Complementary Synthesis for Encoders with Pipeline and Flow Control Mechanism

***Abstract*—Complementary synthesis automatically generates an encoder's decoder that recovers the encoder's inputs from its output. This paper proposes the first complementary synthesis algorithm that can handle flow control and pipeline mechanism widely employed in modern encoders. First, it identifies the flow control variables and infers the flow control predicate. Second, it identifies all pipeline stages in the encoder and infers the flow control predicate for each pipeline stages. Finally, the decoder's Boolean functions that recover each pipeline stage and input are characterized with Craig interpolant. Experimental results indicate that this algorithm can always generate pipelined decoders with flow control mechanism.**

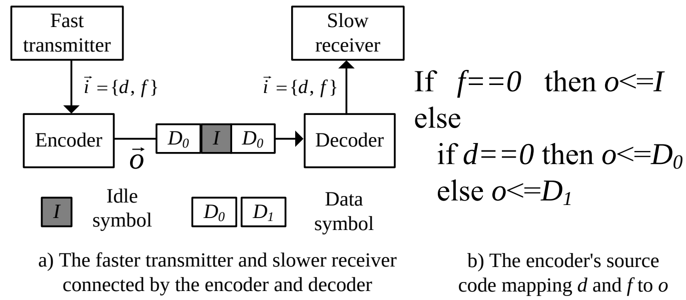

Fig. 1. Encoder with flow control mechanism

I. Introduction

One of the most difficult jobs in designing communication chips is to design and verify complex encoder and decoder pairs. The encoder maps its input $\vec{i}$ to its output $\vec{o}$, while the decoder recovers $\vec{i}$ from $\vec{o}$. Complementary synthesis [1]–[6] eases this job by automatically generating a decoder from an encoder, with the assumption that $\vec{i}$ can always be uniquely determined by and recovered from a bounded sequence of $\vec{o}$.

However, the flow control mechanism [7] in many encoders fails this assumption. Figure 1a) shows a communication system with flow control mechanism that can prevent faster transmitter from overwhelming slower receiver. When receiver can keep up with the transmitter, the transmitter sends the data bit d to the encoder with $f \equiv 1$. According to the encoder's source code in Figure 1b), it maps $\vec{i} = \{d, f\}$ to $\vec{o} = D_d$, which can uniquely determine the value of d and f. However, when receiver can NOT keep up with the transmitter, the transmitter sends $f \equiv 0$ to the encoder, which maps it to the idle symbol I that can uniquely determine only f but not d. This makes it impossible to recover d from $\vec{o}$.

Qin et al. [8] proposed the first complementary synthesis algorithm that can handle flow control mechanism by partitioning $\vec{i}$ into $\vec{f}$ and $\vec{d}$, in which $\vec{f}$ contains all $f \in \vec{i}$ that can always be uniquely determined by $\vec{o}$. It further infers a predicate $valid(\vec{f})$ that enable $\vec{d}$ to be recovered from $\vec{o}$.

This algorithm ignored the encoder's internal pipeline used to cut the datapath and boosts frequency. So the non-pipelined decoder generated by [8] is much slower than the encoder.

To overcome this problem, this paper proposes a novel algorithm to generate pipelined decoders for flow controlled encoder. It first applies Qin et al. [8]'s algorithm to identify $\vec{f}$ and infers $valid(\vec{f})$. It then identifies all state variables in each pipeline stage $\vec{stg}^j$, and partitions them into data vector $\vec{d^j}$ and flow control vector $\vec{f^j}$, and infers predicate $valid(f^j)$ that can make $\vec{d^j}$ to be uniquely determined by the next pipeline stage. It finally characterize the Boolean functions that recover each $\vec{stg}^j$ and $\vec{i}$ with Craig interpolant [9].

Experimental result indicates that this algorithm can always generate pipelined decoder with flow control mechanism.

The remainder of this paper is organized as follows. Section II introduces the background material; Section III introduces the overall framework of our algorithm. Section IV identifies $\vec{f^j}$ and $\vec{d^j}$ in each pipeline stages $\vec{stg}^j$, while Section V characterize the decoder's Boolean functions that recover each pipeline stage $\vec{stg}^j$ and the input vector $\vec{i}$. Sections VI and VII present the experimental results and related works; Finally, Section VIII sums up the conclusion.

