Complementary Synthesis for Pipelined Encoder

***Abstract*—Complementary synthesis automatically generates an encoder's decoder that recovers the encoder's inputs from its output. However, the generated decoders are non-pipelined and therefore much slower than their corresponding pipelined encoders. Thus, this paper proposes a novel algorithm to first identify the encoder's pipeline registers in each pipeline stage, and then build the decoder by characterizing the Boolean functions that recover each pipeline stage and the encoder's input with Craig interpolant. Experimental results on several complex encoders indicate that this algorithm can always correctly generate a pipelined decoders with significantly improved speed.**

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]–[9] 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, all existing complementary synthesis algorithms [1]–[9] ignored the encoder's internal pipeline stages $\vec{stg}^0$ in Fig. 1a), which is used to cut the encoder's datapath and boosts its frequency. So, as shown in Fig. 1b), the generated non-pipelined decoder directly recovers input $\vec{i}$ from output $\vec{o}$ with a huge logic block $C^0 * C^1$. While a proper and faster decoder, as shown in Fig. 1c), should cut this huge logic block $C^0 * C^1$ into two small pieces with the pipeline stage $\vec{stg}^0$, just like its encoder.

Thus, this paper proposes a novel algorithm to generate such pipelined decoder shown in Fig. 1c). It first identify the encoder's pipeline registers in each pipeline stage, and then build the decoder by characterizing the Boolean functions that recover each pipeline stage and the encoder's input with Craig interpolant. [10].

Experimental result indicates that this algorithm can always generate pipelined decoder with significantly improved timing.

This paper is organized as follows. Section II introduces the background material; Section III presents our algorithm framework. Section IV identifies each pipeline stages $\vec{stg}^j$, while Section V characterizes the decoder's Boolean functions that recover each $\vec{stg}^j$ and the input $\vec{i}$. 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 a vector that contains both v and all members of $\vec{v}$. $\vec{v} - v$ is a vector that contains all members of $\vec{v}$ except v. $\vec{a} \cup \vec{b}$ is a vector that contains all members of $\vec{a}$ and $\vec{b}$. $\vec{a} - \vec{b}$ is a vector that contains all members of $\vec{a}$ but no member of $\vec{b}$.

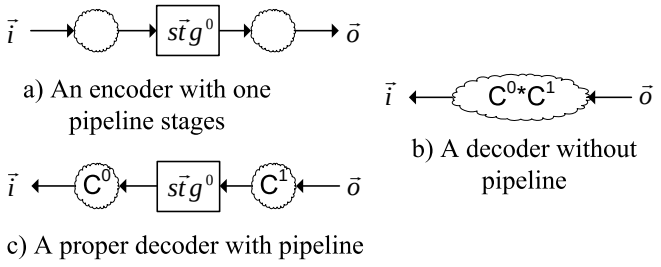


Fig. 1. The pipelined encoder and its decoders

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** [11] 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 $T : \vec{s} \times \vec{i} \to \vec{s} \times \vec{o}$ that computes the next state and output variable vector from the current state and input variable vector.

As shown in Fig. 2, as well as in the remainder of this paper, the state is represented as a gray round corner box, and the transition function T is represented as a white rectangle.

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 $< \vec{s}_n, \dots, \vec{s}_m >$ with

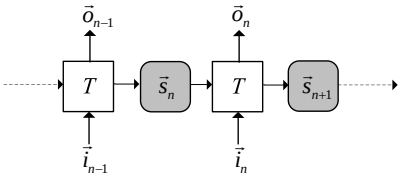


Fig. 2. Modeling the encoder with finite state machine

Algorithm 1: Determine whether i can be uniquely determined by a bounded sequence of $\vec{o}$

Input: The input variable $i \in \vec{i}$.
Output: the answer of whether i can be uniquely determined, and the maximal p, l and r reached.

```
1 p:= 0 ; l:= 0 ; r:= 0 ;
2 while true do
3     p++;  l++;  r++;
4     if Sound(i) then return (true, p, l, r) ;
5     else if Complete(i) then return (false, 0, 0, 0)
```

$\exists \vec{i}_j \vec{o}_j(\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$ 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 determines whether $i \in \vec{i}$ can be uniquely determined by a bounded sequence of $\vec{o}$

Shen et al. [3] proposed Algorithm 1 to determine whether an input variable $i \in \vec{i}$ can be uniquely determined by a bounded sequence of $\vec{o}$, by iteratively calling a sound and a complete approaches until they converge.

