数字系统设计

郑海永

中国海洋大学 电子工程系

2014 年 5 月

VHDL

1. 硬件描述语言
2. VHDL 基础
3. VHDL 实例
4. VHDL 入门
5. VHDL 基本语句
6. VHDL 深入

目录

1. 进程语句与 WAIT 语句
   - 进程机制
   - 进程语句
   - WAIT 语句
2. 顺序语句与并行语句
   - 顺序语句
   - 并行语句

内容提要

1. 进程语句与 WAIT 语句
   - 进程机制
   - 进程语句
   - WAIT 语句
2. 顺序语句与并行语句
   - 顺序语句
   - 并行语句

内容提要

1 进程语句与 WAIT 语句
- 进程机制
- 进程语句
- WAIT 语句

2 顺序语句与并行语句
- 顺序语句
- 并行语句

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的WAIT语句引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的WAIT语句引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的**WAIT**语句引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的**WAIT语句**引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的**WAIT语句**引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的**WAIT语句**引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的**WAIT语句**引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

进程机制

并行语句与顺序语句

- 在设计实体的结构体过程中，所有语句都是并行语句。
- 为了便于只处于仿真阶段的行为描述，VHDL 提供了一种顺序描述语句。
- 借鉴计算机操作系统的进程机制，进程语句和与之相配合的**WAIT语句**引入顺序语句仿真。
- 进程语句将描述行为的一系列顺序描述语句包装成一条并行语句。
- 一条进程语句相当于一个电路模块。
- 进程中的 WAIT 语句具有在仿真时将进程“挂起”和“激活”两种状态之间进行转换的功能。

进程语句是对实体进行行为建模的主要机制。

内容提要

1 进程语句与 WAIT 语句
- 进程机制
- 进程语句
- WAIT 语句

2 顺序语句与并行语句
- 顺序语句
- 并行语句

进程语句

- 进程语句本身是一条并行语句。
- 它可以和其他并行语句一样出现在结构体中。
- 各条进程语句之间也是并行关系，等同于各个电路模块之间是并行工作的。

- 进程内部的各条语句之间是顺序关系。
- 进程内部的所有语句都是用于行为描述的顺序语句。

进程语句

- 进程语句本身是一条并行语句。
- 它可以和其他并行语句一样出现在结构体中。
- 各条进程语句之间也是并行关系，等同于各个电路模块之间是并行工作的。

- 进程内部的各条语句之间是顺序关系。
- 进程内部的所有语句都是用于行为描述的顺序语句。

进程状态

- 活动状态 ⇒ 执行。
- 挂起状态 ⇒ 等待事件的发生。

进程在仿真开始时（事实上是仿真的初始化阶段）进入执行状态，直到它遇到WAIT语句或敏感信号没有变化而被挂起。

进程状态

- 活动状态 ⇒ 执行。
- 挂起状态 ⇒ 等待事件的发生。

进程在仿真开始时（事实上是仿真的初始化阶段）进入执行状态，直到它遇到WAIT语句或敏感信号没有变化而被挂起。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

进程语句格式

```
[ 进程标号: ] [ POSTPONED ] PROCESS [( 敏感信号表 )] IS
    [ { 声明语句 } ]
BEGIN
    [ { 顺序语句 } ]
END [ POSTPONED ] PROCESS [ 进程标号 ];
```

1. 进程标号是该进程的一个名称标识符，可选项。
2. 敏感信号表也是可选项。
   - 敏感信号表中的一个或多个信号值的变化可以激活该进程。
   - 敏感信号表等价于在该进程的最后一条顺序语句是一条隐含的 WAIT 语句。
   - 在含有敏感信号表的进程中，不能再有显式的 WAIT 语句。
3. 声明语句所声明的数据类型、对象和子程序等都是该进程的局部数据环境，不能在进程中声明信号对象。
4. 顺序语句描述了该进程的行为。
5. 延缓进程（VHDL'93）只在一个仿真周期的最后一个 Δ 时间才被激活，可以有效的降低进程的仿真处理频率。

