Complementary Synthesis for Encoder with Flow Control Mechanism

YING QIN and QINGBO WU and HUADONG DAI and YAN JIA and SHENGYU SHEN,
School of Computer, National University of Defense Technology

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

Flow control mechanisms classify the encoder's input variables into two sets: the flow control variables that can always be uniquely determined, and the input data variables whose validness is indicated by a predicate over the flow control variables. When the input data variables are invalid, the decoder is supposed to recognize its invalidness by recovering the flow control variables and ignoring the input data variables.

Thus, a novel algorithm is proposed for the first time to handle such encoders with flow control mechanism. **First**, it identifies all variables that can be uniquely determined, and take them as flow control variables. **Second**, it infers a predicate over these flow control variables that enables all other input data variables to be uniquely determined. **Third**, it characterizes the decoder's Boolean function for each input variable, by first building a conjunction of two formulas that assign conflicting values to this input variable, and then generating a Craig interpolant from its unsatisfiability proof. In addition, for the input data variables, the inferred predicate must be enforced before generating Craig interpolant.

Experimental results on several complex encoders indicate that our algorithm can always correctly identify the flow control variables, 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

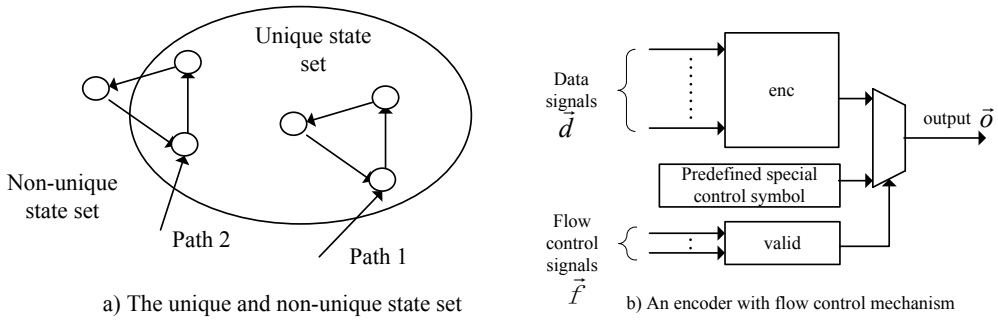
ACM Reference Format:
Ying Qin and QingBo Wu and HuaDong Dai and Yan Jia and ShengYu Shen, 2014. Complementary Synthesis for Encoder with Flow Control Mechanism. *ACM Trans. Des. Autom. Electron. Syst.* 9, 4, Article 39 (March 2010), 17 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

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 variables $\vec{i}$ to its output variables $\vec{o}$ according to some predefined rules, such as Ethernet [wikipedia 2013b] and PCI Express [wikipedia 2013d], while the decoder recovers $\vec{i}$

This work was funded by projects 61070132 and 61133007 supported by National Natural Science Foundation of China, the 863 Project of China under contract 2012AA01A301.
Author's addresses: Ying Qin, QingBo Wu, HuaDong Dai, Yan Jia and ShengYu Shen, School of Computer, National University of Defense Technology.

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



a) The unique and non-unique state set b) An encoder with flow control mechanism

Fig. 1. An encoder with flow control mechanism

from $\vec{o}$. Complementary synthesis [Shen et al. 2009] facilitates this job by automatically generating an encoders' decoder. Just like the path 1 shown in Figure 1a), this algorithm assumes that the encoder eventually reaches and never leaves the unique state set, in which $\vec{i}$ can always be uniquely determined by a bounded sequence of $\vec{o}$. The set of all other states is called the non-unique state set, because $\vec{i}$ cannot be uniquely determined in it by a bounded sequence of $\vec{o}$.

At the same time, high speed communication systems, such as Ethernet [wikipedia 2013b] and PCI Express [wikipedia 2013d], transmit several gigabits per second. It is impossible to distribute so fast a clock to both the transmitter and receiver. So they must be driven by two 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 variables $\vec{i}$, so that the decoder can recognize and discard them.

