phi'_A \wedge \phi'_B$  is unsatisfiable. Thus, a Craig interpolant  $ITP$  can be derived by Mcmillian's algorithm[McMillan 2003] from the unsatisfiability proof of  $\phi'_A \wedge \phi'_B$ , which covers and only covers all valuations of  $\langle \vec{o}_p, \dots, \vec{o}_{p+l+r} \rangle$  that make  $d_{p+l} \equiv 1$ . Thus,  $ITP$  is the decoder's Boolean function that computes  $d \in \vec{d}$ .

Furthermore, when  $valid(\vec{f}_{p+l})$  does not hold, the data bit  $d \in \vec{d}$  can not be uniquely determined, so no function can be used to calculate its value. But this is not a problem because the decoder is supposed to recognize these invalid data by computing the value of control flow vector  $\vec{f}$ , instead of recovering the exact value of  $\vec{d}$ .

## 6. EXPERIMENTAL RESULTS

We have implemented these algorithms and solved the generated SAT instances with Minisat[Eén and Sörensson 2003]. All experiments have been run on a PC with a 2.4GHz Intel Core 2 Q6600 processor, 8 GB memory, and Ubuntu Linux 12.04.

By studying the benchmarks used in our previous papers [Shen et al. 2009;2010; 2011; 2012], we found that most of them have built-in flow control mechanism. But these mechanism are turned off in previous papers by enforcing manually specified assertion[Shen et al. 2009;2010; 2011], or automatically inferred assertions[Shen et al. 2012]. Thus, this paper's algorithms is the first one that allows these flow control mechanisms to be turned on and off freely. We will discuss these benchmarks one by one in the following subsections.

Furthermore, we also study the benchmarks used in [Tu and Jiang 2013], and find that they don't contain the flow control mechanism, so they will not be discussed here.

### 6.1. PCI Express 2.0 encoder

This encoder is compliant to the PCI Express 2.0 standard [PCI-SIG 2009]. After deleting empty line and comments, its source code have 259 lines of verilog. After mapped to LSI10K library, it contains 113 AND2 gates, 212 OR2 gates, 68 inverters and 23 registers, its total area is 879.

The list of input and output signals are shown in Table I. According to the 8b/10b encoding scheme's coding table[wikipedia 2013a], when  $TXDATAK \equiv 0$ ,  $TXDATA$  can be any value. But when  $TXDATAK \equiv 1$ ,  $TXDATA$  can only be 1C, 3C, 5C, 7C, 9C, BC, DC, FC, F7, FB, FD and FE. So we write an assertion to rule out those combinations not in this coding table, and embeds it into the the transition function  $T$ , such that it can be enforced at every cycle in the unrolled state sequences.

Algorithm 2 costs 0.924754 seconds to find out the flow control signal  $CNTLTXEnable_P0$ . And then Algorithm 4 costs 2.067509 seconds to infer the pred-

Table I. The input and output signals of the PCI Express 2.0 encoder

	signal name	width	description
Inputs	<i>TXDATA</i>	8	The data to be encoded
	<i>TXDATAK</i>	1	1 means <i>TXDATA</i> is a controlling character, 0 means <i>TXDATA</i> is normal data
	<i>CNTLTXEnable_P0</i>	1	Indicating the validness of <i>TXDATA</i> and <i>TXDATAK</i>
Outputs	<i>HSS_TXD</i>	10	The encoded data
	<i>HSS_TXELEC_IDLE</i>	1	The electrical idle state

Table II. The input and output signals of the 10G Ethernet encoder

	signal name	width	description
Inputs	<i>encode_data_in</i>	8	The data to be encoded
	<i>konstant</i>	1	1 means <i>encode_data_in</i> is a special character, 0 means <i>encode_data_in</i> is normal data
	<i>bad_code</i>	1	Indicating the validness of <i>konstant</i> and <i>encode_data_in</i>
Outputs	<i>encode_data_out</i>	10	The encoded data

icate  $CNTL\_TXEnable\_P0 \equiv 1$  that makes all other data signals to be uniquely determined. Finally, with the inferred predicate, generating the decoder's Boolean functions for *CNTLTXEnable\_P0*, *TXDATA* and *TXDATAK* costs 3.121821 seconds. After mapped to LSI10K library, the decoder contains 614 AND2, 198 OR2 and 22 registers, its total area is 1778.