D 型触发器——bhv1

```
1  ENTITY dff_1 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_1;

6  ARCHITECTURE bhv1 OF dff_1 IS
7  BEGIN
8    d1: PROCESS(clk)
9    VARIABLE a: Bit;
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a := d;
13       q <= a;
14     END IF;
15   END PROCESS d1;
16 END bhv1;
```

敏感信号表

注意 每个进程的敏感信号表必须完整列出所有敏感信号（包括时钟信号和复位信号）。

- 如果敏感信号列表不完整，可能导致综合前后的仿真结果不一致。
- 如果敏感信号表中含有不必要的信号，则又会降低仿真速度。

```
1 PROCESS(x)
2 BEGIN
3   IF x AND y = '1' THEN
4     z <= '1';
5   ELSE
6     z <= '0';
7   END IF;
8 END PROCESS;
```

行为综合后的 RTL 描述：`z <= x AND y;`

敏感信号表

注意 每个进程的敏感信号表必须完整列出所有敏感信号（包括时钟信号和复位信号）。

- 如果敏感信号列表不完整，可能导致综合前后的仿真结果不一致。
- 如果敏感信号表中含有不必要的信号，则又会降低仿真速度。

```
1 PROCESS(x)
2 BEGIN
3   IF x AND y = '1' THEN
4     z <= '1';
5   ELSE
6     z <= '0';
7   END IF;
8 END PROCESS;
```

行为综合后的 RTL 描述：`z <= x AND y;`

被动进程

- 如果在一个进程的顺序语句中没有对任何信号赋值，则该进程被成为被动进程。
- 被动进程用于各种检查，其功能与断言语句类似，但更加灵活。

进程中的信号赋值语句

```
1 ...
2 SIGNAL x,y,z : Bit;
3 ...
4 PROCESS(y)
5 BEGIN
6   x <= y;
7   z <= NOT x;
8 END PROCESS;
```

```
1 PROCESS(y)
2   VARIABLE x,z : Bit;
3 BEGIN
4   x := y;
5   z := NOT x;
6 END PROCESS;
```

进程中的信号赋值语句

```
1 ...
2 SIGNAL x,y,z : Bit;
3 ...
4 PROCESS(y)
5 BEGIN
6   x <= y;
7   z <= NOT x;
8 END PROCESS;
```

```
1 PROCESS(y)
2   VARIABLE x,z : Bit;
3 BEGIN
4   x := y;
5   z := NOT x;
6 END PROCESS;
```

多个进程驱动同一个信号

```
1  proc_a: PROCESS(clk)
2  BEGIN
3    IF clk'Event AND clk='1' THEN
4      dout <= din_a;
5    END IF;
6  END PROCESS;

8  proc_b: PROCESS(sel_en)
9  BEGIN
10   IF sel_en='1' THEN
11     dout <= din_b;
12   END IF;
13 END PROCESS;
```

进程

- 一个进程中不允许出现两个时钟沿触发。
- 对同一信号赋值的语句应出现在单个进程内。
- 当出现多层IF 语句嵌套时，最好采用CASE 语句替代，一是减少多层嵌套带来的延时，二来可以增强程序的可读性。
- 顺序语句如IF 语句、CASE 语句、LOOP 语句、变量赋值语句等必须出现在进程、函数或子程序内部，而不能单独出现在进程之外。
- 进程内部是顺序执行时，进程之间是并行运行的。
- VHDL 中所有并行语句都可以理解为特殊的进程，只是不以PROCESS 结构出现，其输入信号和判断信号就是隐含的敏感表。

内容提要

1 进程语句与 WAIT 语句
- 进程机制
- 进程语句
- WAIT 语句

2 顺序语句与并行语句
- 顺序语句
- 并行语句

WAIT 语句

- 作用是将正在仿真的进程挂起，并在 WAIT 语句条件为真时，再次激活该进程。
- 功能是将进程在“挂起”和“激活”两种状态之间进行转换。

- 仿真器在初始化阶段要对每个进程仿真一次。
- 仿真期间，一个进程可以被认为是一个无限循环。
- 当进程的最后一条顺序语句仿真完毕之后，又会从该进程的第一条语句开始仿真。
- 当一条进程语句中没有敏感信号表和 WAIT 语句时，仿真器永远不会跳出初始化阶段。

