基于同态映射的从 UML 导出可综合 Verilog 算法*)

沈筱彦[1,2]　陈　杰[1]

(中国科学院微电子研究所　北京 100029)[1]　(山东大学物理与微电子学院　济南 100250)[2]

摘　要　UML 建模因其可显著提高开发效率和代码质量已经成为软件开发领域的一大热点，而硬件设计的日益复杂性也要求我们在更高层次抽象上分析和验证系统行为，故更精细的系统级建模方法变得日趋重要。本文构建了 UML 元模型与可综合 Verilog 间的同态映射，定义了一个从 UML 模型子集导出可综合 Verilog 描述的算法，为 UML 模型对于建模硬件系统提供了形式化的语义，从而使运用 UML 进行硬件系统级建模和系统级上验证系统性能和功能正确性成为可能。

关键词　统一建模语言(UML)，Verilog 硬件描述语言，同态映射

A Homomorphic Mapping Based Algorithm to Generate Synthesizable Verilog from UML

SHEN Xiao-Yan[1,2]　CHEN Jie[2]

(Microelectronics Research and Development Center of the Chinese Academy of Science, Beijing 100029)[1]

(Dept. of Physics and Microelectronics, Shan Dong University, Ji'nan 250100)[2]

Abstract　Modeling with UML has been a hot topic in software development domain since it can significantly improve product quality and productivity. But the constantly increasing complexity of hardware design also demands analysis and verification of system behavior on higher levels of abstraction, so more elaborate system - level modeling techniques are more and more important. In this paper, a homomorphic mapping between UML metamodel and synthesizable Verilog is constructed and a algorithm for deriving synthesizable Verilog specification from a subset of UML models is defined. So it is possible to use UML for hardware modeling and the performance and functionality correctness verifying at system level.

Keywords　UML, Verilog hardware description language, Homomorphic mapping

1　引言

硬件系统设计的日趋复杂使得因需求分析错误导致设计功能错误的风险日益增大，因此使用 UML 对硬件系统建模和验证成为近两年来国际上硬件设计方法学的一个研究热点。Zhu Qiang 等人对使用 UML 的硬件系统设计流程和验证方法做了深入的研究[1,2]。Greaves 和 Soderman 的工作使通过 C 语言从 UML 到硬件描述语言之间相互转换成为可能[3,4]。Mcumber 提出了从 UML 导出 VHDL 模型的算法[5]。由于他们的工作大多仅得到一种不可综合的硬件模型或需要中间语言转换，实用性较差。我们构建了一个 UML 元模型到 Verilog 的同态映射，通过它定义了一个从 UML 图导出可综合 Verilog 描述的算法，从而前端设计师能直接协同运用直观的 UML 和规范的 Verilog 进行硬件建模，确保需求分析的正确性。

本文剩余部分组织如下：第 2 部分说明 UML 元模型及其与 Verilog 间的同态映射；第 3 部分定义了从 UML 类图和状态图导出可综合 Verilog 描述的算法；最后是总结及相关讨论和进一步的研究。

2　UML 元模型到 Verilog 同态映射的构建

2.1　构建同态映射的意义

从 UML 半形式化模型导出 Verilog 代码的关键在于对于 UML 产生精确的语义。通过建立一个 Verilog 的元模型，语义的产生就是构建一个严格的从 UML 元模型到 Verilog 元模型的同态映射来完成的。任何从 UML 导出 Verilog 的规则都不能违背同态映射，这样就保证了导出的 Verilog 描述为 UML 模型提供了精确的语义。

2.2　UML 元模型

UML 元模型[6]是用来描述 UML 图语法的一个 UML 类图，其中的类代表了该种 UML 图的语法构件。我们认为，描述系统静态结构的类图和描述动态时序行为的状态图已足够用于硬件系统的高层次建模，而这两类图在 UML 规范中是用两种分离的元模型描述的。因为在单一的一个 UML 元模型上建立到目标描述语言的同态映射较为方便，故需要将两个元模型合并为一个统一的元模型[5]，如图 1 所示。图中矩形代表类；连线代表关系。关系的类型由其上的多重集决定，为函数、单射、满射和全射四种基本类型的组合[7]，定义为 Type。

