3 VHDL语言

VHDL: VHSIC Hardware Description Language.

3.1 VHDL语言基础
3.2 VHDL基本结构
3.3 VHDL语句
3.4 状态机在VHDL中的实现
3.5 常用电路VHDL程序
3.6 VHDL仿真
3.7 VHDL综合

HDL----Hardware Description Language

一种用于描述数字电路的功能或行为的语言。目的是提为电路设计效率，缩短设计周期，减小设计成本，可在芯片制造前进行有效的仿真和错误检测。

优点：

HDL设计的电路能获得非常抽象级的描述。如基于RTL(Register Transfer Level)描述的IC，可用于不同的工艺。

HDL设计的电路，在设计的前期，就可以完成电路的功能级的验证。

HDL设计的电路类似于计算机编程。

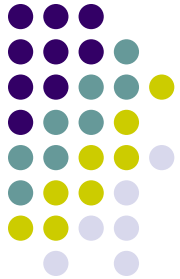
常用的HDL语言：VHDL 、Verilog HDL

VHDL 概述：

VHDL ⟶ VHSIC Hardwarter Description Language

VHSIC ⟶ Very High speed integrated circuit

- **VHDL是美国国防部在20世纪80年代初为实现其高速集成电路硬件VHSIC计划提出的描述语言；**
- **IEEE从1986年开始致力于VHDL标准化工作，融合了其它ASIC芯片制造商开发的硬件描述语言的优点，于93年形成了标准版本（IEEE.std_1164）。**
- **1995年，我国国家技术监督局推荐VHDL做为电子设计自动化硬件描述语言的国家标准。**



VHDL优点:

- 覆盖面广，系统硬件描述能力强，是一个多层次的硬件描述语言;
- VHDL语言具有良好的可读性，既可以被计算机接受，也容易被人们所理解;
- VHDL语言可以与工艺无关编程;
- VHDL语言已做为一种IEEE的工业标准，便于使用、交流和推广。

VHDL语言的不足之处:

设计的最终实现取决于针对目标器件的编程器，工具的不同会导致综合质量不一样。

3.1 VHDL语言基础

3.1.1 标识符（Identifiers）

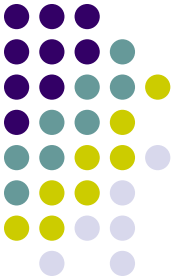
标识符用来定义常数、变量、信号、端口、子程序或参数的名字，由字母（A~Z，a~z）、数字（0~9）和下划线（_）字符组成。

要求：

- 首字符必须是字母
- 末字符不能为下划线
- 不允许出现两个连续的下划线
- 不区分大小写
- VHDL定义的保留字（关键字），不能用作标识符
- 标识符字符最长可以是32个字符。

注释由两个连续的虚线（－－）引导。

关键字（保留字）：



关键字（keyword）是VHDL中具有特别含义的单词，只能做为固定的用途，用户不能用其做为标识符。

例如：**ABS，ACCESS，AFTER，ALL，AND，ARCHITECTURE，ARRAY，ATTRIBUTE，BEGIN，BODY，BUFFER，BUS，CASE，COMPONENT，CONSTANT，DISCONNECT，DOWNTO，ELSE，ELSIF，END，ENTITY，EXIT，FILE，FOR，FUNCTION，GENERIC，GROUP，IF，INPURE，IN，INOUT，IS，LABEL，LIBRARY，LINKAGE，LOOP，MAP，MOD，NAND，NEW，NEXT，NOR，NOT，NULL，OF，ON，OPEN，OR，OTHERS，OUT，PACKAGE，POUT，PROCEDURE，PROCESS，PURE，RANGE，RECODE，REM，REPORT，RETURN，ROL，ROR，SELECT，SHARED，SIGNAL，SLA，SLL，SRA，SUBTYPE，THEN，TRANSPORT，TO，TYPE，UNAFFECTED，UNITS，UNTIL，USE，VARIABLE，WAIT，WHEN，WHILE，WITH，XOR，XNOR**

3.1.2 数据对象（Date Objects）

数据对象包括常量、变量、信号和文件四种类型。

- **常量Constant**

常量是对某一常量名赋予一个固定的值，而且只能赋值一次。通常赋值在程序开始前进行，该值的数据类型则在说明语句中指明。

Constant 常数名：数据类型：＝表达式

Constant Vcc：real:=5.0;　--定义Vcc的数据类型是实数，赋值为5.0V

Constant bus_width：integer := 8;　--定义总线宽度为常数8

常量所赋的值应和定义的数据类型一致；

常量在程序包、实体、构造体或进程的说明性区域内必须加以说明。定义在程序包内的常量可供所含的任何实体、构造体所引用，定义在实体说明内的常量只能在该实体内可见，定义在进程说明性区域中的常量只能在该进程内可见。

➢ 变量Variable

变量只能在进程语句、函数语句和过程语句结构中使用。变量的赋值是直接的，非预设的，分配给变量的值立即成为当前值，变量不能表达“连线”或存储元件，不能设置传输延迟量。

变量定义语句：

Variable 变量名：数据类型 :=初始值;

Variable count: integer 0 to 255:=20； -- 定义count整数变量，变化范围0～255，初始值为20。

变量赋值语句：

目标变量名 := 表达式;

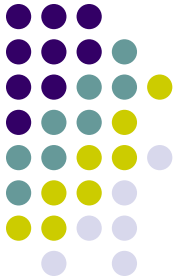
x:=10.0; -- 实数变量赋值为10.0

Y:=1.5+x; -- 运算表达式赋值，注意表达式必须与目标变量的数据类型相同

A(3 to 6):=(“1101”); --位矢量赋值

➢ 信号Signal

信号表示逻辑门的输入或输出，类似于连接线，也可以表达存储元件的状态。信号通常在构造体、程序包和实体中说明。



信号定义语句：

Signal 信号名: 数据类型 :=初始值

Signal clock：bit :=‘0’； --定义时钟信号类型，初始值为0

Signal count：BIT_VECTOR(3 DOWNTO 0); --定义count为4位位矢量

信号赋值语句：

目标信号名 <= 表达式;

x<=9;

Z<=x after 5 ns; -- 在5ns后将x的值赋予z

3.1.2 数据类型

➢ VHDL的预定义数据类型

在VHDL标准程序包STANDARD中定义好，实际使用过程中，已自动包含进VHDL源文件中，不需要通过USE语句显式调用。

- 布尔：(Boolean)

TYPE BOOLEAN IS (FALSE, TRUE); -- 取值为FALSE和TRUE，不是数值，不能运算，一般用于关系运算符

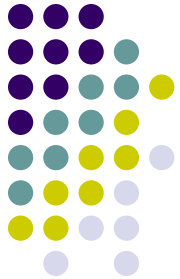
- 位: (Bit)

TYPE BIT IS ('0','1'); --取值为0和1，用于逻辑运算

- 位矢量: (Bit_Vector)

TYPE BIT_VECTOR IS ARRAY (Natural range<>) OF BIT; -- 基于Bit类型的数组，用于逻辑运算

SIGNAL a：Bit_Vector(0 TO 7)； SIGNAL a：Bit_Vector (7 DOWNTO 0)



- 字符：(Character)

TYPE CHARACTER IS (NUL, SOH,STX, …, ' ', '!',…); --通常用 ‘’ 引起来，区分大小写；

- 字符串：(String)

VARIABLE string_var: STRING (1 TO 7);

string_var:="A B C D" ;　　-- 通常用“ ”引起来，区分大小写；

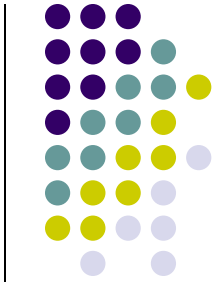
- 整数：(Integer)

取值范围 $-(2^{31}-1) \sim (2^{31}-1)$，可用32位有符号的二进制数表示

variable a：integer range -63 to 63

在实际应用中，VHDL仿真器将Integer做为有符号数处理，而VHDL综合器将Integer做为无符号数处理；

要求用RANGE子句为所定义的数限定范围，以便根据范围来决定表示此信号或变量的二进制数的位数。



- 实数：(Real)

  取值范围 -1.0E38 ～+1.0E38，仅用于仿真不可综合

  1.0 --十进制浮点数

  8＃43.6＃e+4 --八进制浮点数

  43.6E-4 --十进制浮点数

- 时间：(Time)

  物理量数据，完整的包括整数和单位两个部分，用至少一个空格隔开，仅用于仿真不可综合；

  fs,ps,ns,us,ms,sec,min,hr

- 错误等级（Severity Level）：

  表示系统状态，仅用于仿真不可综合；

TYPE severity_level IS (NOTE、WARNING、ERROR、FAILURE);

➢ IEEE预定义标准逻辑位与矢量

- 标准逻辑位（Std_Logic）

U：Uninitialized；　X：Forcing Unkown；　0：Forcing 0

1：Forcing 1　Z：High Impedance　W：Weak Unknown

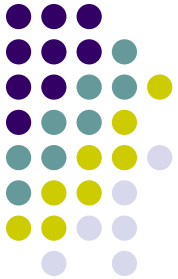
L：Weak 0　H：Weak 1　—：Don’t care

- 标准逻辑位矢量（Std_Logic_vector）

基于Std_Logic类型的数组；

使用Std_Logic和 Std_Logic_Vector要调用IEEE库中的Std_Logic_1164程序包；就综合而言，能够在数字器件中实现的是“一、0、1、Z”四种状态。

在条件语句中，必须要全面考虑Std_Logic的所有可能取值情况，否则综合器可能会插入不希望的锁存器。



➢ 用户自定义

- **TYPE 数据类型名 IS 数据类型定义 OF 基本数据类型**
  或 TYPE 数据类型名 IS 数据类型定义

数组：**type value_type is array (127 downto 0) of integer;**

type matrix_type is array (0 to 15, 0 to 31) of std_logic；

枚举：**type states is (idle，decision，read，write);**

type boolean is (false，true);

type bit is (‘0’， ‘1’)；

- **SUBTYPE 子类型名 IS 基本数据类型定义 RANGE 约束范围**

subtype digit is integer range 0 to 9;

3.1.3 数据类型转换

VHDL为强定义类型语言，不同类型的数据不能进行运算和直接赋值。

- 类型标记法

Variable A: integer; Variable B: real;

A= integer (B); B=real (A);

- 函数法

Conv_interger (A)；--由std_logic转换为integer型，在std_logic_unsigned包。

- 常数转换法 / 常量转换法

具有转换表性质的常数

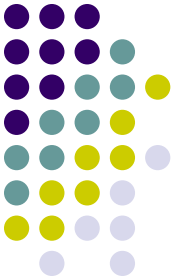
Type conv_table is array(std_logic) of bit;

Constant table: conv_table:=(‘0’|’L’=>’0’, ‘1’|’H’=>’1’, others=>’0’);

Signal a: bit; signal b: std_logic;

A<=table(b); -- 将std_logic型转换为bit型

在“STD_LOGIC_1164”、“STD_LOGIC_ARITH”和“STD_LOGIC_UNSIGNED”的程序包中提供的数据类型变换函数。



程　序　包	函　数　名	功　　能
STD_LOGIC_1164	TO_ STD_LOGIC_VECTOR (A) TO_ BIT_VECTOR (A) TO_ STD_LOGIC (A) TO_ BIT (A)	由 Bit Vector 转换成 Std_Logic_Vector 由 Std_Logic_Vector 转换成 Bit Vector 由 Bit 转换成 Std_Logic 由 Std_Logic 转换成 Bit
STD_LOGIC_ARITH	CONV_ STD_LOGIC_VECTOR (A, 位长) CONV_INTEGE (A)	由 Integer_Unsigned_Signed 转换成 Std_Logic_Vector 由 Unsigned_Signed 转换成 Integer
STD_LOGIC_UNSIGNED	CONV_INTEGE (A)	由 Std_Logic_Vector 转换成 Integer

➢ 属性

属性提供的是关于信号、类型等的指定特性。

‘event：若属性对象有事件发生，则生成布尔值“true”，常用来检查时钟边沿是否有效。

上升沿：**Clock’ EVENT AND Clock=‘1’**

’range：生成一个限制性数组对象的范围

’range: “0 to n”；　　’ reverse_range：“n downto 0”

’left：生成数据类型或数据子类型的左边界值；

’right , ’high, ’low, ’length

➢ 运算符

- 算术运算符：＋， －， *， /， MOD， REM，SLL，SRL，SLA，SRA，ROL，ROR，**，ABS
- 关系运算符：＝， /＝，<， >，<=， >=
- 逻辑运算符：AND，OR，NAND，NOR，XNOR，NOT，XOR
- 赋值运算符：<=，:=
- 关联运算符：=>
- 其他运算符：＋， －，&

并置操作符 &

```
SIGNAL a : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL d : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
 ...
a  <= '1'&'0'&d(1)&'1'  ; -- 元素与元素并置，并置后的数组长度为4
 ...
IF a & d = "101011"  THEN ... –- 在IF条件句中可以使用并置符
```

➢ 运算符优先级别

逻辑、算术运算符（ NOT， **，ABS）

乘法运算符（/ ， MOD， REM， * ）

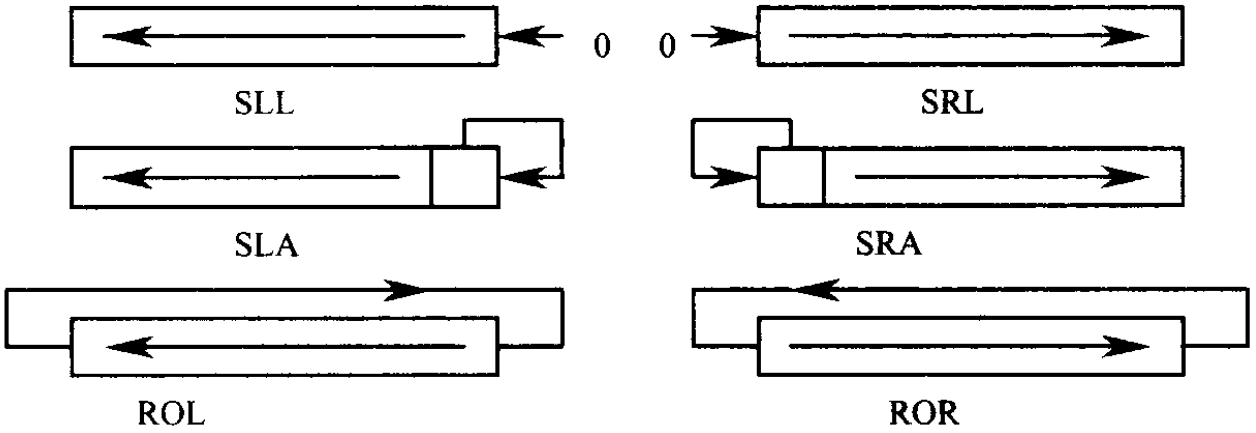
正负运算符：＋， －，

加减、并置运算符：＋， －， &

关系运算符：＝， /＝，<， > ， <=， >=

逻辑运算符：AND，OR，NAND，NOR，XNOR，NOT，XOR

移位运算符的左边为一维数组，其类型必须是BIT或BOOLEAN，右边必须是整数移位次数为整数的绝对值。



移位运算符操作示意图

SLL：将位向量左移，右边移空位补零；
SRL：将位向量右移，左边移空位补零；
SLA：将位向量左移，右边第一位的数值保持原值不变；
SRA：将位向量右移，左边第一位的数值保持原值不变；
ROL和ROR：自循环左右移位。

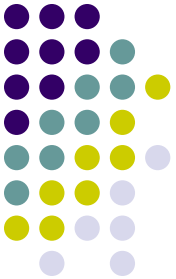
“1100”SLL1 =“1000”　“1100”SRL1 =“0110”　“1100”SLA1 =“1000”
“1100”SRA1 =“1110”　“1100”ROL1 =“1001”　“1100”ROR1 =“0110”

取余运算（a REM b）的符号与a相同，其绝对值小于b的绝对值。

例如：（-5）REM 2=（-1）　　5 REM 2=（1）

取模运算（a MOD b）的符号与b相同，其绝对值小于b的绝对值。

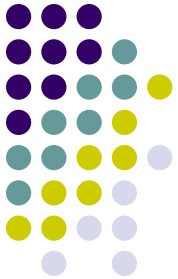
例如：（-5）MOD 2=1　　5 MOD (- 2)=（-1）



3.2 VHDL基本结构

- 实体（Entity）：描述所设计的系统的外部接口信号，定义电路设计中所有的输入和输出端口；
- 结构体 (Architecture)：描述系统内部的结构和行为；
- 包集合 (Package)：存放各设计模块能共享的数据类型、常数和子程序等；
- 配置 (Configuration)：指定实体所对应的结构体；
- 库 (Library)：存放已经编译的实体、结构体、包集合和配置。

VHDL的基本设计单元结构：程序包说明、实体说明和结构体说明三部分。



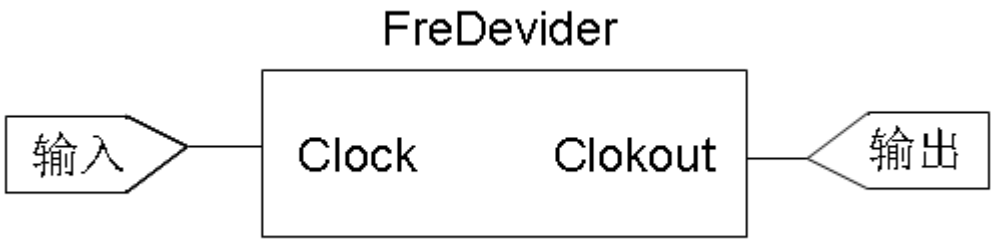
```
LIBRARY IEEE;                                    -- 库、程序包的说明调用
USE IEEE.Std_Logic_1164.ALL;
-------------------------------------------------------------------------
ENTITY FreDevider IS                             -- 实体声明
PORT
(Clock: IN Std_logic;
  Clkout: OUT Std_logic
);
END;
-------------------------------------------------------------------------
ARCHITECTURE Behavior OF FreDevider IS           -- 结构体定义
SIGNAL Clk:Std_Logic;
BEGIN
   PROCESS(Clock)
BEGIN
   IF rising_edge(Clock) THEN
       Clk<=NOT Clk;
   END IF;
END PROCESS;
Clkout<=Clk;
END;
```

3.2.1 实体（Entity）

实体描述了设计单元的输入输出接口信号或引脚，是设计实体经封装后对外的一个通信界面。

```
ENTITY 实体名 IS
[ GENERIC（常数名：数据类型：设定值）]
PORT
( 端口名1：端口方向  端口类型;
  端口名2：端口方向  端口类型;
  .
  .
  端口名n：端口方向  端口类型
);
END [实体名];
```

➢ 实体名由设计者自由命名，用来表示被设计电路芯片的名称，但是必须与**VHDL**程序的文件名称相同。要与文件名一致；




```
ENTITY FreDevider IS
PORT
(Clock: IN Std_logic;
 Clokout: OUT Std_logic
);
END;
```

