论 文

对偶综合的断言推导

沈胜宇*, 秦莹, 王克非, 庞征斌, 张建民, 李思昆

国防科技大学计算机学院, 长沙410073
*沈胜宇E-mail: syshen@nudt.edu.cn, qy123@nudt.edu.cn, lqxiao@nudt.edu.cn, kfwang@nudt.edu.cn, jmzhang@nudt.edu.cn, skli@nudt.edu.cn

国家自然科学基金(批准号: 61070132)资助项目

摘要 对偶综合算法能够自动综合特定编码器的解码器。然而，用户需要手工给出某些配置管脚的值，以防编码器进入非工作状态。

为了避免这种繁琐的工作，本文提出一个自动算法，通过迭代的检测并移除无解码器的情形，以得到正确的断言。

为了发掘在该断言下可能同时存在的多个解码器，另一个算法被提出以将迁移关系$\mathbb{R}$划分为多个可能的解码器。

为了帮助用户挑选正确的解码器，第三个算法被提出来以计算每个解码器存在的前提条件。

在一系列复杂编码器上的实验结果表明，我们的算法始终能够为这些编码器推到断言并生成解码器。进一步的，在存在多个解码器的情况下，用户能够非常容易的通过检查前提条件公式，推断出正确的解码器。

关键词
对偶综合断言推导
因子化Craig插值
函数依赖

1 引言

在设计复杂通讯芯片中，最为困难的任务在于设计电路对(E, E^{-1}), 其中编码器E将输入数据流转换成为特定的编码格式，而解码器E^{-1}则恢复输入数据流。

因此, 对偶综合[1]被提出以自动综合一个编码器的解码器。如图Fig. 1a)所示, 对偶综合包含三步: **i)**手工给出配置管脚上的断言以防止编码器进入非工作状态; **ii)** 通过检验编码器的输入能否被其输出唯一决定，以确定解码器是否存在; **iii)** 特征化解码器的布尔函数。

为了手工给出断言, 用户需要大量阅读文档并进行大量手工尝试. 例如, 最复杂的XFI编码器有120个配置管脚。找出它们的含义和正确的组合是一个非常困难的过程, 作者已经在论文[1]中指出了这一点. 为了避免这一繁杂的工作, 本文给出了以下三个算法.

首先, 一个算法[2]被提出以自动推导该断言, 该算法对应于图Fig. 1b)的第一步. 该算法迭代的发掘导致解码器不存在的配置值, 并使用Craig插值发掘覆盖该配置值的大型公式. 这些发掘的公式将被剔除直至没有更多的非法配置能够被发掘为止。最终的断言就是这些所有被发掘的公式的非与。

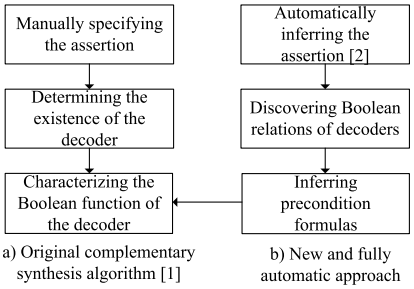

图 1 原始的和新提出的对偶综合流程

其次, 然而在这个被发掘的断言下, 多个解码器有可能同时存在. 因此, 如图. 1b)的第二步所示, 第二个算法被提出以发掘所有这些解码器，通过迭代的测试$\mathbb{R}$, 能够从输出唯一决定输入的布尔关系, 函数依赖于[5]所有被发掘的解码器. 对于失败与此次测试的配置值c, 通过将c与进$\mathbb{R}$可以得到新的布尔关系. 该步骤将迭代直至得到所有的解码器.

第三, 为了帮助用户选择合适的解码器, 第三个算法被提出以特征化特定解码器的前提条件公式, 该条件代表了该解码器存在的配置值全集. 用户通过检查该公式能够非常容易的找到正确的解码器.

例如, 对于XFI编码器的两个被发掘的解码器, 他们的配置公式只引用了三个配置管脚, 只有其中的两个取值不同. 因此, 用户只需找出这两个管脚的含义,而不是全部120个管脚，即可选择到正确的解码器. 更多的细节可以在实验结果中找到, 这表明本文算法能够极大的简化用户花在写断言和选择解码器上的时间. 本文的所有程序和实验结果可以从`http://www.ssypub.org`下载.

