RTL 综合中子程序的综合方法

袁 媛, 谢 巍, 刘明业
(北京理工大学 ASIC 研究所, 北京 100081)

摘 要: 本文论述了 RTL 综合系统中子程序综合方法, 针对以赋值语句为核心的中间数据结构, 采取嵌入方式或例示方式进行处理, 然后进行格式判别、组合逻辑综合及时序逻辑综合. 文章给出具体综合算法, 并在最后给出运行实例.

关键词: RTL 综合; 子程序; 格式判别; 组合逻辑; 时序逻辑; VHDL
中图分类号: TP301 文献标识码: A 文章编号: 0372-2112 (2002) 02-0236-04

Subprogram Synthesis Method In RTL Synthesis System

YUAN Yuan, XIE Wei, LIU Ming-ye
(*ASIC Research Center, Beijing Institute of Technology, Beijing* 100081, *China*)

Abstract: This paper discusses the subprogram synthesis method in RTL synthesis system and brings forward concrete algorithm which aims at particular data structure of RTL synthesis system. At the end of this paper gives experiment results.

Key words: RTL synthesis; subprogram; format judgement; combination logic synthesis; sequential logic synthesis; VHDL

1 引言

RTL 综合是将 RTL 描述自动转换成电路的门级结构的过程, 其关键技术是格式判别、组合逻辑综合及时序逻辑综合[1]. VHDL 语言中的子程序包括过程与函数, 在 RTL 描述中经常采用子程序定义一个或多个常用的模块以提高设计的效率, 因此在 RTL 综合系统中, 只有能够支持用户的子程序描述才能称为是完整的系统. 文献[2]论述了高级综合系统中的一种采用控制子程序方式的综合方法, 但其他有关 RTL 综合的实现方法很少见. RTL 综合与高级综合是两个不同层次上的综合系统, 在子程序综合方法上是不相同的. 本文针对 RTL 综合系统实现过程中的以赋值语句为核心的数据结构论述了 RTL 综合系统子程序的综合方法, 此综合方法已应用在北京理工大学 ASIC 研究所开发的 RTL 综合工具中.

2 RTL 综合中支持子程序的意义

VHDL 语言提供的子程序机制允许将设计中多次用到的一组语句定义为子程序, 在后面的描述中只要按照子程序名称以及对应的参数表来调用该子程序就可以执行这一组语句. 对于用户描述的一个具体设计来说, 子程序的使用不仅能够减小描述的规模, 而且更符合人们的思维习惯, 增强了描述的可读性, 符合 Top-Down 的设计方法, 加强了设计的可维护性, 同时方便了设计的重用, 提高了设计的效率.

此外, 在 IC 设计的很多情况中都是用 C 或者其他高级语言写出要设计实现的功能, 然后用硬件描述语言实现, 如果可直接使用硬件描述语言进行描述, 将会很大程度地提高设计效率. 因此, 在 VHDL 的 RTL 综合系统中提供对子程序的支持, 对于提高设计效率, 缩短设计周期是十分有效的一项措施.

3 子程序综合方法

3.1 嵌入方式

内嵌方式的具体实现是直接将子程序展开到调用的语句处, 相当于将子程序嵌入到调用主体中(并行语句, 如并行赋值语句及进程). 其基本思路为在构造 RTL 综合的数据结构时, 直接将子程序中的语句按照顺序添加到调用该子程序的地方.

嵌入方式使得子程序和调用主体之间的功能单元能够共享, 从而降低功能单元的数目, 而且直接嵌入减少了子程序与调用主体之间的通信时间, 保证了电路的速度, 但是由于对有 N 次调用的情况而言, 同样的控制逻辑要实现 N 次, 控制器的复杂度明显增大, 综合的时间耗费也会增大.

出于上述原因, 在实现子程序综合的过程中, 对于一次调用的子程序采用内嵌方式进行综合.

3.2 例示方式

例示方式的基本思路如下: 遇到有子程序调用时, 根据子程序调用找到被调用的子程序体, 对其进行综合, 对函数体和调用主体之间的接口进行处理, 然后将函数体综合的结果按照类似元件例示的处理方式加入到设计中, 从而实现一次定义多次调用的设计目标.

收稿日期: 2001-01-04; 修回日期: 2001-06-05
基金项目: 国家自然科学基金资助(No. 69973007)

这种处理方式综合出来的结果比较清晰, 符合设计者的思想, 模块化的特点比较明显, 综合的时间耗费也比较小, 而且电路的主体和子程序之间不相关, 可以在一方不执行的时候将电路关闭, 从而可以方便地降低电路的功耗.

对于多次调用的子程序, 可以采用例示方式进行综合.

