VHDL-C++翻译器设计与实现*

吴清平, 刘明业

(北京理工大学 ASIC 研究所,北京 100081)
E-mail: wuqp@263.net
http://www.bit.edu.cn

摘要: VHDL(VHSIC(very high speed integrated circuit) hardware description language)是描述数字系统的硬件描述语言,C++是编写顺序语句程序的高级编程语言.VHDL 编译型模拟器需要采用具有顺序特征的 C++语句表征具有并发特征的 VHDL 电路设计.提出了一种面向对象的 VHDL-C++翻译方法,充分利用了这两种语言的面向对象的特征,采用 C++类来描述 VHDL 的实体、结构体及进程等元素,并通过一个 C++模拟调度核心完成了用顺序语句描述并发电路的工作.通过此方法可将 VHDL 源描述转化为功能等价的 C++代码,并在模拟调度核心的调度下,使用顺序语句模拟出数字系统并发功能,完成编译型模拟器的构造,实现 VHDL 的高速模拟.用这种翻译方法翻译出来的 C++代码具有结构清晰、可扩充性强的特点,与模拟核心形成的编译型模拟器的模拟速度相比,解释型模拟器速度有较大提高.该方法已在模拟系统中得以成功应用.最后给出了部分试验结果,进一步说明了算法的效率和优点.

关 键 词: VHDL(VHSIC(very high speed integrated circuit) hardware description language);翻译器;编译型模拟器;面向对象技术

中图法分类号: TP312 文献标识码: A

VHDL(VHSIC(very high speed integrated circuit) hardware description language)模拟器在 VLSI 高层设计验证中起着重要的作用.设计正确性的快速、有效的检查对加快整个设计流程至关重要.模拟技术的研究已非常深入,从模拟模型、模拟算法到其实现技术上都有很多的研究成果.在模拟模型方面,从最初的二值逻辑模型发展到三值零延迟模型、单位延迟模型以及多值模型[1];在模拟算法方面,从适合于中小规模设计的表驱动模拟算法发展到适合于大规模设计事件驱动的模拟算法[2]以及最新的适合于同步电路的周期模拟算法[3].而在算法实现技术方面,则由易于实现的解释型方法发展到更快速的编译型方法[4~6].

越来越大的设计要求在算法和实现技术上都要有重大的突破,在算法不断改进的前提下,对模拟算法的实现技术也提出了更高、更快的要求.原有的解释型模拟实现技术已经越来越不能满足设计的需要,编译型实现技术应运而生.编译型实现技术将 VHDL 设计转化为在宿主机器上可以直接执行的代码,通过执行这些代码完成模拟任务,它克服了解释型实现技术在模拟执行过程中重复性冗余操作,因而可以大幅提高速度.编译型实现技术可采用两种途径实现:一种是直接将 VHDL 设计转化为本地机器代码(native-code generation)[1],另一种是先将 VHDL 设计转化为其他标准编程语言,而后再经由这种编程语言的编译器生成机器代码.前者较后者有较快的翻译速度(因其步骤少)和执行速度(可考虑更多机器代码优化),但由于技术复杂,实现起来比较困难.而且,由于其与具体系统平台密切相关,可移植性差.因此,实际上多采用第 2 种途径实现.

本文通过探讨 VHDL 与 C++的异同点,提出了一种将 VHDL 翻译成为功能等价 C++代码的方法.已有的一

* 收稿日期: 2001-03-13; 修改日期: 2001-06-21
基金项目: 国家"九五"国防预研基金资助项目(8.1.1.13)
作者简介: 吴清平(1976－),男,湖南郴州人,博士,主要研究领域为 EDA 技术;刘明业(1934－),男,辽宁营口人,教授,博士生导师,主要研究领域为 EDA 技术.