实体名

端口类型

端口名

端口方向

最后一条端口声明语句无分号

➢ 类属说明

类属为设计实体与外界通信的静态信息提供通道，用来规定端口的大小、实体中子元件的数目和实体的定时特性等。

格式：

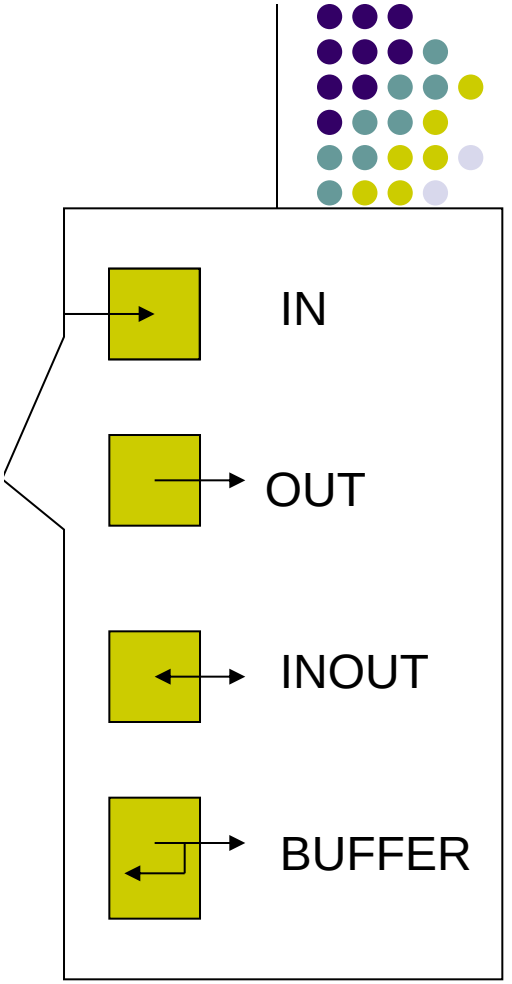
GENERIC（常数名：数据类型：设定值；
：
常数名：数据类型：设定值）；

例如： GENERIC（wide：integer:=32）； --说明宽度为32

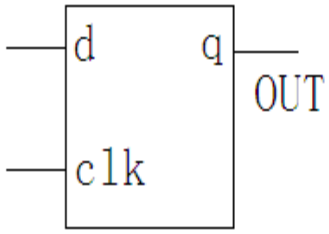
GENERIC（tmp：integer:=1ns）； --说明延时1 ns

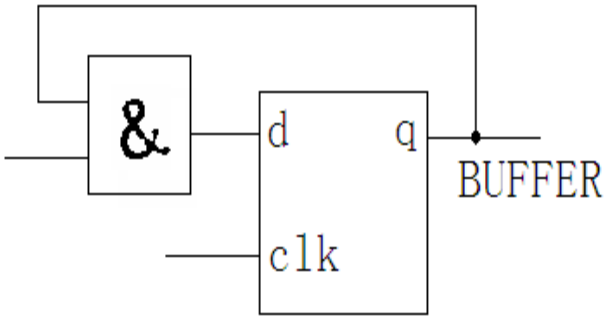
➢ 端口方向：IN， OUT，INOUT，BUFFER

说 明 符	含 义
IN（输入）	信号进入实体内部，内部的信号不能从该端口输出。
OUT（输出）	信号从实体内部输出，不能通过该端口在实体内部反馈使用。
INOUT（双向）	信号不但可以输入到实体内部，还可以从实体内部输出，也允许用于内部反馈。
BUFFER（缓冲）	信号输出到实体外部，同时也可通过该端口在实体内部反馈使用。



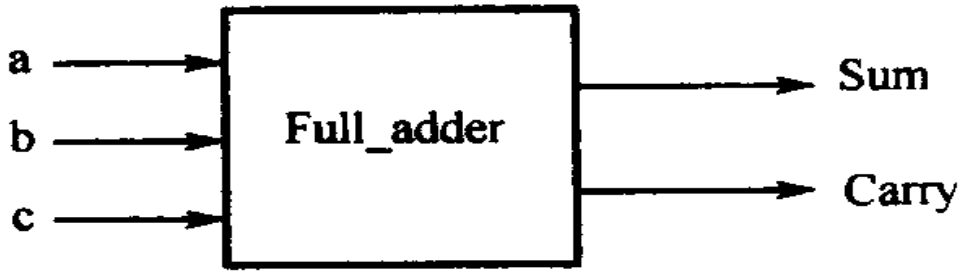
“OUT”和“BUFFER”都可定义输出端口；





若实体内部需要反馈输出信号，则输出端口必须被设置为“**BUFFER**”，而不能为“**OUT**”。

- 同方向、同类型的端口可放在同一个说明语句中。




```
ENTITY  Full_adder  IS
        PORT( a, b, c: IN  BIT;
              sum, carry: OUT  BIT
              );
END  Full_adder;
```

3.2.2 结构体 (Architecture)

结构体定义了设计单元具体的功能，描述了该基本设计单元的行为、元件和内部的连接关系。

ARCHITECTURE 结构体名 **OF** 实体名 **IS**

[声明语句]

BEGIN

功能描述语句

END [结构体名];

用于声明该结构体将用到的信号、数据类型、常数、子程序和元件等。声明的内容是局部的。

具体描述结构体的功能和行为。

- 一个实体可对应多个结构体，每个结构体代表该实体功能的不同实现方案或不同实现方式。同一时刻只有一个结构体起作用，通过CONFIGURATION决定用哪个结构体进行仿真或综合。
- 在结构体描述中，具体给出了输入、输出信号之间的逻辑关系。

```
ARCHITECTURE  Behavior  OF  FreDevider IS      -- 结构体定义
SIGNAL Clk:Std_Logic;                          --信号声明
BEGIN
   PROCESS(Clock)
BEGIN
   IF rising_edge(Clock) THEN
      Clk<=NOT Clk;
   END IF;
END PROCESS;
Clkout<=Clk;
END;
```

顺序语句

进程

功能描述语句

3.2.3 库、程序包的调用

LIBRARY 库名;

USE 库名.程序包名.项目名;

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

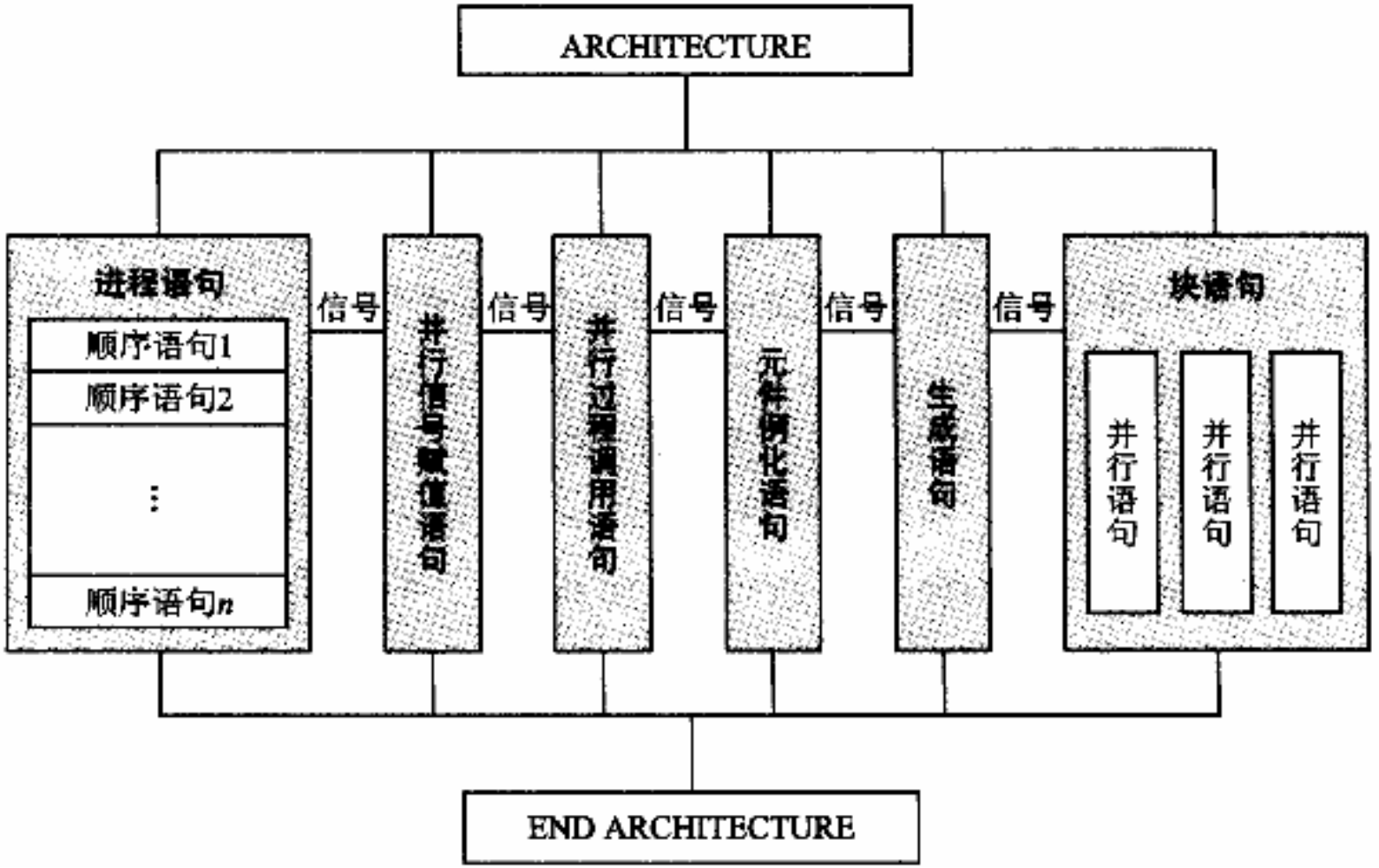
调用此程序包中所有的资源

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE IEEE.Std_Logic_Arith.ALL;
USE IEEE.Std_Logic_Unsigned.ALL;

3.3 VHDL语句

3.3.1 并行语句

在结构体中的执行是同时进行，执行顺序与书写顺序无关。



➢ 并行信号赋值语句

● 简单赋值语句

目标信号的数据类型与右边表达式一致

目标信号名 <= 表达式

```
ARCHITECTURE Behavior OF FreDevider IS
SIGNAL Clk:Std_Logic;
BEGIN
   PROCESS(Clock)
BEGIN
   IF rising_edge(Clock) THEN
      Clk<=NOT Clk;
   END IF;
END PROCESS;
Clkout<=Clk;
```

选择信号赋值语句

WITH 选择表达式 SELECT

赋值目标信号 <= 表达式1 WHEN 选择值1，

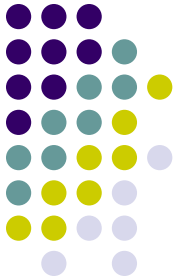
表达式2 WHEN 选择值1，

⋮

表达式n WHEN OTHERS；

选择值要覆盖所有可能情况，若不能一一指定，用OTHERS为其他情况找个出口；

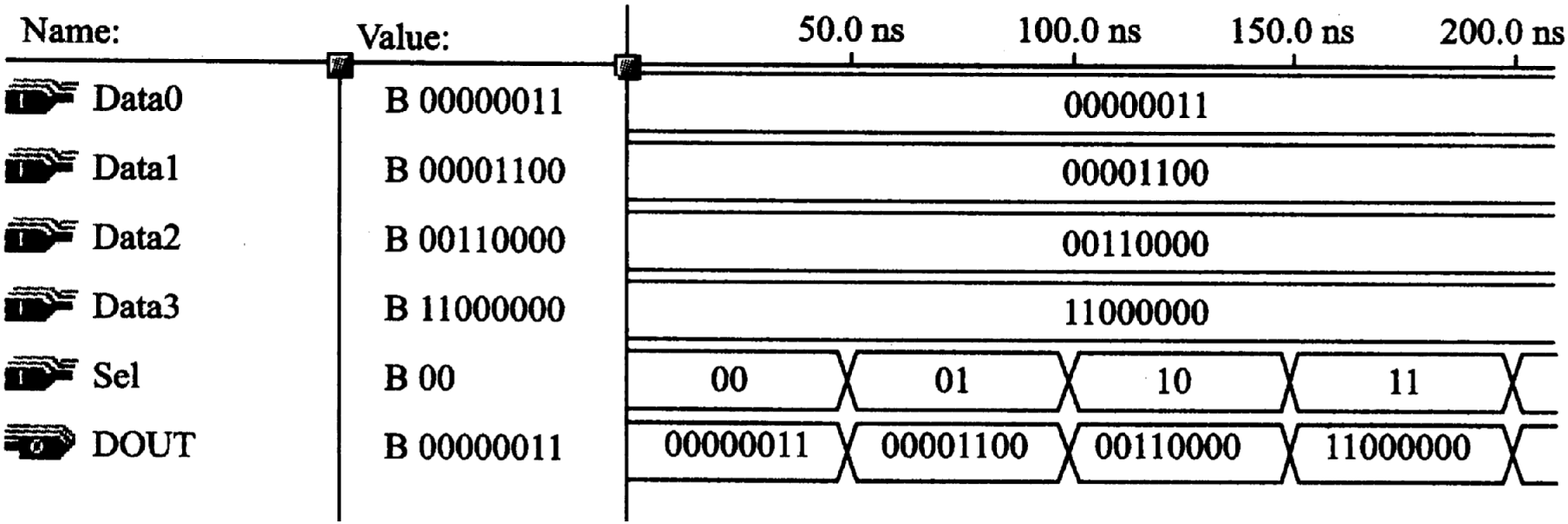
选择值必须互斥，不能出现条件重复或重叠的情况。



```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
ENTITY MUX IS
PORT
( Data0，Data1，Data2，Data3：IN Std_Logic_VECTOR(7 DOWNTO 0);
  Sel：IN Std_Logic_Vector(1 DOWNTO 0);
  DOUT：OUT Std_Logic_Vector(7 DOWNTO 0)
);
END;
ARCHITECTURE DataFlow OF MUX IS
BEGIN
    WITH Sel SELECT
        DOUT<=  Data0  WHEN “00”,
                Data1  WHEN “01”,
                Data2  WHEN “10”,
                Data3  WHEN “11”,
 “00000000” WHEN OTHERS;
END;
```

4X1多路选择器

地址选线Sel	输出DOUT
00	Data0
01	Data1
10	Data2
11	Data3


Name:
Value:
50.0 ns
100.0 ns
150.0 ns
200.0 ns
Data0
B 00000011
00000011
Data1
B 00001100
00001100
Data2
B 00110000
00110000
Data3
B 11000000
11000000
Sel
B 00
00
01
10
11
DOUT
B 00000011
00000011
00001100
00110000
11000000

● 条件信号赋值语句

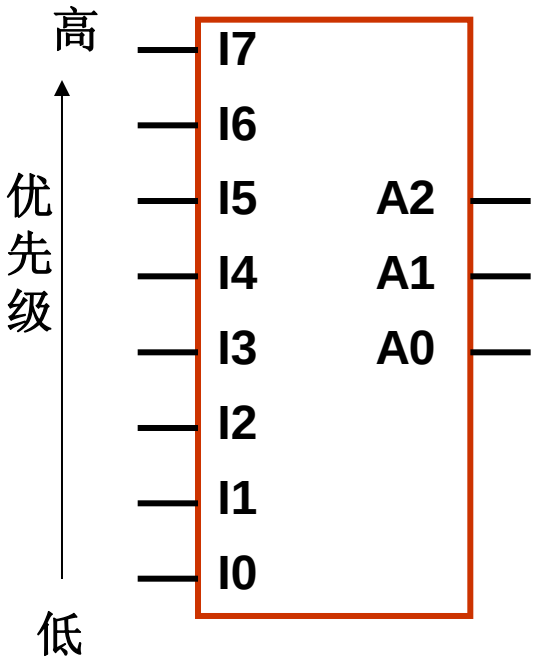
```
赋值目标信号 <= 表达式1  WHEN  赋值条件1  ELSE
              表达式2  WHEN  赋值条件2  ELSE
                    ⋮
              表达式n  WHEN  赋值条件n  ELSE
              表达式;
```

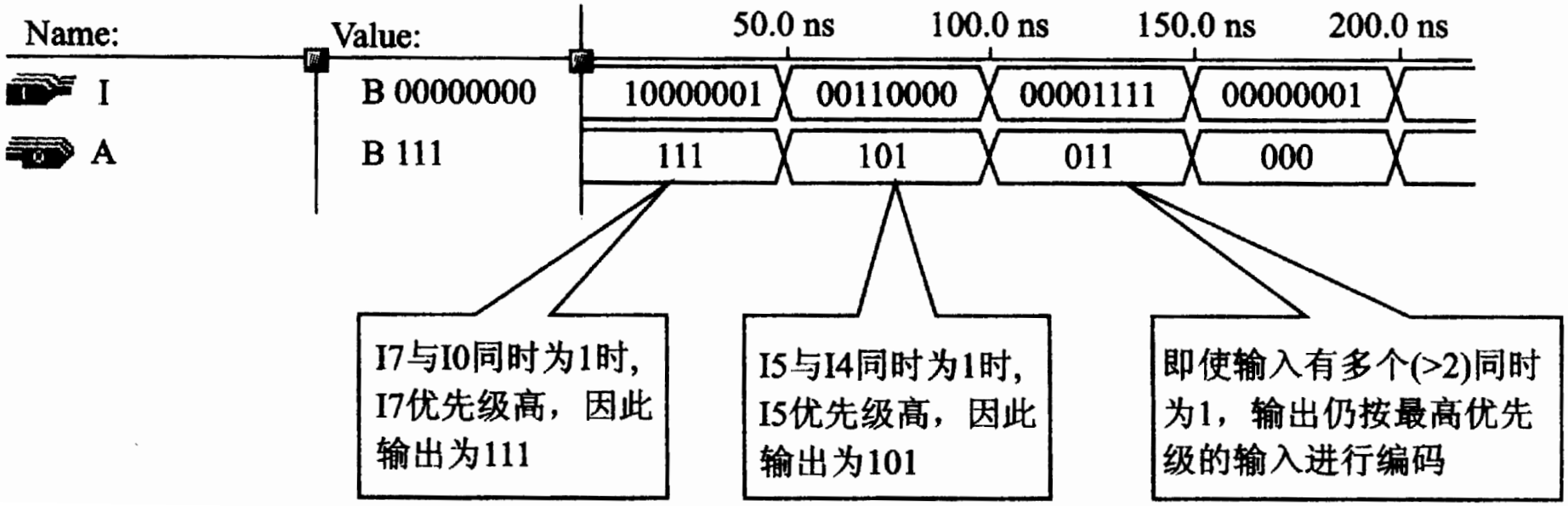
各赋值语句有优先级的差别，按书写顺序从高到低排列；

各赋值条件可以重叠。

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
ENTITY Priority_Encoder IS
PORT
( I:  IN Std_Logic_VECTOR(7 DOWNTO 0);
  A:  OUT Std_Logic_Vector(2 DOWNTO 0)
);
END;
ARCHITECTURE DataFlow OF Priority_Encoder IS
BEGIN
      A<="111"  WHEN I(7)='1'  ELSE
           "110"  WHEN I(6)='1'  ELSE
           "101"  WHEN I(5)='1'  ELSE
           "100"  WHEN I(4)='1'  ELSE
           "011"  WHEN I(3)='1'  ELSE
           "010"  WHEN I(2)='1'  ELSE
           "001"  WHEN I(1)='1'  ELSE
           "000"  WHEN I(0)='1'  ELSE
           "111";
END;
```

8输入优先编码器




Name:
Value:
50.0 ns
100.0 ns
150.0 ns
200.0 ns
I
B 00000000
10000001
00110000
00001111
00000001
A
B 111
111
101
011
000
I7与I0同时为1时，I7优先级高，因此输出为111
I5与I4同时为1时，I5优先级高，因此输出为101
即使输入有多个(>2)同时为1，输出仍按最高优先级的输入进行编码

➢ 进程语句

进程语句定义顺序语句模块，用于将从外部获得的信号值，或内部的运算数据向其他的信号进行赋值。

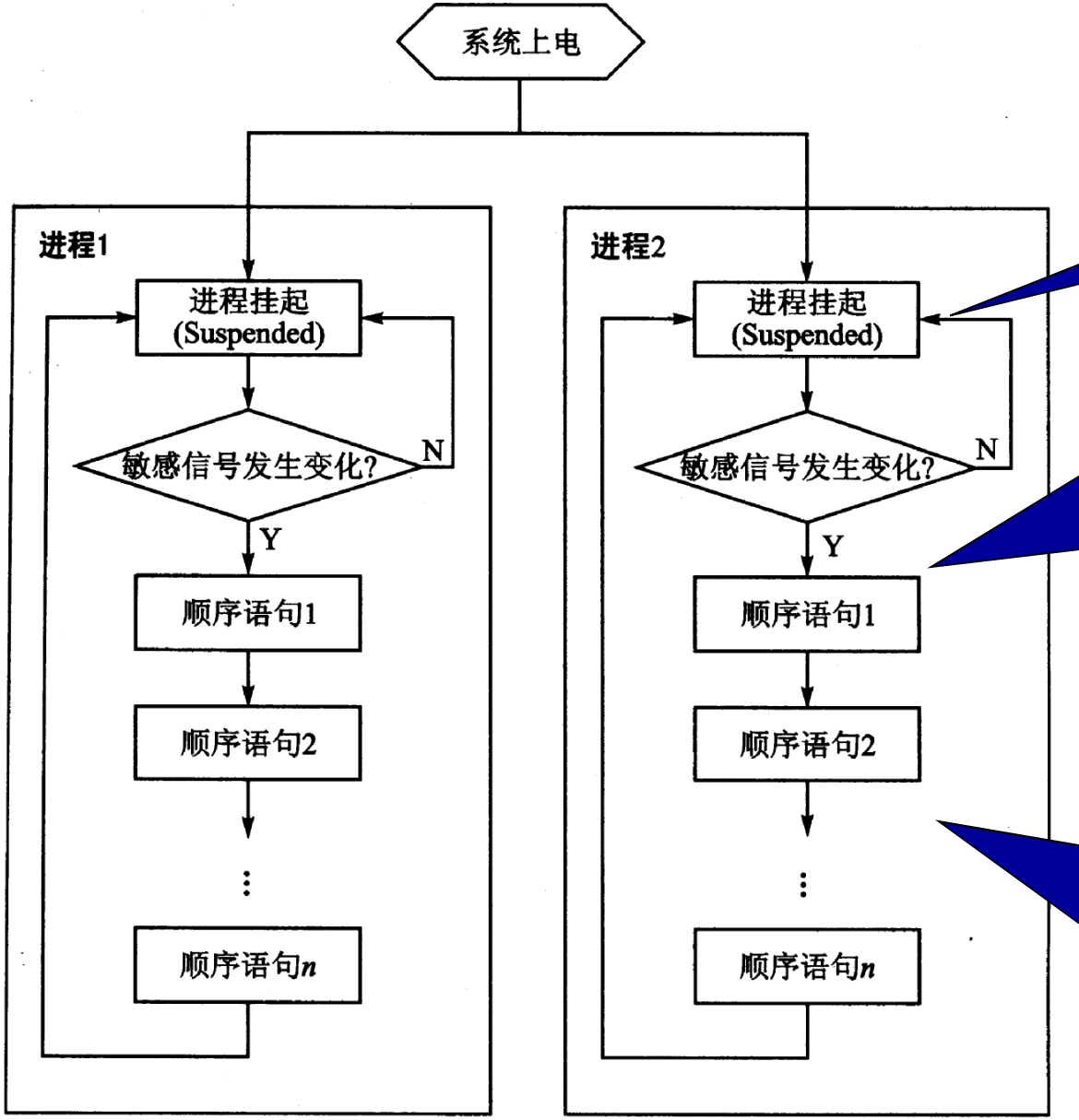
- 进程本身是并行语句，但内部是顺序语句；
- 进程只有在特定的时刻（敏感信号发生变化）才会被激活。

