

6.111 演讲 #6

VHDL语句

并行语句

- 信号赋值

- 例化

- When-Else**

- With-Select-When**

- Process(可以看作顺序语句的一个包)**

顺序语句（必须在一个进程里）

- 信号赋值

- If-Then-Elsif-Else**

- Case-When**

并行语句

信号赋值:

```
outc<=ina AND (inb OR inc);
```

例化:

```
h1: halfadd PORT MAP (a => ina, b => inb,  
                      sum => s1, c => s3); --名称相关
```

或者

```
h1: halfadd PORT MAP (ina, inb, sum, c); --位置相关
```

With-Select-When (注意: 经常使用OTHERS来表示除了'0'和'1'之外的值)
选择条件必须是互斥的, 而且没有遗漏。

```
WITH inc SELECT  
  outc <= ina WHEN '0',  
            inb WHEN '1',  
            inb WHEN OTHERS;
```

Process（进程）语句

进程语句与其他并行语句是并行执行的。

在一个结构体里边可以有多个进程。

进程语句是顺序语句的包

顺序语句建模组合或同步逻辑（或者两者都可以）

同一个进程内的语句是顺序“执行”的（但是用的时候要小心解释这个语句）

信号赋值语句既可以是顺序语句，也可以是并行语句

“变量”在进程内部声明（或者很后边）

信号必须声明在进程外部

进程的语法:

```
label:PROCESS (敏感表)
    VARIABLE --声明
BEGIN
-- 顺序语句
END PROCESS label;
```

进程的标签和变量声明是可选的。

```
PROCESS (敏感表)
BEGIN
--顺序语句
END PROCESS;
```

敏感表是用于仿真的:

在仿真时，当敏感表中的量发生改变的时候，进程就执行敏感表没有综合于实际的逻辑中（除非用了时钟）

顺序语句

信号赋值

```
outc <= ina AND (inb or inc);
```

If-then-elsif-else

```
IF      inc = '0' THEN outc <= ina;  
ELSIF   inc = '1' THEN outc <= inb;  
        ELSE outc <= inc;  
END IF;
```

Case-When:

```
CASE inc IS  
    WHEN '0'      => outc <= ina;  
    WHEN '1'      => outc <= inb;  
    WHEN OTHERS   => outc <= ind;  
END CASE;
```

还有一些没有用到的顺序语句。如果感兴趣，可以找本书看（不同的LOOP语句，等等）。

基本运算符

逻辑 (use ieee.std_logic_1164.all)

AND,OR,NAND,NOR,XOR,XNOR,NOT

关系 (use ieee.std_logic_1164.all)

=,/=<,>,<=,>=

(注意<=和>=也有其他的意思)

算术:

+,-(*也是, 但是不能被综合)

-也被定义为一元运算

并置—为字符串和信号值定义的

&

仿真和综合

综合是为了生成硬件，来完成各个语句指定的任务
(进程和其他并行语句)

仿真用于指定的输入产生输出

只要任何输入改变，并行语句就赋值

如果一个并行语句的赋值改变了另外一个并行语句的输入，该语句也被赋值

进程：你必须列出进程赋值的初始条件
(为此，使用敏感表)

当进程完成的时候，进程中的所有信号被更新
(不是当每一个顺序语句被执行的时候)

综合不考虑敏感表

这种情况下综合会做什么？

```
library ieee;  
use ieee.std_logic_1164.all;  
entity reg is
```

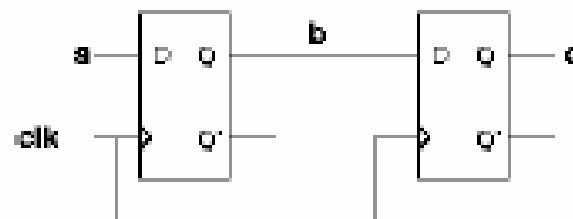
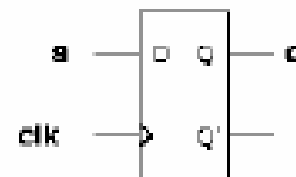
```
port (  
  a, clk : in std_logic;  
  c      : out std_logic);
```

```
end reg;  
architecture top of reg is  
  signal b : std_logic;  
begin -- 顶层
```

```
  reg2: process (clk)  
  begin -- 进程  
    if rising_edge(clk) then  
      b <= a;  
      c <= b;  
    end if;  
  end process;
```

```
end top;
```

是生成一个还是两个触发器？



好了，从.rpt文件里我们已经看出它做了什么。

DESIGN EQUATIONS (16:16:54)

$$c.D = b.Q$$

$$c.C = clk$$

$$b.D = a$$

$$b.C = clk$$

隐式存储器-锁存器例子

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    port (d, g: In std_logic;
          q : out std_logic);
end reg;

architecture top of reg is

begin
    process(d, g)
    begin
        IF g = '1' then q <= d;
            -- 注意没有ELSE
        end IF;
    end process;
end top;
-- this produces q = g*q + d*g (留心你的g's和q's! )
```

显式存储器：锁存器（相同的功能，不同的结构体）

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    port (d, g: In std_logic;
          q    : out std_logic);
end reg;

architecture top of reg is
    signal s1: std_logic;

begin
    q <= s1;
    s1 <= d when g = '1' else s1;
end top;
```

