数字系统设计

郑海永

上课时间地点：周 2/9 10 节/3308

电子信息科学与技术 2010 级和 2011 级

中国海洋大学 电子工程系

2013 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 顺序语句
- 顺序语句

内容提要

1 顺序语句
- 顺序语句

内容提要

1 顺序语句
- 顺序语句

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

1. WAIT 语句
2. 变量赋值语句
3. 信号赋值语句
4. IF 语句
5. CASE 语句
6. NULL 语句
7. LOOP 语句
8. NEXT 语句
9. EXIT 语句
10. 过程调用语句
11. RETURN 语句
12. 断言语句与 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

1. WAIT 语句
2. 变量赋值语句
3. 信号赋值语句
4. IF 语句
5. CASE 语句
6. NULL 语句
7. LOOP 语句
8. NEXT 语句
9. EXIT 语句
10. 过程调用语句
11. RETURN 语句
12. 断言语句与 REPORT 语句

进程语句与 WAIT 语句

1. 进程永远不会停止，它总是处于执行或挂起状态。
2. 进程在仿真开始（仿真的初始化阶段）进入执行状态，直到它遇到 WAIT 语句或敏感列表的信号没有变化而被挂起。
3. 如果进程没有敏感列表或 WAIT 语句，该进程将陷入无限循环，仿真器永远不会跳出初始化阶段。
4. 敏感列表（隐式的 WAIT 语句）和显式的 WAIT 语句不能同时出现在一个进程中。

进程语句与 WAIT 语句

1. 进程永远不会停止，它总是处于执行或挂起状态。
2. 进程在仿真开始（仿真的初始化阶段）进入执行状态，直到它遇到 WAIT 语句或敏感列表的信号没有变化而被挂起。
3. 如果进程没有敏感列表或 WAIT 语句，该进程将陷入无限循环，仿真器永远不会跳出初始化阶段。
4. 敏感列表（隐式的 WAIT 语句）和显式的 WAIT 语句不能同时出现在一个进程中。

进程语句与 WAIT 语句

1. 进程永远不会停止，它总是处于执行或挂起状态。
2. 进程在仿真开始（仿真的初始化阶段）进入执行状态，直到它遇到 WAIT 语句或敏感列表的信号没有变化而被挂起。
3. 如果进程没有敏感列表或 WAIT 语句，该进程将陷入无限循环，仿真器永远不会跳出初始化阶段。
4. 敏感列表（隐式的 WAIT 语句）和显式的 WAIT 语句不能同时出现在一个进程中。

进程语句与 WAIT 语句

1. 进程永远不会停止，它总是处于执行或挂起状态。
2. 进程在仿真开始（仿真的初始化阶段）进入执行状态，直到它遇到 WAIT 语句或敏感列表的信号没有变化而被挂起。
3. 如果进程没有敏感列表或 WAIT 语句，该进程将陷入无限循环，仿真器永远不会跳出初始化阶段。
4. 敏感列表（隐式的 WAIT 语句）和显式的 WAIT 语句不能同时出现在一个进程中。

顺序语句

① WAIT 语句

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

WAIT [sensitivity_clause] [condition_clause] [timeout_clause];

- 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。
- 可以包含敏感（sensitivity）子句、条件（condition）子句、超时（timeout）子句的任何组合，也可以忽略这三个子句。
- 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。

① WAIT 语句

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

- 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。
  - 可以包含敏感（sensitivity）子句、条件（condition）子句、超时（timeout）子句的任何组合，也可以忽略这三个子句。
  - 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。

① WAIT 语句

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

- 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。
- 可以包含敏感（sensitivity）子句、条件（condition）子句、超时（timeout）子句的任何组合，也可以忽略这三个子句。
- 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。

顺序语句

① WAIT 语句

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

- 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。
- 可以包含敏感（sensitivity）子句、条件（condition）子句、超时（timeout）子句的任何组合，也可以忽略这三个子句。
- 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。

① WAIT 语句

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

- 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。
- 可以包含敏感（sensitivity）子句、条件（condition）子句、超时（timeout）子句的任何组合，也可以忽略这三个子句。
- 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。

敏感列表与 WAIT ON

敏感列表

```
1 p1: PROCESS(clear,clk)
2 BEGIN
3   IF clear = '0' THEN
4     q <= "00000000";
5   ELSIF (clk'Event AND clk='1') THEN
6     q <= d;
7   END IF;
8 END PROCESS p1;
```

WAIT ON

```
p2: PROCESS
BEGIN
  IF clear = '0' THEN
    q <= (OTHERS => '0');
  ELSIF (clk'Event AND clk='1') THEN
    q <= d;
  END IF;
  WAIT ON clear,clk;
END PROCESS p2;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

- 等待信号 a 或 b 中有一个值（或两个值）改变。
- ON 后面实际上跟一个敏感列表。

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

- 等待信号 x 的变化，并且 x 的值等于 1（布尔表达式为 True）。
- 如果没有给出敏感子句（ON），那么条件子句（UNTIL）中实际上也包含了一个隐式的敏感子句。
- 如果没有给出条件子句（UNTIL），那么敏感子句（ON）中的信号发生变化实际上给出了一个隐式的 True。

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

- 等待的时间超过 100ns。

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

单一敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON a,b;
```

```
1 WAIT UNTIL x = '1'; <=> WAIT ON x UNTIL x = '1';
2 WAIT ON x; <=> WAIT ON x UNTIL True;
```

```
1 WAIT FOR 100ns;
```

```
1 WAIT;
```

- 使得正在执行的进程在仿真的剩余时间内挂起。

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

- 等待信号 x 变化，并且x 的值等于 1。

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

- 等待信号 clk 变化，并且reset 的值等于 0。

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

- 等待信号 x 的值变化或者等待的时间超过 100ns。

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

- 等待信号 trigger 变化并且值等于 1，或者等待的时间超过 1ms。

组合敏感子句（ON）、条件子句（UNTIL）、超时子句（FOR）

```
1 WAIT ON x UNTIL x = '1'; <=> WAIT UNTIL x = '1';
```

```
1 WAIT ON clk UNTIL reset = '0';
```

```
1 WAIT ON x FOR 100ns;
```

```
1 WAIT UNTIL trigger = '1' FOR 1ms;
```