些翻译方法,有的采用面向函数[4,5]的方式,有的通过指针形式将不同的元件对象连接起来[6],都容易产生层次不清、指针结构混乱、对象访问不方便等问题.与这些方法不同的是,我们的方法基于面向对象的思想,采用类对象表示 VHDL 中任何元素,并使用成员对象的形式构造整个设计,而不使用容易混淆结构的指针形式,使翻译形成的 C++代码具有很好的可读性,并且具有良好的可重用性及可扩充性.本文同时提出了最终生成可模拟执行文件的方法.这种方法在我们自主开发的数字高层次自动设计工具(Talent2000)系统中得到了应用和实现.本文最后将给出一些具体的测试实例及性能比较.

1 VHDL 与 C++语言

VHDL 是在 80 年代初由美国国防部提出的,用于在各开发成员间统一设计文档标准,经过多年的发展,后被 IEEE 接纳为 IEEE1076 标准.C++语言是由贝尔实验室在 C 语言的基础上加入了面向对象的机制发展而来的.它融入面向对象封装性、继承性和多态性的特点,具有结构清晰、功能强大、扩充性强等优点,特别适合于大型软件的设计.

VHDL 和 C++语言都属于高级语言,但是它们有明显的区别,最主要的差别在于,VHDL 是一种硬件描述语言,描述的是一种并发的电路系统;而 C++语言是一种编程语言,描述的是顺序执行的指令序列,它不具备描述硬件结构的语法和功能,如时钟、并发行为等.例如,VHDL 中所有的进程(process)均是并发执行的,在同一个模拟周期内执行的所有进程之间是相互没有影响的,因此它们的模拟次序是无关紧要的;而 C++语言中的函数都是串行执行,各个函数的执行顺序对最终的结果是有影响的.这也是两种语言本身的用途所决定的,VHDL 描述的是硬件电路,电路的各个部分在同一时刻是可以同时动作的;而 C++语言所形成的是运行于一个 CPU 上的一些指令序列,它们只能一条一条地被执行.但是, VHDL 与 C++语言也有很多相似之处,可以作为 VHDL 向 C++转换的基础,归结为如下几点:

(1) VHDL 中保存值的量有 3 种:常量、变量和信号.在 C++中没有信号这个概念.在 VHDL 中,信号与变量的最主要区别在于:当被赋值时,变量值立即有效;而信号值则是在下一个模拟周期时才有效.为了用 C++统一表示它们,可以使用类来封装它们,然后在类中定义一个标识类型(常量、变量或信号)的标记,并在类的成员函数中在操作时针对不同的类型作不同的处理,这样就可以保证操作方式的统一性.

(2) VHDL 以实体配合结构体的方式定义一个设计单元,并通过元件例示的方式将不同的单元连接起来,以组织整个设计.这与 C++程序由对象组成的思想完全相同:C++以类的形式封装一些数据和函数,使用类来定义对象用以表示一个设计单元,并且可以通过类成员对象的形式完成整个程序的组织.这就是说,VHDL 中在一个结构体中例示另一个实体在概念上可以等同于 C++中在一个类中定义另一个类的对象.这样,以 C++类对象来描述 VHDL 中的元件,既符合面向对象的编程思想,又能较清晰地保存 VHDL 描述中设计的层次关系.

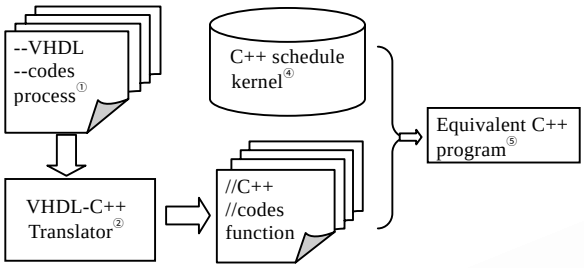
(3) VHDL 中最重要的结构是进程,也是模拟调度的最基本单元,是 VHDL 并发性的集中体现.虽然在 C++中没有直接与此对应的结构,但是可以使用 C++的类成员函数来对它进行模拟.VHDL 中的进程是一个循环过程,因此可以在 C++函数中增加循环语句来模拟进程的重复执行效果,使用一定的机制实现进程的挂起与恢复功能,并且使用一个调度核心对所有的函数在适当的时候进行调用来模拟进程的并发行为.