本文剩下的章节如下组织. 第2节介绍背景资料. 第3节给出断言推到算法. 第4节指出如何发掘解码器并推导他们的前提公式. 第5 和6节给出实验结果和相关工作. 最后, 第7节给出结论.

2 背景知识

为了节约空间, 我们假设读者熟悉命题逻辑可满足性(SAT)概念, 因子化[7], Craig 插值[8], 和函数依赖[5].

编码器使用带配置的*Mealy*有限自动机 $M = (S, s_0, I, C, O, T)$建模, 包含有限状态集合S, 初始状态$s_0 \in S$, 有限的输入字符集合I, 有限的配置字符集合C, 有限的输出字符集合O, 以及迁移函数$T : S \times I \times C \to S \times O$用于从当前状态和输入字符计算次态和输出字符。

我们记第n周期的状态，输入，输出和配置字符分别为s_n, i_n, o_n, 何c_n。并进一步记从第n到第m周期的态，输入，输出和配置字符序列分别为s_n^m, i_n^m, o_n^m, 和c_n^m.

在配置管脚上的一个断言*(*或公式*)*定义为配置字符集合R. $R(c)$ 意味着$c \in R$. 若$R(c)$ 成立, 我们也记R覆盖c.

解码器存在当且仅当其输入能够由其输出唯一决定. 如图. 2a)所示, 这可以形式化的定义为.

Definition 1. 参数对偶条件(PC)*:* 对于编码器E, 断言R, 和参数p, d, 及l, $E \vDash PC(p, d, l, R)$成立当i_n能够被o_{n+d-l}^{n+d-1}唯一决定, 且R覆盖c. 着等价于公式*(1)*中$F_{PC}(p, d, l, R)$的不可满足.

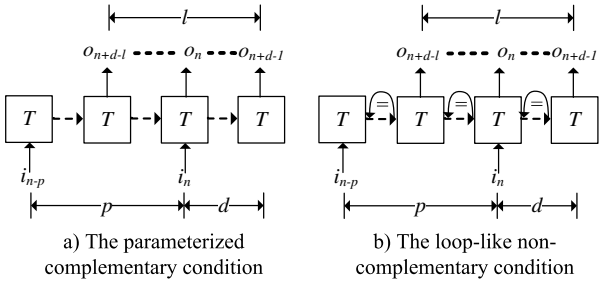

图 2 参数对偶条件和环形非对偶条件

$$F_{PC}(p,d,l,R) \stackrel{def}{=}$$
$$\left\{\begin{array}{lc} & \bigwedge_{m=n-p}^{n+d-1}\{(s_{m+1},o_m)\equiv T(s_m,i_m,c)\} \\ \wedge & \bigwedge_{m=n-p}^{n+d-1}\{(s'_{m+1},o'_m)\equiv T(s'_m,i'_m,c)\} \\ \wedge & \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \\ \wedge & i_n \neq i'_n \\ \wedge & R(c) \end{array}\right\} \tag{1}$$

公式(1)的行2和3对应于E的两个状态序列. 行4强制这两个状态序列的输出相等, 而行5强制他们的输入不等. 最后一行强制c被R覆盖.

基于检查PC[1]的算法简单的从小到大遍历p, d, 和l的各种组合, 直到$F_{PC}(p,d,l,R)$ 不可满足, 这就意味着解码器存在.

3 确认解码器存在性的停机算法

在小节3.1中, 我们将首先定义如何针对一个特定的配置字符，确定是否存在解码器. 而后, 在小节3.2中, 我们将介绍如何通过迭代的剔除无解码器的情形，以推导断言.

3.1 确认解码器的不存在

根据定义1, 对配置字符集合R，解码器存在当且仅当存在参数$< p,d,l >$使得$E \vDash PC(p,d,l,R)$成立. 因此, 直观的, 解码器不存在当且仅当对任意$< p,d,l >$, 我们总能找到$< p',d',l' >$使得$p' > p$,$l' > l$且$d' > d$, 且$E \vDash PC(p',d',l',R)$不成立.