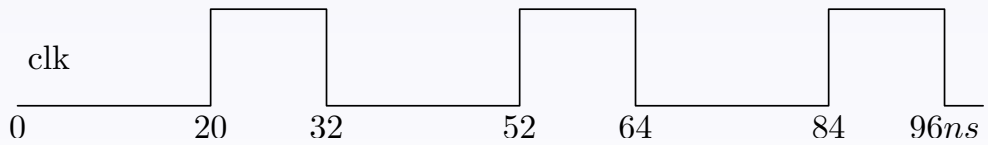
- 如果一个等待语句包含有敏感子句（ON）及条件子句（UNTIL），只有当敏感子句中的任一信号上发生条件时，才测试条件子句中的条件。
- 如果超时子句（FOR）的等待语句中还包含有一个敏感子句（ON）或条件子句（UNTIL），则这些子句会使得进程被更早地重新开始。

顺序语句

```
1 PROCESS
2 BEGIN
3   clk <= '0';
4   WAIT FOR 20 ns;
5   clk <= '1';
6   WAIT FOR 12 ns;
7 END PROCESS;
```

顺序语句

```
1 PROCESS
2 BEGIN
3   clk <= '0';
4   WAIT FOR 20 ns;
5   clk <= '1';
6   WAIT FOR 12 ns;
7 END PROCESS;
```



- 进程永远不会停止，它总是处于执行或挂起状态。
- 所有进程在仿真初始化阶段开始执行，直到挂起为止。
- 没有敏感列表和没有显式 WAIT 语句的进程永远不会挂起它们自己（无限循环）。

```
1 WAIT FOR 0ns;
```

- 表示等待一个 Δ 周期。

```
1 wait0: PROCESS
2 BEGIN
3   WAIT ON data;
4   sig_a <= data;
5   WAIT FOR 0ns;
6   sig_b <= sig_a;
7 END PROCESS wait0;
```

```
1 WAIT FOR 0ns;
```

- 表示等待一个 Δ 周期。

```
1 wait0: PROCESS
2 BEGIN
3   WAIT ON data;
4   sig_a <= data;
5   WAIT FOR 0ns;
6   sig_b <= sig_a;
7 END PROCESS wait0;
```

```
1 WAIT FOR 0ns;
```

- 表示等待一个 Δ 周期。

```
1 wait0: PROCESS
2 BEGIN
3   WAIT ON data;
4   sig_a <= data;
5   WAIT FOR 0ns;
6   sig_b <= sig_a;
7 END PROCESS wait0;
```

```
1 WAIT UNTIL clock = '1';
```

- 当WAIT 语句执行时，它所在的进程首先必须挂起，并一直等待，直到clock 信号上发生事件。
- 事件发生之后，该语句接着会检查布尔表达式的值。
- WAIT 语句执行时，该布尔表达式的值是否为真并不重要。
- 因为WAIT 语句会等到clock 信号上再次发生事件时才会去检查布尔表达式的值。

```
1 WAIT UNTIL True; <=> WAIT;
```

- 进程将会被无限期挂起。
- 因为布尔表达式中没有信号，因此没有事件发生。

```
1 WAIT UNTIL clock = '1';
```

- 当WAIT 语句执行时，它所在的进程首先必须挂起，并一直等待，直到clock 信号上发生事件。
- 事件发生之后，该语句接着会检查布尔表达式的值。
- WAIT 语句执行时，该布尔表达式的值是否为真并不重要。
- 因为WAIT 语句会等到clock 信号上再次发生事件时才会去检查布尔表达式的值。

```
1 WAIT UNTIL True; <=> WAIT;
```

- 进程将会被无限期挂起。
- 因为布尔表达式中没有信号，因此没有事件发生。

```
1 WAIT UNTIL clock = '1';
```

- 当WAIT 语句执行时，它所在的进程首先必须挂起，并一直等待，直到clock 信号上发生事件。
- 事件发生之后，该语句接着会检查布尔表达式的值。
- WAIT 语句执行时，该布尔表达式的值是否为真并不重要。
- 因为WAIT 语句会等到clock 信号上再次发生事件时才会去检查布尔表达式的值。

```
1 WAIT UNTIL True; <=> WAIT;
```

- 进程将会被无限期挂起。
- 因为布尔表达式中没有信号，因此没有事件发生。

```
1 WAIT UNTIL clock = '1';
```

- 当WAIT 语句执行时，它所在的进程首先必须挂起，并一直等待，直到clock 信号上发生事件。
- 事件发生之后，该语句接着会检查布尔表达式的值。
- WAIT 语句执行时，该布尔表达式的值是否为真并不重要。
- 因为WAIT 语句会等到clock 信号上再次发生事件时才会去检查布尔表达式的值。

```
1 WAIT UNTIL True; <=> WAIT;
```

- 进程将会被无限期挂起。
- 因为布尔表达式中没有信号，因此没有事件发生。

- 一般只有WAIT UNTIL格式的等待语句可以被综合器接受；
- 其余格式的等待语句只能在VHDL仿真器中使用。
- 但不同综合器的处理也都可能不同（如WAIT ON与Sensitivity List）。

```
WAIT UNTIL 信号 = Value;
WAIT UNTIL 信号'Event AND 信号 = Value;
WAIT UNTIL NOT 信号'Stable AND 信号 = Value;
```

上升沿

```
1 WAIT UNTIL clock ='1';
2 WAIT UNTIL rising_edge(clock);
3 WAIT UNTIL NOT clock'Stable AND clock ='1';
4 WAIT UNTIL clock ='1' AND clock'Event;
```

- 一般只有WAIT UNTIL格式的等待语句可以被综合器接受；
- 其余格式的等待语句只能在VHDL仿真器中使用。
- 但不同综合器的处理也都可能不同（如WAIT ON与Sensitivity List）。

```
WAIT UNTIL 信号 = Value;
WAIT UNTIL 信号'Event AND 信号 = Value;
WAIT UNTIL NOT 信号'Stable AND 信号 = Value;
```

上升沿

```
1 WAIT UNTIL clock ='1';
2 WAIT UNTIL rising_edge(clock);
3 WAIT UNTIL NOT clock'Stable AND clock ='1';
4 WAIT UNTIL clock ='1' AND clock'Event;
```

- 一般只有WAIT UNTIL格式的等待语句可以被综合器接受；
- 其余格式的等待语句只能在VHDL仿真器中使用。
- 但不同综合器的处理也都可能不同（如WAIT ON与Sensitivity List）。

```
WAIT UNTIL 信号 = Value;
WAIT UNTIL 信号'Event AND 信号 = Value;
WAIT UNTIL NOT 信号'Stable AND 信号 = Value;
```