As shown in Figure 1b), an encoder with flow control mechanism partitions its input variables $\vec{i}$ into two sets: the input data variables $\vec{d}$ to be encoded, and the flow control variables $\vec{f}$ indicating the validness of $\vec{d}$ with a predicate $valid(\vec{f})$.

(1) When $valid(\vec{f}) \equiv 1$, the encoder maps $\vec{d}$ with an encoding function enc to the output variables $\vec{o} := enc(\vec{d})$, in which case both $\vec{d}$ and $\vec{f}$ can be uniquely determined by a bounded sequence of $\vec{o}$.
(2) However, when $valid(\vec{f}) \equiv 0$, the encoder drives some predefined special control symbol 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 with flow control mechanism may visit the non-unique state set infinitely often, that is, its input variables $\vec{i}$ cannot always be uniquely determined by a bounded sequence of $\vec{o}$. 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 $\vec{o}$, that is, they all assume that the encoder will eventually reach and never leave the unique state set. They ensure this with manually specified assertion [Shen et al. 2009;2010; 2011], or automatically inferred assertions [Shen et al. 2012] .

For the first time, we propose in this paper a novel three-step algorithm to handle such encoders with flow control mechanism. **First**, it applies the classical halting complementary synthesis algorithm [Shen et al. 2011] to identify all the flow control variables $\vec{f}$ that can be uniquely determined. **Second**, it infers a sufficient and necessary predicate $valid(\vec{f})$ that enables $\vec{d}$ to be uniquely determined by a bounded sequence of the encoder's output $\vec{o}$. **Finally**, it characterizes the decoder's Boolean function that computes each flow control variable $f \in \vec{f}$ by building two copies of unrolled transition function sequence with common output sequences $\vec{o}$, but different values on f. The conjunction of these two copies is 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 variables $\vec{d}$, their values are meaningful only when $valid(\vec{f}) \equiv 1$. Thus, the decoder's Boolean functions that compute each $d \in \vec{d}$ can be built similarly, but only after enforcing $valid(\vec{f}) \equiv 1$.

The second step of this algorithm seems somewhat similar to that of [Shen et al. 2012] in the sense that both algorithms infer predicates that enable $\vec{d}$ or $\vec{i}$ to be uniquely determined. But the essential difference between them is that the algorithm of [Shen et al. 2012] infers a **global** assertion that must be enforced at all cycles, so that the encoder eventually reaches and never leaves the unique state set, whereas our algorithm infers 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 PCI Express [wikipedia 2013d]), our algorithms can always correctly identify the flow control variables, infer the predicates and generate the decoders.

The remainder of this paper is organized as follows. Section 2 introduces the background material; Section 3 presents the algorithm that identifies the flow control variables, while Section 4 infers the predicate that enables $\vec{d}$ 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 sums up 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 $\vec{v} = (v, \ldots)$. The number of variables in $\vec{v}$ is denoted as $|\vec{v}|$. If a variable v is a member of $\vec{v}$, that is $\vec{v} = (\ldots, v, \ldots)$, 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 the 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 $[\![\vec{v}]\!]$, for instance, $[\![(v_1, v_2)]\!] = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$.

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 $A : V \to B$, 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 $f(v_1 \dots v \dots v_n)$ with respect to variable v are $f_v = f(v_1 \dots 1 \dots v_n)$ and $f_{\overline{v}} = f(v_1 \dots 0 \dots v_n)$, respectively. **Cofactoring** is the action that applies 1 or 0 to v to get f_v or $f_{\overline{v}}$.

Given two Boolean formulas ϕ_A and ϕ_B, with $\phi_A \wedge \phi_B$ unsatisfiable, there exists a formula ϕ_I referring only to the common variables of ϕ_A and ϕ_B such that $\phi_A \Rightarrow \phi_I$ and $\phi_I \wedge \phi_B$ is unsatisfiable. We call ϕ_I the **interpolant**[Craig 1957] of ϕ_A with respect to ϕ_B and use 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 $T : [\![\vec{s}]\!] \times [\![\vec{i}]\!] \rightarrow [\![\vec{s}]\!] \times [\![\vec{o}]\!]$ that computes the next state and output variable vector from the current state and input variable vector.

