ICS 35.080
L 77

中华人民共和国国家标准

GB/T 37979—2019

可编程逻辑器件软件 VHDL 编程安全要求

Safe rule of VHDL for programmable logic device software

2019-08-30 发布　　2020-03-01 实施

国家市场监督管理总局
中国国家标准化管理委员会　发布

目　　次

前言 …… Ⅲ
1　范围 …… 1
2　规范性引用文件 …… 1
3　术语和定义 …… 1
4　缩略语 …… 1
5　安全细则 …… 1
5.1　例化类 …… 1
5.2　结构设计类 …… 4
5.3　敏感列表类 …… 8
5.4　声明定义类 …… 9
5.5　命名类 …… 10
5.6　运算类 …… 13
5.7　循环控制类 …… 16
5.8　分支控制类 …… 16
5.9　时钟类 …… 20
5.10　复位及初始化类 …… 26
5.11　状态机类 …… 30
5.12　综合/约束类 …… 33
5.13　注释类 …… 35
5.14　编码格式类 …… 36
参考文献 …… 37

前　言

本标准按照 GB/T 1.1—2009 给出的规则起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准由全国信息技术标准化技术委员会(SAC/TC 28)提出并归口。

本标准起草单位:中国航天科工集团第三研究院第三〇四研究所、中国电子技术标准化研究院、中核控制系统工程有限公司、中科院国家空间科学中心。

本标准主要起草人:李丽华、王黎、杨楠、王栋、朱琳、张国宇、韩旭东、王颖、刁立峰、刘伟、郑金艳、李文鹏、胡义武、胡勇、郝永乐、赵静、李超、程承、姜晓辉、姚秀娟、高媛、黄勇、张清、张旸旸、刘志凯、梁成华、陈朋、周宁、徐轻尘、石璐。

可编程逻辑器件软件 VHDL 编程安全要求

1 范围

本标准规定了 VHDL 用于可编程逻辑器件软件编程安全细则。

本标准适用于可编程逻辑器件软件 VHDL 的编程安全设计、编码、测试及其使用。ASIC 数字电路设计时也可参照使用。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本(包括所有的修改单)适用于本文件。

GB/T 11457—2006 信息技术 软件工程术语

GB/T 33781—2017 可编程逻辑器件软件开发通用要求

3 术语和定义

GB/T 11457—2006 和 GB/T 33781—2017 界定的术语和定义适用于本文件。

4 缩略语

下列缩略语适用于本文件。

FPGA：现场可编程门阵列(Field Programmable Gate Array)

VHDL：超高速集成电路硬件描述语言(Very-high-speed integrated circuit Hardware Description Language)

5 安全细则

5.1 例化类

5.1.1 端口例化应采取名称映射的方式，避免位置映射。

在进行端口例化时，应采取名称映射的方式，避免位置映射，利于提高程序的可读性和可维护性。

示例：

```
--遵循示例
--端口例化采取名称映射
reset_filter_inst : reset_filter
port map( reset_in    => reset_in,
          clk         => clk,
          reset_out   => reset_out );
```

```
--违背示例
--端口例化采取位置映射
reset_filter_inst : reset_filter
port map( reset_in, clk, reset_out ) ;
```

5.1.2 例化时不应输入端口悬空。

在进行例化时,输入端口应有确定的输入关系,不应悬空。

示例:

```
--遵循示例
……
example_ent_inst : example_ent
port map( a_in  => s_a,
          b_in  => s_b,
          c_out => s_c );
……
entity example_ent is
port(
     a_in : in std_logic;
     b_in : in std_logic;
     c_out : out std_logic );
end example_ent;
……
```

```
--违背示例
……
example_ent_inst : example_ent
port map( a_in  => open, --例化时输入端口悬空
          b_in  => s_b,
          c_out => s_c );
……
entity example_ent is
port(
     a_in : in std_logic;
     b_in : in std_logic;
     c_out : out std_logic );
end example_ent;
……
```

5.1.3 例化时未使用的输出端口宜采用“open”说明。

例化时,如存在不需使用的输出端口,宜采用“open”说明。

示例：

```
--遵循示例
……
example_ent_inst : example_ent
port map( a_in  => s_a,
          b_in  => s_b,
          c_out => s_c,
          d_out => open ); --未使用的端口采用 open 说明
……
entity example_ent is
port(
     a_in : in std_logic;
     b_in : in std_logic;
     c_out : out std_logic;
     d_out : out std_logic );
end example_ent;
……
```

5.1.4 端口例化时不应进行逻辑运算操作。

示例：

```
--遵循示例
……
example_ent_inst : example_ent
port map( a_in  => s_a,
          b_in  => s_b,
          c_out => s_c,
          d_out => not s_d_n );
……
s_d_n <= not s_d;
entity example_ent is
port(
     a_in : in std_logic;
     b_in : in std_logic;
     c_out : out std_logic;
     d_out : out std_logic );
end example_ent;
……
```

```
--违背示例
……
example_ent_inst : example_ent
port map( a_in  => s_a,
          b_in  => s_b,
          c_out => s_c,
          d_out => not s_d ); --端口例化时进行逻辑非运算
……
entity example_ent is
port(
      a_in : in std_logic;
      b_in : in std_logic;
      c_out : out std_logic;
      d_out : out std_logic );
end example_ent;
……
```