```
[进程标号：] PROCESS （敏感信号参数表）
[声明区]；
BEGIN
   顺序语句
END PROCESS [进程标号]；
```

一个进程可以有多个敏感信号，任一敏感信号发生变化都会激活进程

在进程中起作用的局部变量

● 进程的工作原理



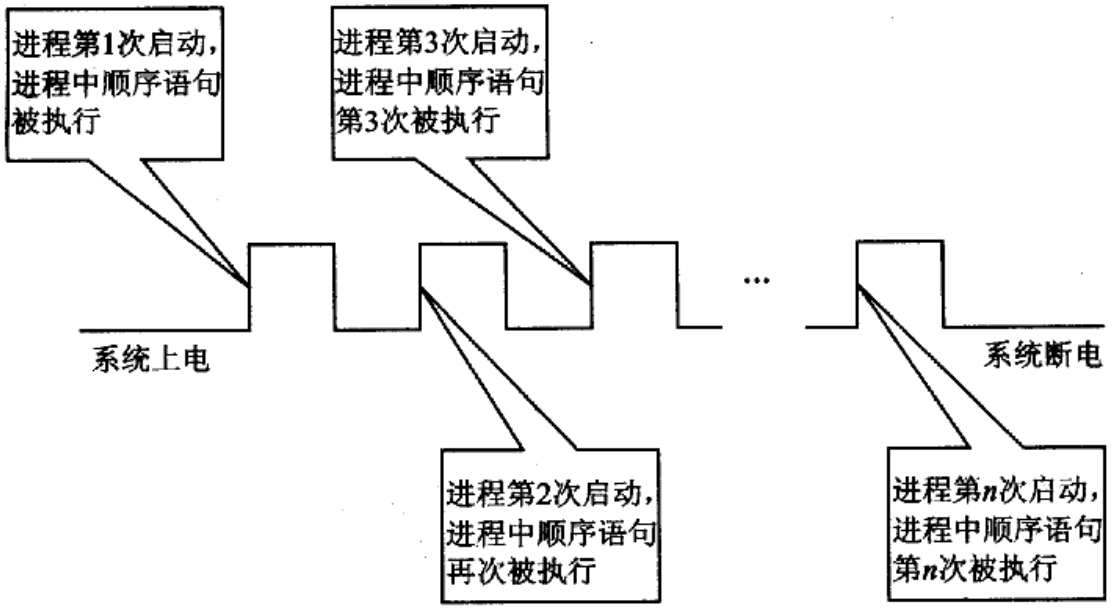
执行过程终止

当某个敏感信号的值发生变化时，每个进程语句立即完成进程内顺序语句所定义的功能行为。

顺序语句所定义的功能行为的结果可以赋值给信号，并通过信号被其他的进程读取或赋值。

● 进程与时钟

在每个上升沿启动一次进程（执行进程内所有的语句）。



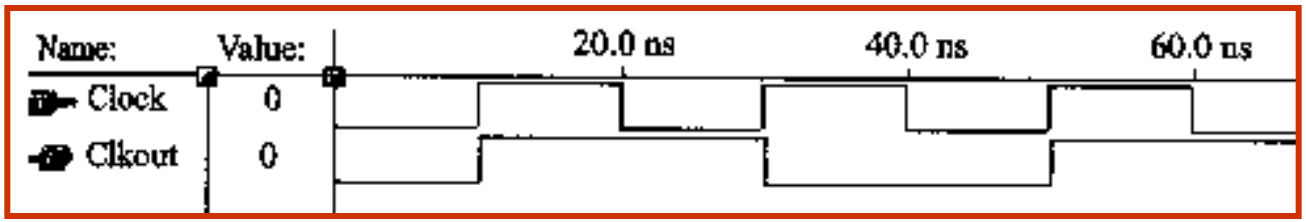
上升沿描述：**Clock' EVENT AND Clock='1'**

下降沿描述：**Clock' EVENT AND Clock='0'**

上升沿描述： **rising_edge (Clock)**

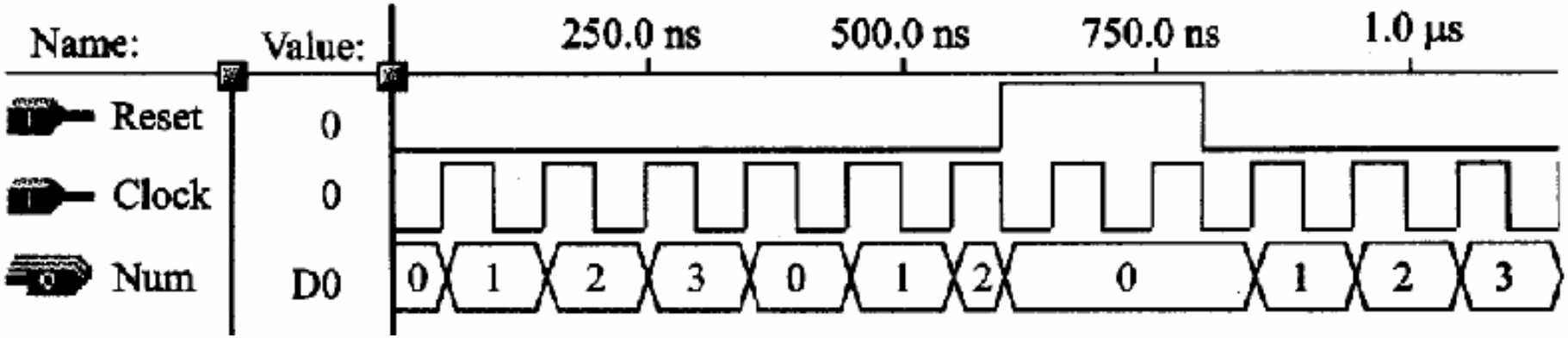
下降沿描述： **falling_edge (Clock)**

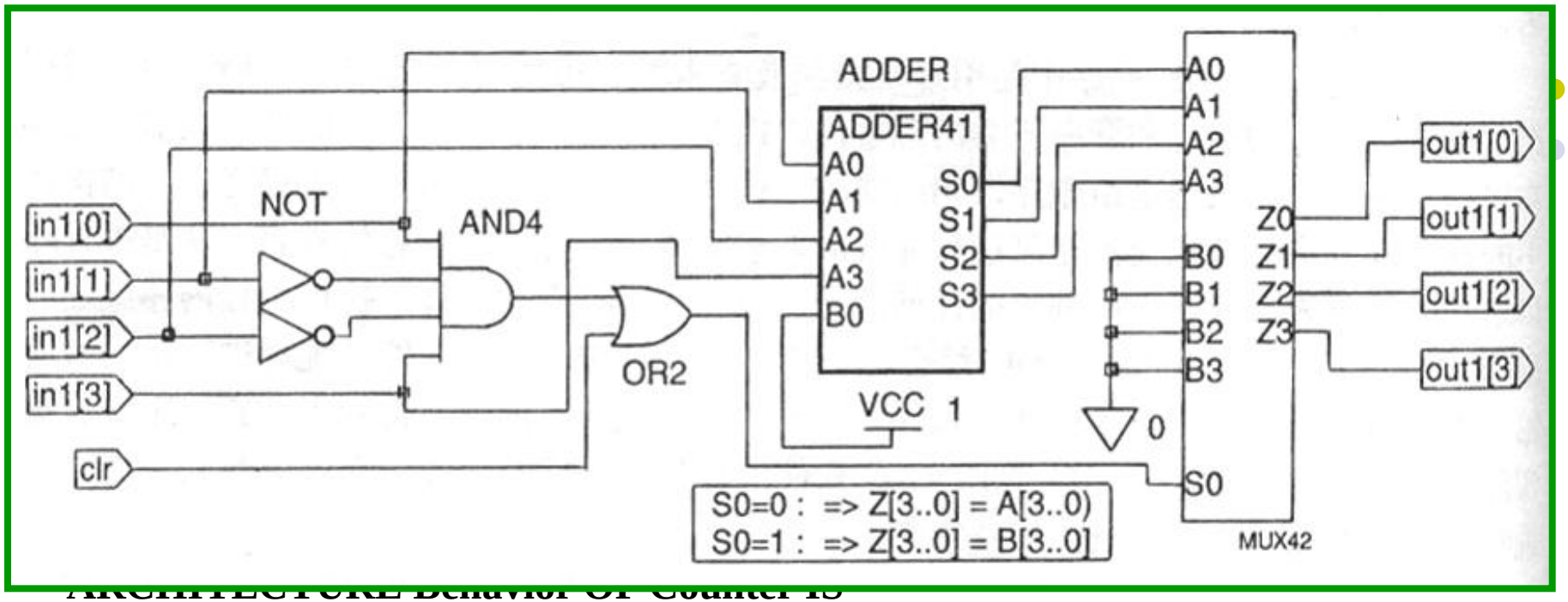
```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
ENTITY FreDevider IS
PORT
( Clock: IN Std_logic;
  Clkout: OUT Std_logic);
END;
ARCHITECTURE Behavior OF FreDevider IS
SIGNAL Clk: Std_Logic;
BEGIN
   PROCESS (Clock)                    ——将时钟作为进程的敏感信号
BEGIN
   IF rising_edge (Clock) THEN
      Clk<=NOT Clk;                   ——在时钟上升沿执行Clk<=NOT Clk
   END IF;
END PROCESS;
Clkout<=Clk;
END;
```



```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
ENTITY Counter IS
PORT
( RESET：  IN Std_Logic;                    ——异步复位信号
  Clock: IN Std_logic;                      ——时钟信号
  NUM: BUFFER Integer RANGE 0 TO 3);        ——计数器输出值
END;
ARCHITECTURE Behavior OF Counter IS
BEGIN
   PROCESS (RESET, Clock)                   ——将复位、时钟作为进程的敏感信号
BEGIN
  IF RESET='1' THEN
       Num<=0;                              ——复位时Num清0
    ELSIF rising_edge (Clock) THEN
       IF  Num=3 THEN
           Num<=0;                          ——如果Num＝3就清0
         ELSE
          Num<=Num+1;                       —— 否则自加1
       END IF;
   END IF;
END PROCESS;
END;
```

仿真波形：


Name:
Value:
250.0 ns
500.0 ns
750.0 ns
1.0 µs
Reset
0
Clock
0
Num
D0
0
1
2
3
0
1
2
0
1
2
3



ARCHITECTURE Behavior OF Counter4 IS

BEGIN

PROCESS (clr, inl) ——进程的敏感信号

BEGIN

IF (clr='1' or inl="1001") THEN

outl<="0000";

ELSE

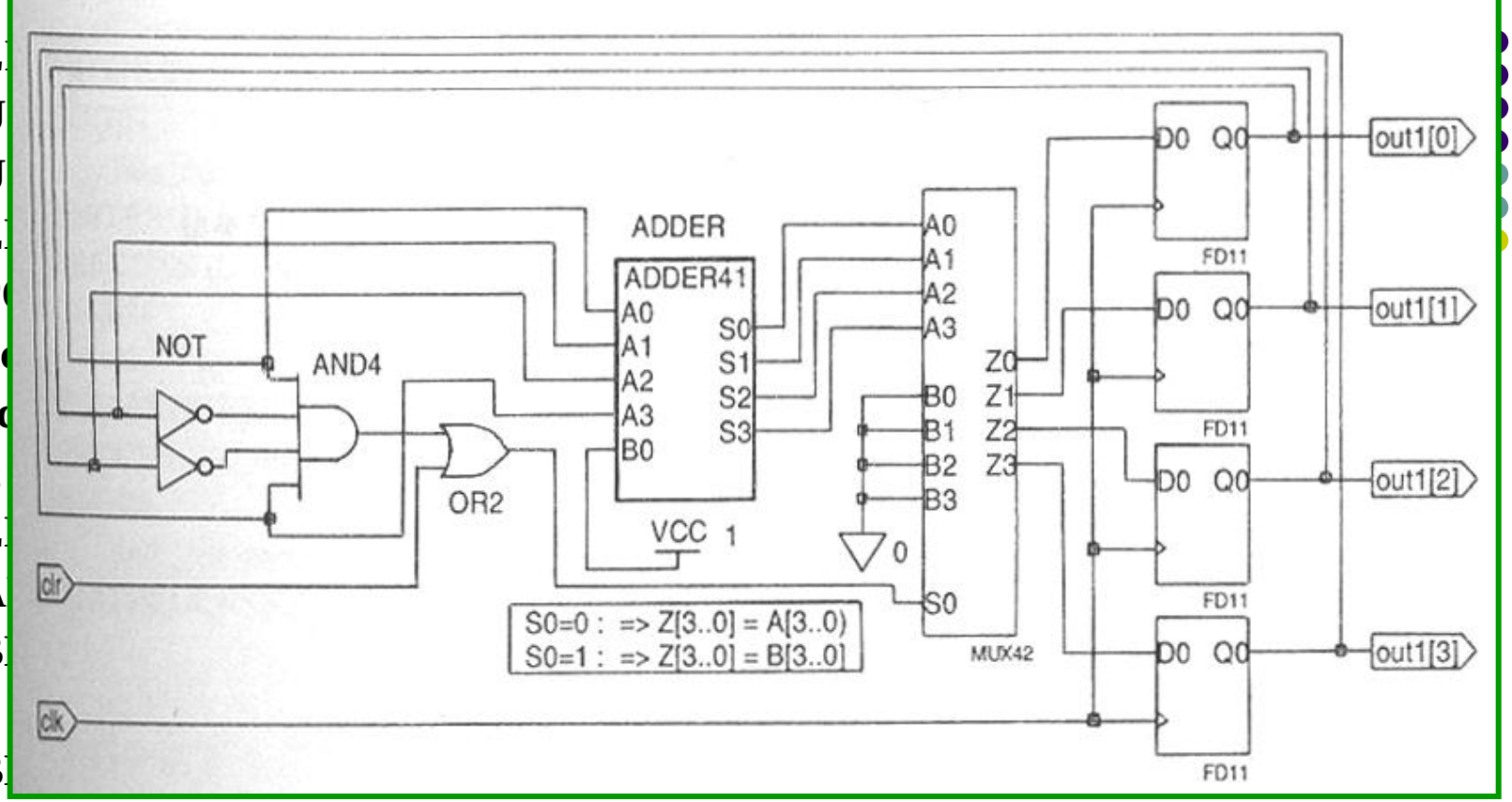
outl<=inl+1;

END IF;

END PROCESS;

END;

重载符号，在库IEEE.Std_Logic_unsigned中预先声明




```
Wait until clk'event and clk='1';
IF (clr='1' or cnt=9) THEN
      cnt<="0000";
 ELSE
       cnt<=cnt+1;
  END IF;
END PROCESS;
END;
```

Wait语句，信号赋值有寄存功能，引入时序器件

● 进程的启动

当process的敏感信号参数表重没有列出任何敏感信号时，进程通过wait语句启动。

```
ARCHITECTURE Behavior OF state IS
BEGIN
   PROCESS                                    ——敏感信号列表为空
BEGIN
    wait until Clock;                         ——等待clock激活进程
       IF  ( drive='1') THEN
         CASE output IS
         WHEN s1 => output <=  s2;
         WHEN s2 => output <= s3;
         WHEN s3 => output <= s4;
         WHEN s4 => output <= s1;
        END CASE;
       END IF;
END PROCESS;
END;
```

进程注意事项：

- 进程本身是并行语句，但内部为顺序语句；
- 进程在敏感信号发生变化时被激活，在使用了敏感表的进程中不能含wait语句；
- 在同一进程中对同一信号多次赋值，只有最后一次生效；
- 在不同进程中，不可对同一信号进行赋值。
- 一个进程不可同时对时钟上、下沿敏感。
- 进程中的信号赋值是在进程[illegible]起时生效的[illegible]性，是[illegible]直接影[illegible]有可读

```
SIGNAL A,B: Integer RANGE 0 TO 7;
        :
 PROCESS (Clock)
  BEGIN
       IF rising_edge (Clock) THEN
                  :
              B<=A+1;
              B<=B+1;
                  :
        END IF;
END PROCESS;
```

```
PROCESS (Clock)
  BEGIN
      IF rising_edge (Clock) THEN
                  :
           ELSIF falling_edge (Clock) THEN;
                  :
        END IF;
END PROCESS;
```

➢ 元件例化语句

较大的电路系统

元件例化引入一种连接关系，将预先设计好的实体定义为元件，并将此元件与当前设计实体中的端口相连接，从而为当前设计实体引入一个新的低一级的设计层次。

电路板上准备接受芯片的插座

要插在电路系统板上的芯片

Component 元件名

port (端口名表);

列出对外通信的各端口名

End component 元件名;

元件定义语句

例化名：元件名 **port map** ([元件端口名=>]连接端口名 , …);

元件例化语句

名字关联方式：port map语句中位置可以任意；

位置关联方式：端口名和关联连接符号可省去，连接端口名的排列方式与所需例化的元件端口定义中的端口名相对应。

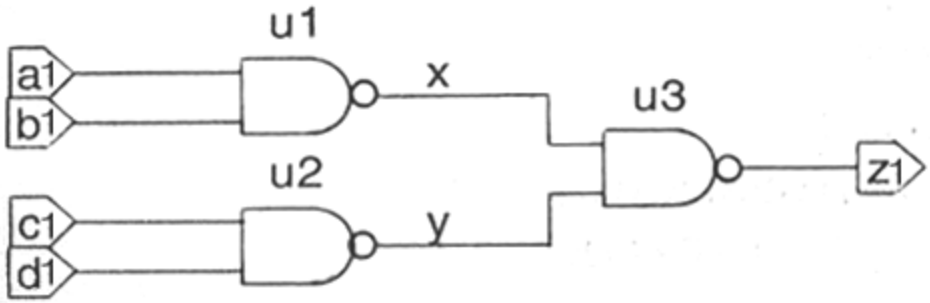
当前系统与准备接入的元件对应端口相连的通信端口。

元件例化：

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
ENTITY ord41 IS
PORT( a1, b1,c1, d1：  IN Std_Logic;
          z1: out std_logic);
END;
ARCHITECTURE ord41behv OF ord41 IS
BEGIN
COMPONENT nd2
PORT( a, b: IN Std_Logic;
          c: out std_logic);
End COMPONENT;
SIGNAL x, y: STD_LOGIC;
BEGIN
u1:nd2 PORT MAP( a1, b1, x);
u2:nd2 PORT MAP( a=>c1, c=>y, b=>d1);
u3:nd2 PORT MAP( x, y, c=>z1);
END;
```