该情形可以用图2b)所示的SAT问题邱恩杰, 该图类似于图2a), 除了三个新的约束被插入以检测s_{n-p}^{n+d-l},$s_{n+d-l+1}^{n}$和s_{n+1}^{n+d}的三个环.

因此, 解码器的不存在可以通过下列定义确认:

Definition 2. 环形非对偶条件(LN): 对编码器E, $E \vDash LN(p,d,l,R)$成立当i_n不能被o_{n+d-l}^{n+d-1} *on* s_{n-p}^{n+d-1}唯一决定, 且在s_{n-p}^{n+d-l}, $s_{n+d-l+1}^{n}$和s_{n+1}^{n+d}上存在环. 这等价于公式*(2)*中$F_{LN}(p,d,l,R)$的可满足性.

$$F_{LN}(p,d,l,R) \stackrel{def}{=} \left\{ \begin{array}{ll} & F_{PC}(p,d,l,R) \\ \wedge & \bigvee_{x=n-p}^{n+d-l-1} \bigvee_{y=x+1}^{n+d-l} \{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \\ \wedge & \bigvee_{x=n+d-l+1}^{n-1} \bigvee_{y=x+1}^{n} \{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \\ \wedge & \bigvee_{x=n+1}^{n+d-1} \bigvee_{y=x+1}^{n+d} \{s_x \equiv s_y \wedge s'_x \equiv s'_y\} \end{array} \right\} \tag{2}$$

3.2 算法实现

Algorithm 1 InferAssertion

1: $NA = \{\}$
2: **for** $x = 0 \rightarrow \infty$ **do**
3: $< p,d,l >=< 2x,x,2x >$
4: **if** $F_{PC}(p,d,l,\bigwedge_{na \in NA} \neg na)$ 不可满足 **then**
5: 停机，最终断言为$\bigwedge_{na \in NA} \neg na$，解码器存在当该断言可满足.
6: **else**
7: **while** $F_{LN}(p,d,l,\bigwedge_{na \in NA} \neg na)$可满足 **do**
8: 假设$A(c)$是c的导致解码器不存在的可满足赋值.
9: $na \leftarrow InferCoveringFormula(A(c))$
10: $NA \leftarrow NA \cup \{na\}$
11: **end while**
12: **end if**
13: **end for**

算法1被用于推到导致解码器存在的配置值集合. 直观的, 他在行3迭代的测试$< p,d,l >$. 对于每个在行8找到的能导致解码器不存在的$A(c)$, 行9的$InferCoveringFormula$将发掘一个大型的公式以覆盖更多的同类赋值, 并在行10剔除他们. 该循环一直持续直至行4的F_{PC}不可满足.

$InferCoveringFormula$ 使用cofactoring [7] 将i_{n-p}^{n+d-1}, $(i')_{n-p}^{n+d-1}$, s_{n-p}, 和$(s')_{n-p}$设置进入F_{LN}, 并使用Craig插值[8]以特征化一个大型公式. 更多的细节参见[2], 该论文已经证明1是停机的.

4 发掘多个解码器

小节4.1介绍并证明了发掘多个解码器的算法及其正确性, 而小节4.2则给出算法以特征化每个解码器的前提条件, 以帮助用户选择正确的解码器.

4.1 构造SAT公式以发掘多个解码器

假设算法1发掘的断言为:

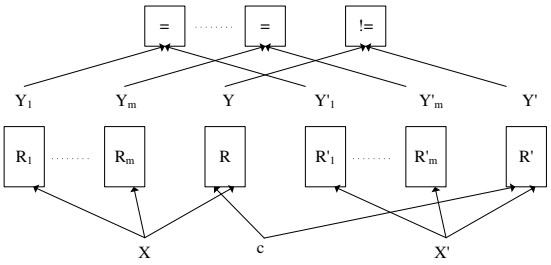

图 3 发掘解码器的公式

$$IA \overset{def}{=} \bigwedge_{na \in NA} \neg na \tag{3}$$

为了简明起见，我们记o_{n+d-l}^{n+d-1}和i_n为:

$$\begin{aligned} X &\overset{def}{=} o_{n+d-l}^{n+d-1} \\ Y &\overset{def}{=} i_n \end{aligned} \tag{4}$$