II. Preliminaries

A. Propositional satisfiability

The Boolean value set is denoted as $\mathbb{B} = \{0, 1\}$. A vector of variables is represented as $\vec{v} = (v, \dots)$. $|\vec{v}|$ is the number of variables in $\vec{v}$. If a variable v is a member of $\vec{v}$, then we say $v \in \vec{v}$; otherwise $v \notin \vec{v}$. $v \cup \vec{v}$ is the new vector that contains both v and all members of $\vec{v}$ $\vec{v} - v$ is he new vector that contains all members of $\vec{v}$ except v, $\vec{a} \cup \vec{b}$ is the new vector that contains all members of $\vec{a}$ and $\vec{b}$

For formula F over variable set V, SAT solvers try to find a satisfying assignment $A : V \to \mathbb{B}$, so that F can be evaluated to 1. If A exists, then F is satisfiable; otherwise unsatisfiable.

For 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 $\phi_I \wedge \phi_B$ is unsatisfiable and $\phi_A \Rightarrow \phi_I$. ϕ_I is the **interpolant** [10] of ϕ_A with respect to ϕ_B.

B. Finite state machine(FSM)

The encoder is modeled by a FSM $M = (\vec{s}, \vec{i}, \vec{o}, T)$, consisting of a state variable vector $\vec{s}$, an input variable vector $\vec{i}$, an output variable vector $\vec{o}$, and a transition function

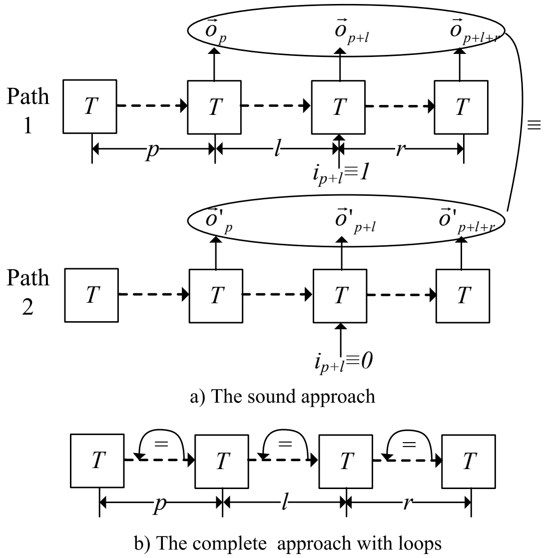

Fig. 2. The sound and complete approximative approaches

$T : \vec{s} \times \vec{i} \to \vec{s} \times \vec{o}$ that computes the next state and output variable vector from the current state and input variable vector. When unrolling transition function T, $s \in \vec{s}$, $i \in \vec{i}$ and $o \in \vec{o}$ at the n-th step are respectively denoted as s_n, i_n and o_n. $\vec{s}$, $\vec{i}$ and $\vec{o}$ at the n-th step are respectively denoted as $\vec{s}_n$, $\vec{i}_n$ and $\vec{o}_n$. A **path** is a state sequence $< \vec{s}_n, \ldots, \vec{s}_m >$ 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 $< \vec{s}_n, \ldots, \vec{s}_m >$ with $\vec{s}_n \equiv \vec{s}_m$.

C. The halting algorithm that identifies flow control vector $\vec{f}$

Qin [8] proposed an algorithm to identify $\vec{f}$ by iteratively calling a sound and a complete approaches until they converge.

1) The sound approach in Figure 2a) shows how to check whether an input variable $i \in \vec{i}$ can be uniquely determined by a bounded sequence of $\vec{o}$: if there exists p, l and r, such that for every output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, i_{p+l} cannot be different. This is equal to the unsatisfiability of $F_{PC}(p, l, r)$ in Equation (1), in which Line 1 and 2 of correspond to the two paths in Figure 2a), Line 3 forces these two paths' output to be the same, while Line 4 forces their i_{p+l} to be different.

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

2) The complete approach in Figure 2b) shows how to determine the non-existence of p, l and r: It is similar to Figure 2a) but with three new 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 as $F_{LN}(p, l, r)$ in Equation (2) with the last three lines corresponding to the three new constraints. If $F_{LN}(p, l, r)$ is satisfiable, then by unrolling the three loops, we can be sure that any larger p, l and r can also make $F_{LN}(p, l, r)$ and $F_{PC}(p, l, r)$ satisfiable.