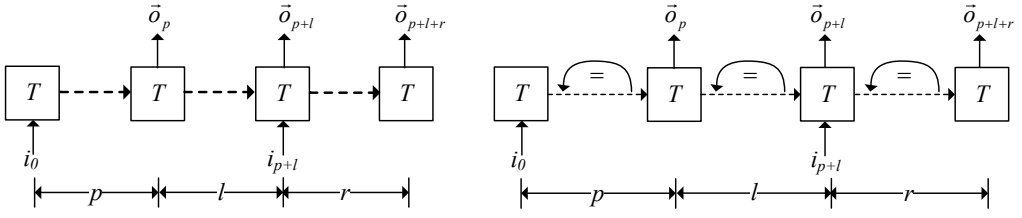
The state variable $s \in \vec{s}$, input variable $i \in \vec{i}$ and output variable $o \in \vec{o}$ at the n-th cycle are respectively denoted as s_n, i_n and o_n. Furthermore, the state, the input and the output variable vectors at the n-th cycle are respectively denoted as $\vec{s}_n$, $\vec{i}_n$ and $\vec{o}_n$. A **path** is a state sequence $< \vec{s}_n, \dots, \vec{s}_m >$ 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 $< \vec{s}_n, \dots, \vec{s}_m >$ with $\vec{s}_n \equiv \vec{s}_m$.

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

All the state-of-the-art complementary synthesis algorithms [Shen et al. 2009; Liu et al. 2011; Tu and Jiang 2013] assume that $\vec{i}$ can be uniquely determined, so they always take $\vec{i}$ as a whole, and never consider individual variables $i \in \vec{i}$. But in this paper, we need to check each $i \in \vec{i}$ one by one, so there may be minor differences 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 three integers p, l and r, such that for any particular valuation of the output sequence $< \vec{o}_p, \dots, \vec{o}_{p+l+r} >$, i_{p+l} cannot take on two different values. This can be checked by solving $F_{PC}(p, l, r)$ in Equation (1).

$$F_{PC}(p,l,r) \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \neq i'_{p+l} \end{array} \right\} \quad (1)$$



a) checking if i_{p+l} can be uniquely determined b) checking if i_{p+l} can **NOT** be uniquely determined

Fig. 2. Checking whether or not the input can be uniquely determined by the output

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 lengths of the two output sequences $< \vec{o}_{p+1}, \ldots, \vec{o}_{p+l} >$ and $< \vec{o}_{p+l+1}, \ldots, \vec{o}_{p+l+r} >$ that are on the left-hand and right-hand sides of i_{p+l}, which is 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 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$.*

On the other hand, if $F_{PC}(p,l,r)$ is satisfiable, then i_{p+l} cannot be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for this particular combination of p, l and r. There are two possible cases:

(1) i_{p+l} can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for larger p, l and r;
(2) i_{p+l} can not be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for any p, l and r at all.

If it is the first case, then by iteratively increasing the value of p, l and r, $F_{PC}(p,l,r)$ will eventually become 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 between 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 $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ and $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$. It is formally defined in Equation (2) with the last three lines corresponding to the three constraints used to detect loops.

$$F_{LN}(p,l,r) \stackrel{def}{=} \left\{ \begin{array}{cc} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \quad (2)$$

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

PROPOSITION 2.2. *If $F_{LN}(p,l,r)$ is satisfiable, then $i_{p'+l'}$ cannot be uniquely determined by $< \vec{o}_{p'}, \ldots, \vec{o}_{p'+l'+r'} >$ for any larger $p' \geq p$, $l' \geq l$ and $r' \geq r$.*

