第一章　面向流水线的对偶综合

1.1　引言

在通讯和多媒体芯片设计项目中，最困难的一个工作是针对特定的协议，如以太网 [1] 和 PCI Express [2]，设计相应的编码器和解码器. 其中编码器负责将输入 $\vec{i}$ 映射至输出 $\vec{o}$，而解码器则负责从 $\vec{o}$ 中恢复 $\vec{i}$。对偶综合 [3–8] 假设 $\vec{i}$ 总能够被 $\vec{o}$ 的一个有限序列唯一决定，以自动产生相应的解码器。其中，解码器的布尔函数可以使用 Jiang et al. [9] 提出的算法，该算法基于 Craig 插值 [10]。

通过研究现有的工业界编码器，我们发现他们都含有流水线结构以提高运行频率。

一个带有流水线的简单编码器如图**??**a) 所示。他的第一级流水线 $\vec{stg}^0$ 包含数个寄存器。其中输入变量 $\vec{i}$ 被用于计算该级流水线 $\vec{stg}^0$，而流水线 $\vec{stg}^0$ 则被用于计算 $\vec{o}$。根据该结构，$\vec{stg}^0$ 能够被 $\vec{o}$ 唯一决定，而 $\vec{i}$ 能够被 $\vec{stg}^0$ 唯一决定。

因此，一个由人类程序员设计的合理解码器, 应当如图**??**b) 所示，从 $\vec{o}$ 中使用组合逻辑 C^1 恢复 $\vec{stg}^0$，并进一步使用组合逻辑 C^0 从 $\vec{stg}^0$ 中恢复 $\vec{i}$。在此类解码器中，关键路径被流水线级 $\vec{stg}^0$ 切断，以改善时序。

然而，目前所有的对偶综合算法 [4–8] 均使用 Jiang 提出的基于 [10] 的算法 [9]。如图**??**c) 所示，这些算法从 $\vec{o}$ 中使用一个大型组合逻辑 $C^0 * C^1$ 直接恢复 $\vec{i}$, 这使得他们变得不必要的很慢，因为没有流水线寄存器切断这段复杂逻辑。

为了产生如图**??**b) 所示的解码器，本文提出了一个新颖的算法。首先找到编码器中每一个流水线级 $\vec{stg}^j$ 中的寄存器，然后特征化每一个流水线级 $\vec{stg}^j$ 的布尔函数，已从下一个流水线级 $\vec{stg}^{j+1}$ 或输出 $\vec{o}$ 之中恢复 $\vec{stg}^j$。最终特征化 $\vec{i}$ 的布尔函数以葱第一个流水线级 $\vec{stg}^0$ 中恢复 $\vec{i}$。

在复杂的工业界实际编码器，如 PCI Express [2] 和以太网 [1]，上的实验表明，该算法总能够产生带有流水线级的快速解码器。.

本文剩余的内容安排如下. 第1.2 节介绍了背景知识; 第1.3 节推导流水线结构，而第1.4 节特征化每一个流水线级和输入的布尔函数; 第1.5 和1.6 节给出实验结果和相关工作; 最后，第1.7 节给出结论。

1.2　背景知识

1.2.1　命题逻辑可满足

布尔集合为 $\mathbb{B} = \{0, 1\}$。变量向量表示为 $\vec{v} = (v, \dots)$。向量 $\vec{v}$ 中的变量个数表示为 $|\vec{v}|$。若一个变量 v 是 $\vec{v}$ 的成员，则记为 $v \in \vec{v}$; 否则 $v \notin \vec{v}$. $v \cup \vec{v}$ 是包含 v 和 $\vec{v}$