The major breakthrough of this paper is the ability to handle invalid data, so it should be very interesting to show how the invalid data are mapped to output vector  $\delta$ . By studying the source code of this encoder, we find that when and only when  $CNTL\_TXEnable\_P0 \equiv 0$  holds, that is, *TXDATA* and *TXDATAK* are invalid, the output electrical idle signal *HSS\_TXELEC\_IDLE* become 1. So the decoder can determined the value of input signal *CNTLTXEnable\_P0* with output signal *HSS\_TXELEC\_IDLE*.

## 6.2. 10G Ethernet encoder

This encoder is compliant to the clause 48 of IEEE 802.3 standard [IEEE 2012a]. After deleting empty line and comments, this encoder have 214 lines of verilog. After mapped to LSI10K library, it contains 65 AND2 gates, 192 OR2 gates, 75 inverters and 17 registers, its total area is 708.

The list of input and output signals are shown in Table II. This encoder also employs an 8b/10b encoding scheme[wikipedia 2013a] with two inputs: the 8-bit *encode\_data\_in* to be encoded, and 1-bit *konstant* indicating a controlling character. According to the coding table in [wikipedia 2013a], when  $konstant \equiv 0$ , *encode\_data\_in* can be any value. But when  $konstant \equiv 1$ , *encode\_data\_in* can only be 1C, 3C, 5C, 7C, 9C, BC, DC, FC, F7, FB, FD and FE. So we write an assertion to exclude those combinations not in this table, and embeds it into the the transition function  $T$ .

Algorithm 2 costs 0.619508 seconds to find out the flow control signal *bad\_code*. And then Algorithm 4 costs 1.443065 seconds to infer the predicate  $bad\_code \equiv 0$  that makes all other data signals to be uniquely determined. Finally, generating the decoder's Boolean functions for *bad\_code*, *encode\_data\_in* and *konstant* costs 2.202401 seconds. After mapped to LSI10K library, the decoder contains 597 AND2, 174 OR2 and 30 registers, its total area is 1752.

Although this encoder uses the same coding mechanism as PCI Express 2.0 encoder mentioned above, its handling of the invalid data is different. This encoder does not have a separate output signal to indicated the validness of the output data, instead, the validness is encoded in *encode\_data\_out*. By study this encoder's source code, we find that when and only when  $bad\_code \equiv 1$ , that is, *encode\_data\_in* and *konstant* are invalid, the output signal *encode\_data\_out* will become 0010111101. So the decoder can use the

Table III. The input and output signals of the UltraSPARC T2 Ethernet encoder

	signal name	width	description
Inputs	<i>txd</i>	8	The data to be encoded
	<i>tx_enc_ctrl_sel</i>	1	Refer to Table IV
	<i>tx_en</i>	1	Transmission enable
	<i>tx_er</i>	1	Transmitting an error character
Outputs	<i>tx_10bdata</i>	10	The encoded data
	<i>txd_eq_crs_ext</i>	10	Transmitting an special error character with <i>tx_er</i> $\equiv$ 1 and <i>txd</i> $\equiv$ 8'h0F
	<i>tx_er_d</i>	1	Transmitting an error character
	<i>tx_en_d</i>	1	Transmission enable
	<i>pos_disp_tx_p</i>	1	Indicating positive parity

output signal *encode\_data\_out* to uniquely determine the value of the flow control signal *bad\_code*.

### 6.3. UltraSPARC T2 Ethernet encoder

This encoder comes from the UltraSPARC T2 opensource processor designed by Sun Microsystems. It is compliant to the clause 36 of IEEE 802.3 standard [IEEE 2012a]. After deleting empty line and comments, this encoder's source code have 864 lines of verilog. After mapped to LSI10K library, it contains 344 AND2 gates, 649 OR2 gates, 128 inverters and 53 registers, its total area is 2485.