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

Input: The input variable 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 |
```

Thus, with Propositions 2.1 and 2.2, Algorithm 1 is a halting algorithm that determines if there exists p, l and r that enable an input variable i_{p+l} to be uniquely determined by the encoder's output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. On the 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 $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ and $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$. This will make $F_{LN}(p,l,r)$ satisfiable in Line 6. Both cases will lead to this Algorithm's termination. Please refer to [Shen et al. 2011] for more details.

3. IDENTIFYING FLOW CONTROL VARIABLES

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

The flow control variables $\vec{f}$ are used to represent 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 cannot recognize the validness of $\vec{d}$.

Thus, Algorithm 2 is proposed to identify $\vec{f}$.

At Line 3, it simply applies Algorithm 1 to each input variable $i \in \vec{i}$ of the encoder, to check whether i_{p+l} can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. If yes, the values of p,l and r are also computed.

At Line 5, the input variable i that can be uniquely determined will be added to the vector $\vec{f}$. And, at the following three lines, the maximal values 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 speed up this testing procedure, when the result of $CheckUniqueness$ is (0,?,?,?) at Line 3, every $j \in \vec{i}$ that has different values for j_{p+l} and j'_{p+l} in the satisfying assignment of $F_{LN}(p,l,r)$ can also be ruled out at Line 12, because their own $F_{LN}(p,l,r)$ is also satisfiable.

ALGORITHM 2: $FindFlowControl(\vec{i})$:Identifying the flow control variables

Input: The input variable vector $\vec{i}$.
Output: $\vec{f} \subset \vec{i}$ is the vector of the encoder's input variables that can be uniquely determined by a bounded sequence of output variable vector $\vec{o}$, and the maximal value of p, l and r .

```
1  f⃗ := <>; p_max := 0 ; l_max := 0 ; r_max := 0 ;
2  foreach i ∈ i⃗ do
3      (uniq,p,l,r) := CheckUniqueness(i);
4      if uniq ≡ 1 then
5          f⃗ := i ∪ f⃗;
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 j ∈ i⃗ do
12             if A(j_{p+l}) ≠ A(j'_{p+l}) then i⃗ := i⃗ − j
13 return (f⃗,p_max,l_max,r_max)
```

4. INFERRING PREDICATE THAT ENABLES THE ENCODER'S INPUT DATA VECTOR TO BE UNIQUELY DETERMINED

In subsection 4.1, we propose an algorithm to characterize a Boolean function that makes a Boolean formula satisfiable. In subsection 4.2, we apply this algorithm to infer $valid(\vec{f})$, the predicate that enable $\vec{d}$ to be uniquely determined by a bounded sequence of $\vec{o}$.

4.1. Characterizing a function that makes a Boolean formula satisfiable

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

We need to characterize a Boolean function $FSAT(\vec{a})$, which 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:

$$FSAT(\vec{a}) \stackrel{def}{=} \begin{cases} 1 & \exists\vec{b}.R(\vec{a},\vec{b},1) \\ 0 & otherwise \end{cases} \tag{3}$$

Thus, a naive algorithm 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 the number of valuations to be enumerated is $2^{|\vec{a}|}$, which will prevent this algorithm from terminating within reasonable time for a long $\vec{a}$.

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

$$R(\vec{a}, A(\vec{b}), 1) \tag{4}$$

Because $R(\vec{a}, A(\vec{b}), 0) \wedge R(\vec{a}, A(\vec{b}), 1)$ is unsatisfiable, the 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 the set of $\vec{a}$ that makes $R(\vec{a}, A(\vec{b}), 1)$ satisfiable. 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

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

Input: The Boolean formula $R(\vec{a},\vec{b},t)$, 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(a⃗) := 0 ;
2 while R(a⃗,b⃗,t) ∧ ¬FSAT(a⃗) is satisfiable do
3     assume A : a⃗ ∪ b⃗ ∪ {t} → {0,1} is the satisfying assignment ;
4     φ_A(a⃗) := R(a⃗, A(b⃗), 1) ;
5     φ_B(a⃗) := R(a⃗, A(b⃗), 0) ;
6     assume ITP(a⃗) is the Craig interpolant of φ_A with respect to φ_B ;
7     FSAT(a⃗) := ITP(a⃗) ∨ FSAT(a⃗) ;
8 return FSAT(a⃗)
```

$R(\vec{a}, A(\vec{b}), 0)$ 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.

