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 is non-pipelined and therefore very slow. On the other hand, many encoders from industrial projects have pipelined structure that can be exploited to resolve this problems.

Thus, we propose a novel algorithm to first find out the encoder's pipeline registers in each pipeline stage, and then characterize all Boolean functions that recover each of these pipeline registers from the registers in the next pipeline stage, and finally characterize the Boolean functions that recover the encoder's input variables from the first pipeline stage.

Experimental results on several complex encoders indicate that this algorithm can always correctly generate a pipelined decoders with significantly improved speed.

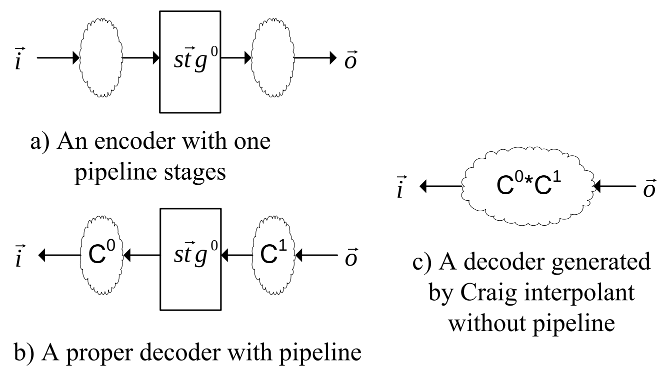

Fig. 1. The pipelined encoder and its decoders

I. 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 $\vec{o}$, while the decoder recovers $\vec{i}$ from $\vec{o}$. Complementary synthesis [11, 9, 10, 8, 5, 6, 12] eases this job by automatically generating a decoder from an encoder, with the assumption that $\vec{i}$ can always be uniquely determined by a bounded sequence of $\vec{o}$. Thus, the decoder's Boolean function can be characterized with the algorithm proposed by Jiang et al. [4] based on Craig interpolant [1].

By studying the structure of many encoders from industrial projects, we find that most of them have a pipeline structure that can be exploited to significantly improve the quality of the generated decoders.

For example, one simple encoder is shown in Figure 1a). It has a pipeline stage $\vec{stg}^0$ with some registers. The inputs variables $\vec{i}$ are used to compute $\vec{stg}^0$, while $\vec{stg}^0$ are used to compute the output variables $\vec{o}$. According to this structure, $\vec{stg}^0$ can be uniquely determined by $\vec{o}$, while $\vec{i}$ can be uniquely determined by $\vec{stg}^0$.

So, a properly designed decoder, often written by human engineers, should be like the one shown in Figure 1b), which recovers $\vec{stg}^0$ from $\vec{o}$ with combinational logic C^1, and further recovers $\vec{i}$ from $\vec{stg}^0$ with combinational logic C^0. In such a decoder, the critical path is cut by registers $\vec{stg}^0$, which improves the circuit speed.

However, all complementary synthesis algorithms [9, 10, 8, 5, 6, 12] generate the decoders with Jiang's algorithm [4] based on Craig interpolant [1]. As shown in Figure 1c), these decoders recover $\vec{i}$ directly from $\vec{o}$ with a large combinational logic $C^0 * C^1$. Such a decoder is unnecessarily slow because there are no registers to cut its critical path.

To generate a proper decoder like Figure 1b) in complementary synthesis, we propose a novel algorithm to first find out the encoder's pipeline registers in each pipeline stage, and then characterize all Boolean functions that recover each of these pipeline registers from the next pipeline stage or output variables $\vec{o}$, and finally characterize the Boolean functions that recover the encoder's input variables from the first pipeline stage.

Experimental results on several complex encoders, such as PCI Express [7] and Ethernet [3], indicate that this algorithm can always correctly generate a pipelined decoder with significantly improved speed.

The remainder of this paper is organized as follows. Section II introduces the background material; Section III infers the pipeline structure, while Section IV characterizes the Boolean functions that recover the input variables and pipeline registers; Sections V and VI present the experimental results and related works; Finally, Section VII sums up the conclusion.