上升沿

```
1 WAIT UNTIL clock ='1';
2 WAIT UNTIL rising_edge(clock);
3 WAIT UNTIL NOT clock'Stable AND clock ='1';
4 WAIT UNTIL clock ='1' AND clock'Event;
```

可综合

该例子请参考这里（2005）。

```
1  ARCHITECTURE a OF latch IS
2  BEGIN
3  PROCESS
4  BEGIN
5    IF ck='1' THEN
6      q <= d;
7      qb <= NOT d;
8    END IF;
9    WAIT ON ck,d; --compiler error
10 END PROCESS;
11 END a;
```

Quartus II web free edition

```
1  Wait Statement must contain condition clause with UNTIL keyword
```

可综合

该例子请参考这里（2005）。

```
1  ARCHITECTURE a OF latch IS
2  BEGIN
3  PROCESS
4  BEGIN
5    IF ck='1' THEN
6      q <= d;
7      qb <= NOT d;
8    END IF;
9    WAIT ON ck,d; --compiler error
10 END PROCESS;
11 END a;
```

Quartus II web free edition

```
1  Wait Statement must contain condition clause with UNTIL keyword
```

② 变量赋值语句

变量名 := 表达式;

```
1 d1: PROCESS(clk)
2 VARIABLE a: Bit;
3 BEGIN
4   IF clk'Event AND clk='1' THEN
5     a := d;
6     q <= a;
7   END IF;
8 END PROCESS d1;
```

② 变量赋值语句

变量名 := 表达式;

```
1 d1: PROCESS(clk)
2 VARIABLE a: Bit;
3 BEGIN
4   IF clk'Event AND clk='1' THEN
5     a := d;
6     q <= a;
7   END IF;
8 END PROCESS d1;
```

变量赋值

- 变量可以在进程或子程序内部声明和使用。
- 当变量赋值语句执行时，计算等式右边的表达式值，其值同时（即在当前仿真事件时）赋给变量对象。
- 通常只有在需要临时保存数据值的时候，才使用变量。

```
1 PROCESS(a)
2   VARIABLE events_on_a: Integer := -1;
3 BEGIN
4   events_on_a := events_on_a + 1;
5 END PROCESS;
```

- 仿真开始时，进程执行一次。
- 之后一旦信号a上有事件发生，进程就会被激活。
- 仿真结束时，变量events_on_a就记录了信号a上所发生事件的总数。

变量赋值

- 变量可以在进程或子程序内部声明和使用。
- 当变量赋值语句执行时，计算等式右边的表达式值，其值同时（即在当前仿真事件时）赋给变量对象。
- 通常只有在需要临时保存数据值的时候，才使用变量。

```
1 PROCESS(a)
2   VARIABLE events_on_a: Integer := -1;
3 BEGIN
4   events_on_a := events_on_a + 1;
5 END PROCESS;
```

- 仿真开始时，进程执行一次。
- 之后一旦信号a 上有事件发生，进程就会被激活。
- 仿真结束时，变量events_on_a 就记录了信号a 上所发生事件的总数。

变量赋值

- 变量可以在进程或子程序内部声明和使用。
- 当变量赋值语句执行时，计算等式右边的表达式值，其值同时（即在当前仿真事件时）赋给变量对象。
- 通常只有在需要临时保存数据值的时候，才使用变量。

```
1 PROCESS(a)
2   VARIABLE events_on_a: Integer := -1;
3 BEGIN
4   events_on_a := events_on_a + 1;
5 END PROCESS;
```

- 仿真开始时，进程执行一次。
- 之后一旦信号a 上有事件发生，进程就会被激活。
- 仿真结束时，变量events_on_a 就记录了信号a 上所发生事件的总数。

共享变量

VHDL'87 不容许使用全局变量

- 由于各条进程语句都是并行执行的，因此在仿真期间不能保证某个进程会比其他任一进程先被执行。
- 当一个进程要对某个全局变量进行赋值，而另一个进程又要读取该全局变量时，就会导致系统行为的不确定。
- 信号是进程间通信的唯一方法。

VHDL'93 引入了全局变量，也称为共享变量。

```
1 SHARED VARIABLE count: Integer;
```

- 在进程或子程序外部声明，但只能在进程或子程序内部使用。
- 可以被多个进程读取和更新。
- 适用于系统级建模和面向对象编程。
- 但同时也可能导致系统行为的不确定。

共享变量

VHDL'87 不容许使用全局变量

- 由于各条进程语句都是并行执行的，因此在仿真期间不能保证某个进程会比其他任一进程先被执行。
- 当一个进程要对某个全局变量进行赋值，而另一个进程又要读取该全局变量时，就会导致系统行为的不确定。
- 信号是进程间通信的唯一方法。

VHDL'93 引入了全局变量，也称为共享变量。

```
1 SHARED VARIABLE count: Integer;
```

- 在进程或子程序外部声明，但只能在进程或子程序内部使用。
- 可以被多个进程读取和更新。
- 适用于系统级建模和面向对象编程。
- 但同时也可能导致系统行为的不确定。

③ 信号赋值语句

信号名 <= [TRANSPORT | REJECT 时间表达式 INERTIAL] 表达式 [AFTER 时间表达式],{···};

```
1  ARCHITECTURE bhv3 OF dff_3 IS
2    SIGNAL a: Bit;
3  BEGIN
4    d3: PROCESS(clk)
5    BEGIN
6      IF clk'Event AND clk='1' THEN
7        a <= d;
8        q <= a;
9      END IF;
10   END PROCESS d3;
11 END bhv3;
```

③ 信号赋值语句

信号名 <= [TRANSPORT | REJECT 时间表达式 INERTIAL] 表达式 [AFTER 时间表达式],{···};

```
1  ARCHITECTURE bhv3 OF dff_3 IS
2    SIGNAL a: Bit;
3  BEGIN
4    d3: PROCESS(clk)
5    BEGIN
6      IF clk'Event AND clk='1' THEN
7        a <= d;
8        q <= a;
9      END IF;
10   END PROCESS d3;
11 END bhv3;
```

信号赋值

- 可以出现在进程内，也可以出现在进程外（并行信号赋值）。
- 首先计算等式右边表达式的值，在指定的延迟时间后赋给信号。
- 右边表达式在语句执行的时候（当前仿真时间）计算，而不是在指定的延迟之后计算。