Based on the foregoing discussion, Algorithm 3 is proposed to characterize $FSAT(\vec{a})$. 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 $FSAT(\vec{a})$. Lines 4 and 5 assign the value of $\vec{b}$ from the satisfying assignment to $R(\vec{a}, \vec{b}, 1)$ 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 ϕ_A and ϕ_B is $\vec{a}$. So a Craig interpolant $ITP(\vec{a})$ can be generated with the Mcmillian's algorithm[McMillan 2003].

$ITP(\vec{a})$ is added to $FSAT(\vec{a})$ 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 $FSAT(\vec{a})$, which means that $FSAT(\vec{a})$ is a Boolean function that covers a bounded and strictly increasing set of valuations of $\vec{a}$. So Algorithm 3 is a halting one.

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

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

$$F^d_{PC}(p,l,r) \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \end{array} \right\} \tag{5}$$

If $F^d_{PC}(p,l,r)$ is satisfiable, then $\vec{d}_{p+l}$ cannot be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. We define a new formula $T_{PC}(p,l,r)$ by collecting the 3rd line of Equation (5):

$$T_{PC}(p,l,r) \stackrel{def}{=} \left\{ \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \right\} \tag{6}$$

By substituting $T_{PC}(p,l,r)$ back into $F^d_{PC}(p,l,r)$, we have a new formula:

$$F'_{PC}(p,l,r,t) \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & t \equiv T_{PC}(p,l,r) \end{array} \right\} \tag{7}$$

It is obvious that $\vec{d}$ cannot be uniquely determined for a particular valuation of p, l and r if $F'_{PC}(p,l,r,1)$ is satisfiable. We further define:

$$\vec{a} \stackrel{def}{=} \vec{f}_{p+l} \tag{8}$$

$$\vec{b} \stackrel{def}{=} \vec{d}_{p+l} \cup \vec{d'}_{p+l} \cup \vec{s}_0 \cup \vec{s'}_0 \cup \bigcup_{0 \leq x \leq p+l+r, x \neq (p+l)} (\vec{i}_x \cup \vec{i'}_x) \tag{9}$$

$\vec{f}_{p+l}$ can be uniquely determined, so we do not need to consider $\vec{f'}_{p+l}$. Thus, $\vec{a} \cup \vec{b}$ is the vector that contains all the input variable vectors $< \vec{i}_0, \ldots, \vec{i}_{p+l+r} >$ and $< \vec{i'}_0, \ldots, \vec{i'}_{p+l+r} >$ at all cycles for the two sequences of unrolled transition function.

It also contains the two initial states $\vec{s}_0$ and $\vec{s'}_0$. In addition, T is a function that computes the next state and the output variable vector from the current state and input variable 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 Boolean function 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:

$$FSAT_{PC}(p,l,r) \stackrel{def}{=} CharacterizingFormulaSAT(F'_{PC}(p,l,r,t),\vec{a},\vec{b},t) \tag{10}$$

Thus, we have the following proposition:

PROPOSITION 4.1. *$FSAT_{PC}(p,l,r)$ is the Boolean function over $\vec{f}_{p+l}$ that makes $\vec{d}_{p+l}$ to be not uniquely determined for a particular p, l and r.*

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

$$F^d_{LN}(p,l,r) \stackrel{def}{=} \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{11}$$

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

$$T_{LN}(p,l,r) \stackrel{def}{=} \left\{ \begin{array}{ll} & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{12}$$