WAIT 语句

- 作用是将正在仿真的进程挂起，并在 WAIT 语句条件为真时，再次激活该进程。
- 功能是将进程在“挂起”和“激活”两种状态之间进行转换。

- 仿真器在初始化阶段要对每个进程仿真一次。
- 仿真期间，一个进程可以被认为是一个无限循环。
- 当进程的最后一条顺序语句仿真完毕之后，又会从该进程的第一条语句开始仿真。
- 当一条进程语句中没有敏感信号表和 WAIT 语句时，仿真器永远不会跳出初始化阶段。

WAIT 语句格式

WAIT [ON 敏感信号表] | [UNTIL 条件表达式] | [FOR 时间表达式];

- 仿真期间当遇到 WAIT 语句时，仿真时钟会停止，进程被挂起，直到该 WAIT 语句中的条件被满足，仿真时钟才会继续前进。
- 敏感信号表中的任何一个或多个信号值发生变化；条件表达式的值为 TRUE；仿真时钟停止的时间超过时间表达式的值。
- 当 WAIT 后面三个可选项都没有时，进程被无限期挂起，在此后的整个仿真期间，进程将不会被再次激活。

WAIT 语句例子

```
1  WAIT ON x FOR 100ns;
2  WAIT UNTIL x = '1';
3  WAIT ON x UNTIL x = '1';

5  p2: PROCESS
6  BEGIN
7    IF clear = '0' THEN
8      q <= (OTHERS => '0');
9    ELSIF (clk'Event AND clk = '1') THEN
10     q <= d;
11   END IF;
12   WAIT ON clear, clk;
13 END PROCESS p2;
```

如果进程语句中含有敏感信号表，则不能再有显式的 WAIT 语句出现。

内容提要

1 进程语句与 WAIT 语句
- 进程机制
- 进程语句
- WAIT 语句

2 顺序语句与并行语句
- 顺序语句
- 并行语句

内容提要

1 进程语句与 WAIT 语句
- 进程机制
- 进程语句
- WAIT 语句

2 顺序语句与并行语句
- 顺序语句
- 并行语句

顺序语句

顺序语句用于在进程语句中描述进程或子程序的行为

- WAIT 语句
- 变量赋值语句
- 信号赋值语句
- IF 语句
- CASE 语句
- NULL 语句
- LOOP 语句
- NEXT 语句
- EXIT 语句
- 过程调用语句
- RETURN 语句
- 断言语句
- REPORT 语句

顺序语句

顺序语句用于在进程语句中描述进程或子程序的行为

- WAIT 语句
- 变量赋值语句
- 信号赋值语句
- IF 语句
- CASE 语句
- NULL 语句
- LOOP 语句
- NEXT 语句
- EXIT 语句
- 过程调用语句
- RETURN 语句
- 断言语句
- REPORT 语句

变量赋值语句

变量名 := 表达式;

```
1  ENTITY dff_1 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_1;
5
6  ARCHITECTURE bhv1 OF dff_1 IS
7  BEGIN
8    d1: PROCESS(clk)
9    VARIABLE a: Bit;
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a := d;
13       q <= a;
14     END IF;
15   END PROCESS d1;
16 END bhv1;
```

变量赋值语句

变量名 := 表达式;

```
1  ENTITY dff_1 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_1;
5
6  ARCHITECTURE bhv1 OF dff_1 IS
7  BEGIN
8    d1: PROCESS(clk)
9    VARIABLE a: Bit;
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a := d;
13       q <= a;
14     END IF;
15   END PROCESS d1;
16 END bhv1;
```

信号赋值语句

信号名 <= [TRANSPORT] 表达式 [AFTER 时间表达式];

```
1  ENTITY dff_3 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_3;
5
6  ARCHITECTURE bhv3 OF dff_3 IS
7    SIGNAL a: Bit;
8  BEGIN
9    d3: PROCESS(clk)
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a <= d;
13       q <= a;
14     END IF;
15   END PROCESS d3;
16 END bhv3;
```