Algorithm 1: Identifying the flow control vector $\vec{f}$

Input: The input variable vector $\vec{i}$.
Output: $\vec{f} \subset \vec{i}$, and the maximal p, l and r reached.

```
1 f := {}; d := {}; p := 0 ; l := 0 ; r := 0 ;
2 while i ≠ {} do
3     assume i ∈ i;
4     p++;   l++;   r++;
5     if F_PC(p,l,r) is unsatisfiable for i then
6         f := i ∪ f ;   i := i − i;
7     else if F_LN(p,l,r) is satisfiable for i then
8         d := i ∪ d ;   i := i − i
9 return (f, p, l, r)
```

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

Algorithm 1 shows an iterative framework to call these two approaches. It is a halting algorithm because if there indeed exists such p, l and r that make $F_{PC}(p, l, r)$ satisfiable for all $i \in \vec{i}$, then they can eventually be found in Line 6; Otherwise, p, l and r will eventually be larger than the length of the encoder's longest non-loop path, which means $F_{LN}(p, l, r)$ is satisfiable for all $i \in \vec{i}$. Both case will terminate the loop.

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

For a particular Boolean relation $R(\vec{a}, \vec{b}, t)$, with $R(\vec{a}, \vec{b}, 0) \wedge R(\vec{a}, \vec{b}, 1)$ unsatisfiable, Subsection 4.1 of [8] proposes an algorithm $CharacterizingFormulaSAT$ to characterize a Boolean function $FSAT_R(\vec{a})$ that covers and only covers all the valuations of $\vec{a}$ that can make $R(\vec{a}, \vec{b}, 1)$ satisfiable.

As shown in Figure 3, it iteratively increases p, l and r, and calls $CharacterizingFormulaSAT$ to characterize $\neg FSAT_{PC}(p, l, r)$, a monotonically growing under-approximation of $valid(\vec{f})$, and $\neg FSAT_{LN}(p, l, r)$, a monotonically shrinking over-approximation of $valid(\vec{f})$. These two approximations will eventually converge to $valid(\vec{f})$.

1) Monotonically growing under-approximation: By replacing i in Equation (1) with $\vec{d}$ inferred in Algorithm 1, and collecting the 3rd line of (1) into Equation (3):

$$T_{PC}(p,l,r) := \left\{ \quad \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \quad \right\} \quad (3)$$

we have:

$$F'^{d}_{PC}(p,l,r,t) :=$$
$$\left\{\begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m)\equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m)\equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t\equiv T_{PC}(p,l,r) \\ \wedge & \vec{d}_{p+l}\neq \vec{d'}_{p+l} \end{array}\right\} \quad (4)$$

If $F'^{d}_{PC}(p,l,r,1)$ is satisfiable, then $\vec{d}_{p+l}$ can't be uniquely determined by $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$. We further define:

$$\vec{a} := \vec{f}_{p+l}$$
$$\vec{b} := \vec{d}_{p+l}\cup\vec{d'}_{p+l}\cup\vec{s}_0\cup\vec{s'}_0\cup \bigcup_{\substack{0\le x\le p+l+r \\ x\neq(p+l)}} (\vec{i}_x\cup\vec{i'}_x) \quad (5)$$

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 steps for the two sequences of unrolled transition function. It also contains the two initial states $\vec{s}_0$ and $\vec{s'}_0$. So $\vec{a}$ and $\vec{b}$ can uniquely determine the value of t in $F'^{d}_{PC}(p,l,r,t)$, which means $R(\vec{a},\vec{b},1)\wedge R(\vec{a},\vec{b},0)$ is unsatisfiable. Thus, for a particular combination of p, l and r, the Boolean function over $\vec{f}_{p+l}$ that makes $F'^{d}_{PC}(p,l,r,1)$ satisfiable can be computed by calling Algorithm **??** with $F'^{d}_{PC}(p,l,r,t)$, $\vec{a}$ and $\vec{b}$ defined above:

$$FSAT_{PC}(p,l,r) :=$$
$$CharacterizingFormulaSAT(F'^{d}_{PC}(p,l,r,t),\vec{a},\vec{b},t) \quad (6)$$

As shown in Figure 3, $\neg FSAT_{PC}(p,l,r)$ is an under-approximation of $valid(\vec{f})$ monotonically growing with respect to p, l and r.