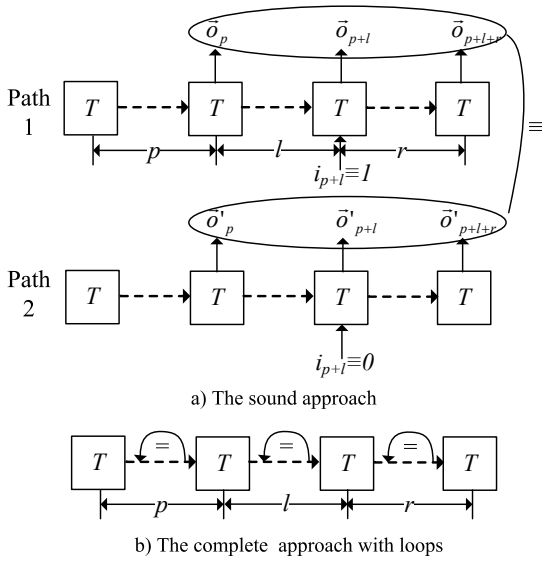
1) The sound approach in Fig. 3a) shows how to check whether an input variable $i \in \vec{i}$ can be uniquely determined by a bounded sequence of $\vec{o}$: if there exists p, l and r, such that for every output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, i_{p+l} cannot be different. This is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

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

This approach is sound because when $F_{PC}(p,l,r)$ is unsatisfiable for a particular p. l and r, $i \in \vec{i}$ can always be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$.

2) The complete approach in Fig. 3b) shows how to check whether an input variable $i \in \vec{i}$ can NOT be uniquely determined by a bounded sequence of $\vec{o}$: It is similar to Fig. 3a) 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.

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



a) The sound approach

b) The complete approach with loops

Fig. 3. The sound and complete approximative approaches

This approach is complete because when $F_{LN}(p,l,r)$ 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. This means $i \in \vec{i}$ can never be uniquely determined by a bounded sequence of $\vec{o}$.

Thus, Algorithm 1 is a halting one because if there indeed exists such p, l and r that make $F_{PC}(p,l,r)$ unsatisfiable, then they can eventually be found in Line 4; Otherwise, p, l and r will eventually be larger than the length of the encoder's longest non-loop path, which makes $F_{LN}(p,l,r)$ satisfiable. Both cases will terminate the loop.

III. Algorithm Framework

As shown in Fig. 4, we assume that the encoder has n pipeline stages $\vec{stg}^j$, where $0 \leq j \leq n-1$. Obviously the set of all state variables in pipeline stages are a subset of $\vec{s}$, that is:

$$\vec{s} \supseteq \bigcup_{0 \leq j \leq n-1} \vec{stg}_j \quad (3)$$

If we take the combinational logic block C^j as a function, then this encoder can be represented by the following equations.

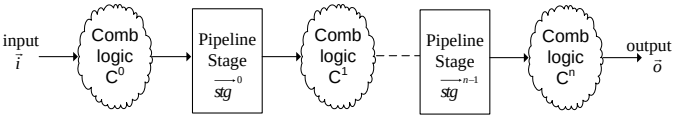


Fig. 4. The encoder's general structure with pipeline

$$\begin{array}{rcl} \vec{stg}^{\,0} & := & C^0(\vec{i}) \\ \vec{stg}^{\,j} & := & C^j(\vec{stg}^{\,j-1}) \quad 1 \le j \le n-1 \\ \vec{o} & := & C^n(\vec{stg}^{\,n-1}) \end{array} \tag{4}$$

So, as shown in Fig. **??**, the transition function T can be seen as a parallel composition $C^0|C^1 \ldots |C^{n-1}|C^n$.