设计2输入与非门：

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
ENTITY nd2 IS
PORT
( a, b：  IN Std_Logic;
  c: out std_logic);
END;
ARCHITECTURE nd2behv OF nd2 IS
BEGIN
   y<=a NAND b;
END;
```

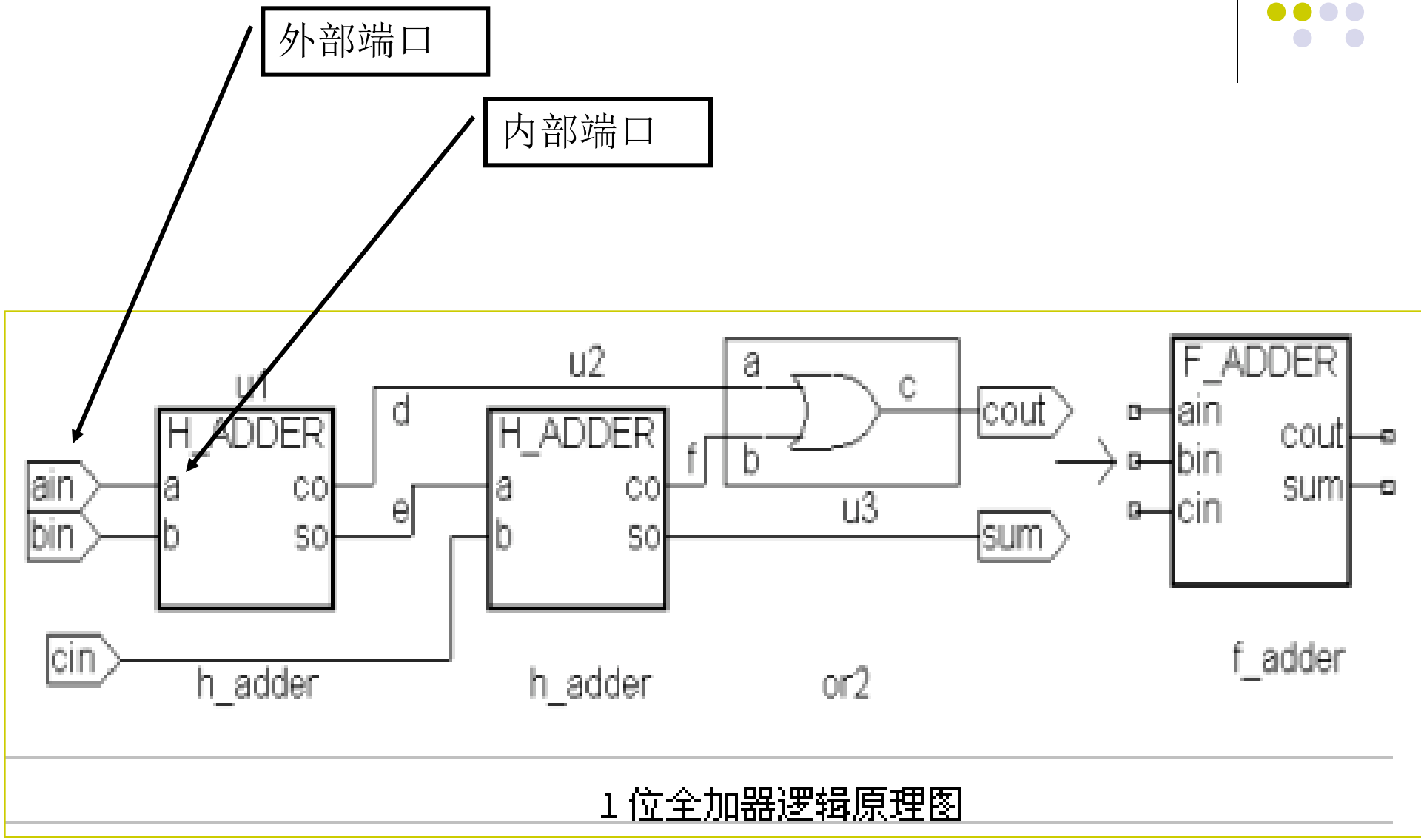


——位置关联方式

——名字关联方式

——混和关联方式

1位二进制全加器



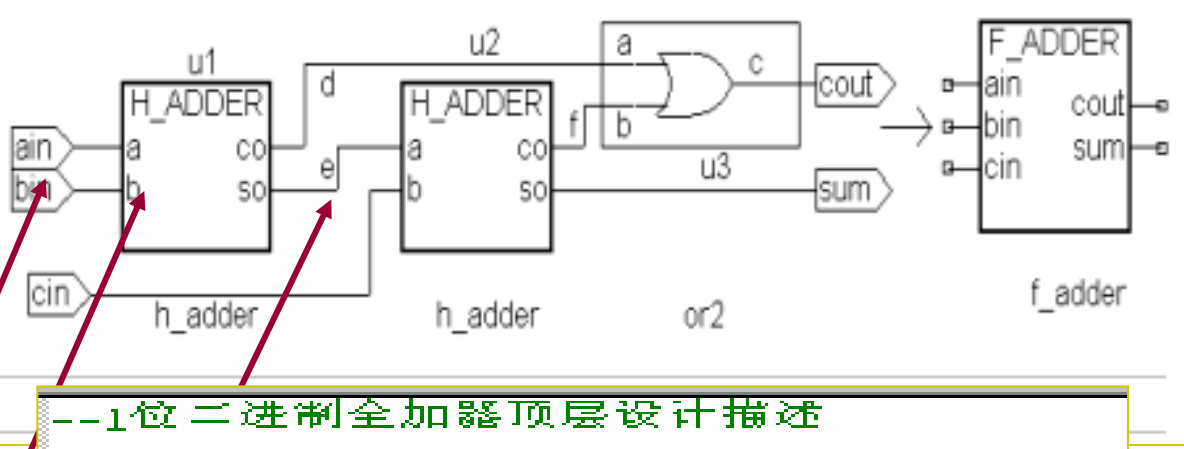
1 位全加器逻辑原理图

```vhdl
LIBRARY  IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY or2 IS
  PORT (a,b : IN STD_LOGIC;
        c : OUT STD_LOGIC );
END ENTITY or2;
ARCHITECTURE ful OF or2 IS
  BEGIN
  c <= a OR b ;
END ARCHITECTURE ful;
  --半加器描述
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY h_adder IS
  PORT (a, b : IN STD_LOGIC;
      co, so : OUT STD_LOGIC);
END ENTITY h_adder;
ARCHITECTURE fhl OF h_adder IS
  BEGIN
  so <= (a OR b)AND(a NAND b);
  co <= NOT( a NAND b);
END ARCHITECTURE fhl;
```

外部端口

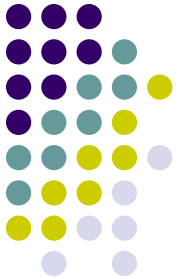
内部端口

端口连线




```vhdl
--1位二进制全加器顶层设计描述
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY f_adder IS
 PORT(ain, bin,  cin : IN STD_LOGIC;
      cout, sum    : OUT STD_LOGIC );
END ENTITY f_adder;
ARCHITECTURE fdl OF f_adder IS
 COMPONENT h_adder
 PORT(  a, b :  IN STD_LOGIC;
      co, so :  OUT STD_LOGIC);
END COMPONENT;
  COMPONENT or2
  PORT (a, b : IN STD_LOGIC;
            c : OUT STD_LOGIC);
END COMPONENT;
 SIGNAL d, e, f  :  STD_LOGIC;
 BEGIN
 u1 : h_adder PORT MAP( a =>ain,
 b =>bin, co=>d, so =>e);
 u2 : h_adder PORT MAP( a =>e, b =>cin,
                        co =>f, so =>sum);
 u3 :    or2     PORT MAP(a =>d, b =>f,
                        c =>cout);
END ARCHITECTURE fdl ;
```

3.3.2 顺序语句



顺序语句仅出现在进程和子程序中。

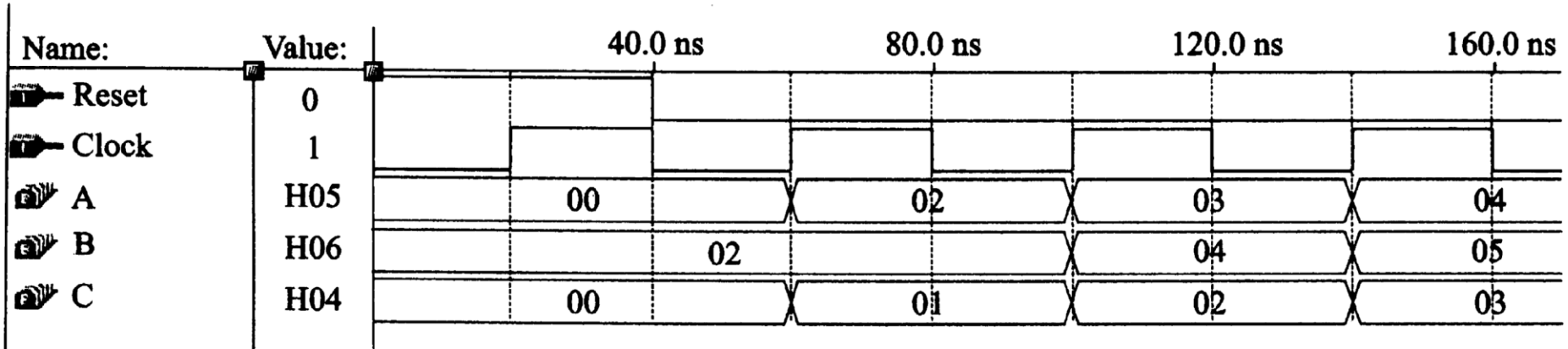
顺序语句综合后，映射为实际的门电路，系统一上电，门电路开始工作。电路可实现逻辑上的顺序执行，实际上所有门电路是并行工作的。

- 赋值语句
- 流程控制语句
- 空操作语句
- 等待语句
- 子程序调用语句
- 返回语句

➢ 赋值语句

```
ENTITY TEST_Signal IS
PORT
( Reset, Clock: IN Std_logic;
  NumA, NumB: OUT Integer RANGE 0 TO 255
);
END;
ARCHITECTURE TEST OF TEST_Signal IS
    SIGNAL A, B: Integer RANGE 0 TO 255;
 BEGIN
    PROCESS (RESET,Clock)
    VARIABLE C: Integer RANGE 0 TO 255;
```

```
BEGIN
      IF RESET='1' THEN
         A<=0; B<=2;C:=0;
      ELSEIF rising_edge(Clock) THEN
         C:=C+1;A<=C+1;B<=A+2;
      END IF;
   END PROCESS;
    Num A<=A; NumB<=B;
END;
```



➢ 流程控制语句

- **IF语句：**

```
IF 条件式 THEN
    顺序语句
END IF;
```

```
IF 条件式 THEN
    顺序语句
ELSE
    顺序语句
END IF;
```

有优先级

```
IF 条件式 THEN
    顺序语句
ELSEIF 条件式2
THEN
    顺序语句

 ELSE
    顺序语句
END IF;
```

用IF语句描述组合逻辑电路时，务必涵盖所有的情况，否则综合后将引入锁存器！

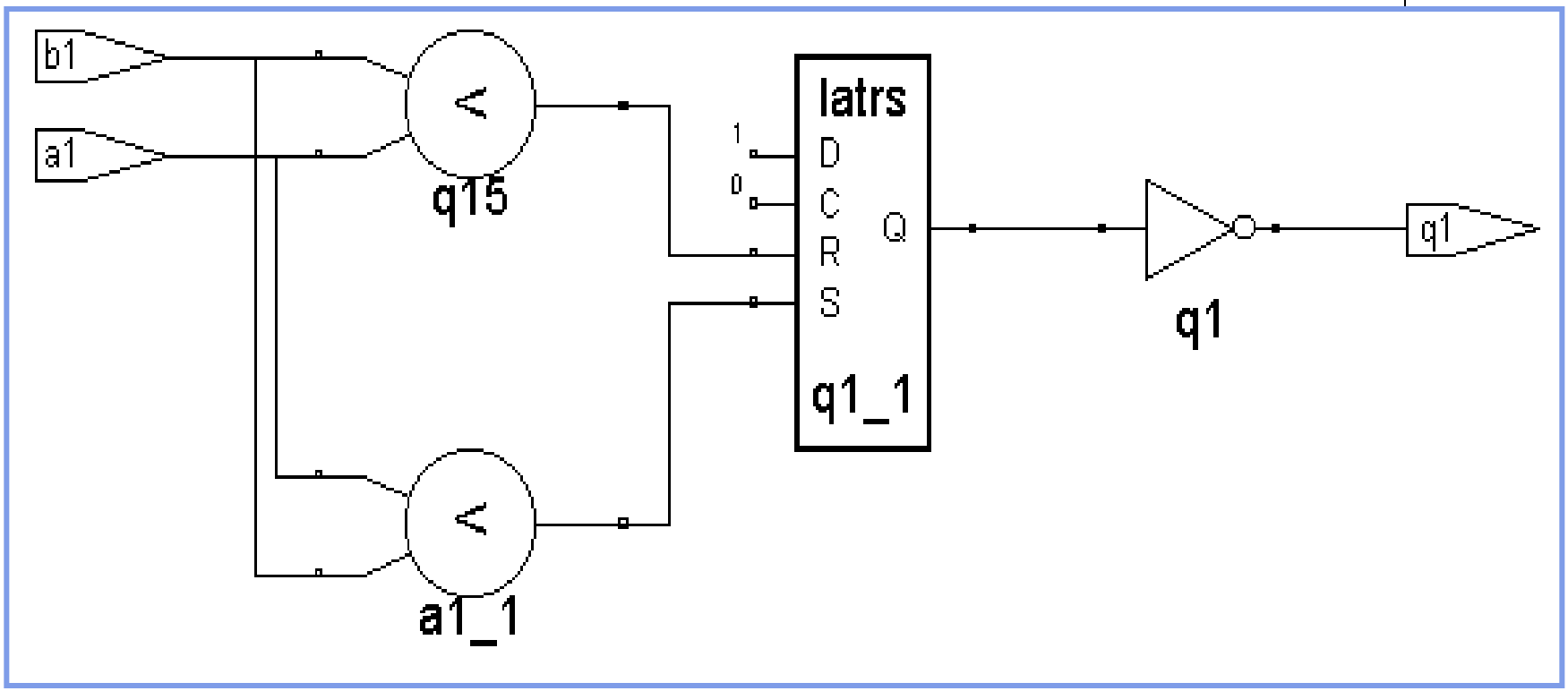
```
ENTITY Encoder IS
PORT
( En: IN Std_logic;
  I: IN Std_logic_Vetor(7 DOWNTO 0);
  A: OUT Std_logic_Vetor(2 DOWNTO 0);
  Idle: OUT Std_logic
);
END;
ARCHITECTURE Behavior OF Encoder IS
BEGIN
   PROCESS (En, I)
BEGIN
   IF En='1' THEN
      IF I(7)='1' THEN
      A<="111";   Idle<='0';
    ELSIF IF I(6)='1' THEN
       A<="110";   Idle<='0';
    ELSIF IF I(5)='1' THEN
       A<="101";   Idle<='0';
      ELSIF IF I(4)='1' THEN
       A<="100";   Idle<='0';
      ELSIF IF I(3)='1' THEN
       A<="011";   Idle<='0';
      ELSIF IF I(2)='1' THEN
       A<="010";   Idle<='0';
      ELSIF IF I(1)='1' THEN
       A<="001";   Idle<='0';
      ELSIF IF I(0)='1' THEN
       A<="000";   Idle<='0';
      ELSE
       A<="000";   Idle<='1';
      END IF;
   END PROCESS;
END;
```

引入ELSE，否则综合器将引入锁存器。

不完整条件语句与时序电路

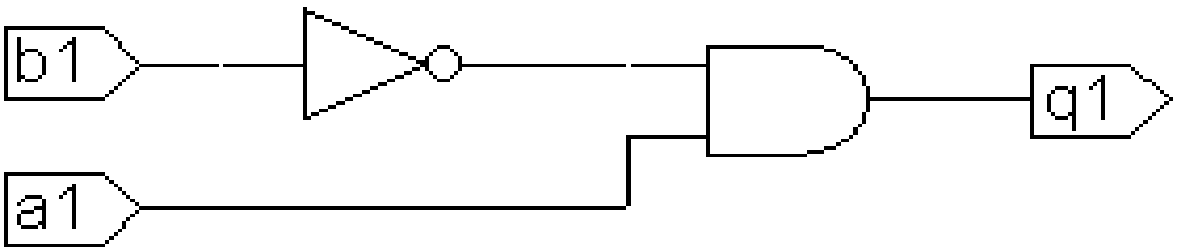
```
ENTITY COMP_BAD IS
  PORT( a1  : IN BIT;
     b1  : IN BIT;
     q1  : OUT BIT    );
END ;
ARCHITECTURE one OF COMP_BAD IS
  BEGIN
   PROCESS (a1,b1)
BEGIN
IF  a1 > b1   THEN  q1 <= '1' ;
ELSIF a1 < b1 THEN  q1 <= '0' ;   -- 未提及当a1=b1时，q1作何操作
END IF;
   END PROCESS ;
END ;
```

综合结果：



完整条件语句

```
ENTITY COMP_GOOD IS
 PORT(a1  :  IN BIT;
      b1  :  IN BIT;
      q1  : OUT BIT  );
 END ;
 ARCHITECTURE one OF COMP_GOOD IS
  BEGIN
   PROCESS (a1,b1)
    BEGIN
     IF  a1 > b1  THEN  q1 <= '1' ;
     ELSE   q1 <= '0' ;
     END IF;
   END PROCESS ;
 END ;
```



➢ CASE语句

```
CASE 表达式 IS
   WHEN 选择值[|选择值 ]=>顺序语句;
   WHEN 选择值[|选择值 ]=>顺序语句;
        ┆
   WHEN OTHERS=>顺序语句;
END CASE;
```

- 选择值不可重复或重叠;
- 当CASE语句的选择值无法覆盖所有的情况时,要用OTHERS指定未能列出的其他所有情况的输出值;

```
LIBRARY IEEE；
USE IEEE.Std_Logic_1164.ALL；
ENTITY MUX IS
PORT
( Data0，Data1，Data2，Data3：IN Std_Logic_VECTOR(7 DOWNTO 0)；
  Sel：IN Std_Logic_Vector(1 DOWNTO 0)；
  DOUT：OUT Std_Logic_Vector(7 DOWNTO 0)
)；
END；

 ARCHITECTURE DataFlow OF MUX IS
    BEGIN
      CASE Sel IS
         WHEN “00”=>  DOUT<=  Data0;
         WHEN “01”=>  DOUT<=  Data1;
         WHEN “10”=>  DOUT<=  Data2;
         WHEN “11”=>  DOUT<=  Data3;
         WHEN OTHERS =>  DOUT<=“00000000” ；
      END CASE；
   END PROCESS;
END;
```

LOOP语句

[LOOP 标号:] FOR 循环变量 IN 循环次数范围 LOOP

顺序语句

P [LOOP 标号];

临时变量，仅在此LOOP中有效，无需事先定义。

….TO….
….DOWNTO…..
从初值开始，每执行完一次后递增(递减)，直到终值为止。