延迟

1. Δ 延迟

   ```
   信号名 <= 表达式;
   ```

2. 传输延迟

   ```
   信号名 <= TRANSPORT 表达式 AFTER 时间表达式;
   ```

3. 惯性延迟

   ```
   信号名 <= [ REJECT 脉冲拒绝限制 INERTIAL ] 表达式 AFTER 时间表达式;
   ```

信号赋值

- 可以出现在进程内，也可以出现在进程外（并行信号赋值）。
- 首先计算等式右边表达式的值，在指定的延迟时间后赋给信号。
- 右边表达式在语句执行的时候（当前仿真时间）计算，而不是在指定的延迟之后计算。

延迟

1. Δ 延迟

   信号名 <= 表达式;

2. 传输延迟

   信号名 <= TRANSPORT 表达式 AFTER 时间表达式;

3. 惯性延迟

   信号名 <= [REJECT 脉冲拒绝限制 INERTIAL] 表达式 AFTER 时间表达式;

Δ 延迟

- Δ 延迟是一段非常小的延迟（无限小）。
- 它不是任何实际的延迟，也不会导致仿真时间的增加。
- 这个延迟建立了一个硬件模型，其模型中事件的发生只需要一段很小的时间。
- 仿真时间的每一个单元都可以看作是由无限个 Δ 延迟组成。

dff_1

```
1 d1: PROCESS(clk)
2   VARIABLE a: Bit;
3 BEGIN
4   IF clk'Event AND clk='1' THEN
5     a := d;
6     q <= a;
7   END IF;
8 END PROCESS d1;
```

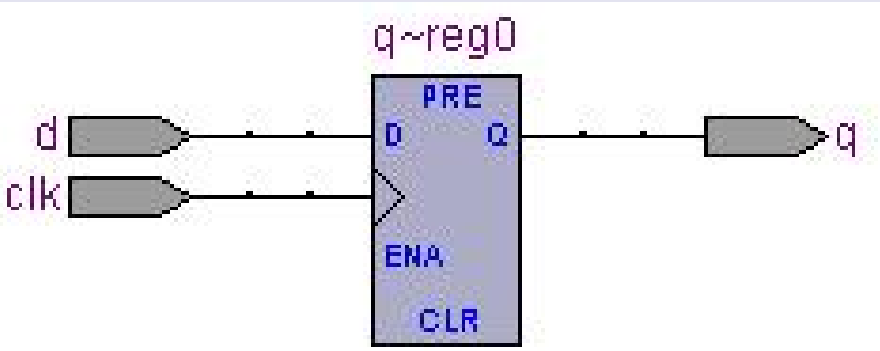
dff_3

```
1 d3: PROCESS(clk)
2 BEGIN
3   IF clk'Event AND clk='1' THEN
4     a <= d;
5     q <= a;
6   END IF;
7 END PROCESS d3;
```

△ 延迟

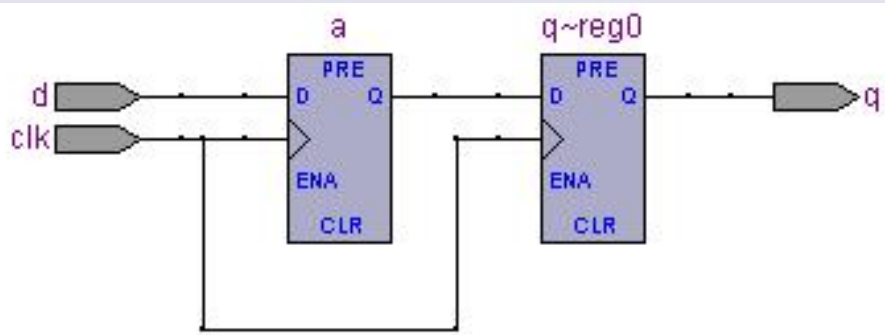
dff_1

```
1 d1: PROCESS(clk)
2   VARIABLE a: Bit;
3 BEGIN
4   IF clk'Event AND clk='1' THEN
5     a := d;
6     q <= a;
7   END IF;
8 END PROCESS d1;
```



dff_3

```
1 d3: PROCESS(clk)
2 BEGIN
3   IF clk'Event AND clk='1' THEN
4     a <= d;
5     q <= a;
6   END IF;
7 END PROCESS d3;
```

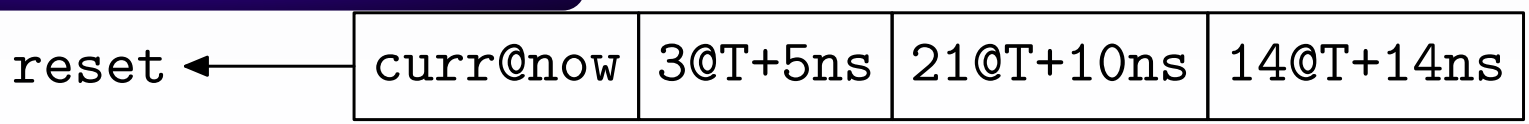


信号驱动器

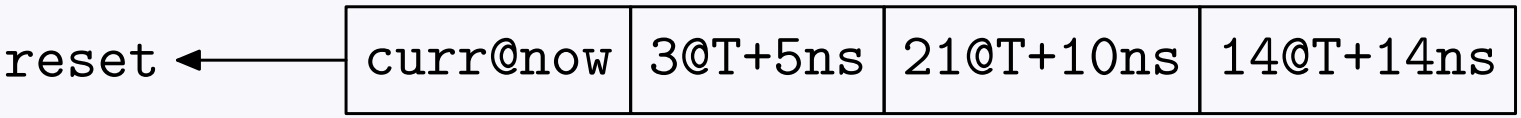
- 进程中赋值的每一个信号都会产生一个驱动器。
- 信号的驱动器保存着信号的当前值和将来所有的值，并把这些值作为一个或多个事件序列来处理。
- 在这些事件序列中存储了将要出现在信号上的值，以及这些值出现的时间。
- 驱动器的所有处理事件都是按时间进行增序排列的。
- 一个驱动器至少包含一个处理事件（信号的初值）。

```
1 PROCESS
2 BEGIN
3   reset <= 3 AFTER 5ns, 21 AFTER 10ns, 14 AFTER 14ns;
4 END PROCESS;
```