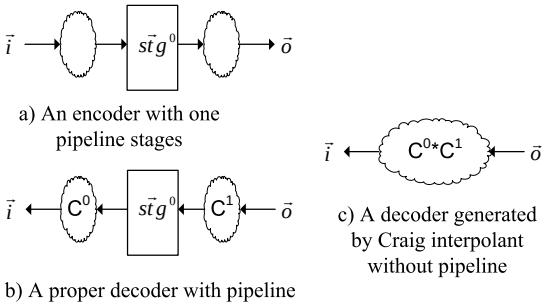

图 1.1　带有流水线的编码器和解码器

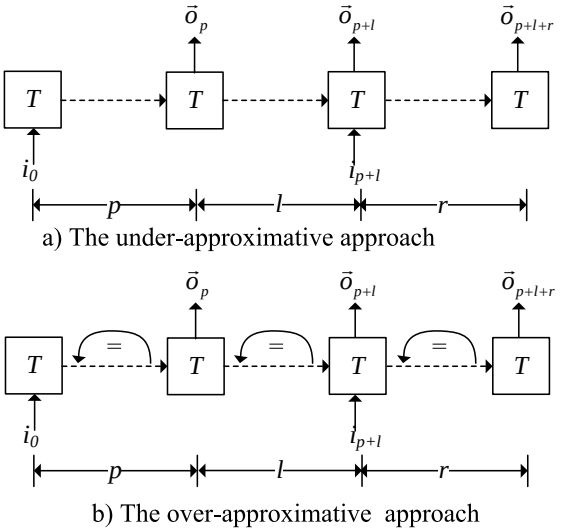

图 1.2　下估计和上估计算法

的所有成员的新向量。$\vec{v}/\vec{w}$ 是一个包含 $\vec{v}$ 的所有成员但是不包含 $\vec{w}$ 的任意成员的新向量。$\vec{a} \cup \vec{b}$ 是包含 $\vec{a}$ 和 $\vec{b}$ 的所有成员的新向量。$\vec{v}$ 的赋值集合记为 $[\![\vec{v}]\!]$，比如，$[\![(v_1, v_2)]\!] = \{(0,0),(0,1),(1,0),(1,1)\}$。

对于变量集合 V 上的公式 F，起命题逻辑可满足 (SAT) 问题是指如何找到 V 的一个赋值函数 $A : V \to \mathbb{B}$，使得 F 能够被赋值为 1。若 A 存在，则 F 是可满足的；否则，是不可满足的。

对于公式 ϕ_A 和 ϕ_B，若 $\phi_A \wedge \phi_B$ 不可满足，则存在公式 ϕ_I 仅包含 ϕ_A 和 ϕ_B 的共同变量，并使得 $\phi_A \Rightarrow \phi_I$ 且 $\phi_I \wedge \phi_B$ 不可满足。ϕ_I 称为 ϕ_A 相对于 ϕ_B 的 Craig 插值 [10]。ϕ_I 可以使用 McMillan 提出的算法 [11] 进行计算，并应用于从一个布尔关系特征化出一个布尔函数 [9]。

1.2.2 游有限状态机

编码器使用有限状态机模型 (FSM) $M = (\vec{s}, \vec{i}, \vec{o}, T)$，其中 $\vec{s}$ 为状态变量向量，$\vec{i}$ 为输入变量向量，$\vec{o}$ 为输出变量向量，$T : [\![\vec{s}]\!] \times [\![\vec{i}]\!] \to [\![\vec{s}]\!] \times [\![\vec{o}]\!]$ 是一个状态迁移函数，用于从当前状态和输入变量向量，计算出下一状态和输出变量向量。

M 的行为可以通过展开迁移关系进行推导。在第 n 步的 $s \in \vec{s}$，$i \in \vec{i}$ 和 $o \in \vec{o}$ 分别表示为 s_n, i_n 和 o_n。另外，在第 n 步的单个状态，输入和输出变量分别表示为 $\vec{s}_n$，$\vec{i}_n$ 和 $\vec{o}_n$。一个路径是指 $< \vec{s}_n, \ldots, \vec{s}_m >$，其中对于 $n \le j < m$，有 $\exists \vec{i}_j \vec{o}_j (\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$。一个环是一个路径 $< \vec{s}_n, \ldots, \vec{s}_m >$ 满足 $\vec{s}_n \equiv \vec{s}_m$。