5.2 结构设计类

5.2.1 每个模块/单元应至少包含 1 个输入/输出端口。

示例：

```
--违背示例
entity example_ent is
port(
--没有定义端口
    ……);
end example_ent;
```

5.2.2 除双向端口外，输出信号不应直接用于驱动同一个模块内部的逻辑。

单向的输出信号不应直接用于驱动同一个模块内部的逻辑，如需使用可采用中间信号的方式进行处理。

示例：

```
--遵循示例
--输出信号反馈驱动同模块内部逻辑
entity top_ent is
port(
      A_in : in std_logic;
      B_in : in std_logic;
      Clk : in std_logic;
      A_r_out : out std_logic;
      B_out : out std_logic );
end top_ent;
architecture behave_arc of top_ent is
signal S_A_r;
```

```
begin
  B_out <= B_in and S_A_r;    --采用中间信号 S_A_r 的方式驱动内部逻辑
  process(clk)
    begin
      if clk'event and clk='1' then
        A_r_out <= A_in;
        S_A_r   <= A_in;
      end if;
  end process;
end behave_arc;
```

```
--违背示例
--输出信号反馈驱动同模块内部逻辑
entity top_ent is
port(
     A_in : in std_logic;
     B_in : in std_logic;
     Clk : in std_logic;
     A_r_out : out std_logic;
     B_out : out std_logic );
end top_ent;

architecture behave_arc of top_ent is
begin
  B_out <= B_in and A_r_out;    --输出端口 A_r_out 直接用于驱动模块内部逻辑 B_out
  process(clk)
    begin
      if clk'event and clk='1' then
        A_r_out <= A_in;
      end if;
  end process;
  end behave_arc;
```

5.2.3 内部信号应避免使用三态逻辑。

大部分可编程逻辑器件除端口外内部无三态逻辑资源，内部信号的三态逻辑无法正确实现。

示例：

```
--遵循示例
entity top_ent is
port(……
     Dout : out std_logic );
……
signal S_inter_data, S_inter_ctrl, S_inter_din : std_logic;
S_inter_data <= S_inter_din when S_inter _ctrl= '1' else S_inter_data;
……
```

```
--违背示例
entity top_ent is
port(……
      Dout : out std_logic );
……
signal S_inter_data, S_inter_ctrl, S_inter_din : std_logic;
S_inter_data <= S_inter_din when S_inter _ctrl= '1' else 'Z';
--内部信号 S_inter_data 不应使用三态门
……
```

5.2.4 顶层模块宜只进行模块例化。

示例：

```
--遵循示例
entity top_ent is
port(
      reset_in : in std_logic;
      clk_in : in std_logic;
      ……
      );
end top_ent;
architecture behave_arc of top_ent is
signal reset,clk: std_logic;
……
component reset_filter
port(
      reset_in : in std_logic;
      clk : in std_logic;
      reset_out : out std_logic );
end component;

component GL33
port(
      PAD : in std_logic;
      GL : out std_logic );
end component;
……
reset_filter_0 : reset_filter
port map( reset_in => reset_in,
          clk      => clk,
          reset_out=> reset );
clk_0 : GL33
port map( pad => clk_in,
          GL => clk );
          ……
end behave_arc;
```

```
--违背示例
ENTITY top_ENT(……);
    ……
    signal S_Dout1_2:std_logic;
    ……
    S_Dout1_2 <= Dout1_out or Dout2_out;
    U0: big_block1 port map(clk =>clk, rst =>rst, D_in => Din1_in,Dout => Dout1_out);
    U1: big_block2port map(clk  =>clk, rst => rst, D_in=> Din2_in,Dout => Dout2_out);
    U3: big_block3 port map(clk =>clk, rst =>rst, D_in => S_Dout1_2,Dout => Dout3_out);   --
在顶层处理的模块例化外,使用了胶连逻辑
    ……
end behave_ARC
```

5.2.5 设计中不应存在多余代码或不可达分支。

程序中的多余代码或不可达分支不利于程序的管理和维护,同时影响语句、分支等的覆盖率。

示例:

```
--违背示例
……
signal S_count : std_logic_vector(3 downto 0);
signal S_flag : std_logic;
……
process (clk,reset_n)
begin
  if (reset_n = '0') then
    S_count <= (others=>'0');
    S_flag <= '0';
  elsif clk'event and clk = '1' then
    if (S_count = 16) then      --该分支不可达
      S_flag <= '1';
    else
      S_count <= S_count + 1;
    end if;
  end if;
……
end process
```