因此, 从o_{n+d-l}^{n+d-1}和c唯一决定i_n的布尔关系可以记为:

$$\mathbb{R}(c, X, Y) \overset{def}{=} F_{PC}(p, d, l, IA) \tag{5}$$

$\mathbb{R}(c, X, Y)$定义了从c和X到Y的映射, 该映射可以记为一个函数f:

$$Y = f(c, X) \tag{6}$$

对于每个配置字符$c_i \in IA$, 存在$\mathbb{R}_{c_i}$:

$$\mathbb{R}_{c_i}(X, Y) \overset{def}{=} \mathbb{R}(c, X, Y) \wedge c \equiv c_i \tag{7}$$

两个不同的c_i和c_j共享同一个$\mathbb{R}_{c_i}$. 因此, IA可以划分为$\{IA_1, \ldots, IA_n\}$, 使得在同一个IA_i的所有c共享相同的$\mathbb{R}_i$, 而在两个不同IA_i和$IA_{i'}$中的c和c'则不是. 因此我们称IA_i为$\mathbb{R}_i$的前提条件, 即IA_i是使得$\mathbb{R}_i$存在的配置字符集合. 当存在多个解码器时, 用户可以通过检查小节4.2中特征化的$\{IA_1, \ldots, IA_n\}$以选择正确的解码器.

因此, $\mathbb{R}_i$定义了一个从X到Y的映射, 可以视为一个函数$f_i : X \to Y$. 因此, f可以写为:

$$f(c, X) = \bigvee_{i=1}^{n} \{IA_i(c) \wedge f_i(X)\} \tag{8}$$

因此, 在这里我们的任务是逐步的找到$\{\mathbb{R}_1, \ldots, \mathbb{R}_n\}$. 假设我们已经找到了数个解码器$\{\mathbb{R}_1, \ldots, \mathbb{R}_m\}$. 为了测试其是否包含$\{\mathbb{R}_1, \ldots, \mathbb{R}_n\}$, 即, 是否所有的解码器都已经被发掘, 我们构造如图3所示的SAT公式:

$$\left\{\begin{array}{cc} & \mathbb{R}(c,X,Y)\wedge\bigwedge_{i=1}^{m}\mathbb{R}_i(X,Y_i) \\ \wedge & \mathbb{R}'(c,X',Y')\wedge\bigwedge_{i=1}^{m}\mathbb{R}'_i(X',Y'_i) \\ \wedge & \bigwedge_{i=1}^{m}Y_i\equiv Y'_i \\ \wedge & Y\neq Y' \end{array}\right\} \tag{9}$$

公式(9)的行1代表$\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$和$\mathbb{R}$的布尔关系. 行2是行1的拷贝. 他们分享的唯一变量是c. 行3强制Y_i和Y'_i取相同的值, 而最后一行则强制Y和Y'不同.

下面的定理证明了, 若公式(9)不满足, 则所有的解码器都已经被发掘.

Theorem 1. *若公式(9)不可满足, 则$\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$包含$\{\mathbb{R}_1,\ldots,\mathbb{R}_n\}$.*

Proof. 使用反证法. 假设$\mathbb{R}_n\notin\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$, 且$IA_n$是其对应的配置字符集合, 且$c_n\in IA_n$.

我们可以构造一个赋值集合A使得$A(c)\equiv c_n$. 因此, 我们有$\{\mathbb{R}(c,X,Y)\wedge A(c)\equiv c_n\}\equiv\mathbb{R}_n(X,Y)$, 也就是说, 我们可以用$A$将图3中的$\mathbb{R}$ 和$\mathbb{R}'$变为$\mathbb{R}_n$.

因为$\mathbb{R}_n\notin\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$, 必然存在赋值$A'$, 使得当将$A'(X)$赋值至$X$并将$A'(X')$赋值至$X'$, 我们可以使$\bigwedge_{i=1}^{m}Y_i\equiv Y'_i$和$Y\neq Y'$同时成立.

因此, 通过合并A和A', 公式(9)变为可满足. 矛盾，得证. □

另一方面, 如果公式(9)可满足, 我们需要证明.