2) Computing monotonically shrinking over-approximation: Similarly, we can define :

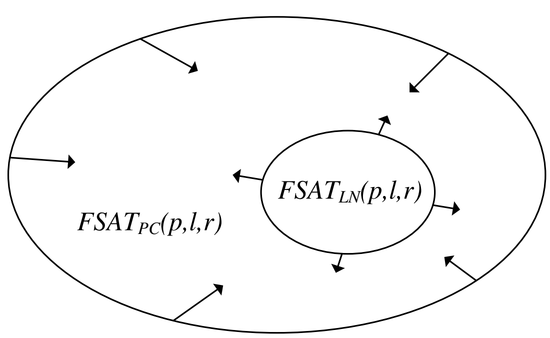

Fig. 3. The monotonicity of $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$

Algorithm 2: Inferring $valid(\vec{f}_{p+l})$

p:= 0; l:= 0; r:= 0 ;
while $\neg FSAT_{LN}(p,l,r)\wedge FSAT_{PC}(p,l,r)$ *is satisfiable* **do**
  p ++ ; l ++ ; r ++ ;
return $\neg FSAT_{LN}(p,l,r)$

$$T_{LN}(p,l,r) :=$$
$$\left\{\begin{array}{cc} & \bigwedge_{m=p}^{p+l+r}\vec{o}_m\equiv\vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x\equiv\vec{s}_y\wedge\vec{s'}_x\equiv\vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x\equiv\vec{s}_y\wedge\vec{s'}_x\equiv\vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x\equiv\vec{s}_y\wedge\vec{s'}_x\equiv\vec{s'}_y\} \end{array}\right\} \quad (7)$$

$$F'^{d}_{LN}(p,l,r,t) :=$$
$$\left\{\begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m)\equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m)\equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t\equiv T_{LN}(p,l,r) \\ \wedge & \vec{d}_{p+l}\neq \vec{d'}_{p+l} \end{array}\right\} \quad (8)$$

$$FSAT_{LN}(p,l,r) :=$$
$$CharacterizingFormulaSAT(F'^{d}_{LN}(p,l,r,t),\vec{a},\vec{b},t) \quad (9)$$

As shown in Figure 3, $\neg FSAT_{LN}(p,l,r)$ is an over-approximation of $valid(\vec{f})$ monotonically shrinking with respect to p, l and r.

3) Inferring $valid(\vec{f})$ with Algorithm 2: It just iteratively increases the value of p, l and r, until $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$ converge. Please refer to [8] for the proofs of its termination and correctness.

III. Algorithm Framework

A. A general model for the encoder

As shown in Figure 4, we assume that the encoder has n pipeline stages $\vec{stg}^j$, where $0\le j\le n-1$. And each pipeline stage $\vec{stg}^j$ can be further partitioned into flow control vector $\vec{f^j}$ and data vector $\vec{d^j}$. The input vector $\vec{i}$, as in [8], can also be partitioned into flow control vector $\vec{f}$ and data vector $\vec{d}$. If we take the combinational logic block C^j as a function, then this encoder can be represented by the following equations.

$$\begin{array}{rcll} \vec{stg}^0 & := & C^0(\vec{i}) & \\ \vec{stg}^j & := & C^j(\vec{stg}^{j-1}) & 1\le j\le n-1 \\ \vec{o} & := & C^n(\vec{stg}^{n-1}) & \end{array} \quad (10)$$

In the remainder of this paper, superscript always means the pipeline stage, while the subscript, as mentioned in Subsection II-B, always means the step index in the unrolled transition function. For example, $\vec{stg}^j$ is the j-th pipeline stage. While $\vec{stg}^j_i$ is the value of this j-th pipeline stage at the i-th step in the unrolled state transition sequence.

Algorithm 3: Minimizing r

1 **for** $r' := r \to 0$ **do**
2   **if** $r' \equiv 0$ *or* $F_{PC}(p, l, r'-1) \wedge valid(\vec{f}_{p+l})$ *is satisfiable for some* $i \in \vec{i}$ **then**
3   break
4 return r'

B. Algorithm framework

With the encoder model shown in Figure 4, our overall algorithm framework is:

1) Calling Algorithm 1 to partition $\vec{i}$ into $\vec{f}$ and $\vec{d}$.
2) Calling Algorithm 2 to infer $valid(\vec{f})$ that enables $\vec{d}$ to be uniquely determined with parameters p, l and r.
3) In Section IV, identifying $\vec{f^j}$ and $\vec{d^j}$ in each pipeline stage $\vec{stg}^j$.
4) In Section V, characterizing the decoder's Boolean functions that recover each pipeline stages $\vec{stg}^j$ and input vector $\vec{i}$.

IV. Inferring the encoder's pipeline structure

A. Minimizing r and l

As Algorithm 2 increases p, l and r simultaneously, there may be some redundancy in the value of l and r. So we need to first minimize r in Algorithm 3.