```
Sum:=0;
FOR i  IN 0 TO 9 LOOP
    Sum:=Sum+i;
END LOOP;
```

循环次数只能用具体数值表达，否则不可综合

```
VARIABLE Length：Integer RANGE 0 TO 15;
        ⋮
FOR  i IN 0 TO Length  LOOP
        ⋮
END LOOP;
```

NEXT语句

NEXT语句主要用在LOOP语句执行中有条件或无条件转向控制，跳向LOOP语句的起点。

NEXT [循环标号] [WHEN 条件];

① NEXT ；——无条件终止当前循环，跳回到本次循环LOOP语句处，开始下一次循环。

② NEXT LOOP 标号；——当有多重LOOP语句嵌套时，跳转到指定标号LOOP语句处，重新开始执行循环操作。

③ NEXT LOOP 标号 WHEN 条件表达式；——条件表达式为TRUE，执行NEXT语句，进入跳转操作，否则向下执行。

➢EXIT语句

EXIT语句主要用在LOOP语句执行中有条件或无条件内部转向控制，跳向LOOP语句的终点，用于退出循环。当程序需要处理保护、出错和警告状态时，语句能提供一个快捷、简便的方法。

EXIT [循环标号] [WHEN 条件]；

① EXIT ；——无条件从当前循环中退出。

② EXIT LOOP 标号；——程序执行退出动作无条件从循环标号所标明的循环中退出。

③ EXIT LOOP 标号 WHEN 条件表达式；——条件表达式为TRUE，程序从当前循环中退出。

NULL语句

NULL为空操作语句，一般用于CASE中，表示在某些情况下对输出不作任何改变，隐含锁存信号。不能用于纯组合逻辑电路。

```
PROCESS (Clock)
BEGIN
  IF rising_edge (Clock)  THEN
        ¦
    CASE Sel IS
      WHEN…
      WHEN OTHERS=>NULL;
    END CASE;
        ¦
    END IF;
END PROCESS;
```

➢ WAIT语句

在进程或过程中执行到WAIT语句时，程序将被挂起，并设置好再次执行的条件。

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

① **WAIT ；**——未设置停止挂起的条件，表示永远挂起。

② **WAIT ON 信号表；**——敏感信号等待语句，敏感信号的变化将结束挂起，再次启动进程。

③ **WAIT UNTIL 条件表达式；**——条件表达式为中所含的信号发生变化，且满足WAIT语句所设条件，则结束挂起，再次启动进程。

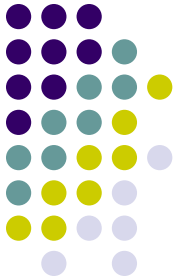
④ **WAIT FOR 时间表达式；**——超时等待语句，从执行当前的WAIT语句开始，在此时间段内，进程处于挂起状态，超过这一时间段后，程序自动恢复执行。

3.3.3 配置语句

配置主要为顶层设计实体指定结构体，或为参与例化的元件实体指定所希望的结构体，以层次方式来对元件例化做结构配置。

```
Configuration 配置名 of 实体名 is
    配置说明
End 配置名;
```

➢ 为顶层设计实体指定结构体

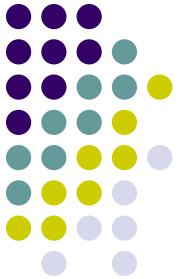


```
Entity  nand is
    port ( a, b: in std_logic;
           c: out std_logic );
End entity nand;
Architecture one of nand is
  begin
   c<=not (a and b);
End architecture one;
Architecture two of nand is
  begin
   c<='1' when (a='0') and (b='0') else
      '1' when (a='0') and (b='1') else
      '1' when (a='1') and (b='0') else
      '0' when (a='1') and (b='1') else
      '0';
End architecture two;
```

```
Configuration 配置名 of 实体名 is
   配置说明
End 配置名;
```

```
Configuration second of nand is
 for two
 end for;
End second;

Configuration first of nand is
   for one
   end for;
End first;
```

➢ 为参与例化的元件实体指定所希望的结构体，以层次方式来对元件例化做结构配置。



```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
Entity  rs is
    port ( r, s: in std_logic;
           q,qf: out std_logic );
End entity rs;
Architecture rsf of rs is
Component nand
PORT( a, b: IN Std_Logic;
          c: out std_logic);
End Component;
BEGIN
u1:nand PORT MAP (a=>s, b=>qf, c=>q);
u2:nand PORT MAP ( a=>q, b=>r, c=>qf);
END rsf;
```

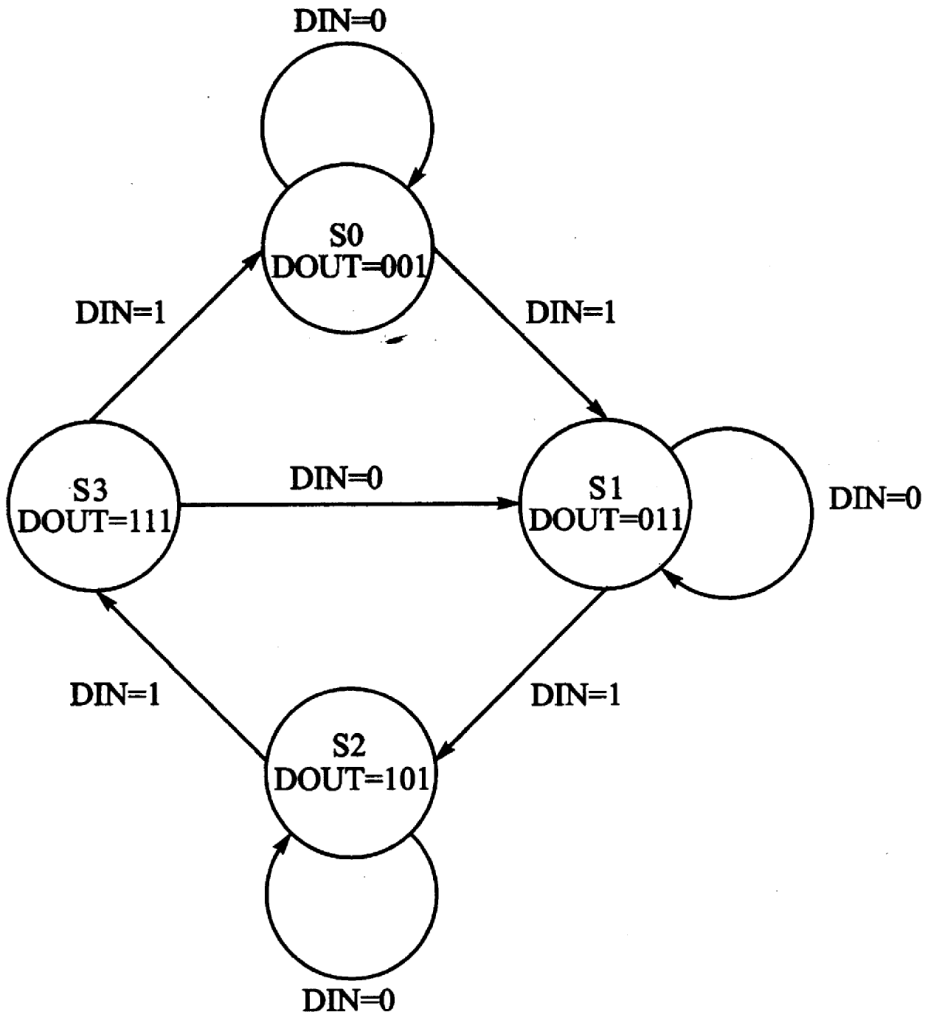
```
Configuration sel of rs is
 for rsf
   for u1,u2:nand
   use entity work.nand (two);
end for;
end for;
End sel;
```

用实体nand构成更高层次设计实体中的元件，由配置语句指定元件实体nand使用哪个结构体。

3.4 状态机在VHDL中的实现

3.4.1 Moore 状态机的VHDL描述

输出仅取决于其所处的状态。

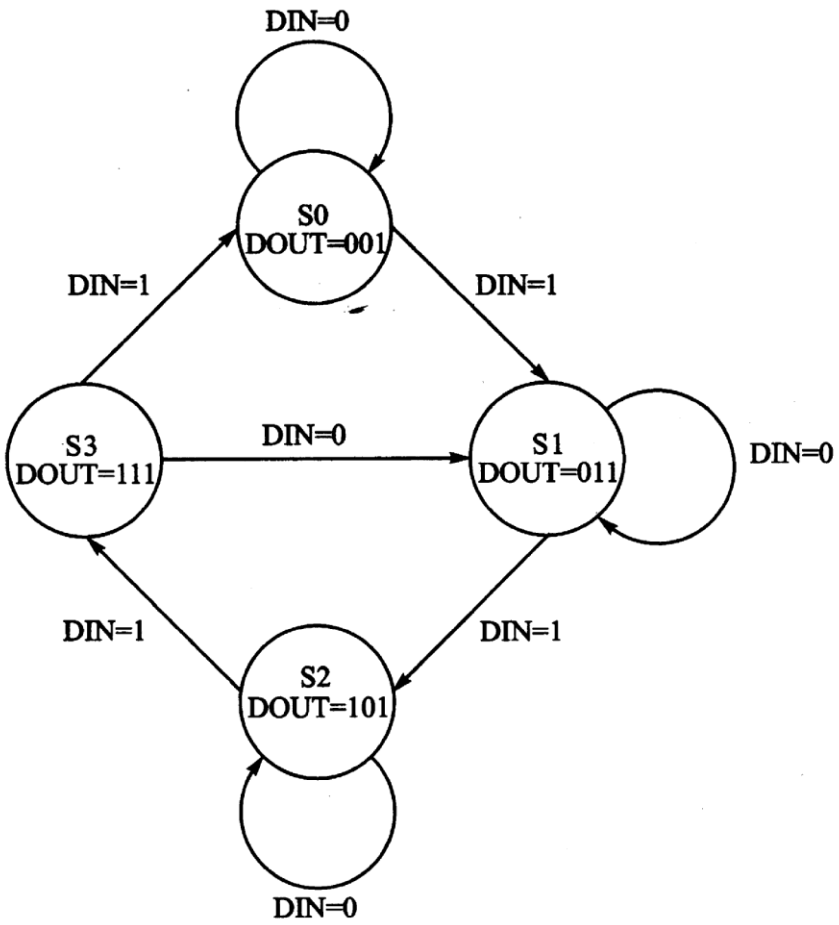


```
LIBRARY IEEE;
USE IEEE.Std_ Logic_1164.ALL;
ENTITY Moore IS
PORT
（Reset, Clock, DIN ： IN Std_Logic;
  DOUT：OUT Std_Logic_Vetor(2 DOWNTO 0));
END;
ARCHITECTURE Mooremachine OF Moore IS
TYPE State_type IS (S0，S1，S2，S3);        ——定义State_type为枚举型数据类型
SIGNAL State：State_type;
BEGIN
Change_State：PROCESS (Reset, Clock)        ——时序逻辑进程
BEGIN
IF Reset='1' THEN
State<=S0;
```

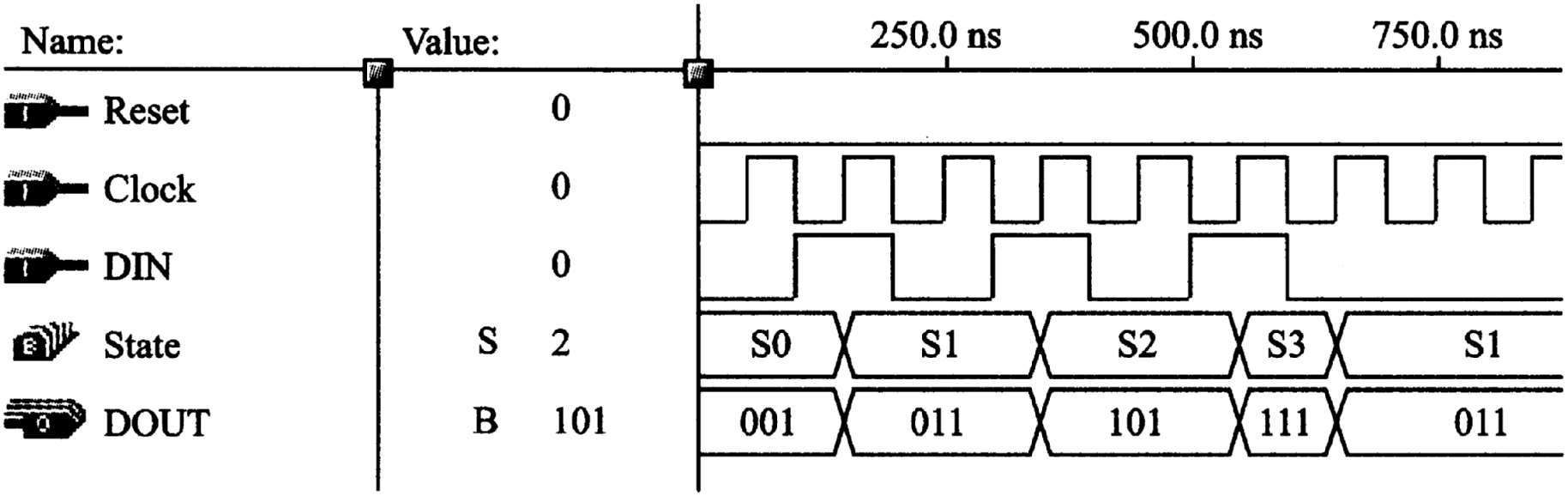
```
ELSEIF rising_edge(Clock) THEN
      CASE  State IS
          WHEN S0=〉 IF DIN=‘1’ THEN State<=S1；END IF；
          WHEN S1=〉 IF DIN=‘1’ THEN State<=S2；END IF；
          WHEN S2=〉 IF DIN=‘1’ THEN State<=S3；END IF；
          WHEN S3=〉 IF DIN=‘1’ THEN State<=S0；
                          ELSE  State<=S1；
                         END IF ；
               END CASE；
        END IF；
END PROCESS；
```

——定义输出进程，组合逻辑

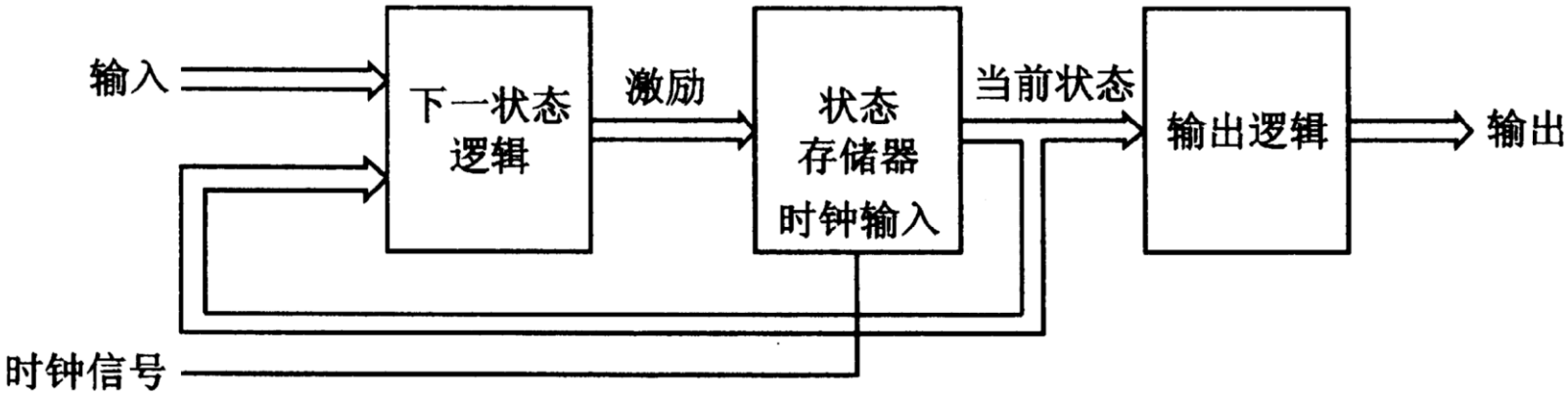
```
Output_Process：  PROCESS（State）
BEGIN
  CASE State IS
      WHEN S0=>DOUT <=“001”；
      WHEN S1=>DOUT <=“011”；
      WHEN S2=>DOUT <=“101”；
      WHEN S3=>DOUT <=“111”；
   END CASE；
  END PROCESS；
END；
```



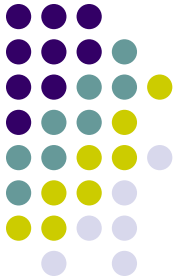
仿真波形图：


Name:
Value:
250.0 ns
500.0 ns
750.0 ns
Reset
0
Clock
0
DIN
0
State
S
2
S0
S1
S2
S3
S1
DOUT
B
101
001
011
101
111
011

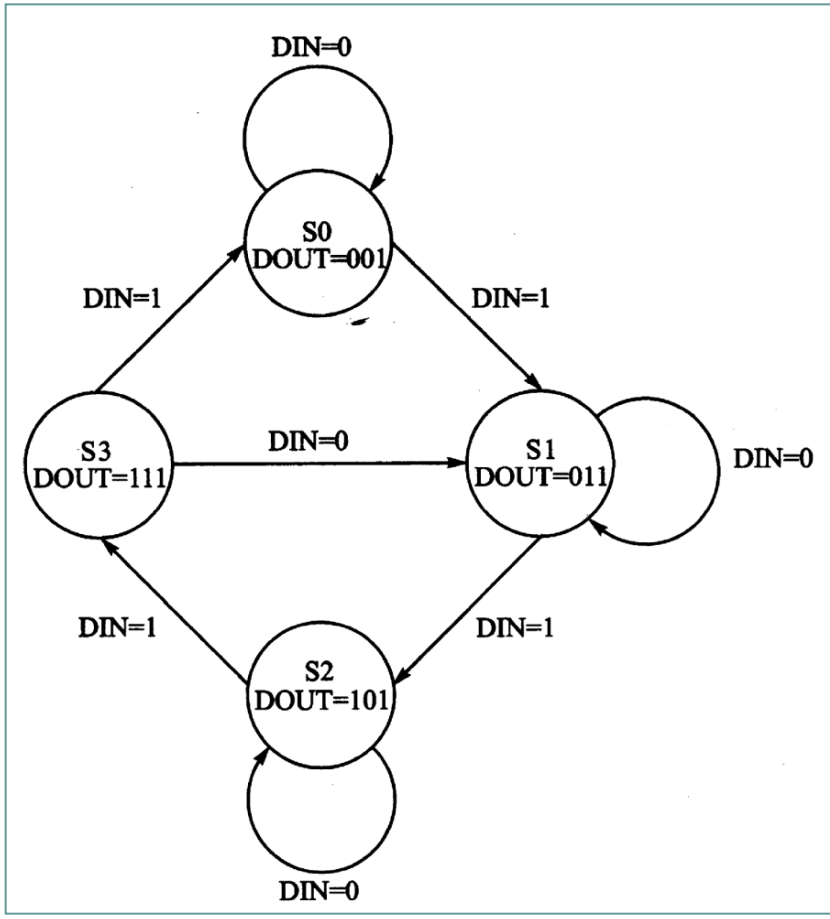
时钟同步Moore状态机结构图：



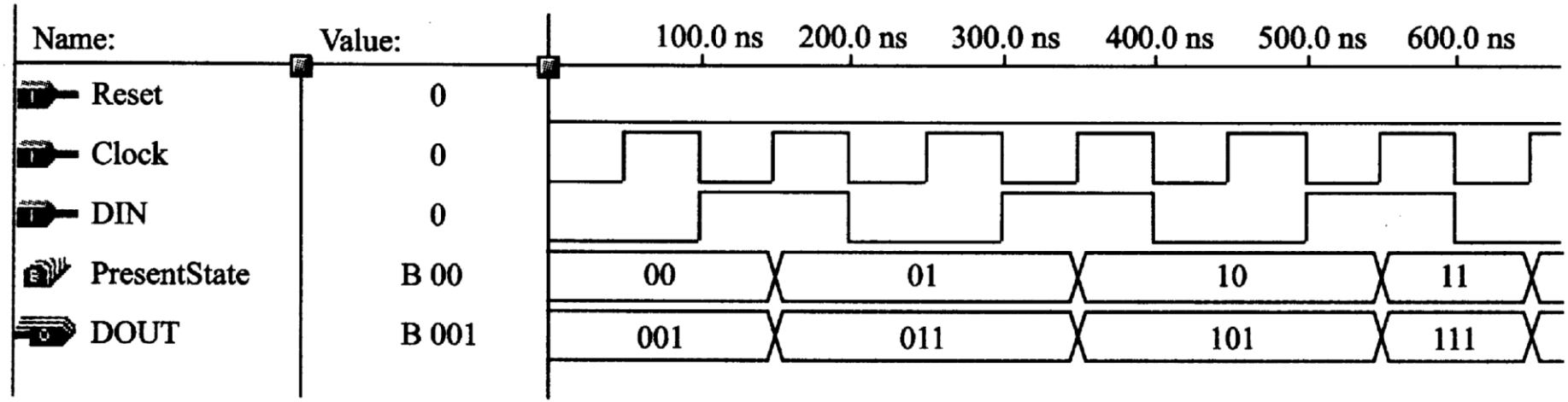
```
LIBRARY IEEE;
USE IEEE.Std_ Logic_1164.ALL;
ENTITY Moore IS
PORT
（ Reset,Clock, DIN ： in Std_Logic;
  DOUT：out Std_Logic_Vetor(2 DOWNTO 0) );
END;
ARCHITECTURE Mooremachine OF Moore IS
   TYPE State_type IS (S0，S1，S2，S3);
   SIGNAL PresentState, NextState ： State_type;  ——定义状态转换信号
  BEGIN
State_Reg: PROCESS (Reset, Clock)          ——状态寄存器
    BEGIN
      IF Reset='1' THEN                    ——异步复位
         PresentState<=S0;
 ELSEIF rising_edge(Clock) THEN
          PresentState<=NextState;        ——时钟上升沿，转换至下一状态
     END IF;
  END PROCESS;
```