1.2.3 用于确定一个输入变量能否被输出向量的有限序列唯一决定的停机算法

该算法 [4] 迭代的展开迁移函数。对于每次迭代，其分别调用一个下估计和一个上估计算法已给出确定的答案。这两个算法分别在1.2.3.1 和1.2.3.2中给出。我们酱紫啊1.2.3.3 中指出这两者将最终收敛。

1.2.3.1 下估计算法

如图1.2a) 所示，在展开的迁移函数序列上，输入变量 $i \in \vec{i}$ 能够被唯一决定的充分条件是，存在 p，l 和 r，使得对于输出向量序列 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 的任意值，i_{p+l} 不能同时为 0 和 1。折等价于公式 (1.1) 中定义的 $F_{PC}(p, l, r)$ 的不可满足性。

$$
F_{PC}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \end{array} \right\} \quad (1.1)
$$

其中，p 是前置状态序列的长度，l 和 r 是两个用于唯一决定输入 i_{p+l} 的输出变量序列 $< \vec{o}_{p+1}, \ldots, \vec{o}_{p+l} >$ 和 $< \vec{o}_{p+l+1}, \ldots, \vec{o}_{p+l+r} >$ 的长度。行 2 对应于图1.2a) 中的路径，行 3 是他的一个拷贝。他们的长度是一样的。行 4 强制两者的输出序列具有相同的赋值，而行 5 强制他们的输入 i_{p+l} 不同。

根据公式 (1.1)，对于 $p' \ge p$，$l' \ge l$ 和 $r' \ge r$，$F_{PC}(p', l', r')$ 的短句集合是 $F_{PC}(p, l, r)$ 的超集。因此, $F_{PC}(p, l, r)$ 的不可满足性可以扩展到任意更大的 $p' \ge p$，$l' \ge l$ 和 $r' \ge r$。

命题 1.1: *若 $F_{PC}(p, l, r)$ 不可满足，则对任意更大的 p, l 和 r，i_{p+l} 可以被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。*

算法 1.1 *CheckUniqueness*(i): 检查 $i \in \vec{i}$ 能否被 $\vec{o}$ 唯一决定的停机算法

```
1: 输入: 输入变量 i ∈ i⃗.
2: 输出: i ∈ i⃗ 是否能够被 o⃗ 唯一决定,p, l 和 r 的取值
3: p:= 0; l:= 0; r:= 0
4: while 1 do
5:     p++; l++; r++
6:     if F_PC(p, l, r) 不可满足 then
7:         return (yes, p, l, r);
8:     else if F_LN(p, l, r) 可满足 then
9:         return (no, p, l, r);
10:    end if
11: end while
```

1.2.3.2 上估计算法

若上一节定义的 $F_{PC}(p,l,r)$ 可满足，则存在两种可能性:

1. i_{p+l} 可以在更大的 p, l 和 r 时，被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定;
2. i_{p+l} 不能被任何 p, l 和 r 情况下的 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。

如果是第一种情况，则通过迭代的增大 p, l 和 r，$F_{PC}(p,l,r)$ 总能够变成不可满足。而如果是第二种情况，则该算法不停机。

因此, 为了得到停机算法我们需要区分上述两种情形。该方法如图1.2b) 所示，该图类似于图1.2a)，不过增加了三个新的约束用于检测在 $<\vec{s}_0,\ldots,\vec{s}_p>$,$<\vec{s}_{p+1},\ldots,\vec{s}_{p+l}>$ 和 $<\vec{s}_{p+l+1},\ldots,\vec{s}_{p+l+r}>$ 上的环。其形式化的定义如公式 (1.2) 其中最后三行对应于新增加的三个环检测约束。

$$F_{LN}(p,l,r) := \left\{\begin{array}{ll} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array}\right\} \quad (1.2)$$

当 $F_{LN}(p,l,r)$ 可满足时，则 i_{p+l} 不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。更重要的是，通过展开这三个环我们可以把 $F_{LN}(p,l,r)$ 的可满足性推广到更大的 p, l 和 r 上。这意味着:

命题 1.2: 如果 $F_{LN}(p,l,r)$ 可满足，则 i_{p+l} 对于任意更大的 p, l 和 r，不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。

其正确性证明的细节见 [4]。

1.2.3.3 完全算法

基于 Propositions 1.1 和1.2，我们有如下算法1.1。该算法搜索 p, l 和 r 使得输入变量 i_{p+l} 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

1. 一方面，如果存在这样的 p, l 和 r，则令 $p' := max(p,l,r), l' := max(p,l,r)$ 和 $r' := max(p,l,r)$。从 Propositions 1.1，我们知道 $F_{PC}(p', l', r')$ 不可满足。因此 $F_{PC}(p,l,r)$ 总能在行5称为不可满足；

2. 另一方面，如果不存在 p, l 和 r，则 p, l 和 r 总会变得比最长的无环路径更长，这意味着在 $< \vec{s}_0, \ldots, \vec{s}_p >, < \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ 和 $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$ 上存在环。这将使得 $F_{LN}(p,l,r)$ 在行7变得可满足。

两种情况都将导致上述算法停机。其正确性和停机性证明的细节见 [4] 。

1.3 推导编码器的流水线结构

1.3.1 流水线的一般性模型

如图1.3所示，我们假设编码器有 n 几流水线。如果我们把组合逻辑 C^j 视为一个函数，则编码器可以用下列的公式表示：

$$\begin{array}{rcll} \vec{stg}^{\,0} & := & C^0(\vec{i}) & \\ \vec{stg}^{\,j} & := & C^j(\vec{stg}^{\,j-1}) & 1 \le j \le n-1 \\ \vec{o} & := & C^n(\vec{stg}^{\,n-1}) & \end{array} \tag{1.3}$$

因此, 每个 C^j 可以视为一个小型的编码器用于从 $\vec{stg}^{\,j-1}$ 或 $\vec{i}$ 中计算 $\vec{stg}^{\,j}$ 或 $\vec{o}$。

在本文的剩余部分，上标始终表示特定的流水线级，, 而下标则如小节1.2.2所述，表示在展开的迁移关系序列中的步数。例如, $\vec{stg}^{\,j}$ 是第 j 个流水线级。而 $\vec{stg}^{\,j}_i$ 则是该第 j 流水线级在展开的迁移函数序列中的第 i 步的值。

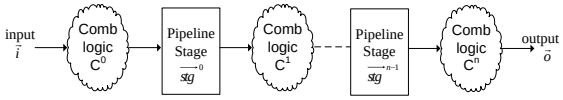

图 1.3 编码器的一般性结构

算法 1.2 *RemoveRedundancy*(p, l, r)

```
1: for r' := r → 0 do
2:     if r' ≡ 0 or F_PC(p, l, r' − 1) 对于某些 i ∈ i⃗ 是可满足的 then
3:         break;
4:     end if
5: end for
6: return r';
```

1.3.2 推导 p, l 和 r

在推导之前，我们首先使用算法1.1 以得到 p, l 和 r 。

因为存在多于一个 $i \in \vec{i}$，我们需要为每一个 $i \in \vec{i}$ 使用算法1.1 以得到他们对用的 p, l 和 r 。

然后我们将最终的 p,l 和 r 设置为所有 $i \in \vec{i}$ 中最大的 p, l 和 r 。根据公式1.1，这些 p, l 和 r 能够使得 $< o_p, \ldots, o_{p+l+r} >$ 唯一决定所有 $i_{p+l} \in \vec{i}_{p+l}$。

1.3.3 压缩 r 和 l

算法1.1 同步增长 p, l 和 r ，因此在 l 和 r 中可能存在冗余。因此我们需要首先在算法1.2中压缩 r。

在行1，当 $F_{PC}(p, l, r' - 1)$ 可满足时，则 r' 是最有一个使得 $F_{PC}(p, l, r')$ 不可满足的，我们将其直接返回。另一方面，当 $r' \equiv 0$，$F_{PC}(p, l, 0)$ 已经在上一次迭代中被测试过, 且结果必然是不可满足。在这种情况下我们返回 0。