The list of input and output signals are shown in Table III. This encoder also employs an 8b/10b encoding scheme[wikipedia 2013a], but with yet another style of flow control mechanism that is significantly different compared to the above two encoders. The data to be encoded is the 8-bit *txd*, but there is no standalone signal to indicate a control symbol. But only a 4-bit *tx\_enc\_ctrl\_sel* used to define the action to be performed, as shown in Table IV. It is obvious that the functionality of the control symbol indication and flow control mechanism are combined in *tx\_enc\_ctrl\_sel*. The last four case in Table IV can never be uniquely determined, because they can not be distinguished from the case of 'PCS\_ENC\_DATA'. So we write an assertion to rule them out.

Algorithm 2 costs 11.750317 seconds to find out the flow control signals *tx\_enc\_ctrl\_sel*, *tx\_en* and *tx\_er*. And then Algorithm 4 costs 27.456717 seconds to infer the predicate *tx\_enc\_ctrl\_sel*  $\equiv$  'PCS\_ENC\_DATA' that makes all other data signals to be uniquely determined. Finally, generating the decoder's Boolean functions for *txd*, *tx\_enc\_ctrl\_sel*, *tx\_en* and *tx\_er* costs 22.156704 seconds. After mapped to LSI10K library, the decoder contains 2245 AND2, 794 OR2 and 22 registers, its total area is 6232.

As shown in the last column of Table IV, the first 5 cases have their particular control symbol values that are assigned to *tx\_10bdata*, so the decoder can recover the value of the flow control signal *tx\_enc\_ctrl\_sel* from the output signal *tx\_10bdata*.

Table IV. Actions to be performed in UltraSPARC T2 Ethernet encoder

'PCS_ENC_K285	sending K28.5 control symbol
'PCS_ENC_SOP	sending K27.7 control symbol
'PCS_ENC_T.CHAR	sending K29.7 control symbol
'PCS_ENC_R.CHAR	sending K23.7 control symbol
'PCS_ENC_H.CHAR	sending K30.7 control symbol
'PCS_ENC_DATA	sending the encoded txd
'PCS_ENC_IDLE2	sending D16.2 data symbol following K28.5
'PCS_ENC_IDLE1	sending D5.6 data symbol
'PCS_ENC_LINK_CONFA	sending D21.5 data symbol following K28.5
'PCS_ENC_LINK_CONFB	sending D2.2 data symbol following K28.5

## 7. RELATED PUBLICATIONS

### 7.1. Complementary synthesis

The first complementary synthesis algorithm was proposed by [Shen et al. 2009]. It checks the decoder's existence by iteratively increasing the bound of unrolled transition function sequence, and generate the decoder's Boolean function by enumerating all satisfying assignments of the decoder's output. Its major shortcomings are that it may not halt and have large runtime overhead in building the decoder.

Shen et al.[2011] and Liu et al.[2011] tackled the halting problem independently by searching for loops in the state sequence. While the runtime overhead problem was addressed in [Shen et al. 2012; Liu et al. 2011] by Craig interpolant[McMillan 2003].

Shen et al.[2012] automatically inferred an assertion for configuration pins, which can lead to the decoder's existence. It can be seen as a special case of Algorithm 4 in Section 4, with a restriction that the inferred assertion must hold on all cycles. While our Algorithm 4 allows states with and without the inferred assertion to be interleaved freely with each other.

Tu and Jiang [2013] proposed a break-through algorithm based on property directed reachability analysis[Bradley 2011; Eén et al. 2011] that can take the encoder's initial state into consideration, such that the infinite history of the encoder and the decoder can be used to generate the decoder's output. This algorithm can handle some special encoders that can not be handle by state-of-the-art algorithms. But for the encoders with flow control mechanism used in our experiments, our algorithm is enough, so we have not implemented their algorithm in our framework.

### 7.2. Program inversion

According to Gulwani[2010], 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[Dijkstra 1979], but it could only handle very small programs and very simple syntax structures.

Glück et al. [2005] inverted 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. [2010; 2011] 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.

### 7.3. Protocol converter synthesis

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. [2008; 2009] first defined a general model for describing the different protocols. Then they provided an algorithm to decide whether there is 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.[2009] improved the algorithm mentioned above with a more efficient design space exploration algorithm.

## 8. CONCLUSIONS

In this paper, we propose the first framework to handle the control flow mechanism in complementary synthesis problem. The experimental result indicates that our framework can successfully handle many complex encoders from real industrial projects, such as PCIE [wikipedia 2013d] and Ethernet [wikipedia 2013b].

## ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

## ACKNOWLEDGMENTS

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

## REFERENCES

- Karin Avnit, Vijay D'Silva, Arcot Sowmya, S. Ramesh, and Sri Parameswaran. 2009. Provably correct on-chip communication: A formal approach to automatic protocol converter synthesis. *ACM Transactions on Design Automation of Electronic Systems* 14, 2 (March 2009), 14:1–14:41. DOI: <http://dx.doi.org/10.1145/1497561.1497562>
- Karin Avnit and Arcot Sowmya. 2009. A formal approach to design space exploration of protocol converters. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2009 (DATE '09)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 129–134.
- Karin Avnit, Vijay D'Silva and Arcot Sowmya, and S. Ramesh and Sri Parameswaran. 2008. A Formal Approach To The Protocol Converter Problem. In *Proceedings of the conference on Design, automation and test in Europe, DATE 2008 (DATE '08)*. ACM Press, Munich, Germany, 294–299. DOI: <http://dx.doi.org/10.1109/DATE.2008.4484695>
- Aaron R. Bradley. 2011. SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation, 12th International Conference, VMCAI 2011*, David A. Schmidt Ranjit Jhala (Ed.). Lecture Notes in Computer Science, Vol. 6538. Springer-Verlag, Berlin Heidelberg, 70–87. DOI: [http://dx.doi.org/10.1007/978-3-642-18275-4\\_7](http://dx.doi.org/10.1007/978-3-642-18275-4_7)
- William Craig. 1957. Linear reasoning: A new form of the herbrand-gentzen theorem. *The Journal of Symbolic Logic* 22, 3 (Sept. 1957), 250–268.
- Edsger W. Dijkstra. 1979. Program Inversion. In *Proceeding of Program Construction, International Summer School*. Springer-Verlag, London, UK, 54–57.
- Niklas Eén, Alan Mishchenko, and Robert K. Brayton. 2011. Efficient implementation of property-directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design, FMCAD 2011 (FMCAD '11)*. FMCAD Inc, Austin, TX, USA, 125–134.
- Niklas Eén and Niklas Sörensson. 2003. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003*, Armando Tacchella Enrico Giunchiglia (Ed.). Lecture Notes in Computer Science, Vol. 2919. Springer-Verlag, Berlin Heidelberg, 502–518. DOI: [http://dx.doi.org/10.1007/978-3-540-24605-3\\_37](http://dx.doi.org/10.1007/978-3-540-24605-3_37)
- Malay K. Ganai, Aarti Gupta, and Pranav Ashar. 2004. Efficient sat-based unbounded symbolic model checking using circuit cofactoring. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design, ICCAD 2004 (ICCAD '04)*. IEEE Computer Society, San Jose, CA, USA, 510–517. DOI: <http://dx.doi.org/10.1109/ICCAD.2004.1382631>
- Robert Glück and Masahiko Kawabe. 2005. A method for automatic program inversion based on LR(0) parsing. *Journal Fundamenta Informaticae* 66, 4 (January 2005), 367–395. DOI: <http://dx.doi.org/10.1109/TCAD.2012.2191288>
- Sumit Gulwani. 2010. Dimensions in program synthesis. In *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming, PPDP 2010 (PPDP '10)*. ACM Press, Hagenberg, Austria, 13–24. DOI: <http://dx.doi.org/10.1145/1836089.1836091>
- IEEE. 2012a. IEEE Standard for Ethernet SECTION FOURTH. (2012). Retrieved January 25, 2013 from [http://standards.ieee.org/getieee802/download/802.3-2012\\_section4.pdf](http://standards.ieee.org/getieee802/download/802.3-2012_section4.pdf)
- IEEE. 2012b. IEEE Standard for Ethernet SECTION THREE. (2012). Retrieved January 25, 2013 from [http://standards.ieee.org/getieee802/download/802.3-2012\\_section3.pdf](http://standards.ieee.org/getieee802/download/802.3-2012_section3.pdf)
- Hsiou-Yuan Liu, Yen-Cheng Chou, Chen-Hsuan Lin, and Jie-Hong R. Jiang. 2011. Towards completely automatic decoder synthesis. In *Proceedings of the 2011 International Conference on*