```
Change_State:PROCESS (PresentState，DIN) ——组合逻辑进程
BEGIN
  CASE Present State IS
    WHEN S0=〉 if DIN='1' then NextState<=S1；
         else  NextState<=S0；   end if；
                DOUT <="001"；
    WHEN S1=〉 if DIN='1' then NextState<=S2；
                  else  NextState<=S1；  end if ；
                DOUT <="011"；
   WHEN S2=〉 if  DIN='1' then NextState<=S3；
                 else  NextState<=S2；  end if ；
               DOUT <="101"；
   WHEN S3=〉 if DIN='1' then NextState<=S0；
               else NextState<=S1；  end if ；
              DOUT <="111"；
   END CASE；
END PROCESS；
END；
```

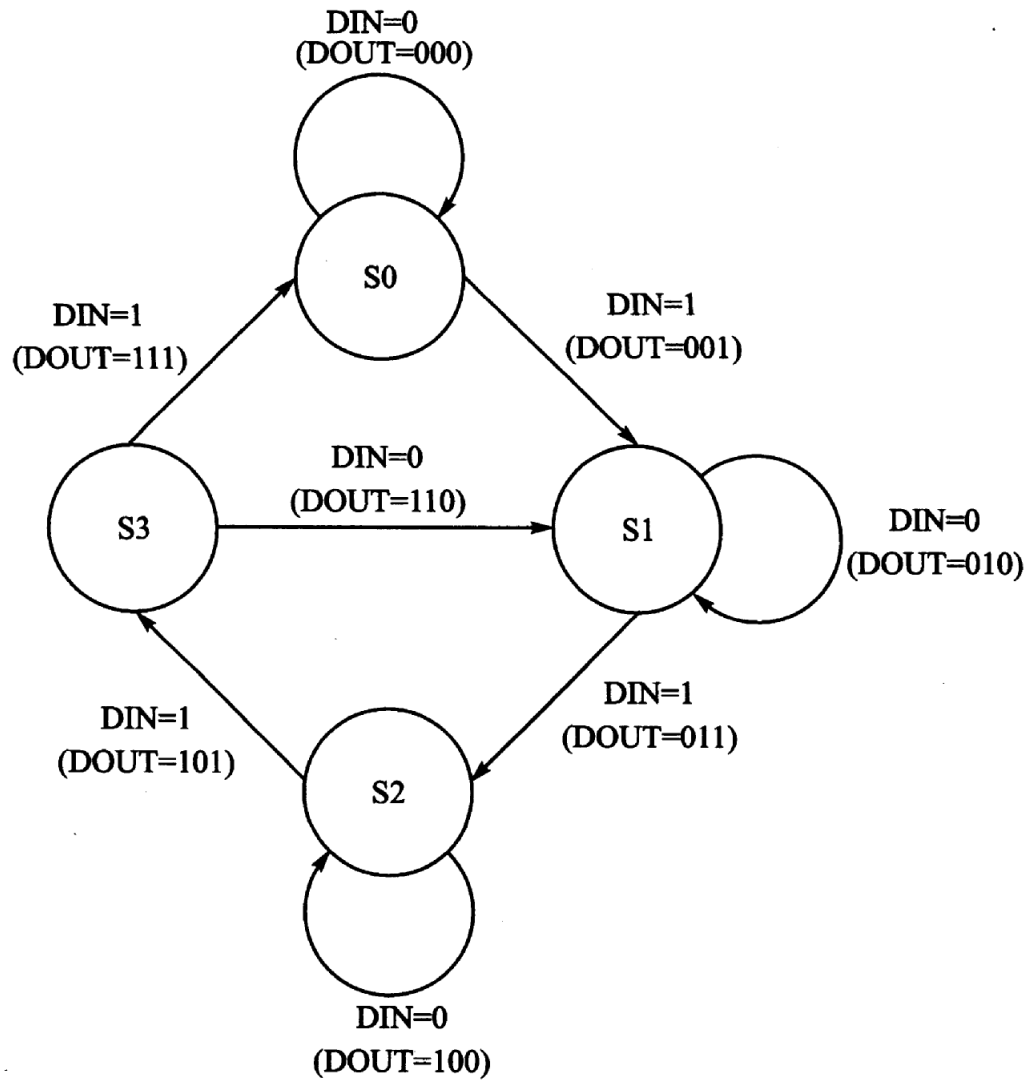


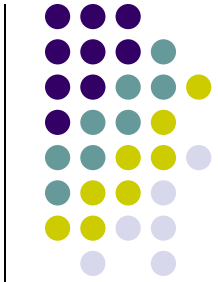
仿真波形图：


Name:
Value:
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
Reset
0
Clock
0
DIN
0
PresentState
B 00
00
01
10
11
DOUT
B 001
001
011
101
111

3.4.2 Mealy 状态机的VHDL描述：

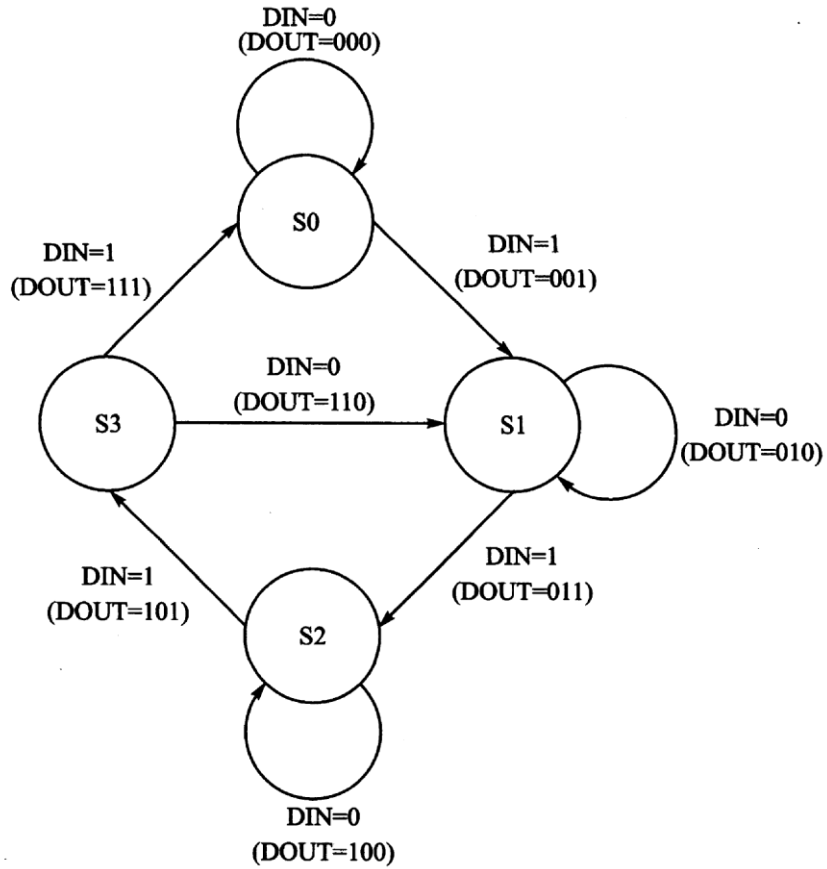
输出由当前状态与输入共同决定。

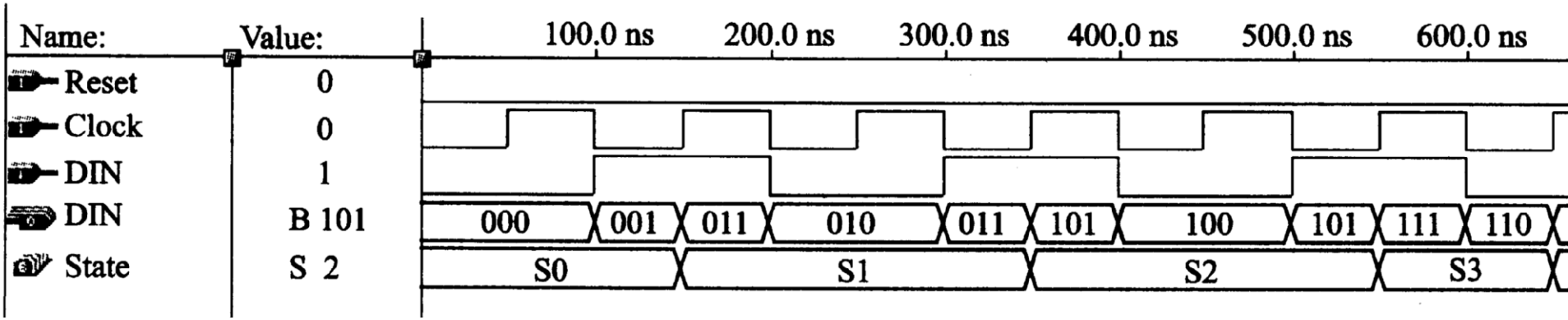




```
LIBRARY IEEE；
USE IEEE.Std_ Logic_1164.ALL；
ENTITY MealyMachine IS
PORT
（ Reset ,Clock,DIN：IN Std_Logic；
   DOUT：OUT Std_Logic_Vetor(2 DOWNTO 0))；
END；
ARCHITECTURE Statemachine OF MealyMachine IS
TYPE State_type IS (S0，S1，S2，S3)；
SIGNAL State：State_type；
BEGIN
Change_State: PROCESS (Reset，Clock)
BEGIN
IF Reset=‘1’ THEN
State<=S0；
```

```
ELSIF rising_edge(Clock) THEN
    CASE  State IS
       WHEN S0=〉 IF DIN='1' THEN State<=S1;  END IF;
       WHEN S1=〉 IF DIN='1' THEN State<=S2;  END IF;
       WHEN S2=〉 IF DIN='1' THEN State<=S3;  END IF;
       WHEN S3=〉 IF DIN='1' THEN State<=S0;
                      ELSE  State<=S1;  END IF ;
    END CASE;
  END IF;
END PROCESS;
Output_Process:  PROCESS（State，DIN）
BEGIN
  CASE State IS
     WHEN S0=>IF DIN='0' THEN DOUT <="000";
                      ELSE DOUT<="001";  END IF;
     WHEN S1=>IF DIN='0' THEN DOUT <="010";
                      ELSE DOUT<="011";  END IF;
     WHEN S2=> IF DIN='0' THEN DOUT <="100";
                      ELSE DOUT<="101";  END IF;
     WHEN S3=> IF DIN='0' THEN DOUT <="110";
                      ELSE DOUT<="111";  END IF;
   END CASE;
  END PROCESS;
END;
```




Name:
Value:
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
Reset
0
Clock
0
DIN
1
DIN
B 101
000
001
011
010
011
101
100
101
111
110
State
S 2
S0
S1
S2
S3

3.4.3 状态机的容错设计

主要针对未定义状态（剩余状态）

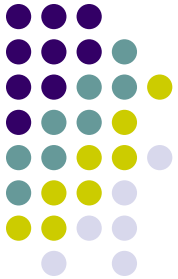
- 单独设计一个状态(ERROR)，用以处理状态机出错的情况，

  用 **WHEN OTHERS=>State<=Error**；

  使状态机从未定义的状态跳转到处理出错情况的状态；

- 直接回复到其他已设定的状态。

3.4.4 状态机设计与寄存器



- 对于所有可能的输入条件，当进程中的输出信号如果没有被完全地与之对应指定，此信号将自动被指定，即在未列出的条件下保持原值，这意味着引入了寄存器。

- 在状态机中，如果存在一个或更多的状态没有被明确指定转换方式，或者对于状态机中的状态值没有规定所有的输出值，就可能引入寄存器。

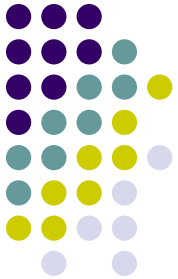
3.5 常用电路的VHDL程序

计数器：

```
--4-bit synchronous counter with count enable,
--asynchronous reset and synchronous load
CLK: in STD_LOGIC;
RESET: in STD_LOGIC;
CE, LOAD, DIR: in STD_LOGIC;
DIN: in STD_LOGIC_VECTOR(3 downto 0);
COUNT: inout STD_LOGIC_VECTOR(3 downto 0);
process (CLK, RESET)
begin
   if RESET='1' then
      COUNT <= "0000";
   elsif CLK='1' and CLK'event then
      if CE='1' then
         if LOAD='1' then
            COUNT <= DIN;
         else
            if DIR='1' then
               COUNT <= COUNT + 1;
            else
               COUNT <= COUNT - 1;
            end if;
         end if;
      end if;
   end if;
end process;
```

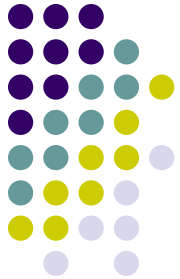
比较器：

```
--N-bit Comparator, synchronous with reset
--Please define the value of N for A and B
CLK,RESET: in STD_LOGIC;
A, B: in STD_LOGIC_VECTOR(N downto 0);
ALB, AGB: out STD_LOGIC;
ALEB, AGEB: out STD_LOGIC;
AEB, ANEB: out STD_LOGIC

process(CLK,RESET)
begin
   if (RESET = '1') then
      ALB  <= '0'; AGB  <= '0'; ALEB <= '0';
      AGEB <= '0'; AEB  <= '0'; ANEB <= '0';
   elsif (CLK'event and CLK ='1') then
    if ( A < B ) then ALB <= '1';
    else ALB <= '0';
    end if;

    if ( A > B ) then AGB <= '1';
    else AGB <= '0';
    end if;

    if ( A <= B ) then ALEB <= '1';
    else ALEB <= '0';
    end if;

    if ( A >= B ) then AGEB <= '1';
    else AGEB <= '0';
    end if;

    if ( A = B ) then AEB <= '1';
    else AEB <= '0';
    end if;

    if ( A /= B ) then ANEB <= '1';
    else ANEB <= '0';
    end if;

   end if;
end process;
```

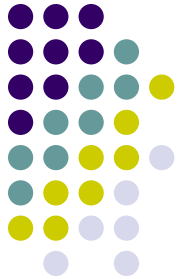


奇数倍分频：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fre_d is
    Port ( clk : in std_logic;
           clkout : out std_logic);
end fre_d;

architecture Behavioral of fre_d is
signal couter:integer range 0 to 2;
signal temp1,temp2:std_logic;
begin
 process(clk)
    begin
       if rising_edge(clk) then
         if couter=2 then
            couter<=0;  temp1<=not temp1;
          else couter<=couter+1;
          end if;
        end if;

       if falling_edge(clk) then
         if couter=1 then
           temp2<=not temp2;
          end if;
        end if;
  end process;
  clkout<= temp1 xor temp2;

end Behavioral;
```



8位奇偶校验电路

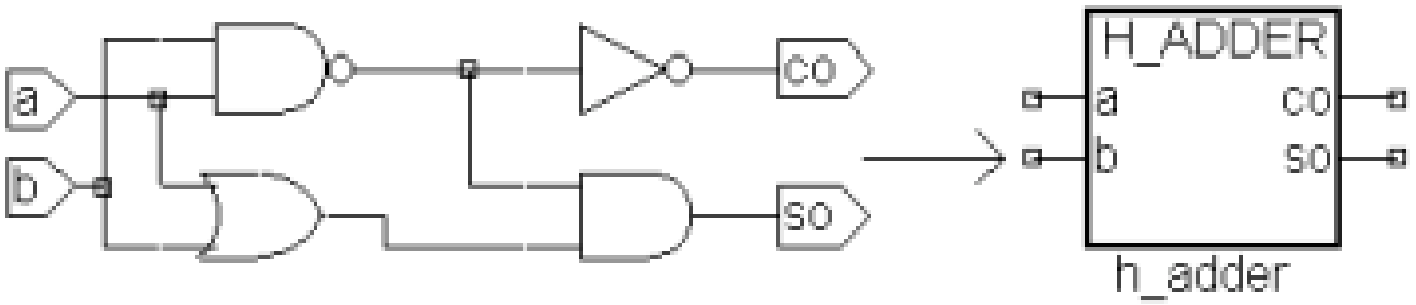


```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY p_check IS
  PORT (a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
y: OUT STD_LOGIC);
END p_check;

ARCHITECTURE opt OF p_check IS
    SIGNAL tmp: STD_LOGIC;
BEGIN
  PROCESS (a)
    BEGIN
      tmp<='0';
      FOR n IN 0 TO 7  LOOP --此循环语 句作为进程语句中的顺序语句使用
tmp <= tmp XOR a(n);
          END LOOP;
y <= tmp
 END..PROCESS;
END opt;
```

半加器

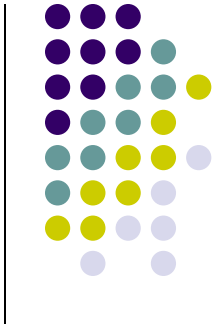
1位二进制半加器

```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY h_adder IS
  PORT (a, b : IN STD_LOGIC;
        co, so : OUT STD_LOGIC);
 END ENTITY h_adder;
 ARCHITECTURE fh1 OF h_adder I
    BEGIN
       so <= (a OR b)AND(a NAND b);
       co <= NOT( a NAND b);
  END ARCHITECTURE fh1;
```