(4) 在 VHDL 中,行为描述中的顺序语句:变量赋值语句、IF 语句、CASE 语句、LOOP 语句、NEXT 语句、EXIT 语句、RETURN 语句、NULL 语句以及过程调用语句在 C++语言中均有类似的语法,作少量的修改即可转换为 C++语句.

2 常量、变量和信号的翻译

在将 VHDL 源描述转化为 C++代码中的过程中,首要的问题是如何用 C++的顺序语句来表征 VHDL 的并行特性,前面介绍了将 VHDL 中的进程转化为 C++的类成员函数的思想,但是单纯按照一一对应关系将 VHDL 源代码转化为 C++代码是不可能体现并行特性的,这必须借助于一个模拟调度核心来完成此工作,模拟调度核心的主要作用是在模拟过程中在适当的时候调整个设计中的各个由进程转化来的 C++类成员函数,以模拟进

程的并发执行过程,可以使用事件驱动(event-driven)的模拟算法来实现.翻译器与模拟核心之间的关系可以用图 1 来表示.



①VHDL 代码,②VHDL-C++翻译器,③C++代码,④C++调度核心,⑤等价 C++程序.

Fig.1 Relation between translator and schedule kernel

图 1 翻译器与调度核心关系

3 翻译器与调度核心

在 VHDL 中可用于保存值的量按照其对象类型分为“常量”、“变量”和“信号”3 类,并且,任何一个量还有其数据类型特征,如整型、BIT 型或数组、聚集等类型.数据类型用于说明此数据可以保存的值的类型,而对象类型用于说明可在此数据上进行的操作的方式,如变量可以被赋值,常量不能被赋值,信号在赋值之后的下一个 delta 周期后其值才有效.对于 C++语言来说,由于不存在“信号”类型这个概念,因此对于“信号”的翻译需要作特殊处理.

首先,在表示方法上以 C++的类对象对应 VHDL 中的常量、变量和信号对象.为每一种 VHDL 中的数据类型定义一个类,在类中定义一个标识符用以区分信号、变量和常量,通过以数据类型类定义对象的方式定义数据量,并通过向构造函数传递参数来区分信号、变量和常量.例如,下面的 VHDL 源描述左侧可翻译为右侧对应的 C++代码.

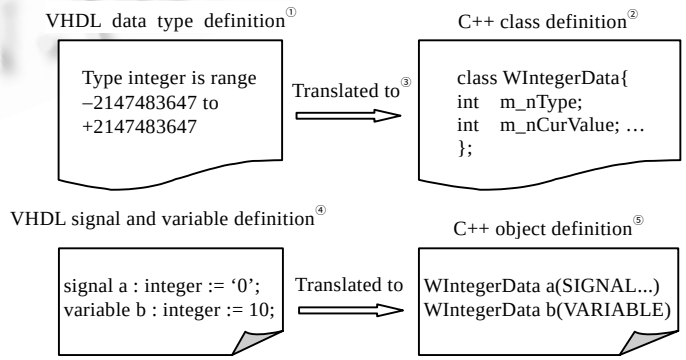
```
signal a : bit := ‘0’;                  ⇒   WBitData a ( “a”, QVHDLObject::SIGNAL, … ,‘0’);
variable b : integer := 10;                 WIntegerData b ( “b”, QVHDLObject::VARIABLE, … ,10 );
constant c : integer := 20;                 WIntegerData c ( “c”, QVHDLObject::CONSTANT, … , 20 );
```

其中 WBitData 和 WIntegerData 分别为 VHDL 中的 bit 和 integer 类型定义的两个类,其对应关系如图 2 所示;CONSTANT,VARIABLE,SIGNAL 是定义在类 QVHDLObject 中的 3 个类静态常量,用以区分常量、变量和信号.也就是说,每一个对象除了保存它的值以外,还保存一个量用以区分它的对象类型,这样在对它进行操作时可根据对象类型区别对待.

