

6.111演讲10

今天的主题：
有关VHDL的更多细节和更多实例
移位寄存器（如同74LS194）

注意实验2设计要在周三之前做完

第1页

VHDL标识符

对大小写不敏感（但是最好不要依赖这个）

首字符必须是字母。

（只有）字母，数字和下划线
连续两个下划线是不允许的。
最后字符不能是下划线。

使用保留字是不允许的。

emacs 的最近版本用颜色区分保留字（和其他）
使用保留字通常会引起一个可以理解的错误注释。

合法的例子

CLK, Three_StateEnable, h23, Reg_12

不合法的例子

_clk, 3_State_Enable, large#num, clk_, Three__State, register, begin

第3页

但是首先,...时钟的惯例

这只是一个惯例，但是应用的很广泛。重要的是器件
什么时间触发。



上升沿触发器件

经常叫它/CLK，因为建立时间
是在时钟信号为低的时候。
大多数寄存器是这个样子的。

下降沿触发器件

经常叫它CLK，因为建立时间
是在时钟信号为高的时候。
JK触发器是这个样子的。

第2页

VHDL的保留字

其中一些是

abs access after begin
array disconnect file
guarded impure postponed
rem unaffected wait

有97个：太多需要记忆。

这是“增量”编译的另外一个好理由。

以编译和一次增加一个块代码开始。

第4页

VHDL数值：在IEEE1164中定义。

你经常用到的数值是'0', '1', '-', 'Z'

'-' (连字符) 表示“无关项”
'Z' (必须大写) 是“高阻”

向量是字符串

注意VHDL是强类型：

a+b只有在a和b有相同的长度时才是有效的

为了给长出1位的数赋值 (调整溢出)

c<=('0'&a)+('0'&b)

也当然必须定义为比a和b长1位。

常数的定义：

'-'是字符

"---"是长度为3的字符串 (向量)

&是连接操作符：

"01"&"111"是"01111"，而'0'和"1111"也是。

第5页

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all; -- 对整数和信号是需要的
entity test_tri is
  port(clk, oe, cnt_enb : in std_logic;
        data : inout std_logic_vector(7 downto 0));
end test_tri;

architecture foo of test_tri is
  signal counter : std_logic_vector(7 downto 0);
begin
  process (oe, counter)
  begin
    if (oe = '1') then data <= counter;
    else data <= "ZZZZZZZ"; -- 注意 Z必须是大写的!
    end if;
  end process;
  process (clk)
  begin
    if rising_edge(clk) then
      if (oe = '0') and (cnt_enb = '1') then
        counter <= counter + 1;
      end if;
    end if;
  end process;
end architecture foo;
```

这是inout (三态) 的使用

第6页

这里的包是很小的包结构

实体不必须和包的声明在同一个文件中

```
library ieee;
use ieee.std_logic_1164.all;
entity mux2to1 is port (
  a, b, sel: in std_logic;
  c: out std_logic);
end mux2to1;

architecture archmux2to1 of mux2to1 is
begin
  c <= (a and not sel) or (b and sel);
end archmux2to1;

library ieee; -- 注意要重复库和use语句
use ieee.std_logic_1164.all;
package mymuxpkg is
  component mux2to1 port (
    a, b, sel: in std_logic; -- identical port list
    c: out std_logic);
  end component;
end mymuxpkg;
```

这个文件有实体和结构体

这个文件有元件声明

第7页

注意我们可以在一些顶层代码使用包：

```
--不，我不认为这有什么用...
library ieee;
use ieee.std_logic_1164.all;
entity toplevel is port (
  s: in std_logic;
  p, q, r: in std_logic_vector(2 downto 0);
  t: out std_logic_vector(2 downto 0));
end toplevel;
use work.mymuxpkg.all; -- 这是我们所谓的包
architecture archtoplevel of toplevel is
  signal i: std_logic_vector(2 downto 0);
begin
  --前两个例化为名称关联
  m0: mux2to1 port map (a=>i(2), b=>r(0), sel=>s, c=>t(0));
  m1: mux2to1 port map (c=>t(1), b=>r(1), a=>i(1), sel=>s);
  --最后一个为位置关联
  m2: mux2to1 port map (i(0), r(2), s, t(2));
  i <= p and not q;
end archtoplevel;
```

第8页

预定义属性

s'event被读作“s事件”，这里s是一个信号名。
rising_edge(event)和 (s'event and event = '1') 是一样的

每次当信号估值的时候都有一个事件出现，无论信号值是否改变。

一个信号估值的时候能强制估计其他信号的值。

第9页

74194 : 双向, 可装载的移位寄存器



S1	S0	QA	QB	QC	QD	
1	1	A	B	C	D	装载
0	1	R	QA	QB	QC	右移
1	0	QA	QB	QC	L	左移
0	0	QA	QB	QC	QD	保持

部件还有一个异步的清零

现在我们将用VHDL写这个功能

第11页

数组属性对类属数组大小是有用的