By replacing the 3rd line and the last three lines of Equation (11) with $T_{LN}(p,l,r)$, we got:

$$F'_{LN}(p,l,r,t) \stackrel{def}{=} \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & t \equiv T_{LN}(p,l,r) \end{array} \right\} \tag{13}$$

Then $\vec{d}$ cannot 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. 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

$$FSAT_{LN}(p,l,r) \stackrel{def}{=} CharacterizingFormulaSAT(F'_{LN}(p,l,r,t),\vec{a},\vec{b},t) \tag{14}$$

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

```
1 p:= p_max; l:= l_max; r:= r_max ;
2 while ¬FSAT_LN(p,l,r) ∧ FSAT_PC(p,l,r) is satisfiable do
3     p ++ ; l ++ ; r ++ ;
4 end
5 return ¬FSAT_LN(p,l,r)
```

Thus we have the following proposition:

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

With Propositions 4.1 and 4.2, the algorithm that infers the predicate $valid(\vec{f}_{p+l})$ is shown in Algorithm 4. It just iteratively increases the value of p, l and r, until $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ is unsatisfiable. 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. $\mathbf{FSAT_{PC}(p,l,r)}$ *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 $r' > l$, we have $F'_{PC}(p',l',r',1) \Rightarrow F'_{PC}(p,l,r,1)$. So, according to Equation (10), we have $FSAT_{PC}(p',l',r') \Rightarrow FSAT_{PC}(p,l,r)$. Thus, $FSAT_{PC}(p,l,r)$ monotonically decreases. □

LEMMA 4.4. $\mathbf{FSAT_{LN}(p,l,r)}$ *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 $F'_{LN}(p',l',r',1)$ satisfiable for all larger p', l' and r'. So, according to Equation (14), we have $FSAT_{LN}(p,l,r) \Rightarrow FSAT_{LN}(p',l',r')$, that is, $FSAT_{LN}(p,l,r)$ monotonically increases. □

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. □

These three lemmas are depicted intuitively in Figure 3, which makes 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. As 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 values 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 $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ and $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$. 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. □

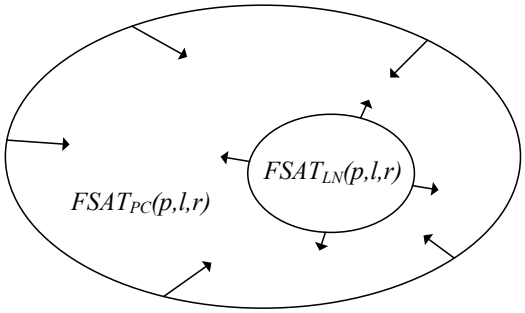


Fig. 3. The monotonicity of $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$

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 enable $\vec{d}$ 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 enable $\vec{d}$ 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 makes $\vec{d}$ 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 Lemmas 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 makes $\vec{d}$ to be **NOT** uniquely determined. This proves the only covering case. □

5. CHARACTERIZING THE DECODER'S BOOLEAN FUNCTION

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

5.1. Characterizing the decoder's Boolean function that computes $\vec{f}$

Each variable $f \in \vec{f}$ can be uniquely determined by a bounded sequence of the encoder's output. So, for each particular valuation of the encoder's output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, f_{p+l} cannot be 0 and 1 at the same time. Thus, the decoder's Boolean function that computes f_{p+l} is exactly the Craig interpolant of ϕ_A with respect to ϕ_B:

$$\phi_A \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & f_{p+l} \equiv 1 \end{array} \right\} \tag{15}$$

$$\phi_B \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & f'_{p+l} \equiv 0 \end{array} \right\} \tag{16}$$

It is obvious that $\phi_A \wedge \phi_B$ equals $F_{PC}(p,l,r)$ in Equation (1), so it is unsatisfiable. The common variable set of ϕ_A and ϕ_B is $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. 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 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 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} 0. Thus, ITP is the decoder's Boolean function that computes $f \in \vec{f}$.