为了使自动生成的 C++源代码的可读性更高,对于信号、变量的赋值语句通过 C++的‘=’运算符的重载来实现.在重载了‘=’运算符之后,所有 VHDL 中的对于信号或变量的赋值语句都转化为 C++中的‘=’赋值语句,如 VHDL 的信号赋值和变量赋值语句:

```
a <= ‘1’;
b := 30;
```

均可以转化为 C++的对象赋值语句:



①VHDL 类型定义,②C++类定义,③被翻译为,④VHDL 信号、变量定义,⑤C++对象定义.

Fig.2 Translation of the definition of data type, signal and variable

图 2 数据类型、信号、变量定义的翻译

```
a = '1';
b = 30;
```

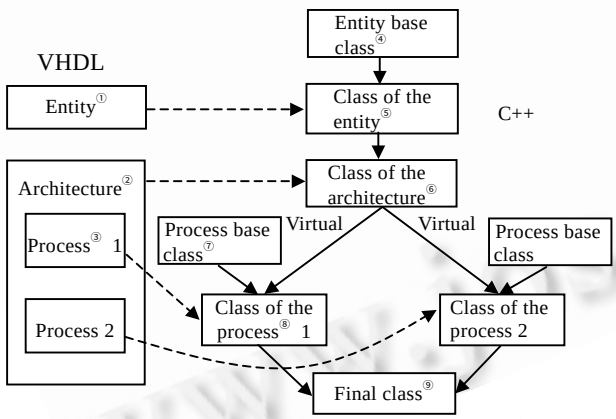
在各种数据类型类中均要求实现对于'='运算符的重载,在这些重载函数中首先要区分对象的类别,根据不同的类别作不同的操作.如与 integer 类型对应的 WIntegerData 类中的重载函数如下:

```
WIntegerData& WIntegerData::operator=(const int    source)
{
    判断 source 是否合法数据
    if (本身是变量)
        直接将参数的值赋予自己
    else if (本身是信号)
        将赋值事件加入全局事件链表
    else if (本身是常量)
        报错,常量不允许赋值
}
```

函数首先判断源值是否是合法数据,如是否在数据类型允许的大小范围内等,然后区分对象类型,如果是信号,则根据事件驱动算法要求生成一个信号更新事件,加入到全局事件链表中,等到下一模拟周期时才对信号的值进行更新;如果是变量,则直接进行赋值;如果是常量,则报告错误.

4 实体、结构体的翻译

实体、结构体是 VHDL 组织整个设计的基本单元,通过在一个实体所对应的结构体中例示另一个实体来形成设计的结构,同时形成设计的树状层次关系.翻译过程的一个首要问题是如何正确而有效地表示源设计,使得在模拟过程中数据的访问方便、高效.已有的一些翻译方法使用结构加指针的方式[5]来完成,这会使翻译结果关系复杂化,结构与源设计相差甚远,导致观察、调试上的困难.本文采用的是对象加引用的方式,采用类对象来表示 VHDL 中的设计单元,通过类成员对象来表示 VHDL 中的元件例示关系,这样就保持了与源设计相同的层次关系,结构清晰、便于观察调试,同时也使模拟调度核心库[7]的设计变得简单.



Base class tree of the final class⑩

Solid line represents the inheritance relation of classes; dashed represents the translation mapping(11).

①实体,②结构体,③VHDL 进程,④实体基类,⑤实体对应的类,⑥结构体对应的类,⑦进程基类,⑧两个进程分别对应的类,⑨终结类,⑩终结类的基类树,(11)实线为派生关系,虚线为翻译转化关系.

Fig.3 Translation mapping from entity, architecture and process to object classes

图 3 实体、结构体、进程和对象类的翻译对应关系

主要思想是通过将VHDL的实体和结构体翻译成对应的 C++类来完成,在 C++实体类中描述设计的接口,C++结构体类中定义在 VHDL 结构体中声明的信号等数据,并且由于结构体是实体的功能实现,它拥有实体所有的属性,这与 C++中的派生关系相同,因此可使 C++中的结构体类从实体类中进行派生.注意到结构体中的进程是模拟的最基本调度单元,它具有某些共用特征,因此定义了一个用以描述进程行为特征的类 QProcess,每一个结构体中的进程都被翻译为一个类,并且让其从 QProcess 中派生;同时,进程又属于结构体,它可以使用结构体中定义的数据(如在结构体中定义的信号等),于是又使进程类从对应的结构体类中派生.当一个结构体中包含多个进程时,为了避免其对应的多个进程类均从同一个结构体类中派生所造成的父类(结构体类)对象的重复,我们使用了 C++虚(virtual)派生方式巧妙地解决此问题(如图 3 所