2.3　UML 到 Verilog 同态映射的概念

令 O 为元模型；$C=\{C_i \mid C_i \in O\}$ 为 O 中的类集；$D=\{x \mid x \in C_i\}$ 为所属 C 中类的实例集；$\mathrm{Re}=\{(C_{i1}, C_{i2}, \mathrm{type}) \mid C_{i1}, C_{i2} \in O$，且 C_{i1}, C_{i2} 间存在类型为 type 的关系$\}$ 为 O 中的关系。

定义 1　定义类预测函数为：bool $T(C, D)$，使得 $T(C_i, x)$ 等于 ture 当且仅当 $x \in C_i$。即：

$$T(C_i, x) \Leftrightarrow x \in C \tag{1}$$

定义 2　定义关联预测函数为：bool $R(\mathrm{Type}, D, D)$，使得 $R(\mathrm{type}, x, y)$ 等于 ture 当且仅当 x, y 所属类在元模型中存在类型为 type 的关联。即：

$$R(\mathrm{type}, x, y) \Leftrightarrow (x \in C_{i1} \wedge y \in C_{i2} \rightarrow (C_{i1}, C_{i2}, \mathrm{type}) \in R_e) \tag{2}$$

*)基金项目：863 重点项目《32 位高性能嵌入式数字信号处理器(DSP)芯片设计与实现》(项目号：2002AA1Z1130)。沈筱彦　硕士研究生，研究方向为数字集成电路设计和视频会议；陈　杰　研究员、SOC 实验室主任、博士生导师，主要研究方向为无线通信系统、数字大规模集成电路等。

定义 3 定义 UML 元模型到 Verilog 元模型的同态映射为能够在建立的 Verilog 元模型上保留 UML 元模型类间对应关系的映射。即令 $x,y\in D;A,B\in C$,UML 元模型到 Verilog 元模型的同态映射 h 为满足如下关系的映射:

$$\forall x,y(T(A,x)\wedge T(B,y)\wedge R(\text{type},x,y))\Rightarrow T(A',h(x))\wedge T(B',h(y))\wedge R(\text{type},h(x),h(y)) \quad (3)$$

图 2 给出了我们建立的 Verilog 元模型。依该元模型生成的代码都是可综合的,不仅方便验证,而且大大缩短了前端设计的周期。对于算法的进一步完善补充和改进,将能有效填补系统级设计和 RTL 代码间的鸿沟。