II. Preliminaries

A. Propositional satisfiability

The Boolean set is $\mathbb{B} = \{0, 1\}$. A variables vector is $\vec{v} = (v, \dots)$. The number of variables in $\vec{v}$ is $|\vec{v}|$. If a variable v is a member of $\vec{v}$, then we say $v \in \vec{v}$; otherwise

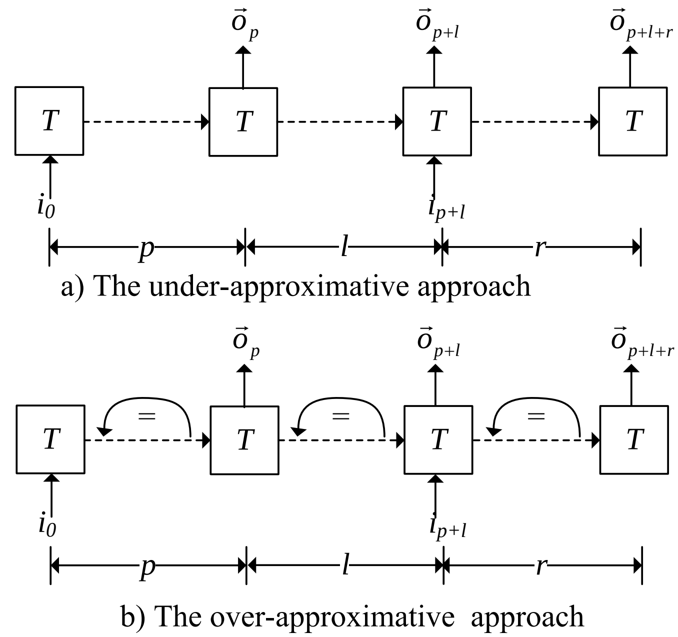

Fig. 2. The under and over-approximative approaches

$v \notin \vec{v}$. $v \cup \vec{v}$ is the vector containing both v and all members of $\vec{v}$. $\vec{v}/\vec{w}$ is the vector containing all members of $\vec{v}$ but no member of $\vec{w}$. $\vec{a} \cup \vec{b}$ is the vector with all members of $\vec{a}$ and $\vec{b}$. The set of truth valuations of $\vec{v}$ is $[\![\vec{v}]\!]$, for instance, $[\![(v_1, v_2)]\!] = \{(0,0),(0,1),(1,0),(1,1)\}$.

The propositional satisfiability problem(SAT) for a formula F over a variable set V is to find a satisfying assignment $A : V \rightarrow \mathbb{B}$, so that F can be evaluated to 1. If A exists, then F is satisfiable; otherwise, it is 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 $\phi_A \Rightarrow \phi_I$ and $\phi_I \wedge \phi_B$ is unsatisfiable. ϕ_I is the **Craig interpolant** [1] of ϕ_A with respect to ϕ_B.

B. Finite state machine

The encoder is modeled by a finite state machine(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 $T : [\![\vec{s}]\!] \times [\![\vec{i}]\!] \rightarrow [\![\vec{s}]\!] \times [\![\vec{o}]\!]$ that computes the next state and output variable vector from the current state and input variable vector.

The behavior of FSM M can be reasoned by unrolling transition function for multiple steps. The state variable $s \in \vec{s}$, input variable $i \in \vec{i}$ and output variable $o \in \vec{o}$ at the n-th step 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 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 to determine if an input variable can be uniquely determined by a bounded sequence of output variable vector

The first halting algorithm [10] iteratively unrolls the transition function. For each iteration, it uses an under-approximative and an over-approximative approaches presented respectively in C.1 and C.2 to determine the answer. We will show in C.3 that these two approaches will eventually converge to a conclusive answer.

C.1 The under-approximative approach

As shown in Figure 2a), on the unrolled transition functions, an input variable $i \in \vec{i}$ can be uniquely determined, if there exist three integers p, l and r, such that for any particular valuation of the output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, i_{p+l} cannot be 0 and 1 at the same time. This is equal to the unsatisfiability of $F_{PC}(p,l,r)$ in Equation (1).

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

Here, p is the length of the prefix state transition sequence. l and r are the lengths of the two output sequences $< \vec{o}_{p+1}, \ldots, \vec{o}_{p+l} >$ and $< \vec{o}_{p+l+1}, \ldots, \vec{o}_{p+l+r} >$ used to determine i_{p+l}. Line 2 of Equation (1) corresponds to the path in Figure 2a), while Line 3 is a copy of it. These two paths are of the same length. Line 4 forces these two paths' output sequences to be the same, while Line 5 forces their i_{p+l} to be different.