```
signal s : std_logic_vector(7 downto 3)
        s'left = 7    s'high = 7
        s'right = 3   s'low = 3
        s'length = 5
```

你甚至可以建立一个多维的索引数组：

```
type rom is array (0 to 6, 3 downto 0) of std_logic;
signal r : rom;
        r'left(1) = 0    r'high(1) = 6
        r'left(2) = 3    r'high(2) = 3
        r'right(1) = 6   r'low(1) = 0
        r'right(2) = 0   r'low(2) = 0
        r'length(1) = 7
        r'length(2) = 4
```

第10页

--可变宽度的移位寄存器 (像194)

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
```

```
entity shift_reg is
  generic (width : integer := 4); -- 开始
  port (data : in std_logic_vector(width-1 downto 0); -- 输入
        s : in std_logic_vector(1 downto 0);
        clk, sl, sr : in std_logic; -- 移动位
        output : out std_logic_vector (width-1 downto 0));
```

```
end shift_reg;
```

注意通过使用类属宽度，我们实际上能用这个代码仿效任意宽度的移位寄存器。194是4位的。

使用位置属性使得这个变化的宽度有效

第12页

```

--目的：194移位寄存器的仿真
architecture first_try of shift_reg is
    signal int : std_logic_vector(width-1 downto 0); -- 在内部使用
    constant right : std_logic_vector(1 downto 0) := "01";
    constant left : std_logic_vector(1 downto 0) := "10";
    constant load : std_logic_vector(1 downto 0) := "11";
    constant hold : std_logic_vector(1 downto 0) := "00";

begin -- first_try
    output <= int;
    shift_reg: process(clk)
    begin
        if rising_edge(clk) then
            case s is
                when right =>
                    int <= sr & int(int'left downto int'right+1);
                when left =>
                    int <= int(int'left-1 downto int'right) & sl;
                when load =>
                    int <= data; when hold => int <= int;
                when others =>
                    int <= (others => '-');
            end case;
        end if;
    end process;
end first_try;

```

第13页

两个属性，有时候很有用

当需要超过16个乘积项的时候，和的分解就出现了

(这当然也依赖于你要编译的部件)

balanced (默认的)有很好的时序，但是使用更多的宏

cascaded使用较少的宏，而且慢

```

attribute sum_split of mysig: signal is cascaded;
attribute sum_split of mysig: signal is balanced;

```

synthesis_off属性被用于使信号因式分解。

对并发信号，对信号因式分解会导致乘积项的减少。也要避免信号被优化的可能性。

寄存器方程是自然因式分解，所以只对组合信号使用synthesis_off。

```

attribute synthesis_off of sel: signal is true;

```

第15页

用户定义属性：通常使用

```

type state_type is (idle, state1, state2);[]
attribute state_encoding of state_type: is sequential;[]
-- 或者 one_hot, zero_hot, gray[]

attribute enum_encoding of state_type: is "11 01 00";[]
-- 或者你想做的任何赋值 []

within an entity: to set pin numbers:
attribute pin_numbers of counter:Entity is
    "clk:13 reset:2" &
    " count(3):3";
-- 注意上面count(3)前面的空格

within an entity: to reserve pin numbers (or avoid contention as in your kits)
attribute pin_avoid of mydesign: entity is "21 24 26";

-- 下面的语句没有什么用处

attribute lab_force of mysig: signal is al;[]
attribute node_num of buried: signal is 202;[]
attribute low_power of mydesign: entity is "b g e";[]
attribute slew_rate of count(3): signal is slow; -- or fast[]

```

第14页

```

--试做在短脉冲捕捉器
library ieee;
use ieee.std_logic_1164.all;
entity spulse is port(
    N_GO, CLK, S_CLK: in std_logic;
    aout, n_aout, xout, p: out std_logic);
end spulse;

architecture behavioral of spulse is
    signal A, N_A, X, N_X, N_CLK : std_logic;
    attribute synthesis_off of A : signal is true;
    attribute synthesis_off of N_A : signal is true;
begin
    A <= (not N_GO) or (not N_A);
    N_A <= (not A) or (not N_X);
    N_X <= (not X);
    P <= X and N_CLK;
    N_CLK <= not (S_CLK);
    aout <= A;
    xout <= X;
    n_aout <= N_A;
    ff: process(CLK)
    begin --进程
        if rising_edge(CLK) then
            X <= A;
        end if;
    end process;
end behavioral;

```

你应当记住这个例子，它使用了synthesis_off指令

下一页是这段代码以及synthesis_off指令注释外的相同代码的报告文件的摘录

第16页

综合的结果：

带有属性synthesis_off

```
DESIGN EQUATIONS
(12:40:06)

p =
  xout.Q * /s_clk

xout.D =
  aout

xout.C =
  clk

/aout =
  n_go * n_aout

/n_aout =
  /xout.Q * aout
```

不带有属性synthesis_off

```
DESIGN EQUATIONS
(12:41:12)

p =
  xout.Q * /s_clk

xout.D =
  aout

xout.C =
  clk

/n_aout =
  aout * /xout.Q

aout =
  aout * /xout.Q
  + /n_go
```

第17页