In Line 2, we enforce the inferred flow control predicate $valid(\vec{f})$ by conjugating it with $F_{PC}(p, l, r'-1)$. When it is satisfiable, then r' is the last one that makes $F_{PC}(p, l, r') \wedge valid(\vec{f}_{p+l})$ unsatisfiable, we return it directly. On the other hand, when $r' \equiv 0$, $F_{PC}(p, l, 0)$ must have been tested in last iteration, and the result must be unsatisfiable. In this case we return 0.

Now, we have a minimized r from Algorithm 3, which can make $\vec{i}_{p+l}$ to be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$.

We further require that :

1) As shown in Figure 5, l can be reduced to 0, which means $\vec{i}_p$ can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+r} >$, that is, the set of future outputs.
2) The above mentioned output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+r} >$ can be further reduced to $\vec{o}_{p+r}$. This means $\vec{o}_{p+r}$ is the only output vector needed to recover the input vector $\vec{i}_p$.

Checking these two requirements equals to checking the unsatisfiability of $F'_{PC}(p, r) \wedge valid(\vec{f}_{p+l})$, with $F'_{PC}(p, r)$ defined below:

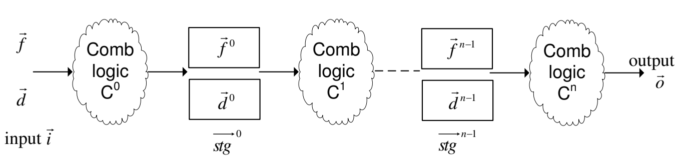

Fig. 4. A general structure of the encoder with pipeline stages and flow control mechanism

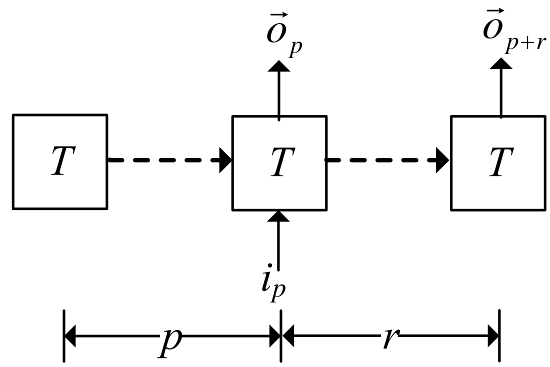

Fig. 5. Recovering input with reduced output sequence

$$F'_{PC}(p, r) :=
\left\{ \begin{array}{cc}
 & \bigwedge_{m=0}^{p+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\
\wedge & \bigwedge_{m=0}^{p+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\
\wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\
\wedge & i_p \equiv 1 \wedge i'_p \equiv 0
\end{array} \right\} \tag{11}$$

This equation seems much stronger than the general requirement in Equation (1). But we will show in experimental results that they are always fulfilled.

B. Inferring pipeline stages

Now, with the inferred p and r, we need to generalize F'_{PC} in Equation (11) to the following new formula that can determine whether a particular variable v at step j can be uniquely determined by a vector $\vec{w}$ at step k. Now v and $\vec{w}$ can be either input, registers or output variables.

$$F''_{PC}(p, r, v, j, \vec{w}, k) :=
\left\{ \begin{array}{cc}
 & \bigwedge_{m=0}^{p+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\
\wedge & \bigwedge_{m=0}^{p+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\
\wedge & \vec{w}_k \equiv \vec{w'}_k \\
\wedge & v_j \equiv 1 \wedge v'_j \equiv 0
\end{array} \right\} \tag{12}$$

Obviously, when $F''_{PC}(p, r, v, j, \vec{w}, k)$ is unsatisfiable, $\vec{w}_k$ can uniquely determine v_j.

For $0 \leq j \leq n-1$, in the j-th pipeline stage $\vec{stg}^j$, its flow control vector $\vec{f^j}$ is exactly the set of registers $s \in \vec{s}$ that can be uniquely determined at the $j - ((n-1) - (p+r))$-th step by $\vec{o}$ at the $p+r$-th step without enforcing $valid(\vec{f}_p)$. It can be formally defined as:

$$\vec{f^j} := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p, r, s, j-D, \vec{o}, p+r) \\ is\ unsatisfiable \end{array} \right\} \tag{13}$$

with:

$$D \quad := \quad (n-1) - (p+r) \tag{14}$$

While the data vector $\vec{d^j}$ in the j-th pipeline stage $\vec{stg}^j$ is the set of registers $s \in \vec{s}$ that can be uniquely determined at the same $j-((n-1)-(p+r))$-th step by $\vec{o}$ at the $p+r$-th step by enforcing $valid(\vec{f_p})$. It can be formally defined as:

$$\vec{d^j} := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,j-D,\vec{o},p+r) \wedge valid(\vec{f_p}) \\ is\ unsatisfiable \end{array} \right\} \quad (15)$$