5.2. Characterizing the decoder's Boolean function that computes $\vec{d}$

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

$$\phi'_A \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & d_{p+l} \equiv 1 \\ \wedge & valid(\vec{f}_{p+l}) \end{array} \right\} \tag{17}$$

$$\phi'_B \stackrel{def}{=} \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & d'_{p+l} \equiv 0 \\ \wedge & valid(\vec{f'}_{p+l}) \end{array} \right\} \tag{18}$$

Each variable $d \in \vec{d}$ can be uniquely determined by the encoder's output only when $valid(\vec{f})$ holds. So, if $valid(\vec{f}_{p+l})$ holds, for each particular valuation of the encoder's output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, d_{p+l} cannot be 0 and 1 at the same time. 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 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 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 input data variable $d \in \vec{d}_{p+l}$ cannot 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 the invalid input data vector by computing the value of control flow vector $\vec{f}$, and ignore 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 mechanisms. This is not a surprise to us, because these benchmarks all come from real industrial projects. We will present the experimental result for them in the following subsections.

On the other hand, we have also found that the benchmarks used in [Tu and Jiang 2013] contain no flow control mechanism, and hence they will not be discussed here.

6.1. PCI Express 2.0 encoder

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

The list of input and output variables is shown in Table I. According to the 8b/10b encoding scheme's coding table[wikipedia 2013a], when $TXDATAK \equiv 0$, $TXDATA$ can be of any value. But when $TXDATAK \equiv 1$, $TXDATA$ can only be 1C, 3C, 5C, 7C,

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

	variable name	width	description
Inputs	$TXDATA$	8	The data to be encoded
	$TXDATAK$	1	1 means $TXDATA$ is a controlling character, 0 means $TXDATA$ is normal data
	$CNTL\_TXEnable\_P0$	1	Indicating the validness of $TXDATA$ and $TXDATAK$
Outputs	HSS_TXD	10	The encoded data
	$HSS\_TXELECIDLE$	1	The electrical idle state

9C, BC, DC, FC, F7, FB, FD and FE. So, we write an assertion to rule out those combinations that are not in this coding table. This assertion is embed into the transition function T, so that it can be enforced at every cycle in the unrolled state sequences.

Algorithm 2 costs 0.924754 seconds to identify the flow control variable $CNTL\_TXEnable\_P0$. And then Algorithm 4 costs 2.067509 seconds to infer the predicate $CNTL\_TXEnable\_P0 \equiv 1$ that enables the input data vector to be uniquely determined. Finally, with the inferred predicate, generating the decoder's Boolean functions for $CNTL\_TXEnable\_P0$, $TXDATA$ and $TXDATAK$ costs 3.121821 seconds. After being 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's algorithms is their ability to handle invalid input data vector. So, it should be very interesting to show how the invalid input data vector is mapped to output variable vector $\vec{o}$. 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 variable $HSS\_TXELECIDLE$ becomes 1. So, the decoder can use the output variable $HSS\_TXELECIDLE$ to uniquely determine the value of flow control variable $CNTL\_TXEnable\_P0$.

6.2. 10G Ethernet encoder

This encoder is compliant with clause 48 of IEEE 802.3 standard [IEEE 2012a]. After deleting empty line and comments, this encoder has 214 lines of verilog. After being 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 variables is 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 of 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 that are not in this table and embed it into the the transition function T.

Algorithm 2 costs 0.619508 seconds to identify the flow control variable bad_code. And then Algorithm 4 costs 1.443065 seconds to infer the predicate $bad\_code \equiv 0$ that enables the input data vector to be uniquely determined. Finally, generating the decoder's Boolean functions for bad_code, $encode\_data\_in$ and $konstant$ costs 2.202401 sec-

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

	variable name	width	description
Inputs	$encode\_data\_in$	8	The data to be encoded
	$konstant$	1	1 means $encode\_data\_in$ is a special character, 0 means $encode\_data\_in$ is normal data
	bad_code	1	Indicating the validness of $konstant$ and $encode\_data\_in$
Outputs	$encode\_data\_out$	10	The encoded data