Theorem 2. *如果公式(9)可满足, 则至少还有一个解码器没有被发掘.*

Proof. 使用反证法. 假设所有的解码器均已被发掘, 也就是说, $\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$包含$\{\mathbb{R}_1,\ldots,\mathbb{R}_n\}$. 因此, 对于使得公式(9)的前三行可满足的任意赋值A, 我们有:

$$Y=\bigvee_{i=1}^{m}\{IA_i(c)\wedge Y_i\}=\bigvee_{i=1}^{m}\{IA_i(c)\wedge Y'_i\}=Y' \tag{10}$$

因此, 公式(9)的最后一行将永不满足. 因此, 公式(9)是不可满足的矛盾，得证. □

对于可满足赋值A, 下面定义的$\mathbb{R}_{m+1}$ 是一个新发掘的解码器:

$$\mathbb{R}_{m+1}\stackrel{def}{=}\mathbb{R}(c,X,Y)\wedge c\equiv A(c) \tag{11}$$

定理3证明了$\mathbb{R}_{m+1}$尚未被发掘:

Theorem 3. $\mathbb{R}_{m+1}\notin\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$

Proof. 反证法. 假设有$0\leqslant i\leqslant m$使得$\mathbb{R}_i\equiv\mathbb{R}_{m+1}$.

由于$\mathbb{R}_i$能从X唯一决定Y, 而$\mathbb{R}$能从X和c唯一决定Y, 和$\mathbb{R}_{m+1}\equiv\mathbb{R}_i$, 很明显通过强制$c$等于$A(c)$在公式(11)，我们能使$Y\equiv Y_i$. 类似的我们有$Y'_i\equiv Y'$.

根据公式(9)的行5, 我们有$Y\equiv Y_i\equiv Y'_i\equiv Y'$. 这和公式(9)最后一行矛盾, 得证. □

基于上述讨论, 下列算法2描述了如何发掘所有的解码器.

算法2中的循环单调的增加$\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$的尺寸. 由于从$X$计算$Y$的函数个数是有限的, 算法2 是停机的.

Algorithm 2 *DiscoveringDecoders*

while 公式(9)可满足 **do**
　插入公式(11)的$\mathbb{R}_{m+1}$进入$\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$
end while
被发掘的解码器的布尔关系集合为$\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$

4.2 特征化$\{IA_1,\ldots,IA_m\}$

假设$\{\mathbb{R}_1,\ldots,\mathbb{R}_m\}$是算法2发掘的解码器全集. 为了帮助用户选择正确的解码器, 我们需要特征化他们的前提条件集合$\{IA_1,\ldots,IA_m\}$. 根据图3, Y和所有的Y_i之间的关系记为:

$$Y=\bigvee_{i=1}^{m}\{IA_i(c)\wedge Y_i\} \tag{12}$$

假设Y和所有Y_i是具有相同长度的向量v, 其中第j个比特是y^j和y_i^j. 因此, 通过划分为比特我们能够重写公式(12), 而Y和所有Y_i的第j比特可以重写为:

$$y^j=\bigvee_{i=1}^{m}\{IA_i^j(c)\wedge y_i^j\} \tag{13}$$

基于Lee et al. [5], 很明显公式(13)是一个函数依赖问题. 基于Lee et al. [5]的算法，使用下面的公式我们能够特征化$IA_i^j(c)$:

$$\begin{aligned}\phi_A &\stackrel{def}{=}\left\{\begin{array}{cc} & \mathbb{R}(c,X,Y)\wedge\bigwedge_{i=1}^{m}\mathbb{R}_i(X,Y_i)\\ \wedge & y^j\equiv 1\end{array}\right\}\\ \phi_B &\stackrel{def}{=}\left\{\begin{array}{cc} & \mathbb{R}'(c,X',Y')\wedge\bigwedge_{i=1}^{m}\mathbb{R}'_i(X',Y'_i)\\ \wedge & \bigwedge_{i=1}^{m}y_i^j\equiv y_i'^j\\ \wedge & y'^j\equiv 0\end{array}\right\}\end{aligned} \tag{14}$$

很明显$\phi_A\wedge\phi_B$非常类似于公式(9), 除了只有第j被约束为相同值, 且y^j和y'^j被约束为不同值.