一个驱动器和三个波形要素

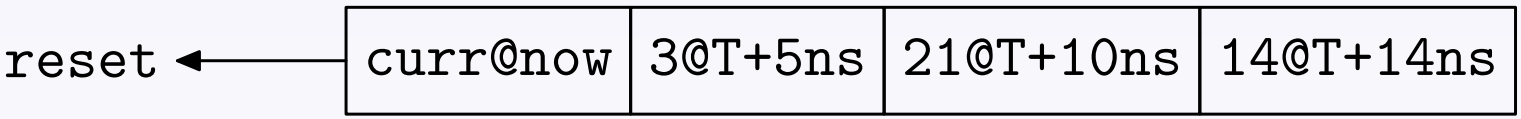


```
1 PROCESS
2 BEGIN
3   reset <= 3 AFTER 5ns, 21 AFTER 10ns, 14 AFTER 14ns;
4 END PROCESS;
```



1. 当信号赋值语句在时间T 执行时，信号reset 的驱动器增加三个新的处理事件。
2. 第一个处理事件是信号的当前值curr@now。
3. 当仿真时间到T+5ns 时，第一个处理事件从驱动器中被删除，reset 的值变为3。
4. 当时间到T+10ns 时，第二个处理事件被删除，reset 的值变为21。
5. 当时间到T+14ns 时，第三个处理事件被删除，reset 的值变为14。

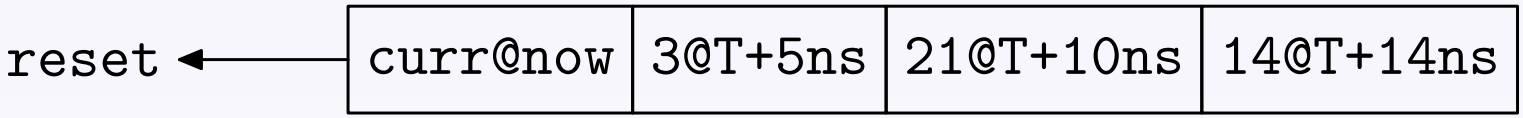
```
1 PROCESS
2 BEGIN
3   reset <= 3 AFTER 5ns, 21 AFTER 10ns, 14 AFTER 14ns;
4 END PROCESS;
```



1. 当信号赋值语句在时间T 执行时，信号reset 的驱动器增加三个新的处理事件。
2. 第一个处理事件是信号的当前值curr@now。
3. 当仿真时间到T+5ns 时，第一个处理事件从驱动器中被删除，reset 的值变为3。
4. 当时间到T+10ns 时，第二个处理事件被删除，reset 的值变为21。
5. 当时间到T+14ns 时，第三个处理事件被删除，reset 的值变为14。

如果在同一进程中有其他的信号赋值语句对 reset 赋值该怎么办？

```
1 PROCESS
2 BEGIN
3   reset <= 3 AFTER 5ns, 21 AFTER 10ns, 14 AFTER 14ns;
4 END PROCESS;
```



1. 当信号赋值语句在时间T 执行时，信号reset 的驱动器增加三个新的处理事件。
2. 第一个处理事件是信号的当前值curr@now。
3. 当仿真时间到T+5ns 时，第一个处理事件从驱动器中被删除，reset 的值变为3。
4. 当时间到T+10ns 时，第二个处理事件被删除，reset 的值变为21。
5. 当时间到T+14ns 时，第三个处理事件被删除，reset 的值变为14。

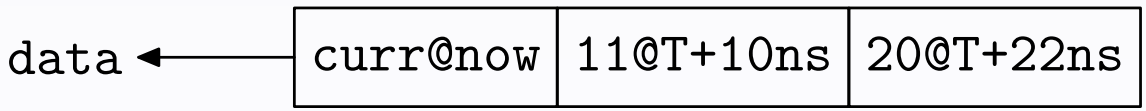
如果在同一进程中有其他的信号赋值语句对 reset 赋值该怎么办？

- 由于进程内的信号只有一个驱动器，第二个信号赋值的处理事件将会修改已经出现在驱动器上的处理事件。
- 取决于是否使用传输或惯性延迟模型。

传输延迟对信号驱动器的影响

```
1 PROCESS
2 BEGIN
3    data <= TRANSPORT 11 AFTER 10ns;
4    data <= TRANSPORT 20 AFTER 22ns;
5    data <= TRANSPORT 35 AFTER 18ns;
6 END PROCESS;
```

1. 当第一条信号赋值语句执行时，事件11@T+10ns 传递到驱动器上；
2. 第二条信号赋值语句执行完后，时间20@T+22ns 传递到驱动器上，因为这个事件的延迟22ns 大于驱动器等待的时间10ns。

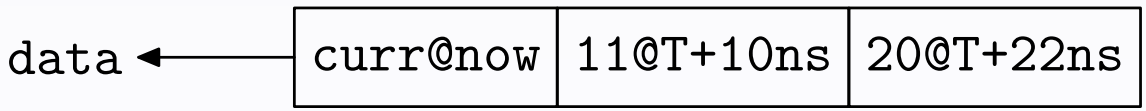


- 当第三条信号赋值语句执行时，新事件35@T+18ns 使得事件20@T+22ns 被删除。
- 这是因为采用了传输延迟，新的事件延迟18ns 小于驱动器上一次的事件延迟22ns。新的事件会删除驱动器上所有比该新事件还要晚发生的事件。

传输延迟对信号驱动器的影响

```
1 PROCESS
2 BEGIN
3    data <= TRANSPORT 11 AFTER 10ns;
4    data <= TRANSPORT 20 AFTER 22ns;
5    data <= TRANSPORT 35 AFTER 18ns;
6 END PROCESS;
```

1. 当第一条信号赋值语句执行时，事件11@T+10ns 传递到驱动器上；
2. 第二条信号赋值语句执行完后，时间20@T+22ns 传递到驱动器上，因为这个事件的延迟22ns 大于驱动器等待的时间10ns。

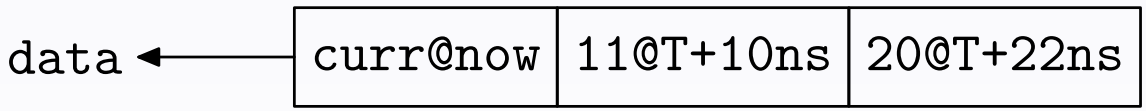


- 当第三条信号赋值语句执行时，新事件35@T+18ns 使得事件20@T+22ns 被删除。
- 这是因为采用了传输延迟，新的事件延迟18ns 小于驱动器上一次的事件延迟22ns。新的事件会删除驱动器上所有比该新事件还要晚发生的事件。