如此, 我们从算法1.2获得了一个压缩后的 r ，它能使 $\vec{i}_{p+l}$ 被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

我们进一步要求：

1. 如图1.4所示，l 可以被压缩为 0, 这意味着 $\vec{i}_p$ 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+r} >$ 唯一决定，也就是说，所有的将来输出。

2. 上述的序列 $< \vec{o}_p, \ldots, \vec{o}_{p+r} >$ 可以被进一步压缩为 $\vec{o}_{p+r}$. 这意味着 $\vec{o}_{p+r}$ 是在恢复 $\vec{i}_p$ 时唯一被需要的输出。

检查上述两个要求等价于下式的不可满足：

图 1.4　从压缩的输出序列中恢复输入

$$F'_{PC}(p,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & i_p \equiv 1 \wedge i'_p \equiv 0 \end{array} \right\} \tag{1.4}$$

该等式看起来似乎远强于等式 (1.1)。我们将在实验结果中指出，该等式总能称为不满足。

1.3.4　推导流水线

基于上述的 p 和 r, 我们将上述公式 (1.4) 中的 F'_{PC} 推广到下述公式以便能够检查 v 在第 j 步是否能够被 $\vec{w}$ 在第 k 步唯一决定。现在 v 和 $\vec{w}$ 可以使输入，输出和状态变量向量。

$$F''_{PC}(p,r,v,j,\vec{w},k) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{w}_k \equiv \vec{w'}_k \\ \wedge & v_j \equiv 1 \wedge v'_j \equiv 0 \end{array} \right\} \tag{1.5}$$

很显然, 当 $F''_{PC}(p,r,v,j,\vec{w},k)$ 不可满足时，$\vec{w}_k$ 能够唯一决定 v_j.

最后一集流水线 $\vec{stg}^{n-1}$ 就是所有能够被在第 $p+r$ 步的 $\vec{o}$ 唯一决定的 $s \in \vec{s}$。其定义为:

$$\vec{stg}^{n-1} := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,p+r,\vec{o},p+r) \\ is\ unsatisfiable \end{array} \right\} \tag{1.6}$$

类似的, 对于 $0 \le j \le n-2$, $\vec{stg}^{j}$ 在第 $j-((n-2)-(p+r-1))$ 步可以被 $\vec{stg}^{j+1}$ 在第 $j-((n-2)-(p+r-1))+1$ 步唯一决定。我们可以定义 $\vec{stg}^{j}$ 为:

$$\begin{array}{rcl} S & := & \vec{s} / \bigcup_{j<k\leq n-2} \vec{stg}^{k} \\ D & := & (n-2)-(p+r-1) \end{array} \tag{1.7}$$

$$\vec{stg}^{j} := \left\{ s \in S \mid F''_{PC}(p,r,s,j-D,\vec{stg}^{j+1},j-D+1) \right\} \tag{1.8}$$

基于等式 (1.6) 和 (1.8), 所有的流水线几都能够被推导出来。

1.3.5 推导唯一决定输入的流水线级

根据图1.3, 定义于 (1.8) 的 $\vec{stg}^{0}$ 是唯一决定 $\vec{i}$ 的流水线级。

然而在实际的解码器中，并不一定是这种情形。因此我们需要从 0 到 $n-1$，搜索最小的 j 使得 $\vec{i}$ 能够被 $\vec{stg}^{j}$ 唯一决定。即, 最小的能够使得 $F''_{PC}(p,r,i,p,\vec{stg}^{j},j-D)$ 对所有 $i \in \vec{i}$ 不可满足的 j。其中 D 定义于等式 (1.7)。

1.4 特征化输入向量和流水线级的布尔函数

1.4.1 特征化最后一个流水线级的布尔函数

根据等式 (1.6), 每个寄存器 $s \in \vec{stg}^{n-1}$ 都能够被 $\vec{o}$ 在第 $p+r$ 步唯一决定。即, $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ 不可满足，且可以被划分为:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+r} \equiv 1 \end{array} \right\} \tag{1.9}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & s'_{p+r} \equiv 0 \end{array} \right\} \tag{1.10}$$

