第32卷 第21期 计 算 机 工 程 2006年11月
Vol.32 No.21 Computer Engineering November 2006
·博士论文· 文章编号：1000—3428(2006)21—0001—02 文献标识码：A 中图分类号：TP312

Verilog语义的ASM表示方法研究

胡燕翔[1,2]
（1. 天津大学电子信息工程学院，天津 300072；2. 天津师范大学计算机学院，天津 300072）

摘 要：使用抽象状态机模型（ASM）对 Verilog 的语义进行研究，给出各类赋值语句和延迟/事件控制结构的形式定义。以此为基础与VHDL进行对比，说明各种赋值语句和延迟/事件控制结构向VHDL的转换方法以及二者在转换前后的差异。
关键词：Verilog语义；抽象状态机；延迟/事件控制

Research on the Formal Semantics of Verilog Based on ASM

HU Yanxiang[1,2]
(1. School of Electronics and Information Engineering, Tianjin University, Tianjin 300072;
2. School of Computer, Tianjin Normal University, Tianjin 300072)

【Abstract】Verilog's formal semantics using the abstract states machine are studied, and the formal definition of assignment statements and delay/event control mechanism is given. Comparing with Borger's definition on VHDL, the key methods on how to translate Verilog description to VHDL are explained. In the end, the simulation differences before and after translation are studied.
【Key words】Verilog formal semantics; Abstract states machine; Delay/Eevent control

随着设计规模的扩大和芯片集成度的提高，以硬件描述语言为基础的设计方法被广泛采用。对于硬件描述语言的语义进行无二义性定义的重要性在于保证各种EDA工具中转换规则的正确性，以及为形式化验证和模型检查工具提供理论基础[1]。

对于VHDL，人们已经使用了许多种方法对其形式化语义进行研究，采用的技术包括表示语义、操作语义、代数语义以及间隔时序逻辑等[1,2]。在这些研究中，Egon Borger等使用抽象状态机模型（亦称Evolving Algebra Machine）从模拟核心与用户进程交互的角度对VHDL'93 的语义进行全面定义，因其具有严格而透明的特点，被认为是一种较为成功的方法[1,3]。其它几种方法由于侧重点有所不同，大多数只对VHDL的子集进行了定义。对于Verilog形式化语义的研究较少，使用的方法主要包括操作语义、表示语义等，且定义的对象都是一个子集[4,5]。

本文使用抽象状态机模型从模拟的角度对 Verilog 中核心的赋值语句和时序控制语句的语义进行形式化定义。目的在于使用这一定义与 Egon Borger 等对 VHDL 的定义进行比较，对 Verilog—VHDL 的模拟共用性进行形式化研究，为 Verilog—VHDL 翻译转换提供理论依据，保证转换规则的正确性和无二义性。

1 抽象状态机模型

抽象状态机模型可被理解为基于抽象数据的伪代码[3]。自提出以来，它已经被用于许多种语言的形式语义定义，如C++，Java等[1]。在抽象状态机模型中，抽象数据元素使用集合元素的方式来表示，用于表征系统当前的状态；状态之间的转换关系则通过集合间代数关系的转换进行表示。

Egon Borger等使用抽象状态机模型提出了一个针对VHDL'93 的语义模型。该模型使用一组转换规则定义了一个抽象的VHDL模拟引擎，用于描述模拟核心与用户进程之间的交互、挂起和恢复，以及对变量、声明信号、端口等的操作[2]。为了使用该模型对Verilog进行定义，作者定义如下规则用于处理Verilog所特有的分层事件队列和内嵌延迟/事件赋值机制。这些规则主要包括：

N/R：表示线网/寄存器类型变量；P：表示进程；value (Expr)：表达式 Expr 的值；

single_waveform := (value(Expr)/Expr, time, wait_cond, P) ：四元组，表示对某一线网/寄存器变量赋值 value (Expr)，该赋值计划发生时间 time，防卫条件为 wait_cond，所在进程为 P；

schedule (single_waveform, event_type)：将类型为 event_type 的赋值 single_waveform 调度到事件队列之中；

event_type：表示事件被模拟核心所调度的类型，包括现用事件（active）、带用事件（inactive）、非阻塞赋值更新事件（non_blocking update）、监视事件（monitor）和将来事件（future）；

driver (N/R)：事件队列中对于线网/寄存器变量的驱动源队列。

其它规则的详细定义请参考文献[2]。

2 赋值与时序控制的形式语义

2.1 进程语句

Verilog 中包括循环执行进程（always）和一次执行进程（initial）。循环执行进程自其中的第一条顺序语句开始循环执行，直至模拟结束。一次执行进程只在模拟开始时执行一次，然后无条件地被挂起。可以通过在最后一条语句之后添加无限等待语句“ wait (0) ”将其视为循环执行进程。

```
if    process_type == always
then    < executed by ASM > ;
else      // process_type == initial
then <    execute by ASM only once> ;
```

基金项目：国防基础科研基金资助项目
作者简介：胡燕翔(1969－)，男，博士后、副教授，主研方向：硬件描述语言及其相关模拟，综合技术，VLSI 设计
收稿日期：2005-11-23 **E-mail：**yanxianghu@263.net