With this encoder model, our algorithm framework is:

1) In Section IV, identifying all state variables in each pipeline stage $\vec{stg}^{\,j}$.
2) In Section V, characterizing $(C^j)^{-1}$, the inverse functions of C^j shown in Fig. 4 , to recover each pipeline stages $\vec{stg}^{\,j}$ and input vector $\vec{i}$ from the next pipeline stage $\vec{stg}^{\,j+1}$ or the output vector $\vec{o}$.

IV. Identifying pipeline stages

A. Finding out p,l and r

We first call Algorithm 1 for each $i \in \vec{i}$. If there exists an i that make Algorithm 1 to return $(false, 0, 0, 0)$, then their must be a bug in the encoder that prevents i from being recovered from $\vec{o}$.

Otherwise, assume that Algorithm 1 returns p_i, l_i and r_i for $i \in \vec{i}$, we set the final p, l and r to the largest ones returned from Algorithm 1:

$$\begin{array}{r} p := max_{i \in \vec{i}}\{p_i\} \\ l := max_{i \in \vec{i}}\{l_i\} \\ r := max_{i \in \vec{i}}\{r_i\} \end{array} \tag{5}$$

Obviously, for every $i \in \vec{i}$, the clause set of $F_{PC}(p,l,r)$ is a super set of $F_{PC}(p_i,l_i,r_i)$. So $F_{PC}(p,l,r)$ is also unsatisfiable for $i \in \vec{i}$. This means all $i \in \vec{i}$ can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$.

B. Minimizing r

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.

With this notation, we can substitute Equation (4) into Fig. 3a). This leads to the following observations, which is also shown intuitively in Fig. 5:

1) $\vec{i}_{p+l}$, the value of input $\vec{i}$ at step $p+l$, can be uniquely determined by $\vec{stg}^{\,0}_{p+l}$, the value of the 1st pipeline stage $\vec{stg}^{\,0}$ at the same step $p+l$;
2) $\vec{stg}^{\,0}_{p+l}$, the value of the 1st pipeline stage $\vec{stg}^{\,0}$ at step $p+l$, can be uniquely determine by $\vec{stg}^{\,1}_{p+l+1}$, the value of the 2nd pipeline stage $\vec{stg}^{\,1}$ at the next step $p+l+1$.
3) ...
4) $\vec{stg}^{\,j}_{p+l+j}$, the value of $\vec{stg}^{\,j}$ at step $p+l+j$, can be uniquely determine by $\vec{stg}^{\,j+1}_{p+l+j+1}$, the value of next pipeline stage $\vec{stg}^{\,j+1}$ at the next step $p+l+j+1$.

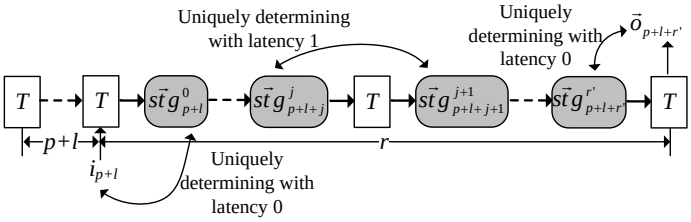


Fig. 5. Unrolled transition function with pipeline stages

5) ...
6) $\vec{stg}^{\,r'}_{p+l+r'}$, the value of the last pipeline stage $\vec{stg}^{\,r'}$ at step $p+l+r'$, can be uniquely determined by $\vec{o}_{p+l+r'}$, the value of output $\vec{o}$ at the same step $p+l+r'$.

With these observations, it is obvious that $\vec{i}_{p+l}$ can be uniquely determined by $\vec{o}_{p+l+r'}$. By comparing this conclusion to Fig. 3 and Algorithm 1, we can be sure that $r' \le r$. To find out r', we define the following new formula:

$$F'_{PC}(p,l,r') := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{o}_{p+l+r'} \equiv \vec{o'}_{p+l+r'} \\ \wedge & i_p \equiv 1 \wedge i'_p \equiv 0 \end{array} \right\} \tag{6}$$

Compared to Equation (1), this new formula tries to use only $\vec{o}_{p+l+r'}$ instead of sequence $< \vec{o}_{p+l}, \ldots, \vec{o}_{p+l+r} >$ to uniquely determine $\vec{i}_{p+l}$.