示进程类的派生关系).最后,为了使得所有这些类(实体类、结构体类、进程类)最终组合成一个对应与 VHDL 中一个元件的概念,可以再定义一个类(可以称之为终结类),并让其从所有进程类中派生,这样将所有的类组合在一起,一个终结类对象的定义就相当于一个设计单元的例示,对应关系如图 3 所示.

这种方法充分利用 C++的封装性和继承性的特点,每一个实体对应于一个实体类,每一个结构体对应于一个结构体类,每一个进程对应于一个进程类.并且为了提高代码了利用率,定义了实体基类和进程基类,将实体和进程所具有的特征封装在其中,如将进程在模拟调度时所需要的一些函数与变量封装在进程类中,所有的进程均由它进行派生,这不仅减少了代码的编写工作,同时也便于扩充和修改.图 4 给出了一个具体的翻译实例,进一步说明了这种转化关系.

5 进程翻译

进程是模拟器调度的基本单元,进程的翻译对于模拟调度起着关键作用.我们通过将进程语句翻译为函数完成了功能上的表示,而后需要解决的主要问题是如何通过 C++函数的顺序特征实现 VHDL 进程的并发特征.VHDL 进程并发性体现在进程可以通过 wait 语句将自身挂起,进而使其他同一时刻激活进程得以执行,并且在适当的时候,此进程还会被激活并恢复执行.而在标准 C++语言中没有这样的功能,必须采用一定的机制来模拟此过程.

前面讨论到将结构体中的每一个 VHDL 进程都翻译为一个 C++进程类,这样在 VHDL 进程中定义的对象等均可以直接定义为 C++进程类中的成员变量,而进程中的顺序语句则翻译为进程类的一个成员函数(*Simulate*()函数).模拟器所调度时以进程类对象为调度对象,当进程被激活需要模拟时,模拟将调用进程类对象的 *Simulate*()函数.

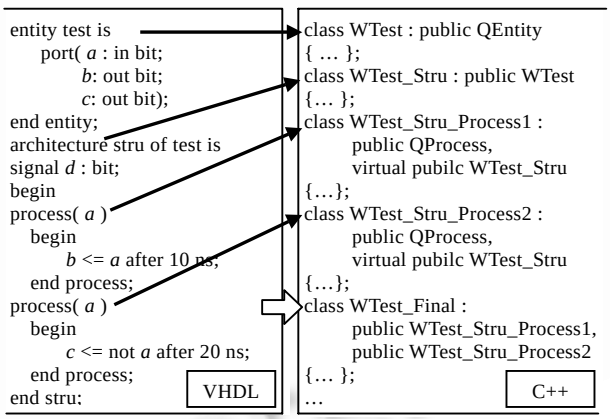


Fig.4 A translation example of entity, architecture and process

图 4 实体、结构体、进程翻译实例

我们所设计的类层次结构,为进程的挂起与恢复提供了很大的便利.实现进程类中函数的挂起和恢复主要需要解决两个问题:第 1 个是运行上下文的保存与恢复问题,第 2 个是模拟当前位置的保存与恢复问题.下面分别就这两个问题进行讨论.

由于 VHDL 中的实体、结构体及进程中定义的信号、变量等均转化为对应 C++类的成员变量,因此,如果在类成员函数 *Simulate*()中不定义任何局部变量,则可以证明此函数挂起和恢复时就不存在任何运行上下文的保存与恢复工作.这是因为类成员变量在类对象未消亡之前是不会消亡的,而这里的实体、结构体类对象在整个模拟过程中显然都是存在的,因此,类成员函数的挂起和恢复对类成员变量不会产生任何影响,故无须任何保存或恢复上下文的工作.