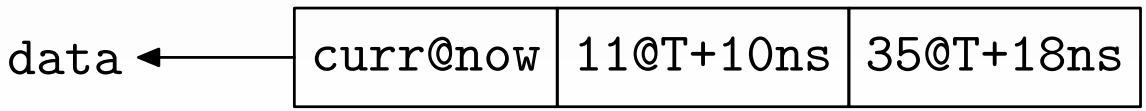
传输延迟对信号驱动器的影响

```
1 PROCESS
2 BEGIN
3    data <= TRANSPORT 11 AFTER 10ns;
4    data <= TRANSPORT 20 AFTER 22ns;
5    data <= TRANSPORT 35 AFTER 18ns;
6 END PROCESS;
```

1. 当第一条信号赋值语句执行时，事件11@T+10ns 传递到驱动器上；
2. 第二条信号赋值语句执行完后，时间20@T+22ns 传递到驱动器上，因为这个事件的延迟22ns 大于驱动器等待的时间10ns。



- 当第三条信号赋值语句执行时，新事件35@T+18ns 使得事件20@T+22ns 被删除。
- 这是因为采用了传输延迟，新的事件延迟18ns 小于驱动器上一次的事件延迟22ns。新的事件会删除驱动器上所有比该新事件还要晚发生的事件。



惯性延迟对信号驱动器的影响

- 采用惯性延迟时，信号值和延迟值都会影响事件的删除和添加。
- 如果一个新事件的延迟比当前事件早，将会删除后者而不管这两个事件的信号值大小。
- 对于落在第一个新事件时间窗口内的事件，即对于发生在新事件发生时间 − 脉冲拒绝限制到新事件发生时间内的事件，将会检查它们的值。如果它们的值与新事件的值不同，删除这些事件；否则，保留这些事件。

```
1 PROCESS
2 BEGIN
3   data <= 11 AFTER 10ns;
4   data <= REJECT 15ns INERTIAL 22 AFTER 20ns;
5   data <= 33 AFTER 15ns;
6 END PROCESS;
```

1. 事件11@10ns 首先被添加到驱动器。
2. 第二个事件22@20ns 使得驱动器上的事件11@10ns 被删除。这是因为落在20ns（第一个新事件的时间）和(20ns-15ns)（第一个新事件的时间减去脉冲拒绝限制）窗

惯性延迟对信号驱动器的影响

```
1 PROCESS
2 BEGIN
3    data <= 11 AFTER 10ns;
4    data <= REJECT 15ns INERTIAL 22 AFTER 20ns;
5    data <= 33 AFTER 15ns;
6 END PROCESS;
```

1. 事件11@10ns 首先被添加到驱动器。
2. 第二个事件22@20ns 使得驱动器上的事件11@10ns 被删除。这是因为落在20ns（第一个新事件的时间）和(20ns-15ns)（第一个新事件的时间减去脉冲拒绝限制）窗口的事件是11@10ns，而它的值11 与添加的第一个新事件的值22 不同。
- 第三条赋值语句的执行使得事件22@20ns 从驱动器上被删除。因为新事件的延迟15ns 小于驱动器当前事件的延迟20ns（类似传输延迟情况）。

顺序语句

惯性延迟对信号驱动器的影响

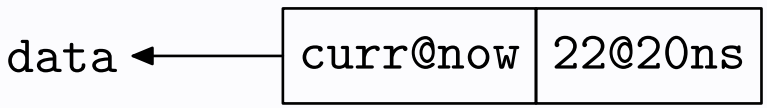
```
1 PROCESS
2 BEGIN
3    data <= 11 AFTER 10ns;
4    data <= REJECT 15ns INERTIAL 22 AFTER 20ns;
5    data <= 33 AFTER 15ns;
6 END PROCESS;
```

1. 事件11@10ns 首先被添加到驱动器。
2. 第二个事件22@20ns 使得驱动器上的事件11@10ns 被删除。这是因为落在20ns（第一个新事件的时间）和(20ns-15ns)（第一个新事件的时间减去脉冲拒绝限制）窗口的事件是11@10ns，而它的值11 与添加的第一个新事件的值22 不同。
- 第三条赋值语句的执行使得事件22@20ns 从驱动器上被删除。因为新事件的延迟15ns 小于驱动器当前事件的延迟20ns（类似传输延迟情况）。

惯性延迟对信号驱动器的影响

```
1 PROCESS
2 BEGIN
3    data <= 11 AFTER 10ns;
4    data <= REJECT 15ns INERTIAL 22 AFTER 20ns;
5    data <= 33 AFTER 15ns;
6 END PROCESS;
```

1. 事件11@10ns 首先被添加到驱动器。
2. 第二个事件22@20ns 使得驱动器上的事件11@10ns 被删除。这是因为落在20ns（第一个新事件的时间）和(20ns-15ns)（第一个新事件的时间减去脉冲拒绝限制）窗口的事件是11@10ns，而它的值11 与添加的第一个新事件的值22 不同。

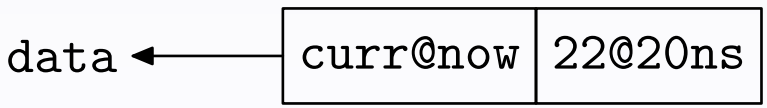


- 第三条赋值语句的执行使得事件22@20ns 从驱动器上被删除。因为新事件的延迟15ns 小于驱动器当前事件的延迟20ns（类似传输延迟情况）。

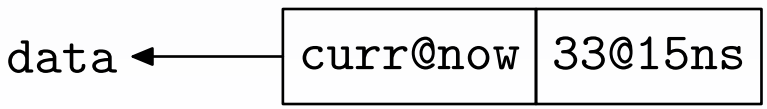
惯性延迟对信号驱动器的影响

```
1 PROCESS
2 BEGIN
3    data <= 11 AFTER 10ns;
4    data <= REJECT 15ns INERTIAL 22 AFTER 20ns;
5    data <= 33 AFTER 15ns;
6 END PROCESS;
```

1. 事件11@10ns 首先被添加到驱动器。
2. 第二个事件22@20ns 使得驱动器上的事件11@10ns 被删除。这是因为落在20ns（第一个新事件的时间）和(20ns-15ns)（第一个新事件的时间减去脉冲拒绝限制）窗口的事件是11@10ns，而它的值11 与添加的第一个新事件的值22 不同。