Algorithm 2 is proposed to use this new formula to find out r'. In Line 2, if $F'_{PC}(p,l,r'-1)$ is satisfiable for $i \in \vec{i}$, then r' is the last one that makes $F'_{PC}(p,l,r')$ 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, with p and l defined in Equation (5) and r' founded by Algorithm 2, we need to generalize F'_{PC} in Equation (6) to the following new formula that can determine whether v_j, the value of variable v at step j can be uniquely determined by $\vec{w}_k$, the value of a vector $\vec{w}$ at step k. Now v_j and $\vec{w}_k$ can be either input, state or output variables (vectors).

Algorithm 2: Minimizing r

```
1 for r' := r → 0 do
2     if r' ≡ 0 or F'_PC(p,l,r'-1) is satisfiable for some
      i ∈ i⃗ then
3         break
4 return r'
```

$$F''_{PC}(p,l,r',v_j,\vec{w}_k) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{w}_k \equiv \vec{w'}_k \\ \wedge & v_j \equiv 1 \wedge v'_j \equiv 0 \end{array} \right\} \quad (7)$$

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

According to Fig. 5, for $0 \le j \le r'$, the state variable set in the j-th pipeline stage is exactly the set of state variables $s \in \vec{s}$ that can be uniquely determined at the $p+l+j$-th step by $\vec{o}$ at the $p+l+r'$-th step. It can be formally defined as:

$$\vec{stg}^{\,j} := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,l,r',s_{p+l+j},\vec{o}_{p+l+r'}) \\ is\ unsatisfiable \end{array} \right\} \quad (8)$$

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

A. Characterizing the Boolean functions recovering the last pipeline stage $\vec{stg}^{\,r'}$

According to Equation (8), every state variable $s \in \vec{stg}^{\,r'}$ at the $p+l+r'$-th step can be uniquely determined by $\vec{o}_{p+l+r'}$. That is, $F''_{PC}(p,l,r',s_{p+l+r'},\vec{o}_{p+l+r'})$ is unsatisfiable and can be partitioned into :

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+l+r'} \equiv 1 \end{array} \right\} \quad (9)$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+l+r'} \equiv \vec{o'}_{p+l+r'} \\ \wedge & s'_{p+l+r'} \equiv 0 \end{array} \right\} \quad (10)$$

According to [10], a Craig interpolant ϕ_I of ϕ_A with respect to ϕ_B can be constructed, which refer only to $\vec{o}_{p+l+r'}$, the common variables of ϕ_A and ϕ_B. And ϕ_I covers all the valuations of $\vec{o}_{p+l+r'}$ that can make $s_{p+l+r'} \equiv 1$. At the same time, $\phi_I \wedge \phi_B$ is unsatisfiable, which means ϕ_I covers nothing that can make $s_{p+l+r'} \equiv 0$.

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

B. Characterizing the Boolean functions recovering other pipeline stages $\vec{stg}^{\,j}$

According to Fig. 5, $\vec{stg}^{\,j}_{p+l+j}$ can be uniquely determined by $\vec{stg}^{\,j+1}_{p+l+j+1}$. So for every $s \in \vec{stg}^{\,j}$, we can partition the unsatisfiable formula $F''_{PC}(p,l,r',s_{p+l+j},\vec{stg}^{\,j+1}_{p+l+j+1})$ into:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+l+j} \equiv 1 \end{array} \right\} \quad (11)$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{stg}^{\,j+1}_{p+l+j+1} \equiv \vec{stg'}^{\,j+1}_{p+l+j+1} \\ \wedge & s'_{p+l+j} \equiv 0 \end{array} \right\} \quad (12)$$

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 $s \in \vec{stg}^{\,j}$ from $\vec{stg}^{\,j+1}$.