According to Equation (1), for $p' \geq p$, $l' \geq l$ and $r' \geq r$, the clause set of $F_{PC}(p', l', r')$ is a super set of $F_{PC}(p,l,r)$. So, the bounded proof of $F_{PC}(p,l,r)$'s unsatisfiability can be generalized to unbounded cases.

Proposition 1 *If $F_{PC}(p,l,r)$ is unsatisfiable, then i_{p+l} can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for all larger p, l and r.*

C.2 The over-approximative approach

If $F_{PC}(p,l,r)$ presented in the last subsection is satisfiable, there are two possibilities:

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

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

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

```
1 p:= 0; l:= 0; r:= 0;
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
```

If it is the 1st case, then by iteratively increasing p, l and r, $F_{PC}(p,l,r)$ will eventually become unsatisfiable. But if it is the 2nd case, this method will never terminate.

So, to obtain a halting algorithm, we need to distinguish these two cases. One such solution is shown in Figure 2b), which is similar to Figure 2a) but with three additional constraints used 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 new constraints used to detect loops.

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

When $F_{LN}(p,l,r)$ is satisfiable, then i_{p+l} can't be uniquely determined by $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$. More importantly, by unrolling these three loops, we can generalize the satisfiability of $F_{LN}(p,l,r)$ to all larger p, l and r. This means:

Proposition 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 all larger p, l and r.*

Please refer to [10] for more detail of this.

C.3 The full algorithm

With Propositions 1 and 2, we can generalize their bounded proof to unbounded cases. This leads to the halting Algorithm 1 that search for p, l and r that enable an input variable i_{p+l} to be uniquely determined by the output sequence $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$:

1. On the one hand, if there exists such p, l and r, then let $p' := max(p,l,r)$, $l' := max(p,l,r)$ and $r' := max(p,l,r)$. From Propositions 1, we know that $F_{PC}(p',l',r')$ is unsatisfiable. So eventually $F_{PC}(p,l,r)$ will become unsatisfiable in Line 4;
2. On the other hand, if there doesn't exist such p, l and r, then eventually p, l and r will be larger than the encoder's 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 [10] for more detail of this algorithm's correctness and termination proof.

III. Inferring the encoder's pipeline structure

A. A general model for the encoder

As shown in Figure 3, we assume that the the encoder have n pipeline stages. If we take the combinational logic block C^j as a function, then this encoder can be represented by the following equations.

$$\begin{array}{rcll} \vec{stg}^{\,0} & := & C^0(\vec{i}) & \\ \vec{stg}^{\,j} & := & C^j(\vec{stg}^{\,j-1}) & 1 \leq j \leq n-1 \\ \vec{o} & := & C^n(\vec{stg}^{\,n-1}) & \end{array} \quad (3)$$

Thus, each C^j can be seen as a small encoder that computes $\vec{stg}^{\,j}$ or $\vec{o}$ from $\vec{stg}^{\,j-1}$ or $\vec{i}$.

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

B. Inferring p, l and r

Before inferring the pipeline stages, we first apply Algorithm 1 to infer the value of p, l and r that can make the output sequence $<o_p,\ldots,o_{p+l+r}>$ uniquely determine all $i_{p+l} \in \vec{i}_{p+l}$.

As there are more than one $i \in \vec{i}$, we need to apply Algorithm 1 for each $i \in \vec{i}$ to get the p, l and r for each of them.