- 第三条赋值语句的执行使得事件22@20ns 从驱动器上被删除。因为新事件的延迟15ns 小于驱动器当前事件的延迟20ns（类似传输延迟情况）。



传输延迟

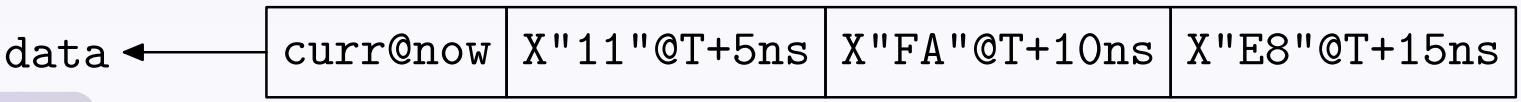
```
1 PROCESS
2 BEGIN
3    data <= TRANSPORT X"01" AFTER 5ns, X"FA" AFTER 10ns,
4                                X"E8" AFTER 15ns;
5    data <= TRANSPORT X"B5" AFTER 12ns;
6 END PROCESS;
```

惯性延迟

```
1 PROCESS
2 BEGIN
3    data <= 1 AFTER 5ns, 21 AFTER 9ns, 6 AFTER 10ns,
4              12 AFTER 19ns;
5    data <= REJECT 4ns INERTIAL 6 AFTER 12ns, 20 AFTER 19ns;
6 END PROCESS;
```

传输延迟

```
1 PROCESS
2 BEGIN
3   data <= TRANSPORT X"01" AFTER 5ns, X"FA" AFTER 10ns,
4                     X"E8" AFTER 15ns;
5   data <= TRANSPORT X"B5" AFTER 12ns;
6 END PROCESS;
```



惯性延迟

```
1 PROCESS
2 BEGIN
3   data <= 1 AFTER 5ns, 21 AFTER 9ns, 6 AFTER 10ns,
4           12 AFTER 19ns;
5   data <= REJECT 4ns INERTIAL 6 AFTER 12ns, 20 AFTER 19ns;
6 END PROCESS;
```

传输延迟

```
1 PROCESS
2 BEGIN
3   data <= TRANSPORT X"01" AFTER 5ns, X"FA" AFTER 10ns,
4                     X"E8" AFTER 15ns;
5   data <= TRANSPORT X"B5" AFTER 12ns;
6 END PROCESS;
```

data ←	curr@now	X"11"@T+5ns	X"FA"@T+10ns	X"E8"@T+15ns
data ←	curr@now	X"11"@T+5ns	X"FA"@T+10ns	X"B5"@T+12ns

惯性延迟

```
1 PROCESS
2 BEGIN
3   data <= 1 AFTER 5ns, 21 AFTER 9ns, 6 AFTER 10ns,
4           12 AFTER 19ns;
5   data <= REJECT 4ns INERTIAL 6 AFTER 12ns, 20 AFTER 19ns;
6 END PROCESS;
```

传输延迟

```
1 PROCESS
2 BEGIN
3   data <= TRANSPORT X"01" AFTER 5ns, X"FA" AFTER 10ns,
4                     X"E8" AFTER 15ns;
5   data <= TRANSPORT X"B5" AFTER 12ns;
6 END PROCESS;
```

惯性延迟

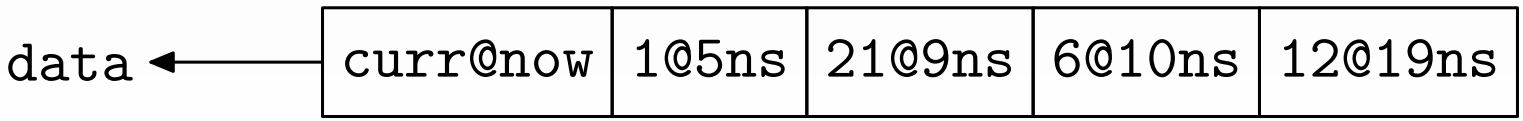
```
1 PROCESS
2 BEGIN
3   data <= 1 AFTER 5ns, 21 AFTER 9ns, 6 AFTER 10ns,
4           12 AFTER 19ns;
5   data <= REJECT 4ns INERTIAL 6 AFTER 12ns, 20 AFTER 19ns;
6 END PROCESS;
```

传输延迟

```
1  PROCESS
2  BEGIN
3    data <= TRANSPORT X"01" AFTER 5ns, X"FA" AFTER 10ns,
4                      X"E8" AFTER 15ns;
5    data <= TRANSPORT X"B5" AFTER 12ns;
6  END PROCESS;
```

惯性延迟

```
1  PROCESS
2  BEGIN
3    data <= 1 AFTER 5ns, 21 AFTER 9ns, 6 AFTER 10ns,
4              12 AFTER 19ns;
5    data <= REJECT 4ns INERTIAL 6 AFTER 12ns, 20 AFTER 19ns;
6  END PROCESS;
```



传输延迟

```
1 PROCESS
2 BEGIN
3   data <= TRANSPORT X"01" AFTER 5ns, X"FA" AFTER 10ns,
4                     X"E8" AFTER 15ns;
5   data <= TRANSPORT X"B5" AFTER 12ns;
6 END PROCESS;
```

惯性延迟

```
1 PROCESS
2 BEGIN
3   data <= 1 AFTER 5ns, 21 AFTER 9ns, 6 AFTER 10ns,
4           12 AFTER 19ns;
5   data <= REJECT 4ns INERTIAL 6 AFTER 12ns, 20 AFTER 19ns;
6 END PROCESS;
```

data ←	curr@now	1@5ns	21@9ns	6@10ns	12@19ns
data ←	curr@now	1@5ns	6@10ns	6@12ns	20@19ns

传输延迟与惯性延迟对驱动器的影响

传输延迟

- 在第一个新事件的延迟时间之后，发生在驱动器上的所有事件被删除。

惯性延迟

- 在第一个新事件的延迟时间之后，发生在驱动器上的所有事件被删除。
- 对于驱动器上所有发生在第一个新事件时间和新事件时间减去脉冲拒绝限制之间的老事件，删除其中与第一个新事件值不同的老事件。

传输延迟与惯性延迟对驱动器的影响

传输延迟

- 在第一个新事件的延迟时间之后，发生在驱动器上的所有事件被删除。

惯性延迟

- 在第一个新事件的延迟时间之后，发生在驱动器上的所有事件被删除。
- 对于驱动器上所有发生在第一个新事件时间和新事件时间减去脉冲拒绝限制之间的老事件，删除其中与第一个新事件值不同的老事件。