V. Characterizing the Boolean functions recovering input variables and pipeline registers

A. Characterizing the Boolean functions recovering the last pipeline stage

According to Equation (13), every registers $s \in \vec{f}^{n-1}$ can be uniquely determined by $\vec{o}$ at the $p+r$-th step, that is, $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ is unsatisfiable and can be partitioned into :

$$\phi_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+r} \equiv 1 \end{array} \right\} \quad (16)$$

$$\phi_B := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & s'_{p+r} \equiv 0 \end{array} \right\} \quad (17)$$

As $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ equals to $\phi_A \wedge \phi_B$, so $\phi_A \wedge \phi_B$ is unsatisfiable. And the common variables of ϕ_A and ϕ_B is $\vec{o}_{p+r}$.

According to [9], a Craig interpolant ϕ_I of ϕ_A with respect to ϕ_B can be constructed, which refer only to $\vec{o}_{p+r}$, and covers all the valuations of $\vec{o}_{p+r}$ that can make $s_{p+r} \equiv 1$. At the same time, $\phi_I \wedge \phi_B$ is unsatisfiable, which means ϕ_I covers nothing that can make $s_{p+r} \equiv 0$.

Thus, ϕ_I can be used as the decoder's Boolean function that recovers $s \in \vec{f}^{n-1}$ from $\vec{o}$.

By replacing $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ with $F''_{PC}(p,r,s,p+r,\vec{o},p+r) \wedge valid(f_p)$, we can similarly characterize the Boolean function that recovers $s \in \vec{d}^{n-1}$.

B. Characterizing the Boolean functions recovering other pipeline stages

According to Figure 4, $\vec{f^j}$ at the $j-D$-step can be uniquely determined by $\vec{stg}^{j+1}$ at the $j-D+1$-th step. So we can partition the unsatisfiable formula $F''_{PC}(p,r,s,j-D,\vec{stg}^{j+1},j-D+1)$ into the following two equations:

$$\phi_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{j-D} \equiv 1 \end{array} \right\} \quad (18)$$

$$\phi_B := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{stg}^{j+1}_{j-D+1} \equiv \vec{stg'}^{j+1}_{j-D+1} \\ \wedge & s'_{j-D} \equiv 0 \end{array} \right\} \quad (19)$$

Again, a Craig interpolant ϕ_I of ϕ_A with respect to ϕ_B can be constructed, and used as the decoder's Boolean function that recovers $s \in \vec{f^j}$ from $\vec{stg}^{j+1}$.

Similarly, by replacing $F''_{PC}(p,r,s,j-D,\vec{stg}^{j+1},j-D+1)$ with $F''_{PC}(p,r,s,j-D,\vec{stg}^{j+1},j-D+1) \wedge valid(f_p)$, we can characterize the Boolean function that recovers $s \in \vec{d^j}$ from $\vec{stg}^{j+1}$.

C. Characterizing the Boolean functions recovering the encoder's input variables

According to Figure 4, $\vec{f}$ at the p-step can be uniquely determined by $\vec{stg}^0$ at the p-th step. $F''_{PC}(p,r,i,p,\vec{stg}^0,p)$ is unsatisfiable and can be partitioned into :

$$\phi_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & i_p \equiv 1 \end{array} \right\} \quad (20)$$

$$\phi_B := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{stg}^0_p \equiv \vec{stg'}^0_p \\ \wedge & i'_p \equiv 0 \end{array} \right\} \quad (21)$$

Again, the Craig interpolant ϕ_I of ϕ_A with respect to ϕ_B can be used as the decoder's Boolean function that recovers $i \in \vec{f}$ from $\vec{stg}^0$.

Similarly, by replacing $F''_{PC}(p,r,i,p,\vec{stg}^0,p)$ with $F''_{PC}(p,r,i,p,\vec{stg}^0,p) \wedge valid(\vec{f_p})$, we can characterize the Boolean function that recovers $i \in \vec{d}$ from $\vec{stg}^0$.

VI. Experimental results

We have implemented these algorithms in OCaml language, and solved the generated CNF formulas with MiniSat 1.14 [11]. All experiments have been run on a server with 16 Intel Xeon E5648 processors at 2.67GHz, 192GB memory, and CentOS 5.4 Linux.