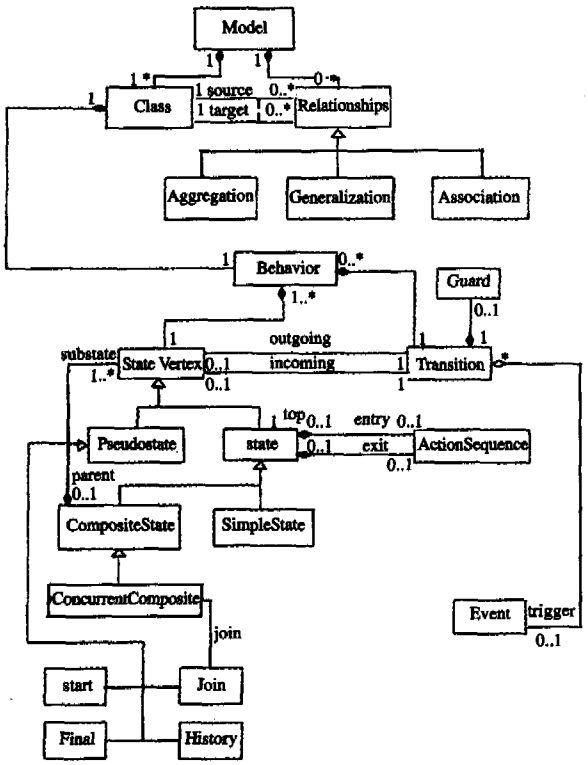

图 1 统一的 UML 类图和状态图元模型

3 导出可综合 Verilog 算法

我们设计了一个满足第 2 节构建的同态映射的从 UML 类图和状态图导出可综合 Verilog 的算法。限于篇幅,我们仅列出较主要的部分。

3.1 UML 类图导出算法

对类图进行拓扑排序,要求每个泛化关系的子类在拓扑排序中位于父类之后,并且每个聚合关系的全局类位于部分类之后。

定理 1 对类图按泛化和聚合关系进行拓扑排序,一定存在一个序列 s,使得每个泛化关系的子类位于父类之后,并且每个聚合关系的全局类位于部分类之后。

证明:假设不存在这样的一个序列,则说明在类图中存在环 A,B,…,D,A。由于泛化和聚合关系都是传递性的,所以推出 A 同时是 A 的父类和子类或 A 同时是 A 的部分和整体,显然这是错误的,故存在满足要求的拓扑序列 s。

```
for (顺序取拓扑排序中的每个类 C){
  if (没有产生与 C 同名的模块){产生与类 C 同名的一个如下模块:
    module Cname_m (端口表);
      端口声明;变量声明;模块语句;
    endmodule }
  for 每个属于类 C 的变量 X{
    if (X 是 public)
      在端口声明中增加该变量;
    else 在变量声明中增加该变量;}
for 每个与类 C 相关联的关系 R {
  if (R 是聚合)
    模块语句部分增加子模块实例化语句。
  else if (R 是泛化,C 为父类){
    if (泛化子模块不存在)
      将类 C 的模块复制并改名为泛化子类的名字;
    else 插入 C 的端口和变量;}
  else if (R 是关联,C 为源端)
    在端口声明中增加不重名信号 t,端口类型为 output。
  else if (R 是关联,C 为宿端)
    在端口声明中增加不重名信号 t,端口类型为 input。
  else continue;}}
```

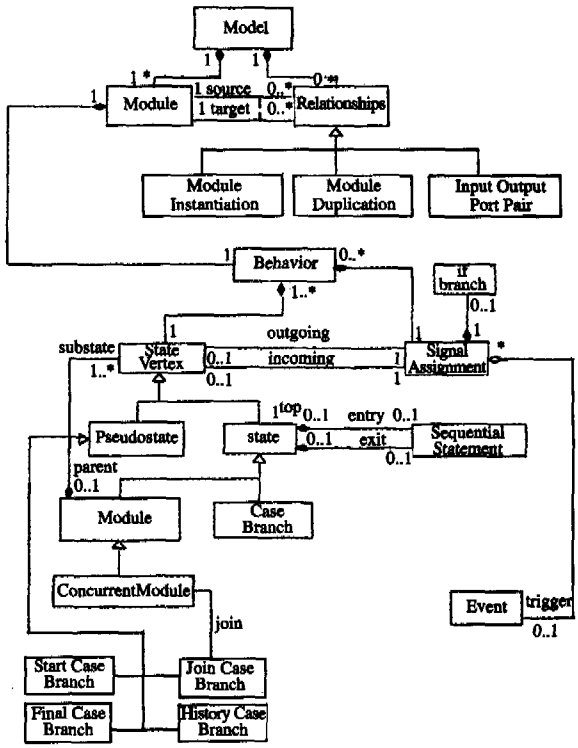

图 2 可综合 Verilog 元模型

定理 2 该算法满足 class 到 module 的同态映射,对于三种关系的导出代码也满足相应的 source/target 关系。

证明:class 到 module 的同态映射是显然的。又一个模块可以包含 0 个或多个实例化语句;可以在 0 个或多个模块中实例化;可以复制成多个模块;可以由多个模块合并而来;可以有 0 个或多个作为关联源态的输出端口;还可以有 0 个或多个作为关联宿态的输入端口。所以,它们满足相应的 source/target 关系。

3.2 UML 状态图导出算法

新建一状态名宏文件,初始化初始和终结状态对应的宏 START 和 FINAL 和 Finish 常量;