由于 $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ 等价于 $\phi_A \wedge \phi_B$, 因此 $\phi_A \wedge \phi_B$ 也是不可满足的。而 ϕ_A 和 ϕ_B 的共同变量集合是 $\vec{o}_{p+r}$。

根据 [9], ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以构造出来，其中仅包含 $\vec{o}_{p+r}$，并覆盖所有使 $s_{p+r} \equiv 1$ 得 $\vec{o}_{p+r}$ 的赋值。同时, $\phi_I \wedge \phi_B$ 不可满足，这意味着 ϕ_I 不能使 $s_{p+r} \equiv 0$。

因此, ϕ_I 能作为解码器的布尔函数以从 $\vec{o}$ 中恢复 $s \in \vec{stg}^{n-1}$ 。

1.4.2 特征化其他流水线级的布尔函数

类似于上一小节, 我们可以将公式 (1.8) 中的不可满足公式 $F''_{PC}(p, r, s, j - D, \vec{stg}^{j+1}, j - D + 1)$ 划分如下:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & s_{j-D} \equiv 1 \end{array} \right\} \quad (1.11)$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{j+1}_{j-D+1} \equiv \vec{stg'}^{j+1}_{j-D+1} \\ \wedge & s'_{j-D} \equiv 0 \end{array} \right\} \quad (1.12)$$

ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 能够被构造出来，并作为从 $\vec{stg}^{j+1}$ 恢复 $s \in \vec{stg}^{j}$ 的布尔函数。

1.4.3 特征化输入变量的布尔函数

根据小节1.3，我们找到了使得 $F''_{PC}(p, r, i, p, \vec{stg}^{j}, j - D)$ 对所有 $i \in \vec{i}$ 都不满足的 j，其中 D 在的公式 (1.7) 中定义。$F''_{PC}(p, r, i, p, \vec{stg}^{j}, j - D)$ 是不可满足的并可划分为:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & i_p \equiv 1 \end{array} \right\} \quad (1.13)$$

表 1.1　Benchmarks and experimental results

Names	编码器				由 [4] 产生的解码器			本文产生的解码器			
	# in/out	# reg	面积	编码器描述	运行时间	延迟 (ns)	面积	运行时间	延迟 (ns)	面积	寄存器个数
pcie	10/11	23	326	PCIE 2.0 [2]	0.37	7.20	624	3.57	5.89	652	9/12
xgxs	10/10	16	453	以太网 clause 48 [1]	0.21	7.02	540	1.57	5.93	829	13
t2eth	14/14	49	2252	以太网 clause 36 [1]	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	以太网 clause 49 [1]							

$\phi_B :=$

$$\left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{\,j}_{j-D} \equiv \vec{stg'}^{\,j}_{j-D} \\ \wedge & i'_p \equiv 0 \end{array} \right\} \quad (1.14)$$

和上一小节类似, ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以作为从 $\vec{stg}^{\,j}$ 中恢复 $i \in \vec{i}$ 的布尔函数。

1.5　实验结果

我们在 OCaml 语言中实现了上述算法，并使用 MiniSat 1.14 [12] 求解相应的公式。使用一台包含 16 个 Intel Xeon E5648 2.67GHz 处理器，192GB 存储器, 和 CentOS 5.4 Linux 操作系统的服务器。

表1.1 给出了本文使用的 benchmarks 。第 2 和 3 列分别给出了每个 benchmark 的输入, 输出和寄存器数量。第 4 列将这些编码器映射至 LSI10K 库的面积。本文中，所有的面积和延时都是在相同的设定下得到的。

第 6 到 8 列分别给出了 [4] 算法在产生误流水线的解码器时的运行时间，延迟和面积。而第 9 到 11 列分别给出了本文算法的运行时间，延迟和面积。而最后一列给出了每一级流水线包含的寄存器个数。