A. Comparing timing and area

Table I shows the benchmarks used in this paper. The 2nd and 3rd column show respectively the number of inputs, outputs and registers of each benchmark. The 4th column shows the area of the encoder when mapped to LSI10K library with Design Compiler. In this paper, all area and delay are obtained in the same setting.

The 6th to 8th columns show respectively the run time of [2]'s algorithm to generate the decoder without pipeline, and the delay and area of the generated decoder. While the 9th to 11th columns show respectively the run time of this paper's algorithm to generate the pipelined decoder, and the delay and area of the generated decoder.

Comparing the 7th and the 10th column indicates that the decoders' delay have been significantly improved.

One thing that is a little bit surprise is, the two largest benchmarks scrambler and xfi do not have pipeline stages inside. We study their code and confirm that this is true. Their area are so large because they use much wider datapaths with 64 to 72 bits.

TABLE I
BENCHMARKS AND EXPERIMENTAL RESULTS

Names	The encoders				decoder generated by [2]			decoder generated by this paper		
	# in/out	# reg	area	Description of Encoders	run time	delay (ns)	area	run time	delay (ns)	area
pcie	10/11	23	326	PCIE 2.0 [12]	0.37	7.20	624	8.08	5.89	652
xgxs	10/10	16	453	Ethernet clause 48 [13]	0.21	7.02	540	4.25	5.93	829
t2eth	14/14	49	2252	Ethernet clause 36 [13]	12.7	6.54	434	430.4	6.12	877
scrambler	64/64	58	1034	inserting 01 flipping	no pipeline stages found					
xfi	72/66	72	7772	Ethernet clause 49 [13]						

TABLE II
INFERRED PIPELINE STAGES OF PCIE

	input	pipeline stage 0	pipeline stage 1
flow control vector	CNTL_TXEnable_P0	InputDataEnable_P0_reg	OutputData_P0_reg[9:0] OutputElecIdle_P0_reg
flow control predicate	CNTL_TXEnable_P0	InputDataEnable_P0_reg	true
data vector	TXDATA[7:0] TXDATAK	InputData_P0_reg[7:0] InputDataK_P0_reg	

TABLE IV
INFERRED PIPELINE STAGES OF T2ETHER

	input	pipeline stage 0	pipeline stage 1	pipeline stage 2
flow control vector	tx_enc_ctrl_sel[3:0]	qout_reg_0_8 qout_reg_2_4 qout_reg_1_4	qout_reg_0_9 qout_reg_1_5 qout_reg_2_5 qout_reg_0_10	qout_reg[9:0]_2
data vector	txd[7:0]	qout_reg[7:0]	qout_reg[7:0]_1	

B. Inferred pipeline stages of pcie

For the benchmark pcie, there are two pipeline stages, whose flow control vector and data vector are respectively shown in Table II.

One issue to be noticed that is the data vector at pipeline stage 1 is empty, while all registers in that stages are recognized as flow control vector. We inspect the encoder's source code and find that these registers are directly feed to output. So they can actually be uniquely determined by $\vec{o}$. This doesn't affect the correctness of the generated decoder, because the functionality of flow control vector never depend on the inferred flow control predicate.

C. Inferred pipeline stages of xgxs

For the benchmark xgxs, there are only 1 pipeline stage, whose flow control vector and data vector are respectively shown in Table III.

D. Inferred pipeline stages of t2ether

For the benchmark t2ether, there are four pipeline stages shown in Table IV. The control flow predicates are fairly complex, so we list them below instead of in Table IV. The input control flow predicate f is :

$$(tx\_enc\_ctrl\_sel[2]\ \&\ tx\_enc\_ctrl\_sel[3])|$$
$$(tx\_enc\_ctrl\_sel[2]\ \&\ !tx\_enc\_ctrl\_sel[3]\ \&\ !tx\_enc\_ctrl\_sel[0]\ \&$$
$$(!tx\_enc\_ctrl\_sel[2]\ \&\ tx\_enc\_ctrl\_sel[3])|$$
$$(!tx\_enc\_ctrl\_sel[2]\ \&\ !tx\_enc\_ctrl\_sel[3]\ \&\ tx\_enc\_ctrl\_sel[0])$$

The flow control predicate $valid(\vec{f^0})$ for the 0-th pipeline stage is :

$$(qout\_reg\_2\_4\ \&\ qout\_reg\_1\_4\ \&\ !qout\_reg\_0\_8)|$$
$$(!qout\_reg\_2\_4\ \&\ qout\_reg\_0\_8) \tag{23}$$