调用子程序 1;

```
for (广度优先遍历每个状态 S){
  if (不存在以 S 名为宏的常量 MS)
      宏文件中增加表示 S 状态名的常量 MS;
    在子程序 1 生成的两个 always 语句的 case 分支部分插入一个以 MS 为分支项值的分支;
  if (S 是简单状态) call 子程序 2;
  else if (S 是复合状态){
      if (S 含非正交状态)
        call 子程序 3 生成新的模块;
      else (S 是只含正交状态){
          for (每个 S 中的正交状态 C)
            call 子程序 4 为每个正交状态生成一个新的模块;}
          记录所有以正交状态为宿的转换源状态,在它所对应的活动语句中增加将所有子程序 4 产生的 css 信号赋值为正交状态对应的宏;}
    else 调用子程序 5;}
```

初始化子程序 1;

在调用该函数的模块的端口部分声明一个名为 cs 的 reg

型输出端口;在变量部分声明一个名为 ns 的 reg 型变量;在语句部分生成两个如下形式 always 语句块,其中第一个 always 语句的触发事件是时钟信号,第二个的触发事件待定。

```
always (posedge clk)
  begin
    case(cs)case 分支部分; endcase
    cs <=ns;
  end
always (触发事件表)
  begin
    case (cs)case 分支部分;
    default: ns <= 默认态;endcase
  end
```

生成简单状态对应代码的子程序 2:

S 的活动语句放入第一个 always 中 S 的对应分支中;

在活动语句后插入如下代码:

```
case (ns)case 分支语句;default: //空语句
for (以 S 为源的每个转换 T){
  宏文件中加入 T 次态宏常量 TS;
  在 case (ns)分支部分插入以 TS 为分支项的分支,T 的动作序列为其语句部分;
  在第二个 always 语句触发事件表中插入 T 的监护条件和触发事件,在其 case 分支部分插入以 TS 为分支项的分支,分支语句为如下形式:
  if (! 监护条件)ns = cs
  else if(触发事件)ns = TS;}
```

生成含非正交状态对应模块的子程序 3:

在宏文件中增加相应常量;

在父模块中新增不重名变量 child;

生成如下形式的模块:

```
module Sname_m (pcs,…,state);
  input wire pcs; 其他父模块信号;
  output reg state; 模块语句部分;
endmodule
```

在父模块中实例化该模块,将 cs 信号映射到 pcs 端口,child 变量映射到 state 端口,并映射其他相应端口;

call 子函数 1;

```
for 按广度优先遍历每个子状态 S
  if S 为简单状态、复合状态和伪状态
    分别 call 子程序 2、3、5;
  else call 子程序 4;
```

生成正交状态对应模块的子程序 4:

在宏文件中增加相应常量;

在父模块中新增不重名变量 css 和 child;

生成如下形式的模块:

```
module Cname_m (pcs,…,state);
  input wire pcs; 其他父模块信号;
  output reg state; 模块语句部分;
endmodule
```

在父模块中实例化该模块,将 css 信号映射到 pcs 端口,将 child 变量映射到 state 端口,并映射其他相应端口;

call 子函数 1;

```
for 按广度优先遍历每个子状态 S
  if S 为简单状态、复合状态和伪状态
    分别 call 子程序 2、3、5;
  else call 子程序 4;
  生成伪状态对应代码的子程序 5;
if (S 是初始状态 && S 位于复合状态中)
{  宏文件中增加表示 S 唯一转换 T 的次态的常量 TS;
   第二个 always 语句触发事件表中插入 pcs 信号;
   在其 case 分支部分插入以 TS 为分支项的分支,分支语句为如下形式:
    if(pcs == MS)ns(= TS;}
else if(S 是终结状态 && 位于复合状态中)
{  以类似子程序 2 的方式将复合状态的出口动作序列加入到第一个 always 语句;
   在第二个 always 语句的 case 分支中将 ns 赋值为初始状态对应的宏;}
else if (S 是合并状态){
    宏文件中加入 S 上唯一转换 T 的次态宏常量 TS;
for (每个正交状态到 S 的转换){
   在第一个 always 语句对应 case 分支部分插入如下代码:
    if (child == Final) css <= Finish;
   将子函数 4 得到的 css 信号加入到第二个 always 语句触发事件表中;
   在第二个 always 语句对应 case 分支中加入如下语句:
   if (所有的 css 信号都等于 Finish)
     ns <= 合并状态转换次态;}}
else //历史状态{ 宏文件中增加表示 S 唯一转换 T 的次态的常量 TS;
   增加 history 变量,该复合状态中的每个转换都改变 history 为转换次态;
   在第二个 always 语句触发事件表中插入 pcs 信号;
   在其 case 分支部分插入以 TS 为分支项的分支,分支语句为如下形式:
     if(pcs== MS) ns <= history; }
```