1 位半加器逻辑原理图

含异步清0和同步时钟使能的4位加法计数器

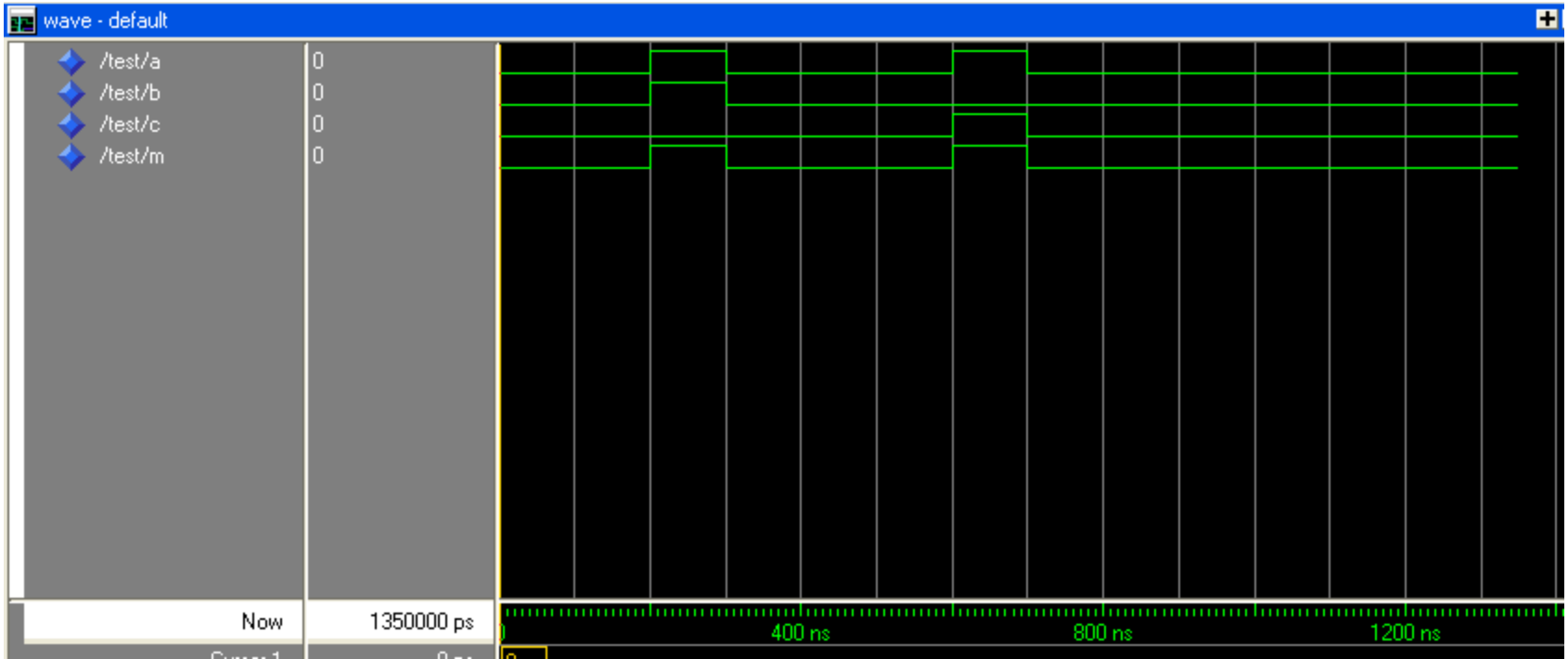


```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT4B IS
   PORT (CLK : IN STD_LOGIC;   RST : IN
STD_LOGIC;
            ENA : IN STD_LOGIC;
        OUTY : OUT STD_LOGIC_VECTOR(3
DOWNTO 0);
        COUT : OUT STD_LOGIC    );
  END CNT4B;
ARCHITECTURE behav OF CNT4B IS
   SIGNAL CQI : STD_LOGIC_VECTOR(3
DOWNTO 0);
SIGNAL CQI : STD_LOGIC_VECTOR(3
DOWNTO 0);
BEGIN
P_REG: PROCESS(CLK, RST, ENA)
       BEGIN
```

```
P_REG: PROCESS(CLK, RST, ENA)
       BEGIN
        IF RST = '1' THEN   CQI <= "0000";
         ELSIF CLK'EVENT AND CLK = '1'
THEN
       IF ENA = '1' THEN  CQI <= CQI + 1;
                ELSE  CQI <=  "0000";
          END IF;
        END IF;
         OUTY <= CQI ;
       END PROCESS P_REG ;
    COUT <= CQI(0) AND CQI(1) AND
CQI(2) AND CQI(3); --进位输出
END  behav;
```

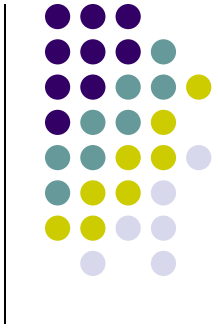
表决器：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY maj IS
   PORT(a,b,c: IN bit; m : OUT bit);
END maj;
--Dataflow style architecture
ARCHITECTURE concurrent OF maj IS
BEGIN
    WITH a&b&c SELECT
      m <= '1' WHEN "110"|"101"|"011"|"111",
           '0' WHEN OTHERS;
END concurrent;
```



三态门：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity buffer is
    Port ( OE : in std_logic;
           DATAIN : in std_logic;
           DATAOUT : in std_logic);
end buffer;

architecture Behavioral of buffer is
begin
 PROCESS(OE,DATAIN)
  if OE='0' then
     DATAOUT<='Z';
    ELSE DATAOUT<=DATAIN;
  END IF;
 end process;
end Behavioral;
```

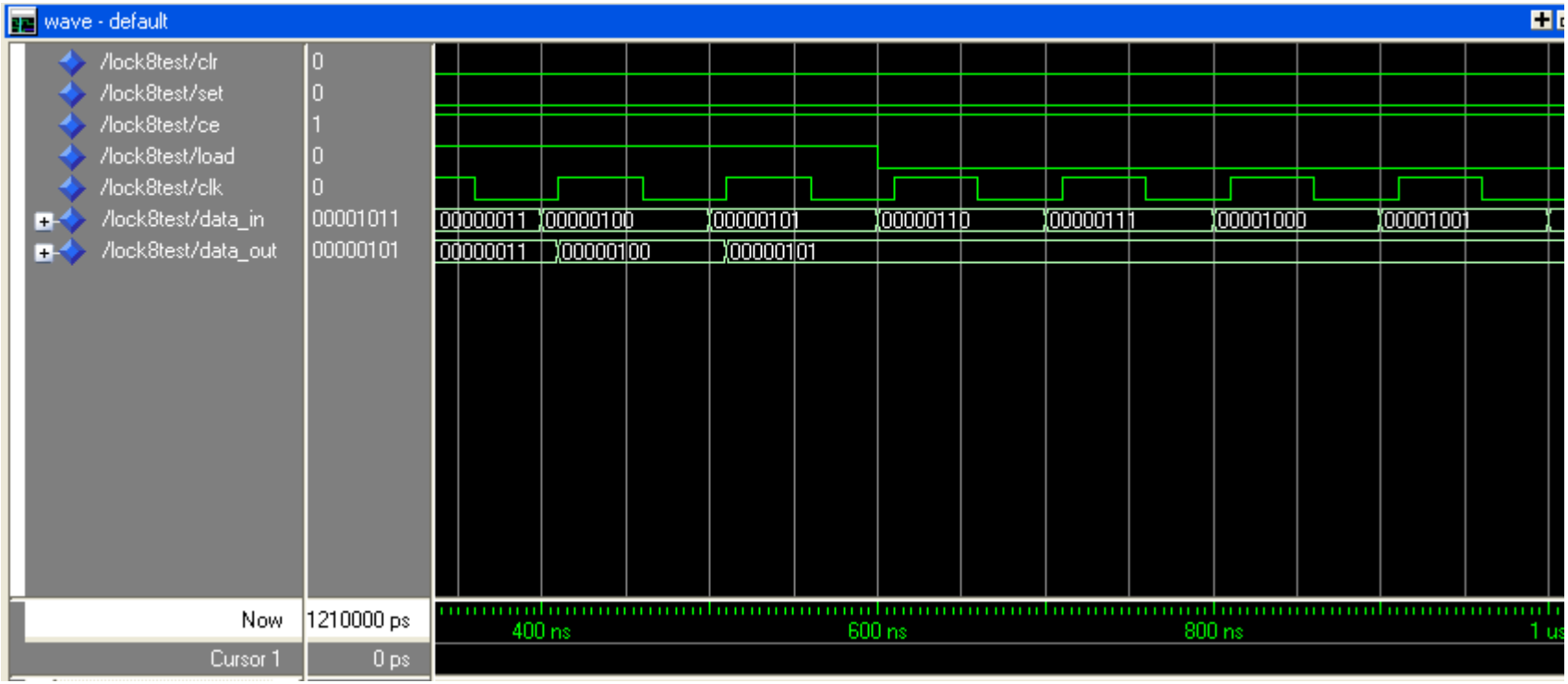


三态总线：

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
ENTITY prebus IS
    PORT(
        my_in  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        sel    : IN STD_LOGIC;
        my_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END prebus;
ARCHITECTURE cpld OF prebus IS
BEGIN
    my_out <= "ZZZZZZZZ"
    WHEN (sel = '1')
    ELSE my_in;
END cpld;
```

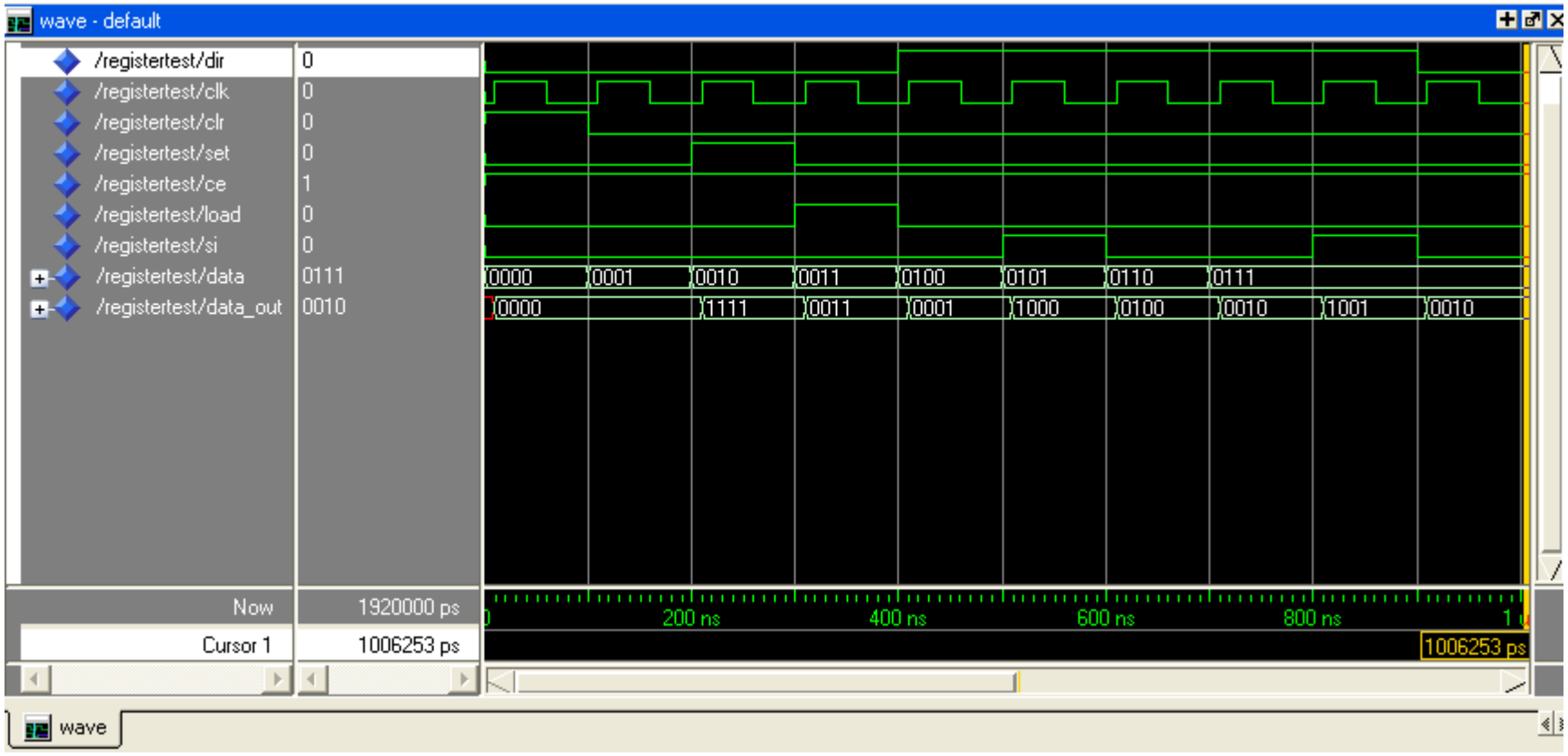
八位锁存器:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ffd is
     port (
          CLR : in std_logic;
          SET : in std_logic;
          CE : in std_logic;
          LOAD : in std_logic;
          CLK : in std_logic;
          DATA_IN : in std_logic_vector (7 downto 0);
          DATA_OUT : out std_logic_vector (7 downto 0)
     );
end entity;

architecture ffd_arch of ffd is
signal TEMP_DATA_OUT: std_logic_vector (7 downto 0);
begin
     process (CLK)
     begin

          if rising_edge(CLK) then
               if CE = '1' then
                    if CLR = '1' then
                         TEMP_DATA_OUT <= (others => '0');
                    elsif SET = '1' then
                         TEMP_DATA_OUT <= (others => '1');
                    elsif LOAD = '1' then
                         TEMP_DATA_OUT <= DATA_IN;
                    end if;
               end if;
          end if;
     end process;
     DATA_OUT <= TEMP_DATA_OUT;
end architecture;
```


wave - default
/lock8test/clr
/lock8test/set
/lock8test/ce
/lock8test/load
/lock8test/clk
/lock8test/data_in
/lock8test/data_out
0
0
1
0
0
00001011
00000101
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00000011
00000100
00000101
Now
1210000 ps
Cursor 1
0 ps
400 ns
600 ns
800 ns
1 us

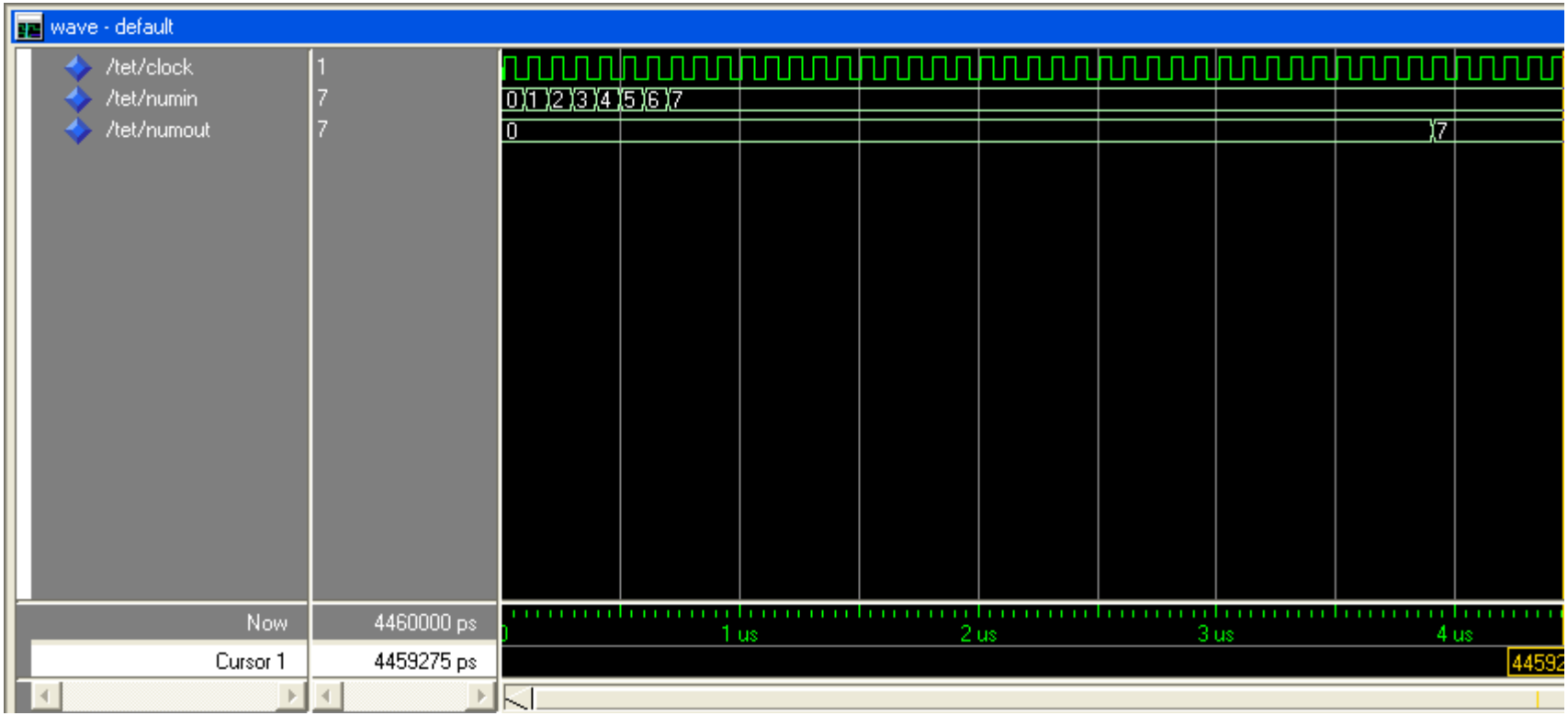
移位寄存器：

```
library IEEE;
use IEEE.std_logic_1164.all;

entity shft_reg is
     port (
          DIR : in std_logic;
          CLK : in std_logic;
          CLR : in std_logic;
          SET : in std_logic;
          CE : in std_logic;
          LOAD : in std_logic;
          SI : in std_logic;
          DATA : in std_logic_vector(3 downto 0);
          data_out : out std_logic_vector(3 downto 0)
     );
end entity;

architecture shft_reg_arch of shft_reg is
signal TEMP_data_out : std_logic_vector(3 downto 0);
begin
     process(CLK)
     begin
          if rising_edge(CLK) then
               if CE = '1' then
                    if CLR = '1' then
                         TEMP_data_out <= "0000";
                    elsif SET = '1' then
                         TEMP_data_out <= "1111";
                    elsif LOAD = '1' then
                         TEMP_data_out <= DATA;
                    else
                         if DIR = '1' then
                              TEMP_data_out <= SI & TEMP_data_out(3 downto 1);
                         else
                              TEMP_data_out <= TEMP_data_out(2 downto 0) & SI;
                         end if;
                    end if;
               end if;
          end if;
     end process;

     data_out <= TEMP_data_out;
end architecture;
```


wave - default
/registertest/dir
/registertest/clk
/registertest/clr
/registertest/set
/registertest/ce
/registertest/load
/registertest/si
/registertest/data
/registertest/data_out
0
0
0
0
1
0
0
0111
0010
0000
0001
0010
0011
0100
0101
0110
0111
0000
1111
0011
0001
1000
0100
0010
1001
0010
Now
1920000 ps
Cursor 1
1006253 ps
200 ns
400 ns
600 ns
800 ns
1006253 ps
wave