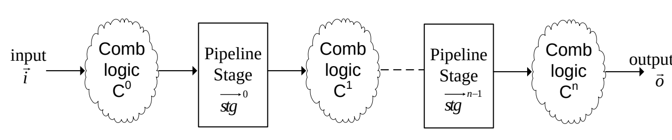

Fig. 3. A general structure of the encoder

Algorithm 2: *RemoveRedundancy*(p, l, 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'
```

And then we set the final p,l and r to be the maximal p, l and r of all $i \in \vec{i}$ respectively. According to Equation 1, these values of p, l and r can indeed make the output sequence $< o_p, \ldots, o_{p+l+r} >$ uniquely determine all $i_{p+l} \in \vec{i}_{p+l}$.

C. Minimizing r and l

As Algorithm 1 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 2

In Line 2, when $F_{PC}(p, l, r'-1)$ is satisfiable, then r' is the last one that makes it unsatisfiable, we return it directly. On the other hand, when $r' \equiv 0$, $F_{PC}(p, l, 0)$ must have been tested in the last iteration, and the result must be unsatisfiable. In this case we return 0.

Now, we have a minimized r from Algorithm 2, 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 4, 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 the following equation.

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

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

D. Inferring pipeline stages

Now, with the inferred p and r, we need to generalize F'_{PC} in Equation (4) to the following new formula that

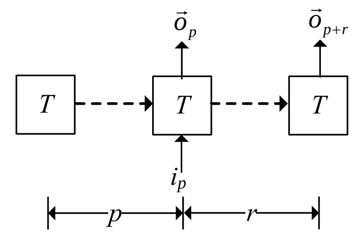

Fig. 4. Recovering input with reduced output sequence

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.

$$F''_{PC}(p, r, v, j, \vec{w}, k) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{w}_k \equiv \vec{w'}_k \\ \wedge & v_j \equiv 1 \wedge v'_j \equiv 0 \end{array} \right\} \quad (5)$$

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

The last pipeline stage $\vec{stg}^{n-1}$ is exactly the set of registers $s \in \vec{s}$ that can be uniquely determined at the $p+r$-th step by $\vec{o}$. It can be formally defined as:

$$\vec{stg}^{n-1} := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p, r, s, p+r, \vec{o}, p+r) \\ is\ unsatisfiable \end{array} \right\} \quad (6)$$

Similarly, for $0 \le j \le n-2$, $\vec{stg}^j$ at $j-((n-2)-(p+r-1))$-th step can be uniquely determined by $\vec{stg}^{j+1}$ at $j-((n-2)-(p+r-1))+1$-th step. So we can recursively defined $\vec{stg}^j$ as :

$$\begin{array}{ccc} S & := & \vec{s}/\bigcup_{j<k\le n-2} \vec{stg}^k \\ D & := & (n-2)-(p+r-1) \end{array} \quad (7)$$

$$\vec{stg}^j := \left\{ s \in S \mid \begin{array}{c} F''_{PC}(p, r, s, j-D, \vec{stg}^{j+1}, j-D+1) \\ is\ unsatisfiable \end{array} \right\} \quad (8)$$

With Equation (6) and (8), all the pipeline stages can now be inferred.

E. Inferring the pipeline stage that uniquely determines input vector

According to Figure 3, $\vec{stg}^0$ defined in Equation (8) is exactly the pipeline stage that uniquely determined the input vector $\vec{i}$.

But in real encoders, this may not be the case. So we need to search for the smallest j from 0 to $n-1$ that can make $\vec{i}$ to be uniquely determined by $\vec{stg}^j$, that is, the smallest j that can make $F''_{PC}(p, r, i, p, \vec{stg}^j, j-D)$ unsatisfiable for all $i \in \vec{i}$, with D defined in Equation (7).

IV. Characterizing the Boolean function of input variables and pipeline registers

A. Characterizing the Boolean function of the last pipeline stage

According to Equation (6), every registers $s \in \vec{stg}^{n-1}$ 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 :

$$\phi_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & s_{p+r} \equiv 1 \end{array} \right\} \tag{9}$$

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

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 [4], 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 valuation 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 $s \in \vec{stg}^{n-1}$ from $\vec{o}$.

B. Characterizing the Boolean function of the other pipeline stages

Similar to last subsection, we can partition the unsatisfiable formula $F''_{PC}(p, r, s, j-D, \vec{stg}^{j+1}, j-D+1)$ in Equation (8) into the following two equations:

$$\phi_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & s_{j-D} \equiv 1 \end{array} \right\} \tag{11}$$

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

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 function of the encoder's input variables

According to Subsection III.E, we have found the smallest j that can make $F''_{PC}(p, r, i, p, \vec{stg}^j, j-D)$ unsatisfiable for all $in \in \vec{i}$, with D defined in Equation (7). $F''_{PC}(p, r, i, p, \vec{stg}^j, j-D)$ is unsatisfiable and can be partitioned into :

$$\phi_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & i_p \equiv 1 \end{array} \right\} \tag{13}$$