4　子程序综合实现方案

4.1　数据结构的建立

4.1.1　赋值语句链表

将 VHDL 语言的 RTL 源描述按照自身语句的层次划分, 相互之间用指针链接起来, 形成双向链表, 这样就用指针将源描述的各个语句都链接起来, 通过某一条语句能够查找到另外一条语句的内容. 其中每一双向链表(包括纵向和横向) 的终点都是赋值语句.

得到语句双向链表后, 还需要构造综合所需的核心数据结构——赋值语句链表(如图 1 所示). 首先从双向链表的表据结构, 称为关联链表, 如图 2 所示. 图 2 中的关联链表主要由三个相互关联的链表组成: 信号/变量赋值语句链表中的内容除了图 2 所示的赋值语句及条件以外, 还根据各个赋值语句及相应条件在源描述中的层次及顺序在关联链表中设置赋值语句及条件的层号及赋值次序; 变量顺序链表是将信号赋值语句及变量赋值语句中出现的所有变量按照出现的次序提取出来, 同时提取的还有相应的条件、赋值/被赋值等内容.

为适应不同处理的需要, 信号赋值语句链表、变量赋值语句链表以及变量顺序链表三者之间相互关联, 有些内容是重叠的, 它们相互关联, 可以从一个链表得到其他两个链表所需的内容, 因此称为关联链表.

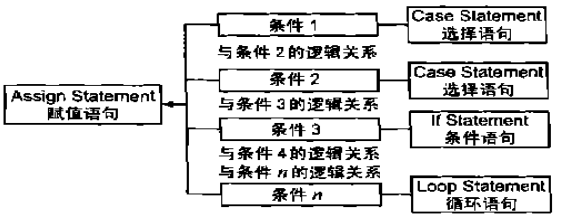

图 1　赋值语句链表

头开始查找, 一旦找到赋值语句, 就向前回溯, 找到赋值语句中目标信号的所有相关条件, 包括该赋值语句所有上层的 if、case 以及循环语句中相应的条件. 由图 1 可知, 最后有效的是赋值语句, 而在赋值语句之前的各种语句都可以看成是此赋值语句的条件, 因此可以建立以赋值语句为核心的中间数据格式.

4.1.2　关联链表

根据图 1 的赋值语句链表可以得到子程序综合的主要数

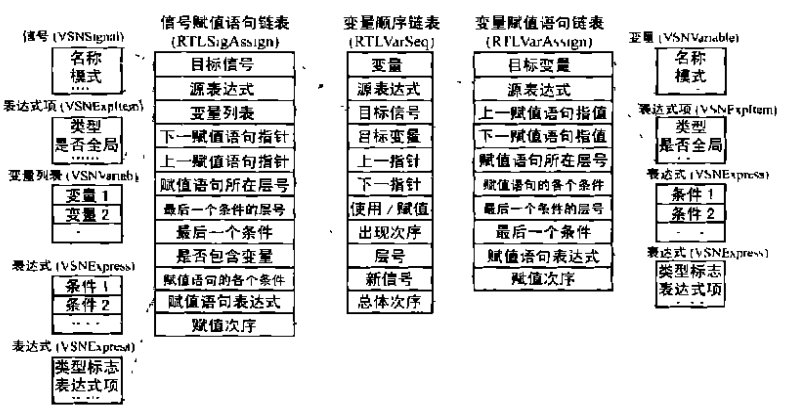

图 2　关联链表

4.2　嵌入方式综合算法

对一次调用的函数与过程采用嵌入方式进行综合, 主要的处理过程包括语句的嵌入、子程序内部局部变量的处理以及传入传出参数的处理, 之后还需要进行格式判别、组合逻辑综合及时序逻辑综合. 嵌入方式综合实现算法如算法 1 及算法 2 所示:

算法 1: 对一次调用子程序的处理算法

将所有并行语句都转化为等价进程;

分别将各个进程中的所有语句链接为双向语句链表;

num = 进程个数;

while (*num* ! = 0) do

{　*lStmntLst* = 进程语句链表头;

　if (*lStmntLst* 中的语句为子程序调用语句)

　{　得到子程序体;

　　得到子程序体中的局部变量列表, 并链接到进程的局部变量列表末尾;

　　lFuncStmntLst = 子程序体语句链表头;

　　while (*lFuncStmntLst* ! = *NULL*) do

　　{　if (语句为 RETURN 语句)

　　　调用算法 2 对返回值进行处理;

　　lFuncStmntLst = *lFuncStmntLst*→Next; }

　paraNum = 子程序形参的数目;

　for (int *I* = 0; *I* < *paraNum*; *I* + +)

　　将形参用子程序调用的实参代替;

　将整个子程序语句链表链接到子程序调用处;

　根据子程序调用语句的层号及顺序设置子程序体中各语句的层号及顺序; }

else

　lStmntLst = *lStmntLst*→Next;

num = *num* - 1; }

根据处理后的语句链表建立如图 2 所示的数据结构关联链表;

调用算法 3 进行格式判别、组合逻辑综合及时序逻辑综合.