对于第 2 个问题,由于任何一个进程中 WAIT 语句的个数在模拟之前是确定的,因此进程的挂起和恢复的位置个数也是确定的,在确定了可能的挂起位置个数之后,此问题可以转化为函数的多入口及多出口问题.即当进程存在一条 wait 语句时,它转化为一个由 1 个入口 1 个出口的函数;当进程存在两条 WAIT 语句时,它转化为一个由 2 个入口 2 个出口的函数,以此类推.当函数模拟到 wait 语句(函数的出口)时,就从函数中返回,当需要恢复进程继续模拟时,从上次返回的出口的下一个入口进入,通过这种方式完成函数的挂起与重入.由于在此成员函数中不定义任何局部变量,而仅仅操作类成员变量,此成员函数完全是代码的集合,前面已经证明这种挂起和恢复的方式是正确的.为了保存函数在下次进入时的入口,在进程基类中定义变量 *m_nMaxEntrance* 保存此函数的最大入口数、*m_nCurEntrance* 保存当前入口位置,通过这些变量跟踪函数的挂起和恢复状态.

函数的多入口和多出口的实现方式是多种多样的,这里采用 C++的 switch 开关语句来完成此工作.由于每个进程的入口数是确定的,各个入口之间的语句也是确定的,因此这种转化是比较直观、简单的.图 5 例可以说明这种转化关系.其中,变量 *m_nCurEntrance*、函数 *waitfor*()、*waiton*()等都是在进程基类 QProcess 中定义的,并且在函数 *waitfor*()和 *waiton*()中完成了对函数下次入口的调整:使变量 *m_nCurEntrance* 加 1,由于进程是一个无限循环的过程,当 *m_nCurEntrance* 超过此函数的最大入口时,恢复到函数的第 1 个入口,例如以下函数:

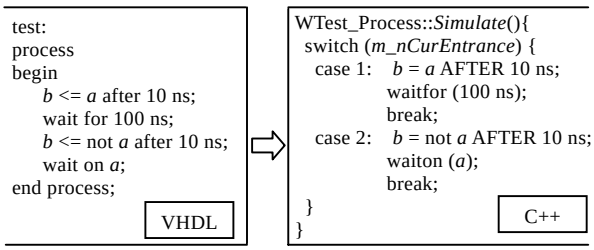


Fig.5 A translation example of process
图 5 进程的翻译实例

```
QProcess::MoveToNextEntrance()
{
    m_nCurEntrance ++;
    if (m_nCurEntrance>m_nMaxEntrance)
        m_nCurEntrance=1;
}
```

正如前面讨论到的,在进程基类 QProcess 中定义进程调度所需的变量、函数,使得整个翻译工作简单、便捷,又便于升级修改,如当需要对模拟调度过程进行某些修改时,通过修改 QProcess 即可完成,而无须对所有的翻译结果进行修改,这正是面向对象设计的优点.

6 实验结果分析

上述方法已在我们自行开发的 Talent2000 数字系统高层次自动设计工具中实现,并使用本系统进行了多例测试,表 1 给出了 5 个典型实例的模拟结果.每一个测试例都首先使用了系统原有的解释型模拟器进行模拟,然后使用按本文提出的翻译方法实现的编译型模拟器进行模拟,它先将 VHDL 设计源描述转化为 C++代码,经过 Visual C++6.0 编译,最终与调度核心代码链接成可执行文件.表 1 列出了两种方式的模拟结果,并给出了编译型模拟较解释型模拟的提速比例.所有测试均在 Windows 98 环境下完成,机器主频为 400.

Table 1 Efficiency comparison between the two simulations
表 **1** 两种模拟效率的对比

Name①	Function②	Interpreted simulation③ (s)	Compiled simulation④ (s)	Promotion ratio⑤ (%)
ellipf	Five stage ellipse filter with 175 test vectors⑥	12.8	3.4	276
decoder_stru	Structure description 2-4 decoder with 575 test vectors⑦	13.5	4.9	175
decoder_beha	Behaviors description 2-4 decoder with 1016 test vectors⑧	8.5	4.7	80
odd_even	Parity checker with 564 test vectors⑨	9.6	4.6	108
tlc	Traffic light controller with 368 test vectors⑩	8.4	3.9	115

①测试例名,②功能,③解释型模拟,④编译型模拟,⑤提速比例,⑥五阶椭圆滤波器,175 组测试向量,⑦结构描述二四译码器,575 组测试向量,⑧行为描述二四译码器,1016 组测试向量,⑨奇偶校验器,564 组测试向量,⑩交通灯控制器,368 组测试向量.