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

According to Fig. 5, $\vec{i}_{p+l}$ can be uniquely determined by the 1st pipeline stage $\vec{stg}^{\,0}_{p+l}$. So for every input $i \in \vec{i}$, $F''_{PC}(p,l,r',i_{p+l},\vec{stg}^{\,0}_{p+l})$ is unsatisfiable and can be partitioned into :

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & i_{p+l} \equiv 1 \end{array} \right\} \quad (13)$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r'}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{stg}^{\,0}_{p+l} \equiv \vec{stg'}^{\,0}_{p+l} \\ \wedge & i'_{p+l} \equiv 0 \end{array} \right\} \quad (14)$$

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 $i \in \vec{i}$ 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 [12]. 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

We use the benchmarks from Qin et al. [9], whose experimental results are shown in Table I. pcie is a PCI Express [13] encoder, while xgxs and t2eth are two Ethernet [14] encoders.

The 2nd and 3rd column of Table I 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.

TABLE I
Benchmarks and experimental results

Names	The encoders			decoder generated by [3]			decoder generated by this paper		
	# in/out	# reg	area	run time	delay (ns)	area	run time	delay (ns)	area
pcie	10/11	23	326	0.37	7.20	624	8.08	5.89	652
xgxs	10/10	16	453	0.21	7.02	540	4.25	5.93	829
t2eth	14/14	49	2252	12.7	6.54	434	430.4	6.12	877

TABLE II
INFERRED PIPELINE STAGES OF PCIE

	input	pipeline stage 0	pipeline stage 1
$\vec{f}$ or $\vec{f^j}$	CNTL_TXEnable_P0	InputDataEnable_P0	OutputData_P0[9:0] OutputElecIdle_P0
$\vec{d}$ or $\vec{d^j}$	TXDATA[7:0] TXDATAK	InputData_P0[7:0] InputDataK_P0	

The 5th to 7th columns show respectively the run time of [3]'s algorithm to generate the decoder without pipeline, and the delay and area of the generated decoder. While the 8th to 10th 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 6th and the 9th column indicates that the decoders' delay have been significantly improved.

B. Inferred pipeline stages of pcie

The benchmark pcie has two pipeline stages, whose flow control vector and data vector are respectively shown in Table II. The inferred $valid(\vec{f})$ is $CNTL\_TXEnable\_P0$.

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 fed 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

The benchmark xgxs has only 1 pipeline stage, whose flow control vector and data vector are respectively shown in Table III. The inferred $valid(\vec{f})$ is $!bad\_code$.

D. Inferred pipeline stages of t2ether

The benchmark t2ether has four pipeline stages shown in Table IV. The inferred $valid(\vec{f})$ is:

$$\begin{aligned}&(tx\_enc\_ctrl\_sel[2]\ \&\ tx\_enc\_ctrl\_sel[3]\ )\ |\\&(tx\_enc\_ctrl\_sel[2]\ \&\ !tx\_enc\_ctrl\_sel[3]\ \&\\&!tx\_enc\_ctrl\_sel[0]\ \&\ tx\_enc\_ctrl\_sel[1]\ )\ |\\&(!tx\_enc\_ctrl\_sel[2]\ \&\ tx\_enc\_ctrl\_sel[3]\ )\ |\\&(!tx\_enc\_ctrl\_sel[2]\ \&\ !tx\_enc\_ctrl\_sel[3]\ \&\\&\ tx\_enc\_ctrl\_sel[0])\end{aligned} \tag{15}$$

TABLE III
INFERRED PIPELINE STAGES OF XGXS

	input	pipeline stage 0
$\vec{f}$ or $\vec{f^j}$	bad_code	bad_code_reg
$\vec{d}$ or $\vec{d^j}$	encode_data_in[7:0] konstant	ip_data_latch[2:0] plus34_latch data_out_latch[5:0] konstant_latch kx_latch minus34b_latch