因此, $\phi_A\wedge\phi_B$不可满足. 而插值公式$ITP:C\times\mathbb{B}^m\to\mathbb{B}$ 的支撑集为$\{c,y_1^j,\ldots,y_m^j\}$. 根据公式(13), ITP是$\bigvee_{i=1}^{m}\{IA_i^j(c)\wedge y_i^j\}$的上估计. 因此, $IA_i^j(c)$ 的上估计可以通过设置y_i^j 为1, 而其他y_k^j为0得到:

$$ITP\wedge\bigwedge_{k\neq i}y_k^j\equiv 0\wedge y_i^j\equiv 1 \tag{15}$$

由于$\phi_A\wedge\phi_B$不可满足, 该上估计$IA_i^j(c)$ 可以使$y^j\equiv 1$. 因此, 我们可以将其视为$IA_i^j(c)$. 因此, $IA_i(c)$可以定义为:

$$IA_i(c)=\bigwedge_{j=0}^{v-1}IA_i^j(c) \tag{16}$$

在小节5.3, 我们将展示用户通过检查IA_i, 可以很容易的选择正确的解码器.

表 1 Benchmarks的相关信息

	XGXS	XFI	scrambler	PCIE	T2 Ethernet
#line of verilog	214	466	24	1139	1073
#regs	15	135	58	22	48
Data path width	8	64	66	10	10
#Config pin	3	120	1	16	26

表 2 实验结果

		XG-XS	XFI	scra-mbler	PCI-E	T2 E-ther
[10]	时间(秒)	0.07	17.84	2.70	0.47	30.59
	d, p, l	1,2,1	0,3,2	0,2,2	2,2,1	4,2,1
本文	时间1	4.53	264.19	13.03	10.39	426.12
	时间2	0.11	12.11	1.26	0.27	3.07
	时间3	0.13	13.69	1.49	0.23	2.86
	d, p, l	1,5,1	0,5,2	0,5,2	2,5,1	4,5,1

5 实验结果

我们实现了上述算法并使用Minisat[6]求解生成的SAT公式. 所有的实验在一台有2.4GHz Intel Core 2 Q6600 处理器, 8 GB内存, 和Ubuntu 10.04 Linux操作系统的PC上进行.

表1给出了下列编码器的信息: XGXS和XFI编码器兼容于IEEE-802.3ae 2002标准[4]的clause 48 and 49 ; 一个同于保证输入数据流具有足够0-1翻转的加扰器scrambler; 一个PCI-E物理编码子层[3]; OpenSparc T2 处理器的以太网模块.

5.1 和从前的工作进行比较

表2比较了[10]和本文的算法. 其中[10]需要一个手工给出断言, 而本文算法不需要.

表2的行2给出[10]的停机算法的运行时间, 行3给出发掘的d, p, 和l.

我们的算法包括三步: 推到断言, 发掘解码器, 推到前提条件. 他们的运行时间在表2的行4到6给出. 第7行给出发掘的d, p, 和l.

通过比较行2和4, 很明显我们的算法比[10]慢很多, 这是由复杂的$InferCoveringFormula$造成的. 第5和6行表明第二和三步相对来说占用的时间很少.

而第3和7行表明发掘的参数有微小的差别, 这是由与在手写断言和自动推理断言之间存在差别造成的. 后者比前者包含多得多的情形.

5.2 推导的断言

For XGXS: ((!bad_code))

For XFI: ((RESET&!TEST_MODE)|(!RESET&DATA _VALID&!TEST_MODE))

For scrambler: True

For PCI-E: ((CNTL_RESETN_P0&!TXELECIDLE& !CNTL_Loopback_P0&CNTL_TXEnable_P0))