- Computer-Aided Design, ICCAD 2011 (ICCAD '11)*. IEEE Press, San Jose, CA, USA, 389–395. DOI: <http://dx.doi.org/10.1109/ICCAD.2011.6105359>
- Kenneth L. McMillan. 2003. Interpolation and sat-based model checking. In *Computer Aided Verification, 15th International Conference, CAV 2003*, Fabio Somenzi Warren A. Hunt Jr. (Ed.). Lecture Notes in Computer Science, Vol. 2725. Springer-Verlag, Berlin Heidelberg, 1–13. DOI: [http://dx.doi.org/10.1007/978-3-540-45069-6\\_1](http://dx.doi.org/10.1007/978-3-540-45069-6_1)
- PCI-SIG. 2009. PCI Express Base 2.1 Specification. (2009). Retrieved January 25, 2013 from [http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI\\_Express\\_Base\\_r2.1.04Mar09.pdf](http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_r2.1.04Mar09.pdf)
- ShengYu Shen, Ying Qin, KeFei Wang, Zhengbin Pang, Jianmin Zhang, and Sikun Li. 2012. Inferring Assertion for Complementary Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 8 (August 2012), 31:1288–31:1292. DOI: <http://dx.doi.org/10.1109/TCAD.2012.2190735>
- ShengYu Shen, Ying Qin, KeFei Wang, LiQuan Xiao, Jianmin Zhang, and Sikun Li. 2010. Synthesizing Complementary Circuits Automatically. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 8 (August 2010), 29:1191–29:1202. DOI: <http://dx.doi.org/10.1109/TCAD.2010.2049152>
- ShengYu Shen, Ying Qin, LiQuan Xiao, KeFei Wang, Jianmin Zhang, and Sikun Li. 2011. A Halting Algorithm to Determine the Existence of the Decoder. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 10 (October 2011), 30:1556–30:1563. DOI: <http://dx.doi.org/10.1109/TCAD.2011.2159792>
- ShengYu Shen, Jianmin Zhang, Ying Qin, and Sikun Li. 2009. Synthesizing complementary circuits automatically. In *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD '09)*. IEEE Press, San Jose, CA, USA, 381–388. DOI: <http://dx.doi.org/10.1145/1687399.1687472>
- S. Srivastava, S. Gulwani, S. Chaudhuri, and J. Foster. 2010. *Program inversion revisited*. Technical Report MSR-TR-2010-34. Microsoft Research.
- Saurabh Srivastava, Sumit Gulwani, Swarat Chaudhuri, and Jeffrey S. Foster. 2011. Path-based inductive synthesis for program inversion. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation, PLDI 2011 (PLDI '11)*. ACM Press, San Jose, CA, USA, 492–503. DOI: <http://dx.doi.org/10.1145/1993498.1993557>
- Kuan-Hua Tu and Jie-Hong R. Jiang. 2013. Synthesis of feedback decoders for initialized encoders. In *Proceedings of the 50th Annual Design Automation Conference, DAC 2013 (DAC '13)*. ACM Press, Austin, TX, USA, 1–6. DOI: <http://dx.doi.org/10.1145/2463209.2488794>
- wikipedia. 2013a. 8b/10b encoding. (25 Jan. 2013). Retrieved January 25, 2013 from [http://en.wikipedia.org/wiki/8b/10b\\_encoding](http://en.wikipedia.org/wiki/8b/10b_encoding)
- wikipedia. 2013b. Ethernet. (25 Jan. 2013). Retrieved January 25, 2013 from <http://en.wikipedia.org/wiki/Ethernet>
- wikipedia. 2013c. Flow control. (25 Jan. 2013). Retrieved January 25, 2013 from [http://en.wikipedia.org/wiki/Flow\\_control\\_\(data\)](http://en.wikipedia.org/wiki/Flow_control_(data))
- wikipedia. 2013d. PCI Express. (25 Jan. 2013). Retrieved January 25, 2013 from [http://en.wikipedia.org/wiki/PCI\\_Express](http://en.wikipedia.org/wiki/PCI_Express)

Received February 2014; revised March 2014; accepted June 2014