2.2 连续赋值语句

连续赋值语句可以视为对所有变量操作数敏感的循环执行进程。

```
if process does <assign #delay    N = Expr >
if delay == 0
single_waveform := (value(Expr), Tc, Null, P) ;
schedule (single_waveform, active) ;//现用事件
driver (N) := single_waveform ^ driver(N) ;
else
single_waveform := (Expr, Tc+delay, Null, P) ;
schedule (single_waveform, active) ;
driver (N) := single_waveform ^ driver(N)
```

连续赋值语句对线网左值每次只能赋予一个值，因此使用 single_waveform 来表示。与 VHDL 并发信号赋值语句相比，Verilog 连续赋值语句中的延迟相当于惯性延迟。在上面的定义中，为了体现惯性延迟的“过滤”作用，被调度到 Tc 时刻的是表达式 Expr，而不是调度时刻表达式的值 value(Expr)。这样，表达式的值在赋值执行的时候被计算出来，以此来达到事件取消的目的。这种表示方法较文献[3,6]中使用的事件取消方法更加直观，便于具体实现。

2.3 非阻塞赋值语句

(1)带有内嵌延迟控制的非阻塞赋值语句

```
if process does < R <= #delay Expr >
if delay == 0
single_waveform := ( value( Expr ), Tc, Null, P );
schedule ( single_waveform, non_blocking update);
//非阻塞赋值更新类型
driver (R) := single_waveform ^ driver(R);
suspended (P) := false;
else
single_waveform := ( value( Expr ), Tc+delay, Null, P);
schedule ( single_waveform, non_blocking update);
driver (R) := single_waveform ^ driver(R);
suspended (P) := false;
```

(2)带有内嵌事件控制的非阻塞赋值语句

```
if process does < R <= @evnet_expr Expr >
if event_expr == Null
single_waveform := ( value( Expr ), Tc, Null, P )        // 调度到当
//前模拟时刻
schedule ( single_waveform, non_blocking update);
//非阻塞赋值更新类型
driver (R) := single_waveform ^ driver(R);//加入事件队列
suspended (P) := false;
else
single_waveform := ( value( Expr ), Null, event_expr, P);
schedule ( single_waveform, future);
driver (R) := single_waveform ^ driver(R);
suspended (P) := false;
if R/N ∈condsignal (event_expr)
waitsignal (R/N) := waitsignal (R/N) ∪{ P };
```

同 VHDL 的顺序信号赋值语句相同，非阻塞赋值语句对于寄存器左值的赋值并不立即生效并且不会阻塞所在进程的继续执行。Verilog 过程赋值语句中的延迟相当于传输延迟，因此赋值表达式的当前值被调度。

2.4 阻塞赋值语句

(1)带有内嵌延迟控制的阻塞赋值语句

```
if process does < R = #delay Expr >
if delay==0
single_waveform := (value(Expr), Tc, Null, P);
schedule (single_waveform, inactive);
//待用事件类型
driver (R) := single_waveform ^ driver(R);
suspended (P) :=True;
else
single_waveform := (value(Expr), Tc+delay, Null, R);
schedule (single_waveform, inactive);
driver (R) := single_waveform ^ driver(R);
suspended (P) := True;
Timeout(P) := Tc+delay;
```

(2)带有内嵌事件控制的阻塞赋值语句

```
if process does < R = @evnet_expr Expr >
if event_expr == Null
single_waveform := (value(Expr), Tc, Null, P);
//调度到当前模拟时刻
schedule (single_waveform, inactive);
driver (R) := single_waveform ^ driver(R);
suspended (P) :=True;
else
single_waveform := (value(Expr), Null, evnet_expr, P);
schedule (single_waveform, future);
driver (R) := single_waveform ^ driver(R);
suspended (P) := True;
wait_cond (P) := evnet_expr;
if R/N ∈condsignal (event_expr)
waitsignal (R/N) := waitsignal (R/N) ∪{P}};
```

从上面的定义可以看到，阻塞赋值语句会将所在的进程挂起，直至指定的延迟时间到达或等待的事件发生。与 VHDL 的顺序信号赋值语句相反，阻塞赋值语句对于寄存器左值的赋值立即生效，带有零延迟的阻塞赋值语句类似于 VHDL 中变量赋值的效果。带有延迟/事件控制的阻塞赋值语句可以等效地展开为 VHDL 的变量赋值、wait 语句和信号赋值语句的组合。

2.5 延迟/事件控制

(1)延迟控制

```
if process does < # delay >
  timeout(P) := Tc+delay;
  //设定进程恢复执行时间
  suspended(P) := True;//挂起进程
```

可见对于 Verilog 中的延迟控制可以等效地使用 VHDL 中的 wait after 语句来表示将所在进程挂起，延迟指定的时间，然后恢复执行。

(2)事件控制

```
if process does < @ event_expr >
wait_cond (P) := event_expr;
suspended (P) := True;
if R/N ∈condsignal (event_expr)
waitsignal (R/N) := waitsignal (R/N) ∪{P}
//进程并入对该信号敏感的进程链表
```

可见对于 Verilog 中的事件控制可以等效地使用 VHDL 中的 wait on 或 wait until 语句来表示将所在进程挂起，等待事件的发生。

(3)wait 语句

```
if process does < wait (Expr) >
if      value (Expr) == false
  suspended (P) := True;
wait_cond (P) := Expr;
```

（下转第 21 页）