比较第 7 和 10 列可以看到解码器的延迟得到了较大的改善，而最后一列指出确实存在很深的流水线，其中 t2ether 包含 4 级流水线。

有一点非常有意思的是, 两个最大的 benchmarks scrambler 和 xfi 没有检测到流水线。我们研究了其代码，并确认了这一点。他们的面积如此之大是因为使用了 64 到 72 位宽的数据路径。

1.6 相关工作

沈等 [3] 提出了第一个对偶综合算法。他通过迭代的展开迁移函数来检测解码器的存在性。并通过遍历可满足赋值了产生解码器函数。但是该算法是不停机的，并且在产生解码器时非常慢。

沈 et al.[4] 和刘 at al.[6] 分别独立处理了停机问题，方法是在状态序列中检测环形路径。而解码器构造中运行时间太长的问题则在 [5, 6] 中通过 Craig 插值 [9] 解决。

沈 et al. [5] 推导了能够使得解码器存在的配置断言。

屠 et al.[8] 提出了一个突破性的算法能够在构造解码器时考虑整个无限的输入历史。

1.7 结论

本文提出了第一个能够处理流水线的对偶综合算法。实验结果表明，本算法能够针对多个复杂的实际工业界编码器，正确的推导流水线结构并生成对应的流水线解码器。.

参考文献

[1] IEEE. IEEE Standard for Ethernet SECTION FOURTH. 2012. http://standards.ieee.org/getieee802/download/802.3-2012_section4.pdf.

[2] PCI-SIG. PCI Express Base 2.1 Specification. 2009. http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_r2_1_04Mar09.pdf.

[3] Shen S, Zhang J, Qin Y, et al. Synthesizing complementary circuits automatically [C/OL]. In Proceedings of the 2009 International Conference on Computer-Aided Design. San Jose, CA, USA, 2009: 381–388. http://dx.doi.org/10.1145/1687399.1687472.

[4] Shen S, Qin Y, Xiao L, et al. A Halting Algorithm to Determine the Existence of the Decoder [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2011, 30 (10): 30:1556–30:1563. http://doi.acm.org/10.1109/TCAD.2011.2159792.

[5] Shen S, Qin Y, Wang K, et al. Inferring Assertion for Complementary Synthesis [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2012, 31 (8): 31:1288–31:1292. http://doi.acm.org/10.1109/TCAD.2012.2190735.

[6] Liu H-Y, Chou Y-C, Lin C-H, et al. Towards completely automatic decoder synthesis [C/OL]. In Proceedings of the 2011 International Conference on Computer-Aided Design, ICCAD 2011. San Jose, CA, USA, 2011: 389–395. http://dx.doi.org/10.1109/ICCAD.2011.6105359.

[7] Liu H-Y, Chou Y-C, Lin C-H, et al. Automatic Decoder Synthesis: Methods and Case Studies [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2012, 31 (9): 31:1319–31:1331. http://doi.acm.org/10.1109/TCAD.2012.2191288.

[8] Tu K-H, Jiang J-H R. Synthesis of feedback decoders for initialized encoders [C/OL]. In Proceedings of the 50th Annual Design Automation Conference, DAC 2013. Austin, TX, USA, 2013: 1–6. http://dx.doi.org/10.1145/2463209.2488794.

[9] Jie-Hong Roland Jiang W-L H, Hsuan-Po Lin. Interpolating functions from large Boolean relations [C]. In Proceedings of 2009 International Conference on Computer-Aided Design. 2009: 779–784.

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

[11] McMillan K L. Interpolation and sat-based model checking [M] // Warren A Hunt Jr F S. Computer Aided Verification, 15th International Conference, CAV 2003Vol.2725. Berlin Heidelberg: Springer-Verlag, 2003: 2003: 1–13.

[12] Eén N, Sörensson N. An extensible sat-solver [M] // Enrico Giunchiglia A T. Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003Vol.2919. Berlin Heidelberg: Springer-Verlag, 2003: 2003: 502–518.