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

	variable name	width	description
Inputs	txd	8	The data to be encoded
	$tx\_enc\_ctrl\_sel$	1	Refer to Table IV
	tx_en	1	Transmission enable
	tx_er	1	Transmitting an error character
Outputs	$tx\_10bdata$	10	The encoded data
	$txd\_eq\_crs\_ext$	10	Transmitting an special error character with $tx\_er \equiv 1$ and $txd \equiv 8'h0F$
	tx_er_d	1	Transmitting an error character
	tx_en_d	1	Transmission enable
	$pos\_disp\_tx\_p$	1	Indicating positive parity

onds. After being 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 does the PCI Express 2.0 encoder mentioned above, the way it handle the invalid input data vector is different. This encoder does not have a separate output variable to indicate the validness of the output data; instead, the validness and exact value of all input variables are both encoded in $encode\_data\_out$. By studying 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 variable $encode\_data\_out$ will become 0010111101. So the decoder can use the output variable $encode\_data\_out$ to uniquely determine the value of the flow control variable bad_code.

6.3. UltraSPARC T2 Ethernet encoder

This encoder comes from the UltraSPARC T2 open source processor designed by Sun Microsystems. It is compliant with clause 36 of IEEE 802.3 standard [IEEE 2012a]. After deleting empty line and comments, this encoder's source code has 864 lines of verilog. After being 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 variables is 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 from that of the above two encoders. The data to be encoded is the 8-bit txd, but there is no standalone variable to indicate the 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 functionalities of the control symbol indication and flow control mechanism are combined in $tx\_enc\_ctrl\_sel$. The last four cases in Table IV can never be uniquely determined, because they cannot be distinguished from the case of 'PCS_ENC_DATA. So we write an assertion to rule them out.

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

The name of action	The meaning of action
'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

Algorithm 2 costs 11.750317 seconds to identify the flow control variables $tx\_enc\_ctrl\_sel$, 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 enables the input data vector 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 being 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 own particular control symbol values assigned to $tx\_10bdata$, so the decoder can recover the value of the flow control variable $tx\_enc\_ctrl\_sel$ from $tx\_10bdata$.

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 generates 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 that it has 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 the restriction that the inferred assertion must hold on all cycles, to prevent the encoder from leaving the unique state set. Our Algorithm 4, on the other hand, is the first algorithm that allows states with and without the inferred assertion to be interleaved freely with each other, which make it possible to handle encoder with flow control mechanism.

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, so 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 cannot be handled by the state-of-the-art algorithms. But for the encoders with flow control mechanism used in our experiments, our algorithm is enough, and therefore we have not implemented their algorithm in our framework.

7.2. Program inversion

According to Gulwani[2010], program inversion involves deriving 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], which could handle only 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, and 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 a process that automatically generates a translator between two different communication protocols. This is relevant to our work, because both focus on synthesizing communication circuits.

Avnit et al. [2008; 2009] first defined a general model for describing different protocols, and then 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 the greatest fixed point for the update function of the buffer's control states. Latter, they [2009] improved their algorithm with a more efficient design space exploration algorithm.

8. CONCLUSIONS

In this paper, we propose, for the first time, a framework to handle flow control mechanism in complementary synthesis problem. Experimental results indicate that our framework can always successfully handle many complex encoders from real industrial projects, such as PCI Express [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 the anonymous reviewers for their fruitful suggestions.

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'Silvaand Arcot Sowmya, and S. Rameshand 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

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

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

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

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

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

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

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)

wikipedia. 2013d. PCI Express. (25 Jan. 2013). Retrieved January 25, 2013 from http://en.wikipedia.org/wiki/PCI_Express

Received February 2014; revised March 2014; accepted June 2014

Online Appendix to: Complementary Synthesis for Encoder with Flow Control Mechanism

YING QIN and QINGBO WU and HUADONG DAI and YAN JIA and SHENGYU SHEN,
School of Computer, National University of Defense Technology

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