显式存储器：锁存器（相同的功能，另外一个结构体）
（注意这个比前一个更冗长）

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    port (d, g: In std_logic;
          q   : out std_logic);
end reg;

architecture top of reg is
    signal s1: std_logic;

begin
    process (s1, d, g)
    begin
        if g = '1' then s1 <= d;
        else s1 <= s1;
        end if;
    end process;
end top;
```

时钟寄存器（隐式存储器）：这是T触发器

```
library ieee;
use ieee.std_logic_1164.all;
entity clked_t is
    port (t, clk : in std_logic;
          q       : out std_logic);
end clked_t;

architecture top of clked_t is
    signal s1: std_logic;
begin
    q <= s1;
    process (clk, s1)
    begin
        if rising_edge(clk) then
            if t = '1'
                then s1 <= NOT s1;
            end if;
        end if;
    end process;
end top;

--这生成 q.D=t*/q.Q+/t*q.Q
--      和 q.C=clk
```

时钟寄存器（显式存储器）：这是T触发器

```
library ieee;
use ieee.std_logic_1164.all;
entity ckd_t is
    port (t, clk : in std_logic;
          q       : out std_logic);
end ckd_t;
```

```
architecture top of ckd_t is
    signal s1: std_logic;
begin
    q <= s1;
    process (clk, s1)
    begin
        if rising_edge(clk) then
            if t = '1'
                then s1 <= NOT s1;
                else s1 <= s1;
            end if;
        end if;
    end process;
end top;
```

--这生成 $q.D = t \oplus q$ 和 $q.C = clk$

--

例子：建一个计数器

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ctr is
    generic (width: Integer := 4); -- 允许方便的修改宽度

port(
    clk: In std_logic;
    n_clr, n_ld, enp, ent: In std_logic;
    data In std_logic_vector (width-1 downto 0);
    cnt out std_logic_vector (width-1 downto 0);
    rco out std_logic);
end ctr;
```

注意我们使用了generic来定义一个数，这个数在使用的时候可以很容易的进行重新定义。这是这个计数器的实体。结构体在下一页。

Architecture behavioral of ctr ls

```
signal Intcnt, allones: std_logic_vector (width-1 downto 0);
```

```
begin
```

```
  clocked: process (clk)
```

```
  begin
```

```
    if rising_edge(clk) then
```

```
      if n_clr = '0' then
```

```
        Intcnt <= (others => '0');
```

```
      elsif n_ld = '0' then
```

```
        Intcnt <= data;
```

```
      elsif (enp = '1') and (ent = '1') then
```

```
        Intcnt <= Intcnt + 1;
```

```
      end if;
```

```
    end if;
```

```
  end process clocked;
```

```
  allones <= (others => '1');
```

```
  zco <= '1' when ((ent = '1') AND (Intcnt = allones))
```

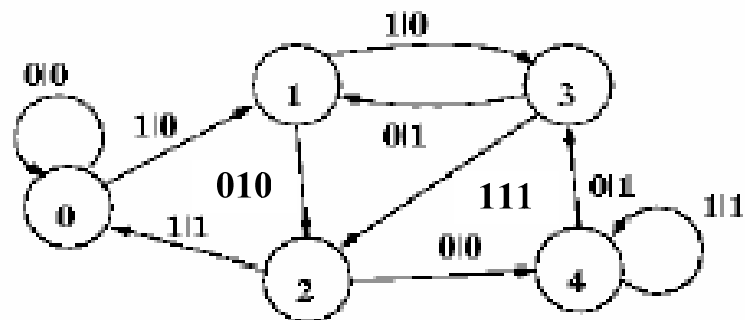
```
    else '0';
```

```
  cnt <= Intcnt;
```

```
end behavioral;
```

用VHDL构造的有限状态机

二进制逐位除5。这是一个简单的有限状态机，当前的状态就是不断得到的余数。输出是一位一位相除的值。



这是相配的实体声明。

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity divby5 is port
(
    x, clk : In std_logic;
    y      : out std_logic);
end divby5;
```

```

architecture state_machine If divby5 is
    type StateType is (state0, state1, state2, state3, state4);
    signal p_s, n_s : StateType;
begin
    fsm: process (p_s, x)
    begin
        case p_s is
            when state0 => y <= '0';
                if x = '1' then
                    n_s <= state1;
                else
                    n_s <= state0;
                end if;
            when state1 => y <= '0';
                if x = '1' then
                    n_s <= state3;
                else
                    n_s <= state2;
                end if;
            when state2 =>
                if x = '1' then
                    n_s <= state0;
                    y <= '1';
                else

```

下转下一页

```

        n_s <= state4;
        y <= '0';
    end if;
    when state3 => y <= '1';
    if x = '1' then
        n_s <= state2;
    else
        n_s <= state1;
    end if;
    when state4 => y <= '1';
    if x = '1' then
        n_s <= state4;
    else
        n_s <= state3;
    end if;
    when others => n_s <= state0; -- 避免陷阱状态
end case
end process fsm;
state-clocked : process (clk)
begin
    if rising_edge(clk) then
        p_s <= n_s;
    end if;
end process state_clocked;
end architecture state_machine;

```