The flow control predicate $valid(\vec{f^1})$ for the 1-th pipeline stage is :

$$(qout\_reg\_2\_5\ \&\ qout\_reg\_1\_5\ \&\ qout\_reg\_0\_10\ \&\ !qout\_reg\_0\_$$
$$(qout\_reg\_2\_5\ \&\ qout\_reg\_1\_5\ \&\ !qout\_reg\_0\_10)|$$
$$(qout\_reg\_2\_5\ \&\ !qout\_reg\_1\_5\ \&\ !qout\_reg\_0\_10)|$$
$$(!qout\_reg\_2\_5\ \&\ qout\_reg\_0\_10\ \&\ qout\_reg\_0\_9)|$$
$$(!qout\_reg\_2\_5\ \&\ !qout\_reg\_0\_10)$$

The flow control predicates $valid(\vec{f^2})$ and $valid(\vec{f^3})$ for the last two pipeline stages are all $true$.

TABLE III
INFERRED PIPELINE STAGES OF XGXS

	input	pipeline stage 0
flow control vector	bad_code	bad_code_reg_reg
flow control predicate	!bad_code	!bad_code_reg_reg
data vector	encode_data_in[7:0] konstant	ip_data_latch_reg[2:0] plus34_latch_reg data_out_latch_reg[5:0] konstant_latch_reg kx_latch_reg minus34b_latch_reg

VII. RELATED PUBLICATIONS

The first complementary synthesis algorithm was proposed by [1]. 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 it is too slow in building the decoder.

Shen et al. [2] and Liu et al. [4] tackled the halting problem independently by searching for loops in the state sequence, while the runtime overhead problem was addressed in [3], [4] by Craig interpolant [10].

Shen et al. [3] automatically inferred an assertion for configuration pins, which can lead to the decoder's existence.

Qin et al. [8] proposed the first algorithm can handle encoder with flow control mechanism. But it can not handle pipeline stages.

Tu and Jiang [6] proposed a break-through algorithm that recover the encoder's input by considering its initial and reachable states.

VIII. CONCLUSIONS

This paper proposes the first complementary synthesis algorithm that can handle encoders with pipeline stages and flow control mechanism. Experimental result indicates that the proposed algorithm can always correctly generate pipelined decoder with flow control mechanism.

REFERENCES

[1] S. Shen, J. Zhang, Y. Qin, and S. Li, "Synthesizing complementary circuits automatically," in ICCAD '09, pp. 381–388.

[2] S. Shen, Y. Qin, L. Xiao, K. Wang, J. Zhang, and S. Li., "A halting algorithm to determine the existence of the decoder," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 10, pp. 30:1556–30:1563.

[3] S. Shen, Y. Qin, K. Wang, Z. Pang, J. Zhang, and S. Li., "Inferring assertion for complementary synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 8, pp. 31:1288–31:1292.

[4] H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang, "Towards completely automatic decoder synthesis," in ICCAD '11, pp. 389–395.

[5] H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang., "Automatic decoder synthesis: Methods and case studies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 9, pp. 31:1319–31:1331.

[6] K.-H. Tu and J.-H. R. Jiang, "Synthesis of feedback decoders for initialized encoders," in DAC '13, pp. 1–6.

[7] D. Abts and J. Kim, *High Performance Datacenter Networks*, 1st ed., ser. Synthesis Lectures on Computer Architecture. Morgan and Claypool, 2011, vol. 14, ch. 1.6, pp. 7–9.

[8] Y. Qin, S. Shen, Q. Wu, H. Dai, and Y. Jia., "Complementary synthesis for encoder with flow control mechanism," *accepted by ACM Transactions on Design Automation of Electronic Systems*.

[9] J. R. Jiang, H. Lin, and W. Hung, "Interpolating functions from large boolean relations," in ICCAD'09, pp. 779–784.

[10] W. Craig, "Linear reasoning: A new form of the herbrand-gentzen theorem," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 250–268, Sep. 1957.

[11] N. Eén and N. Sörensson, "An extensible sat-solver," in SAT'03., pp. 502–518.

[12] M. Jackson, R. Budruk, J. Winkles, and D. Anderson, *PCI Express Technology 3.0*. Mindshare Press, 2012.

[13] IEEE, "Ieee standard for ethernet section fourth," 2012. [Online]. Available: http://standards.ieee.org/getieee802/download/802.3-2012_section4.pdf