TABLE IV
INFERRED PIPELINE STAGES OF T2ETHER

	input	pipeline stage 0	pipeline stage 1
$\vec{f}$ or $\vec{f^j}$	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
$\vec{d}$ or $\vec{d^j}$	txd[7:0]	qout_reg[7:0]	qout_reg[7:0]_1
	pipeline stage 2	pipeline stage 3	
$\vec{f}$ or $\vec{f^j}$	qout_reg[9:0]_2	qout_reg[7:1]_3 qout_reg_8_1 qout_reg_9_1 qout_reg_3_4 qout_reg_0_4 qout_reg_3_5	qout_reg_0_7 sync1_reg1 sync1_reg Q_reg1 Q_reg
$\vec{d}$ or $\vec{d^j}$			

VII. RELATED PUBLICATIONS

Shen et al. [1] proposed the first complementary synthesis algorithm. It checks the decoder's existence by iteratively increasing the length of unrolled transition function sequence, and generates the decoder's Boolean functions 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. [3] and Liu et al. [6] tackled the halting problem independently by searching for loops in the state sequence.

Shen et al. [2] speedups building decoder by inferring hidden XOR gates that makes the solution space irregular, while Shen et al. [5] and Liu et al. [6] finds out a much better solution based on Craig interpolant [11].

Shen et al. [4] automatically inferred an assertion for configuration pins, which can lead to the decoder's existence. Shen et al. [5] further discovers that there may exists multiple decoders for a single assertion inferred by [4]. It iteratively find outs all these decoders and their corresponding refined assertions with functional dependency [15].

Tu and Jiang [8] proposed a break-through algorithm that recover the encoder's input by considering its initial and reachable states.

Qin et al. [9] proposed the first algorithm that can handle encoder with flow control mechanism. But it can not handle pipeline stages.

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 *Proceedings of the 2009 International Conference on Computer-Aided Design*, ser. ICCAD '09. San Jose, CA, USA: IEEE Press, 2009, pp. 381–388. [Online]. Available: http://dx.doi.org/10.1145/1687399.1687472

[2] S. Shen, Y. Qin, K. Wang, L. Xiao, J. Zhang, and S. Li., "Synthesizing complementary circuits automatically," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 8, pp. 29:1191–29:1202, August 2010. [Online]. Available: http://doi.acm.org/10.1109/TCAD.2010.2049152

[3] 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, October 2011. [Online]. Available: http://doi.acm.org/10.1109/TCAD.2011.2159792

[4] S. Shen, Y. Qin, and J. Zhang, "Inferring assertion for complementary synthesis," in *Proceedings of the 2011 International Conference on Computer-Aided Design*, ser. ICCAD '11. San Jose, CA, USA: IEEE Press, 2011, pp. 404–411. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2011.6105361

[5] 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, August 2012. [Online]. Available: http://doi.acm.org/10.1109/TCAD.2012.2190735

[6] H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang, "Towards completely automatic decoder synthesis," in *Proceedings of the 2011 International Conference on Computer-Aided Design, ICCAD 2011*, ser. ICCAD '11. San Jose, CA, USA: IEEE Press, 2011, pp. 389–395. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2011.6105359

[7] 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, September 2012. [Online]. Available: http://doi.acm.org/10.1109/TCAD.2012.2191288

[8] K.-H. Tu and J.-H. R. Jiang, "Synthesis of feedback decoders for initialized encoders," in *Proceedings of the 50th Annual Design Automation Conference, DAC 2013*, ser. DAC '13. Austin, TX, USA: ACM Press, 2013, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1145/2463209.2488794

[9] 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*.

[10] J. R. Jiang, H. Lin, and W. Hung, "Interpolating functions from large boolean relations," in *2009 International Conference on Computer-Aided Design, ICCAD 2009, San Jose, CA, USA, November 2-5, 2009*, J. S. Roychowdhury, Ed. ACM, 2009, pp. 779–784. [Online]. Available: http://doi.acm.org/10.1145/1687399.1687544

[11] 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.

[12] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24605-3_37

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

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

[15] C. Lee, J. R. Jiang, C. Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental SAT solving," in *2007 International Conference on Computer-Aided Design, ICCAD 2007, San Jose, CA, USA, November 5-8, 2007*, G. G. E. Gielen, Ed. IEEE Computer Society, 2007, pp. 227–233. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2007.4397270