For T2 ethernet: ((tx_enc_conf_sel[1]&tx_enc _conf_sel[3]&!txd_sel[1]&!txd_sel[0]&!jit ter_study_pci[0]&
set_tx)|(tx_enc_conf_sel[1]&!tx_enc_conf_ sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_stu dy_pci[0]&!jitter_s
tx_enc_conf_sel[0]&tx_enc_conf_sel[2]&link _up_loc)|(tx_enc_conf_sel[1]&!tx_enc_conf _sel[3]&!txd_sel[1]&
udy_pci[0]&!jitter_study_pci[1]&!reset_tx& tx_enc_conf_sel[0]&!tx_enc_conf_sel[2])| (tx_enc_conf_sel[1]&
!txd_sel[1]&!txd_sel[0]&!jitter_study_pci[0] &!jitter_study_pci[1]&!reset_tx&!tx_enc_ conf_sel[0]&link_
[1]&tx_enc_conf_sel[3]&!txd_sel[1]&!txd_ sel[0]&!jitter_study_pci[0]&!jitter_study_pci [1]&!reset_tx&tx
_conf_sel[1]&tx_enc_conf_sel[3]&!txd_sel[1] &!txd_sel[0]&!jitter_study_pci[0]&!jitter_st udy_pci[1]&!res
nk_up_loc)|(!tx_enc_conf_sel[1]&!tx_enc_con f_sel[3]&!txd_sel[1]&!txd_sel[0]&!jitter_stu dy_pci[0]&!jitt
link_up_loc))

对于具有复杂断言的T2 ethernet, 使用小节4的算法辅助选择解码器非常重要.

5.3 针对多个解码器

对于scrambler的两个解码器, 他们的前提条件分别是*reset*和!*reset*. 通过检查他们的Verilog源代码, 我们发现*reset*为$True$时被用于复位. 所以, 当*reset*为$False$时将工作于正常模式. 因此, 第二个解码器是正确的解码器.

对于XFI编码器[4]的两个解码器, 他们的前提条件分别是!$TEST\_MODE\&RESET$ 和!$TEST\_MODE$ &!$RESET\&DATA\_VALID$. 他们之间唯一的差别在于$RESET$ 和$DATA_VALID$. 通过检查XFI的源代码, 我们发现$RESET$为$True$时被用于复位XFI, 而$DATA_VALID$为$True$时代表数据为真. 因此, 当$RESET$为$False$且$DATA_VA$为$True$时XFI工作于正常模式. 因此, 第二个解码器时正确的.

6 相关工作

对偶综合首先在[1]被提出. Shen et al. [9] 和Liu et al. [11] 分别通过不可满足核提取和Craig插值提升其性能. Shen et al. [10] 和Liu et al. [11] 提出确认解码器存在的停机算法.

K. Avnit et al. [12] 在两个通讯协议之间自动生成翻译自动机, 通过计算缓冲区控制函数的不动点.

7 Conclusions

本文提出了为对偶综合自动推导断言并发掘解码器的算法. 实验结果表明该算法能够极大的降低用户的时间开销.

Acknowledgment

本文作者感谢编辑和未具名的审稿人的辛勤工作.
本文的工作受到中国国家自然科学基金会项目60603088和61070132的资助.

参考文献

1 S. Shen, J. Zhang, Y. Qin, and S. Li, "Synthesizing complementary circuits automatically," in *ICCAD09*. IEEE, Nov. 2009, pp. 381–388.
2 S. Shen, Y. Qin, J. Zhang, and S. Li, "Inferring Assertion for Complementary Synthesis,"in *ICCAD11*. ACM, Nov. 2011, pp. 404–411.
3 "PCI Express Base Specification Revision 1.0". [Online]. Available: `http://www.pcisig.com`
4 "IEEE Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation", IEEE Std. 802.3, 2002.
5 C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental SAT solving," in *ICCAD07*. IEEE, Nov. 2007, pp. 227–233.
6 N. Eén and N. Sörensson. An extensible sat-solver. In *SAT03*, pages 502–518. Springer, May 2003.
7 M. K. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *ICCAD04*. IEEE, Nov. 2004, pp. 510–517.
8 K. L. McMillan, "Interpolation and SAT-based model checking," in *CAV03*. Springer, July 2003, pp. 1–13.
9 S. Shen, Y. Qin, K. Wang, L. Xiao, J. Zhang, and S. Li, "Synthesizing complementary circuits automatically," *IEEE trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 8, pp. 1191–1202, Aug. 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 trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 10, pp. 1556–1563, Oct. 2011.
11 S. Liu, Y. Chou, C. Lin, and J. Jiang, "Towards completely automatic decoder synthesis," in *ICCAD11*. ACM, Nov. 2011, pp. 389–395.
12 K. Avnit, V. D'Silva, A. Sowmya, S. Ramesh, and S. Parameswaran, "A formal approach to the protocol converter problem," in *DATE08*. IEEE, Mar. 2008, pp. 294–299.