算法 2: RETURN 语句中返回值的处理

(此算法主要讨论函数调用中返回值的处理, 过程体中也可能有 RETURN 语句, 但是 RETURN 语句中没有返回值, 因此

对于过程体中的 RETURN 语句, 只要将其直接链接到进程的语句链表中, 在算法 3 中再进行综合处理即可)

funcPtr = 函数调用对应的函数体; *lStmntLst* = 函数体语句链表;

while (*lStmntLst* ! = *NULL*)

{ if (*lStmntLst* 中的语句为 RETURN 语句)

{ *returnExp* = 返回值表达式; *newExp* = 新建赋值语句;

newExpSrc = *newExp* 的源表达式; *newExpTgt* = *newExp* 的目标表达式;

callTgt = 函数调用的目标表达式;

newExpTgt = *callTgt*; *newExpSrc* = *returnExp*;

将 *newExp* 替换原有 RETURN 语句链接到 *lStmntLst* 中; }

lStmntLst = *lStmntLst*→Next; }

4.3 例示方式综合算法

对于多次调用的子程序, 不能简单地将子程序体嵌入到调用处, 原因有三:

(1) 相同的逻辑(组合或时序)要实现 *N* 次, 效率不高, 实现上也过于繁复;

(2) 每一次调用的参数各不相同, 简单地将实参代替形参会破坏子程序体的完整性, 使得其他调用的处理发生错误;

(3) 综合的时间耗费过大.

基于上述原因, 对于多次调用的子程序采用例示方式进行综合. 例示方式综合实现算法如算法 3 所示:

算法 3: 对多次调用子程序的处理

将所有并行语句都转化为等价进程;

分别将各个进程中的所有语句链接为图 1 所示的语句链表; *num* = 进程个数;

while (*num* ! = 0) do

{ *lStmntLst* = 进程语句链表头;

建立图 2 所示的数据结构关联链表, 如果是子程序调用则先保留, 不做处理;

if (*lStmntLst* 中的语句为子程序调用语句)

{ if (该调用是该子程序的第一次调用)

{ 得到子程序体;

得到子程序体中的局部变量列表, 并链接到进程的局部变量列表末尾;

lSubStmntLst = 子程序体语句链表头;

while (*lFuncStmntLst* ! = *NULL*) do

{ if (语句为信号赋值语句语句)

{ 将所有形参与临时变量相连;

将该语句及相应条件添加到关联链表中的信号赋值链表的相应位置上;

信号赋值链表中的各个数据中的临时变量由实参替换; }

if(语句为变量赋值语句)

{ 将所有形参与临时变量相连;

将变量赋值语句及相应条件等内容添加到关联链表中的变量赋值链表的相应位置上;

将变量的相关内容提取出来添加到变量顺序链表相应位置;

链表中各个数据中的临时变量由实参替换; }

lFuncStmntLst = *lFuncStmntLst*→Next; }

}

else //不是第一次调用

{ 找到处理第一次调用后得到的关联链表;

记录链表中各个数据中的临时变量应当对应的实参; }

}

lStmntLst = *lStmntLst*→*Next*; *num* = *num* - 1; }

调用算法 3 进行格式判别、组合逻辑综合及时序逻辑综合.

4.4 格式判别、组合逻辑综合及时序逻辑综合算法

格式判别、组合逻辑综合及时序逻辑综合算法如算法 4 所示.

算法 4: 格式判别、时序逻辑综合与组合逻辑综合算法

(1) 如果是一次调用子程序, 转 2, 如果是多次调用子程序, 则进入子程序在关联链表中对应的内容进行综合, 并根据临时变量列表及实参表处理接口;

(2) 处理变量顺序链表中的每一个变量;

(3) 划分赋值语句块;

(4) 在赋值语句块中寻找赋值语句群;

(5) 在赋值语句群中寻找赋值语句组;

(6) 判别赋值语句组是组合逻辑还是时序逻辑, 如果赋值语句组是完整赋值语句, 则为组合逻辑, 否则为时序逻辑.

(7) 如果是组合逻辑, 则根据赋值的目标表达式、源表达式以及条件生成组合元件;

(8) 如果是时序逻辑, 则根据赋值的目标表达式、源表达式以及条件生成时序元件.