多驱动源信号——决断信号

- 不管进程中一个信号被赋值多少次，此信号只有一个驱动器；
- 每个进程为每个信号至多生成一个驱动器。
- 每条并行信号赋值语句都为所赋值的信号生成一个驱动器。

如果同一个信号有多个赋值，将会发生什么情况呢？

- 信号有多于一个的驱动器，需要采用一种方法计算信号的有效值（决断函数）。
- 决断信号具有多个驱动源和一个决断函数。

多驱动源信号——决断信号

- 不管进程中一个信号被赋值多少次，此信号只有一个驱动器；
- 每个进程为每个信号至多生成一个驱动器。
- 每条并行信号赋值语句都为所赋值的信号生成一个驱动器。

如果同一个信号有多个赋值，将会发生什么情况呢？

- 信号有多于一个的驱动器，需要采用一种方法计算信号的有效值（决断函数）。
- 决断信号具有多个驱动源和一个决断函数。

多驱动源信号——决断信号

- 不管进程中一个信号被赋值多少次，此信号只有一个驱动器；
- 每个进程为每个信号至多生成一个驱动器。
- 每条并行信号赋值语句都为所赋值的信号生成一个驱动器。

如果同一个信号有多个赋值，将会发生什么情况呢？

- 信号有多于一个的驱动器，需要采用一种方法计算信号的有效值（决断函数）。
- 决断信号具有多个驱动源和一个决断函数。

多驱动源信号——决断信号

- 不管进程中一个信号被赋值多少次，此信号只有一个驱动器；
- 每个进程为每个信号至多生成一个驱动器。
- 每条并行信号赋值语句都为所赋值的信号生成一个驱动器。

如果同一个信号有多个赋值，将会发生什么情况呢？

- 信号有多于一个的驱动器，需要采用一种方法计算信号的有效值（决断函数）。
- 决断信号具有多个驱动源和一个决断函数。

多驱动源信号——决断信号

- 不管进程中一个信号被赋值多少次，此信号只有一个驱动器；
- 每个进程为每个信号至多生成一个驱动器。
- 每条并行信号赋值语句都为所赋值的信号生成一个驱动器。

如果同一个信号有多个赋值，将会发生什么情况呢？

- 信号有多于一个的驱动器，需要采用一种方法计算信号的有效值(决断函数)。
- 决断信号具有多个驱动源和一个决断函数。

信号与变量

现象

- 信号赋值是有延迟的，即使在不考虑器件实际延迟的行为仿真和RTL仿真中，也会因为引入的仿真Δ机制，使得信号的赋值具有延迟；而变量赋值是没有延迟的。
- 信号除具有当前值之外还有很多属性；而变量只有当前值。
- 信号值的变化可以激活被挂起的进程；而变量无此功能。
- 使用全局信号不会导致系统行为的不确定性；而使用全局变量（共享变量）则可能导致系统行为的不确定。

本质

- 信号与硬件中互连元件端口的“连线”相对应；而变量在硬件中没有明确的对应物，变量只是为了便于设计实体的行为描述而定义的数据暂存区。

信号与变量

现象

- 信号赋值是有延迟的，即使在不考虑器件实际延迟的行为仿真和RTL仿真中，也会因为引入的仿真Δ机制，使得信号的赋值具有延迟；而变量赋值是没有延迟的。
- 信号除具有当前值之外还有很多属性；而变量只有当前值。
- 信号值的变化可以激活被挂起的进程；而变量无此功能。
- 使用全局信号不会导致系统行为的不确定性；而使用全局变量（共享变量）则可能导致系统行为的不确定。

本质

- 信号与硬件中互连元件端口的“连线”相对应；而变量在硬件中没有明确的对应物，变量只是为了便于设计实体的行为描述而定义的数据暂存区。

信号与变量

现象

- 信号赋值是有延迟的，即使在不考虑器件实际延迟的行为仿真和RTL仿真中，也会因为引入的仿真Δ机制，使得信号的赋值具有延迟；而变量赋值是没有延迟的。
- 信号除具有当前值之外还有很多属性；而变量只有当前值。
- 信号值的变化可以激活被挂起的进程；而变量无此功能。
- 使用全局信号不会导致系统行为的不确定性；而使用全局变量（共享变量）则可能导致系统行为的不确定。

本质

- 信号与硬件中互连元件端口的“连线”相对应；而变量在硬件中没有明确的对应物，变量只是为了便于设计实体的行为描述而定义的数据暂存区。

信号与变量

现象

- 信号赋值是有延迟的，即使在不考虑器件实际延迟的行为仿真和RTL仿真中，也会因为引入的仿真Δ机制，使得信号的赋值具有延迟；而变量赋值是没有延迟的。
- 信号除具有当前值之外还有很多属性；而变量只有当前值。
- 信号值的变化可以激活被挂起的进程；而变量无此功能。
- 使用全局信号不会导致系统行为的不确定性；而使用全局变量（共享变量）则可能导致系统行为的不确定。

本质

- 信号与硬件中互连元件端口的“连线”相对应；而变量在硬件中没有明确的对应物，变量只是为了便于设计实体的行为描述而定义的数据暂存区。

信号与变量

现象

- 信号赋值是有延迟的，即使在不考虑器件实际延迟的行为仿真和RTL仿真中，也会因为引入的仿真Δ机制，使得信号的赋值具有延迟；而变量赋值是没有延迟的。
- 信号除具有当前值之外还有很多属性；而变量只有当前值。
- 信号值的变化可以激活被挂起的进程；而变量无此功能。
- 使用全局信号不会导致系统行为的不确定性；而使用全局变量（共享变量）则可能导致系统行为的不确定。

本质

- 信号与硬件中互连元件端口的“连线”相对应；而变量在硬件中没有明确的对应物，变量只是为了便于设计实体的行为描述而定义的数据暂存区。

信号与变量

现象

- 信号赋值是有延迟的，即使在不考虑器件实际延迟的行为仿真和RTL仿真中，也会因为引入的仿真Δ机制，使得信号的赋值具有延迟；而变量赋值是没有延迟的。
- 信号除具有当前值之外还有很多属性；而变量只有当前值。
- 信号值的变化可以激活被挂起的进程；而变量无此功能。
- 使用全局信号不会导致系统行为的不确定性；而使用全局变量（共享变量）则可能导致系统行为的不确定。

本质

- 信号与硬件中互连元件端口的“连线”相对应；而变量在硬件中没有明确的对应物，变量只是为了便于设计实体的行为描述而定义的数据暂存区。