$$\phi_B := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^j_{j-D} \equiv \vec{stg'}^j_{j-D} \\ \wedge & i'_p \equiv 0 \end{array} \right\} \tag{14}$$

Similar to last subsection, 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 $\vec{stg}^j$.

V. Experimental Results

We have implemented these algorithms in OCaml language, and solved the generated CNF formulas with MiniSat 1.14 [2]. All experiments have been run on a server with 16 Intel Xeon E5648 processors at 2.67GHz, 192GB memory, and CentOS 5.4 Linux.

Table I shows the benchmarks used in this paper. The 2nd to 3rd column show respectively the number of inputs, outputs and registers of each benchmark. The area 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 6-th to 8-th columns show respectively the run time of [10]'s algorithm to generate the decoder without pipeline, and the delay and area of the generated decoder. While the 9-th to 11-th 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. The last column shows the number of registers in each pipeline stage.

Comparing the 7rd and the 10-th column indicates that the decoders' delay have been significantly improved. And the the last column shows that there actually exist very deep pipeline, especially the t2eth with 4 pipeline stages.

One thing that is a little bit surprise is, the two largest benchmarks scrambler and xfi do not have pipeline stages

TABLE I
Benchmarks and experimental results

Names	The encoders				decoder generated by [10]			decoder generated by this paper			
	# in/out	# reg	area	Description of Encoders	run time	delay (ns)	area	run time	delay (ns)	area	# reg
pcie	10/11	23	326	PCIE 2.0 [7]	0.37	7.20	624	3.57	5.89	652	9/12
xgxs	10/10	16	453	Ethernet clause 48 [3]	0.21	7.02	540	1.57	5.93	829	13
t2eth	14/14	49	2252	Ethernet clause 36 [3]	12.7	6.54	434	47.2	6.12	877	8/8/10/20
scrambler	64/64	58	1034	inserting 01 flipping	no pipeline stages found						
xfi	72/66	72	7772	Ethernet clause 49 [3]							

inside. We study their code and confirm that they actually don't have such pipeline stages. Their area are so large because they use much wider datapaths with 64 to 72 bits.

VI. RELATED PUBLICATIONS

The first complementary synthesis algorithm was proposed by Shen et al.[11]. 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. But this algorithm may not halt and is too slow in building the decoder.

The halting problem was independently tackled in Shen et al.[10] and Liu at al.[5] by searching for loops in the state sequence, while the runtime overhead problem was addressed in [8, 5] by interpolant [4].

Shen et al. [8] inferred an assertion for configuration pins that can lead to the decoder's existence. Tu et al.[12] proposed a breakthrough algorithm that use the encoder's infinite history to generate the decoder's output.

VII. Conclusions

This paper proposes the first complementary synthesis algorithm that can handle pipelined encoders. Experimental results on several complex encoders indicate that this algorithm can always correctly infer the encoder's pipeline structure, and generate a pipelined decoder that is significantly faster.

References

[1] W. Craig. Linear reasoning: A new form of the herbrand-gentzen theorem. *The Journal of Symbolic Logic*, 22(3):250–268, Sept. 1957.

[2] N. Eén and N. Sörensson. An extensible sat-solver. In SAT 2003, pages 502–518, 2003.

[3] IEEE. Ieee standard for ethernet section 4, 2012.

[4] W.-L. H. Jie-Hong Roland Jiang, Hsuan-Po Lin. Interpolating functions from large boolean relations. In ICCAD '09, pages 779–784, 2009.

[5] H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang. Towards completely automatic decoder synthesis. In ICCAD '11, pages 389–395, 2011.

[6] H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang. Automatic decoder synthesis: Methods and case studies. *IEEE Tran. on CAD of IC and Sys.*, 31(9):31:1319–31:1331, September 2012.

[7] PCI-SIG. Pci express base 2.1 specification, 2009.

[8] S. Shen, Y. Qin, K. Wang, Z. Pang, J. Zhang, and S. Li. Inferring assertion for complementary synthesis. *IEEE Tran. on CAD of IC and Sys.*, 31(8):31:1288–31:1292, August 2012.

[9] S. Shen, Y. Qin, K. Wang, L. Xiao, J. Zhang, and S. Li. Synthesizing complementary circuits automatically. *IEEE Tran. on CAD of IC and Sys.*, 29(8):29:1191–29:1202, August 2010.

[10] S. Shen, Y. Qin, L. Xiao, K. Wang, J. Zhang, and S. Li. A halting algorithm to determine the existence of the decoder. *IEEE Tran. on CAD of IC and Sys.*, 30(10):30:1556–30:1563, October 2011.

[11] S. Shen, J. Zhang, Y. Qin, and S. Li. Synthesizing complementary circuits automatically. In ICCAD '09, pages 381–388, 2009.

[12] K.-H. Tu and J.-H. R. Jiang. Synthesis of feedback decoders for initialized encoders. In DAC '13, pages 1–6, 2013.