对算法 3 的说明:

(1) 步骤 2 是变量的综合, 是格式判别之前的预处理, 经过变量的综合后, 关联链表中的变量都被消除, 只保留信号赋值语句链表, 但功能与消除之前是等价的.

(2) 在步骤 3 中, 每一赋值语句都有对应的一组执行条件, 这些条件按层次加以区分. 假设当前块中第一条赋值语句具有 *n* 个条件, 且最后一个条件为 *Cond n*; 那么所有与该赋值语句处于同一块的赋值语句应满足下列条件, 根据这些条件划分赋值语句块:

(a) 赋值语句中最后一个条件(条件 *k*)的层次号相同;

(b) 赋值语句中最后一个条件应该相同或包含在 not (*cond n*)中.

(3) 在步骤 4 中, 将赋值语句划分为不同的块之后, 在每一个块中查找群. 设群的第一条赋值语句的层次号为 *m*, 那么所有与该赋值语句处于同一群的赋值语句的层次号应当都为 *m*.

(4) 在步骤 5 中, 设组中第一条赋值语句有条件 1、条件

2、……、条件 n, 那么所有与该赋值语句处于同一群的赋值语句应满足 a 和 b 两个条件, 根据这两个条件寻找赋值语句组:

(a) 条件 1 应相同或包含在 not($cond1$) 中;

(b) 条件 2、……、条件 n 都相同.

5　实验结果

表 1 中列出了一些实例[3,4] 的测试结果, 包括一次调用的函数与过程以及多次调用的函数与过程. 经过模拟验证, 实例 1、实例 2 以及表 1 中实例的测试结果都是正确的.

表 **1**　综合测试结果

序号	测试例	一次/多次调用	函数/过程	综合方式	综合结果				
					描述规模(行)	CPU 运行时间(秒)	功能单元数	网表规模(行)	模拟验证
1	Procall	多次	函数	例示	30	0.039	6	90	通过
2	Andor	多次	过程	例示	28	0.042	8	112	通过
3	Convert	一次	函数	嵌入	23	0.025	3	65	通过
4	Funcsig	一次	过程	嵌入	20	0.018	4	67	通过
5	Shifter	多次	函数	例示	71	0.389	32	229	通过
6	Lesser	多次	过程	例示	22	0.023	32	237	通过

6　结论

在本文论述的子程序综合方法中, 数据结构的建立、嵌入方式/ 例示方式处理、格式判别、组合逻辑综合以及时序逻辑综合是一个整体中的各个环节, 缺一不可, 只有按整个流程进行处理才是真正意义上的子程序综合. 关联链表是针对 RTL 综合建立的数据结构, 它简单明了, 层次清晰, 包含丰富的信息且可以适应不同处理的需要, 对后续环节的处理非常关键; 嵌入方式/ 例示方式处理是子程序综合的重点; 而格式判别、组合逻辑综合与时序逻辑综合是 RTL 综合的关键技术, 对任何类型的 RTL 描述的综合都是必不可少的. 文章后的测试结果表明包含上述各个环节的子程序综合方法是正确且有效的.

在子程序综合中还有一些重要的问题, 如预定义函数的处理, 递归函数的处理, 这些函数在结构及使用方式上与普通子程序有所不同, 且综合起来困难更大, 需要进一步探讨.

致谢: 感谢从本研究所毕业的张俭峰硕士在本课题中所做的前期研究工作.

参考文献:

[1] 谢巍, 袁媛, 刘明业. RTL 综合中的格式判别 [J]. 计算机学报, 2001(1): 99- 105.

[2] HWANG Cheng-tsung, WENG Hsiao-chien, HSU Yu-chin, Lee Mike Tien-chien. On the control-subroutine implementation of subprogram synthesis [A]. Proceeding of the Asia and South Pacific Design Automation Conference, (ASP-DAC) [C], Japan, 1997: 28- 31.

[3] IEEE P1076. 6. VHDL Register Transfer Level Synthesis-Level 1 [S].

[4] IEEE Std 1076-1993. IEEE Standard VHDL Language Reference Manual [S].

作者简介:

袁　媛　女. 1976 年 9 月出生于江西宜春, 1997 年获长沙铁道学院计算机软件专业学士学位, 1997 年起于北京理工大学计算机应用专业攻读硕博连读制博士学位. 研究领域包括电子设计自动化, RTL 综合等. E-mail: yuanr-y@ 263. net.

谢　巍　男. 1974 年 8 月出生于江西南昌, 1997 年获北京理工大学计算机应用专业学士学位, 1997 年起于该校同一专业攻读硕博连读制博士学位. 研究领域包括 RTL 综合、高级综合等.