在所做的测试例中,编译型模拟速度与解释型模拟速度相比都有不同程度的提高,这与编译型系统的自身优点是分不开的,也充分说明前面所述 VHDL-C++翻译方法所生成的 C++代码具有高效率的特征.

7 小 结

本文阐述了一种面向对象的 VHDL 源代码转化为 C++代码的方法,通过这种方法生成的 C++代码保留了原 VHDL 设计的层次结构关系,具有层次清晰、可读性好、高效等特点.由此构造的编译型模拟器具有模拟速度高、可扩充性强的优点,为超大规模集成电路的高速模拟提供了有效的技术.

References:

[1] Birger, A. The state of simulation in russia. In: Proceedings of the 30th ACM/IEEE Design Automation Conference. New York: ACM Press, 1993. 712~715.

[2] Ganguly, N. HSIM1 and HSIM2: object oriented algorithms for VHDL simulation. In: Proceedings of the 7th International Conference on VLSI Design. IEEE Computer Society Press, 1994. 175~178.

[3] Palnitrak, S. Cycle simulation techniques. In: Proceedings of the IEEE International Verilog HDL Conference. IEEE Computer Society Press, 1995. 2~8.

[4] Maurer, P.M. Two new techniques for unit-delay compiled simulation. IEEE Transactions on Computer-Aided Design, 1992, 11(9):1120~1130.

[5] Lewis, D.M. A hierarchical compiled code even-driven logic simulator. IEEE Transactions on Computer-Aided Design, 1991, 10(6):726~737.

[6] Bian, Ji-nian, Chen, Jing. V2C++——the C++ implementation of a VHDL translation simulator. Journal of Computer-Aided Design and Computer Graphics, 1998,10(2):167~172 (in Chinese).

[7] Darmont, J. DESP-C++: a discrete-event simulation package for C++. Software-Practice and Experience, 2000,30(1):37~60.

附中文参考文献:

[6] 边计年,陈菁.V2C++——一个用 C++实现的 VHDL 翻译型模拟器.计算机辅助设计与图形学学报,1998,10(2):167~172.

Design and Implementation of a VHDL-C++ Translator*

WU Qing-ping, LIU Ming-ye

(*ASIC Research Center*, *Beijing Institute of Technology*, *Beijing* 100081, *China*)

E-mail: wuqp@263.net

http://www.bit.edu.cn

Abstract: VHDL (VHSIC (very high speed integrated circuit) hardware description language) is a language for the description of digital hardware system, and C++ is a programming language for coding sequential statements. VHDL compiled simulators use sequential C++ language to model circuits in VHDL with concurrent characterization. An object-oriented method of translating concurrent VHDL codes into sequential C++ codes is presented in this paper. This method takes the object-oriented characteristic of the two languages into consideration and makes the translation very smooth. Using class of C++ to model entity, architecture and process of VHDL, and combining with a simulation kernel, it accomplishes the job of modeling concurrent actions using sequential statements. By this method, VHDL codes can be translated to C++ codes with the same function, and the C++ codes then can be compiled and linked with simulation kernel code to an executable file, which is the compiled simulator. The execution of this file is the simulation of the design of VHDL. This method is well-structured and easily-extended, and the simulator got by this method is more efficient than the traditional one. This method has been successfully applied in the simulator. The performance and efficiency of the method are verified at the end of of this paper.

Key words: VHDL (VHSIC (very high speed integrated circuit) hardware description language); translator; compiled-simulator; object-oriented technique

* Received March 13, 2001; accepted June 21, 2001
Supported by the Defence Pre-Research Project of the 'Ninth Five-Year-Plan' of China under Grant No.8.1.1.13