信号赋值语句

信号名 <= [TRANSPORT] 表达式 [AFTER 时间表达式];

```
1  ENTITY dff_3 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_3;
5
6  ARCHITECTURE bhv3 OF dff_3 IS
7    SIGNAL a: Bit;
8  BEGIN
9    d3: PROCESS(clk)
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a <= d;
13       q <= a;
14     END IF;
15   END PROCESS d3;
16 END bhv3;
```

IF 语句

```
IF 布尔表达式 THEN
    { 顺序语句 }
[ { ELSIF 布尔表达式 THEN
    { 顺序语句 } } ]
[ ELSE
    { 顺序语句 } ]
END IF;
```

CASE 语句

```
CASE 选择表达式 IS
    { WHEN 值域 =>
        { 顺序语句 } }
    [ WHEN OTHERS =>
        { 顺序语句 } ]
END CASE;
```

NULL 语句

```
NULL [ AFTER 时间表达式 ];
```

LOOP 语句

```
[ 语句标号: ][ 重复模式 ] LOOP
    { 顺序语句 }
END LOOP [ 语句标号 ];
```

NEXT 语句

NEXT [LOOP 语句标号][WHEN 布尔表达式];

EXIT 语句

```
EXIT [ LOOP 语句标号 ][ WHEN 布尔表达式 ];
```

过程调用语句

过程名（实参表）;

RETURN 语句

RETURN [表达式];

断言语句与 REPORT 语句

```
ASSERT 布尔表达式 [ REPORT 信息 ][ SEVERITY 错误等级 ];
```

内容提要

1 进程语句与 WAIT 语句
- 进程机制
- 进程语句
- WAIT 语句

2 顺序语句与并行语句
- 顺序语句
- 并行语句

并行语句

结构体中的语句都是并行语句

- 进程语句
- 块语句
- 并行信号赋值语句
- 并行过程调用语句
- 并行断言语句
- 元件例化语句
- 生成语句

并行语句

结构体中的语句都是并行语句

- 进程语句
- 块语句
- 并行信号赋值语句
- 并行过程调用语句
- 并行断言语句
- 元件例化语句
- 生成语句

块语句

```
[ 块标号: ] BLOCK [ (保护表达式) ]
    [ 类属子句 ]
    [ 端口子句 ]
    [ 块声明语句 ]
BEGIN
    { 并行语句 }
END BLOCK [ 块标号 ];
```

并行信号赋值语句

```
[ POSTPONED ] 信号名 <= [ GUARDED ] [ TRANSPORT ]
        [ { 表达式 [ AFTER 时间表达式 ] WHEN 布尔表达式 ELSE } ]
        表达式 [ AFTER 时间表达式 ];

[ POSTPONED ] WITH 选择表达式 SELECT
    信号名 <= [ GUARDED ] [ TRANSPORT ]
        { 表达式 [ AFTER 时间表达式 ] WHEN 值域, }
        表达式 [ AFTER 时间表达式 ] WHEN 值域 |OTHERS;
```

并行信号赋值语句

```
[ POSTPONED ] 信号名 <= [ GUARDED ] [ TRANSPORT ]
        [ { 表达式 [ AFTER 时间表达式 ] WHEN 布尔表达式 ELSE } ]
        表达式 [ AFTER 时间表达式 ];

[ POSTPONED ] WITH 选择表达式 SELECT
    信号名 <= [ GUARDED ] [ TRANSPORT ]
        { 表达式 [ AFTER 时间表达式 ] WHEN 值域, }
        表达式 [ AFTER 时间表达式 ] WHEN 值域 |OTHERS;
```

并行过程调用语句

[POSTPONED] 过程名 (实参表);

并行断言语句

[POSTPONED] ASSERT 布尔表达式 [REPORT 信息][SEVERITY 错误等级];

元件例化语句

[例化元件名] 模板元件名 [GENERIC MAP（类属实参表）] PORT MAP（信号实参表）;

生成语句

[生成标号:] 生成模式 GENERATE
    { 并行语句 }
END GENERATE [生成标号];