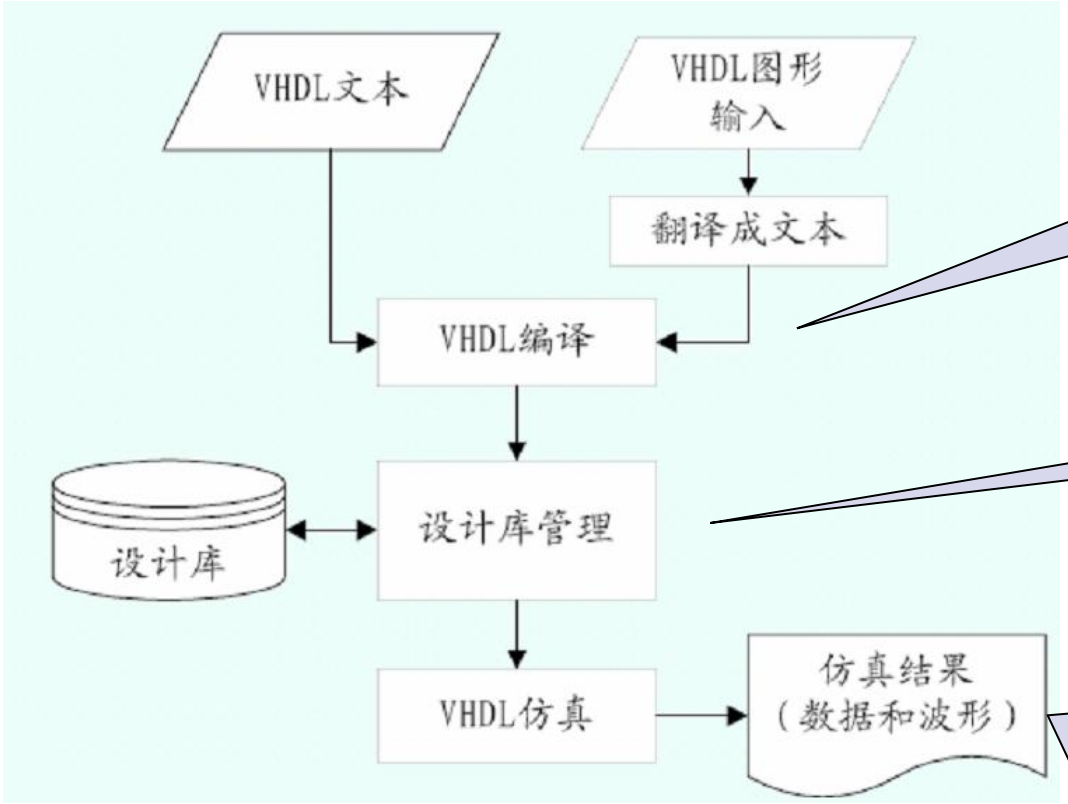
键盘消抖电路：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Antiwitter is
    Port ( clock : in std_logic;
           numin : in integer range 0 to 15;
           numout : out integer range 0 to 15);
end Antiwitter;

architecture Behavioral of Antiwitter is
   signal tempnum: integer range 0 to 15;
   signal counter: integer range 0 to 31;
   signal start:std_logic;
  begin

    process(clock)
     begin
      if rising_edge(clock) then
        if start='0' then tempnum<=15; numout<=15; start<='1';
          else
            if numin/=tempnum then tempnum<=numin; counter<=0;
              else
                if counter=31 then numout<=numin; counter<=0;
                   else counter<=counter+1;
                end if;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```


wave - default
/tet/clock
/tet/numin
/tet/numout
1
7
7
0 1 2 3 4 5 6 7
0
7
Now
4460000 ps
Cursor 1
4459275 ps
1 us
2 us
3 us
4 us

3.6 VHDL仿真

仿真（Simulation，也称模拟），不接触具体的硬件系统利用计算机对电路设计的逻辑行为和运行功能进行模拟检测，较大规模的VHDL系统设计的最后完成必须经历多层次的仿真测试过程，包括针对系统的VHDL行为仿真、分模块的时序仿真和硬件仿真，直至最后系统级的硬件仿真测试。



对VHDL源文件进行语法及语义检查，将其转换为中间数据格式。

保存中间数据结果，提供VHDL源程序中需要的设计库和程序包。

功能仿真是在未经布线和适配前，用VHDL源程序综合后的文件进行仿真。时序仿真是将设计综合后，由适配器完成具体芯片的映射后得到的文件进行仿真。

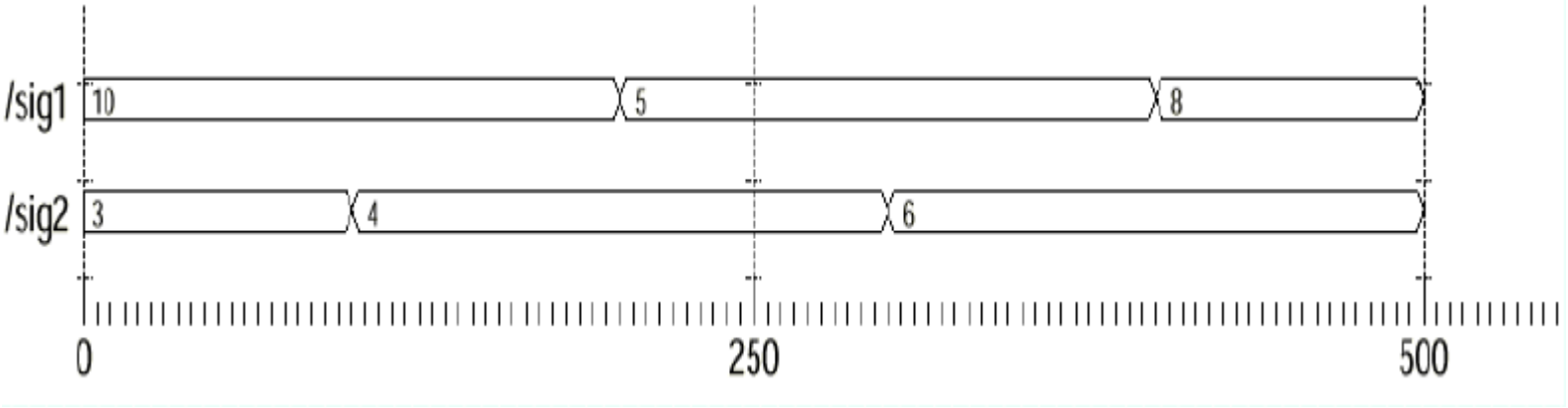
3.6.1 仿真激励信号的产生

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ADDER4 IS
    PORT ( a, b : IN INTEGER RANGE 0 TO 15;
                  c : OUT INTEGER RANGE 0 TO 15 );
END ADDER4;
ARCHITECTURE one OF ADDER4 IS
    BEGIN
    c <= a + b;
END one;
```

➢ 方法一

① 用VHDL写一个波形信号发生器

```
ENTITY SIGGEN IS
    PORT ( sig1 : OUT INTEGER RANGE 0 TO 15;
           sig2 : OUT INTEGER RANGE 0 TO 15 );
END;
ARCHITECTURE Sim OF SIGGEN IS
    BEGIN
        sig1 <= 10, 5 AFTER 200 ns, 8 AFTER 400 ns;
        sig2 <= 3, 4 AFTER 100 ns, 6 AFTER 300 ns;
END;
```

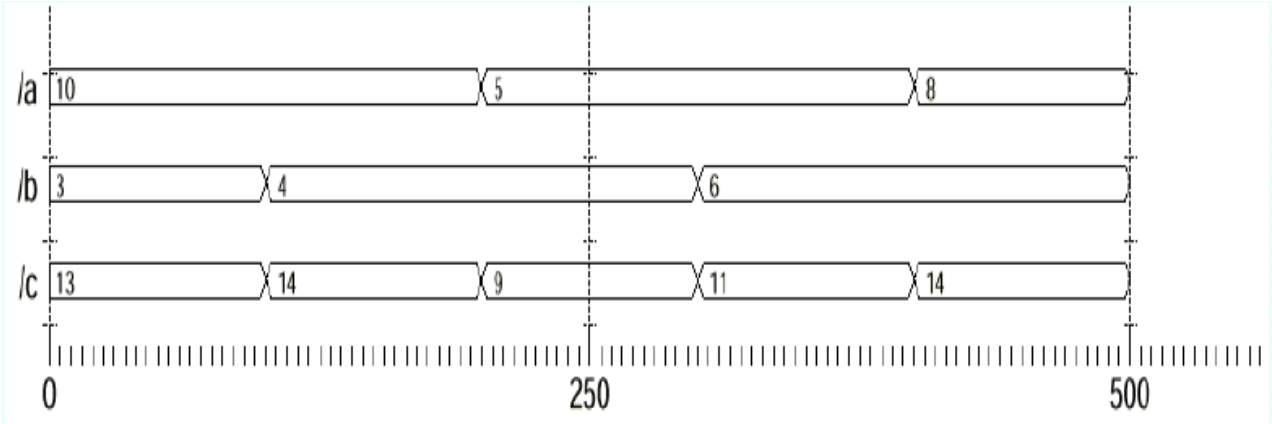


SIGGEN的仿真输出波形

② 将波形信号发生器与ADDER4组装为一个VHDL仿真测试模块

```
ENTITY BENCH IS
END;
ARCHITECTURE one OF BENCH IS
    COMPONENT ADDER4
        PORT ( a, b : integer range 0 to 15;
               c : OUT INTEGER RANGE 0 TO 15 );
    END COMPONENT;
    COMPONENT SIGGEN
        PORT ( sig1 : OUT INTEGER RANGE 0 TO 15;
               sig2 : OUT INTEGER RANGE 0 TO 15 );
    END COMPONENT;
        SIGNAL a, b, c : INTEGER RANGE 0 TO 15;
BEGIN
    U1 : ADDER4 PORT MAP (a, b, c);
    U2 : SIGGEN PORT MAP (sig1=>a, sig2=>b);
END;
```

BENCH仿真波形图



➢ 利用仿真器的波形设置命令施加激励信号

force命令的格式如下：

force <信号名> <值> [<时间>][, <值> <时间> …] [-repeat <周期>]

force a 0 （强制信号的当前值为0）

force b 0 0, 1 10 （强制信号b在时刻0的值为0，在时刻10的值为1）

force clk 0 0, 1 15 –repeat 20 （clk为周期信号，周期为20）

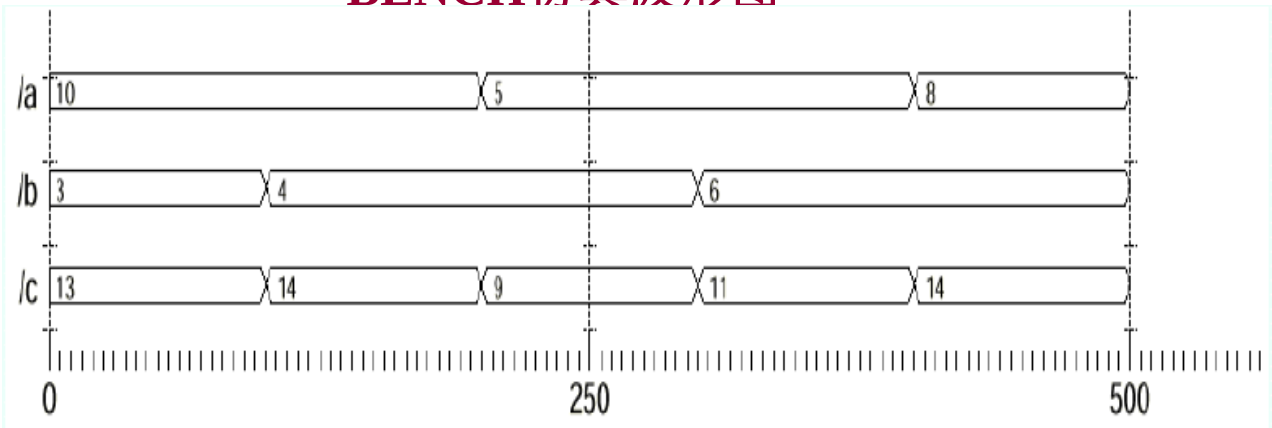
对ADDER4的结构体进行仿真：

① 初始化仿真过程后，在命令行中输入命令：

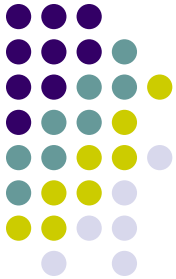
force a 10 0, 5 200, 8 400
force b 3 0, 4 100, 6 300

② 执行RUN命令。

BENCH仿真波形图

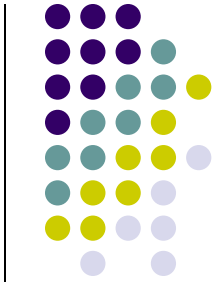


3.6.2 VHDL测试基准（Test Bench）



8位计数器源程序：

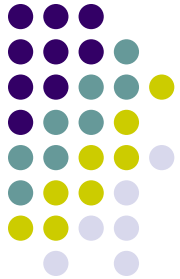
```
Library IEEE;
use IEEE.std_logic_1164.all;
entity counter8 is
    port (CLK, CE, LOAD, DIR, RESET: in STD_LOGIC;
          DIN: in INTEGER range 0 to 255;
          COUNT: out INTEGER range 0 to 255 );
end counter8;
architecture counter8_arch of counter8 is
    begin
     process (CLK, RESET)
          variable COUNTER: INTEGER range 0 to 255;
    begin
          if RESET='1' then COUNTER := 0;
          elsif CLK='1' and CLK'event then
              if LOAD='1' then COUNTER := DIN;
```



```
Else
            if CE='1' then
                if DIR='1' then
                        if COUNTER =255 then COUNTER := 0;
                            else COUNTER := COUNTER + 1;
                        end if;
                    else
                          if COUNTER =0 then COUNTER := 255;
                            else COUNTER := COUNTER-1;
                          end if;
                        end if;
                    end if;
                end if;
              end if;
            COUNT <= COUNTER;
      end process;
end counter8_arch;
```

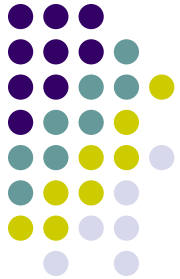
测试基准文件（Test Bench）：

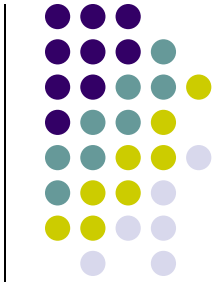
```
Entity testbench is end testbench;
Architecture testbench_arch of testbench is
File RESULTS: TEXT open WRITE_MODE is "results.txt";
Component counter8
    port ( CLK: in STD_LOGIC;
           RESET: in STD_LOGIC;
           CE, LOAD, DIR: in STD_LOGIC;
           DIN: in INTEGER range 0 to 255;
           COUNT: out INTEGER range 0 to 255 );
end component;
    shared variable end_sim : BOOLEAN := false;
    signal CLK, RESET, CE, LOAD, DIR: STD_LOGIC;
    signal DIN: INTEGER range 0 to 255;
    signal COUNT: INTEGER range 0 to 255;
    procedure WRITE_RESULTS (
           CLK，CE，LOAD，LOAD，RESET : STD_LOGIC;
           DIN，COUNT : INTEGER ) is
           Variable V_OUT : LINE;
```



```
Begin
        write(V_OUT, now, right, 16, ps);        ——输入时间
        write(V_OUT, CLK, right, 2);
        write(V_OUT, RESET, right, 2);
        write(V_OUT, CE, right, 2);
        write(V_OUT, LOAD, right, 2);
        write(V_OUT, DIR, right, 2);
        write(V_OUT, DIN, right, 257);
        --write outputs
        write(V_OUT, COUNT, right, 257);
        writeline(RESULTS,V_OUT);
        end WRITE_RESULTS;
begin
        UUT: COUNTER8
        port map (CLK => CLK,RESET => RESET,
                  CE => CE, LOAD => LOAD,
                  DIR => DIR, DIN => DIN,
                  COUNT => COUNT );
```

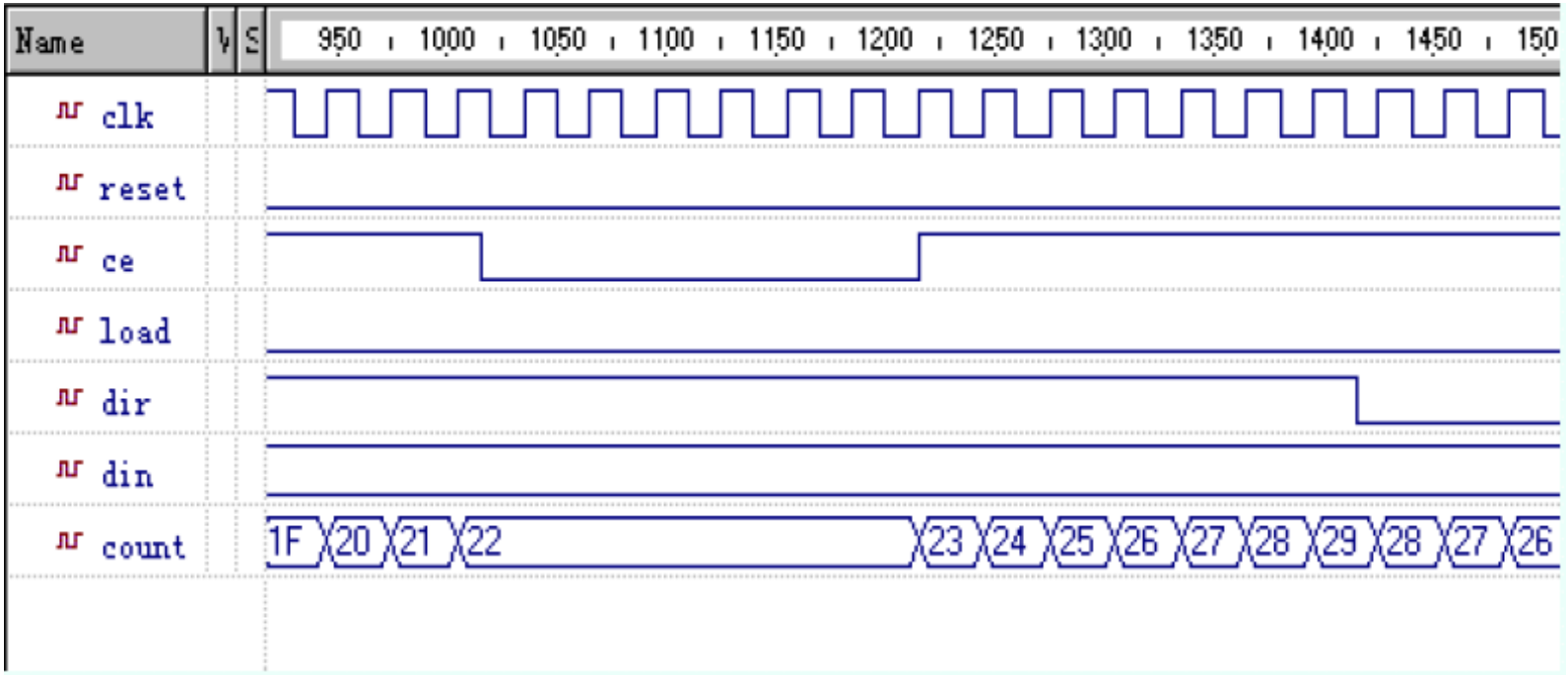
```
CLK_IN: process
Begin
if end_sim = false then CLK <= '0';
Wait for 15 ns;
CLk <='1';
Wait for 15 ns;
   Else
            Wait;
            end if;
     end process;
STIMULUS: process
Begin
     RESET <= '1';
      CE <= ‘1’;                          ——计数使能
         DIR <= ‘1’;                      —— 加法计数
       DIN <= 250;                        —— 输入数据
        LOAD <= ‘0’;                      ——禁止加载输入的数据
        wait for 15 ns;
       RESET <= '0';
     wait for 1 us;
            CE <= ‘0’;                    ——禁止计数脉冲信号进入
wait for 200 ns;
            CE <= '1';
     wait for 200 ns;
```



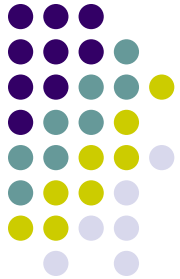


```
                DIR <= '0';
        wait for 500 ns;
                LOAD <= '1';
        wait for 60 ns;
                LOAD <= '0';
        wait for 500 ns;
                DIN <= 60;
                DIR <= '1';
                LOAD <= '1';
        wait for 60 ns;
                LOAD <= '0';
        wait for 1 us;
                CE <= '0';
        wait for 500 ns;
                CE <= '1';
        wait for 500 ns;
                end_sim :=true;
        wait;
        end process;
WRITE_TO_FILE: WRITE_RESULTS(CLK,RESET,CE,LOAD,DIR,DIN,COUNT);
End testbench_arch;
```

8位计数器测试基准仿真部分波形图



3.7 VHDL综合



把VHDL描述转化为门级电路描述，设计过程中的每一步都可称为一个综合环节。

(1) 从自然语言转换到VHDL语言算法表示，即自然语言综合；

(2) 从算法表示转换到寄存器传输级(Register Transport Level，RTL)，即从行为域到结构域的综合，即行为综合；

(3) RTL级表示转换到逻辑门(包括触发器)的表示，即逻辑综合；

(4) 从逻辑门表示转换到版图表示(ASIC设计)，或转换到FPGA的配置网表文件，可称为版图综合或结构综合。有了版图信息就可以把芯片生产出来了。有了对应的配置文件，就可以使对应的FPGA变成具有专门功能的电路器件。