该算法也满足第 2 节构建的同态映射。限于篇幅,不再详述。

3.3 算法的相关讨论

在子程序 2 中,若转换不含监护条件或触发事件,则不需要相应的 if 分支;当 UML 状态图中存在到某个正交子状态的转换时,生成代码中父模块 cs 信号赋值为合并状态名,并相应赋值每个 css 信号,并发地激活了每个正交子状态,这也消除了 UML 状态图中到并发状态转换的二义性语义;非组合状态中的初始状态将其忽略,组合状态中它到实状态转换隐含以 pcs 信号等于组合状态名来作为监护条件的;非组合状态中的初始状态将其忽略,对于具有出口动作的组合状态,若无终结状态,也必须自动生成一个终结状态。

总结与讨论 本文研究了将 UML 运用在硬件建模领域来实现系统级的建模的方法,构建了一个从 UML 到可综合 Verilog 的同态映射;定义了从 UML 类图和状态图导出相应 Verilog 描述的算法,使得从 UML 到 Verilog 的自动代码生成成为可能。这将把 UML 在设计分析阶段的直观性与 Verilog 在验证综合阶段的严谨性相结合,缩短了系统建模方案选择的时间,加快了代码编写的进程。

为了支持上述可综合 Verilog 代码生成算法,我们设计一个根据 Rational Rose 的 mdl 文件,自动生成 Verilog 代码的程序,它与 Soderman 的 V2Verilog 工具[4] 相结合,该方法已在中国科学院微电子研究所的国家 863 重点项目《32 位高性能嵌入式数字信号处理器(DSP)芯片设计与实现》中得以应用,收到了良好效果。

在算法的研究方面,还存在有不完善之处和需要改进的地方。如算法在多继承上的处理困难;在对监护条件的处理上,当两个监护条件互斥时,将会产生不必要甚至错误的分支。判断文本条件的互斥,涉及到相当复杂的算法,一种可选的方法是显式标明两个互斥条件。

进一步的工作是关于运用 UML 的硬件系统设计流程的研究、增加更多的 UML 自动分析映射技术、改进现有算法、如何进行相应的 EDA 设计工具及编译环境的开发。

参 考 文 献

1 zhu Q, matsuda A, et al. System-on-chip Validation using UML and CWL. In: Hardware/Software Codesign and syst em synthesis (ISSS 2004), 2004. 92~97

2 zhu Q, matsuda A, et al. An object oriented design process for system-on-chip using UML. In: Proc. of the 15 th Int Symp- osium on System Synthesis (ISSS 2002), Kyoto, Japan. 249~254

3 Greaves D J. Implementing C Designs in Hardware: A Full- Featured ANSI C to RTL Verilog Compiler in Action. In: Verilog HDL Conference and VHDL international users Forum 1998 IVC/VIUF Proceedings 1998 International ,1998. 22~29

4 Soderman D ,Panchul Y. A Verilog to C Compiler. In: Rapid System Prototy- ping 2000. KSP 2000. Proceeding. 11th international workshop on, 2000. 122~127

5 Mcumber W E. A generic framework for formalizing object-oriented modeling notations for embedded systems development. [Ph D dissertation]. Department of Computer Science and Enginerring, Michigan State University, 2000

6 Object Management Group. Unified Modeling Language specification. USA: Framingham, Mass, 1998

7 Bourdeau R H, Cheng Betty H C. A Formal Semantics for Object Model Diagrams. IEEE transaction on software engineering, 1995, 21(10): 799~821