5.2.6 设计中不应产生非预期的锁存器。

锁存器是电平触发,易受毛刺干扰,应避免非预期锁存器的产生。

示例:

```
--产生锁存器的实现
data_reg <= data when enable = '1' else data_reg;
```

```
--无锁存器的实现
data_reg <= data when enable = '1' else '0';
```

5.2.7 设计模块划分应以功能独立性为原则，一个文件宜对应一个设计模块。

5.2.8 宜按照不同的设计目标进行模块划分。

设计模块划分应考虑不同设计目标，区分面积敏感和速度敏感的部分，便于给不同模块进行不同约束和优化。

5.2.9 宜将相关的组合逻辑和可以复用的逻辑划分在同一个模块层次中。

将相关的组合逻辑和可以复用的逻辑划分在同一模块层次中，有利于综合工具优化时序关键路径，减少面积消耗，提高代码可读性。

5.3 敏感列表类

应保证进程中的敏感列表完整、正确，避免敏感信号多余或缺失。组合逻辑敏感信号列表一般应包括所有影响进程输出的信号，时序逻辑敏感信号列表一般仅包括时钟和异步复位信号。

示例：

```
--遵循示例
process (a, b,enable)
begin
  if enable = '0' then
    out1 <= a;
  else
    out1 <= b;
  end if;
end process;
```

```
--违背示例
process (a, b)      --组合逻辑中敏感列表缺少 enable 信号
begin
  if enable = '0' then
    out1 <= a;
  else
    out1 <= b;
  end if;
end process;
```

```
--违背示例
process (clk, rst_n, enable) --时序逻辑中敏感列表 enable 信号多余
  begin
    if rst_n = '0'then
      out_reg <= '0';
    elsif clk'event and clk = '1'then
      if enable = '0' then
        out_reg <= a;
      end if
    end if;
end process;
```

5.4 声明定义类

5.4.1 多位信号、变量定义时应设置位宽。

示例：

```
--遵循示例
--信号、变量定义时应设置位宽
……
signal S_i : std_logic_vector (1 downto 0);
……
S_i <= "01";
……
```

5.4.2 避免出现多余的信号。

设计中所有的信号应至少被使用过一次，避免出现没有使用的声明对象。

示例：

```
--违背示例
entity sig_ent is
port(
      rst_n : in std_logic;
      clk : in std_logic;
      cnt_out : out std_logic_vector (5 downto 0) );
end sig_ent;

architecture behave_arc of sig_ent is
signal S_count : std_logic_vector (5 downto 0);
signal cnt : std_logic_vector (5 downto 0);    --程序中未使用，多余信号
begin
  cnt_out <= S_count(5 downto 0);
process (rst_n, clk)
  begin
    if rst_n ='0' then
      S_count <= (others => '0');
    elsif clk'event and clk = '1'then
      S_count <= S_count +1;
    end if;
  end process;
end behave_arc;
```

5.4.3 条件判断表达式中避免使用“?”“X”“Z”等状态。

示例：

```
--违背示例
entity top_ent is
port(
      D_in : in std_logic;
      D_out : out std_logic );
end top_ent;
……
signal S_Ctrl_in : std_logic;
  D_out <= D_in when S_Ctrl_in = 'Z'else '1';  --条件表达式中应避免使用"Z"状态
    ……
```

```
--违背示例
entity top_ent is
port(
      D_in : in std_logic;
      D_out : out std_logic);
end top_ent;
……
signal S_Ctrl_in : std_logic_vector(2 downto 0);
  D_out <= D_in when S_Ctrl_in = "1? 0 "else '1';  --条件表达式中应避免使用"?"状态
……
```

5.4.4 不应在一个进程中使用多个 event control 段。

当 process 语句 event control 段存在多个表达式时，会造成不同综合工具的综合差异，不利于代码的移植。

示例：

```
--违背示例
……
signal S_a, S_b : std_logic;
signal S_r : std_logic;
……
process (S_a, S_b)
  begin
    if S_a'event and S_a = '1'then
      if S_b'event and S_b = '1'then--不应在一个进程中使用多个 event control 段
        S_r = S_a xor S_b;
      end if;
    end if;
  end process;
……
```

5.5 命名类

5.5.1 信号、变量等自定义短标识符的命名不应与关键字重复。

示例：

```
--违背示例
……
signal S_a ：std_logic；
signal Label ：std_logic；      --信号命名与关键字重复
……
```

5.5.2 信号、变量、实体名等不同类型的自定义的短标识符不应采用相同的名称。

5.5.3 信号、变量等自定义的短标识符的命名要求如下：

a） 应只包含字母、数字字符和下划线；

b） 应以英文字母开头；

c） 末字符不应使用下划线；

d） 不应连续使用下划线。

5.5.4 不应使用大小写英文字母区分不同的命名。

5.5.5 时钟的命名宜与一般的端口名字相区别。

时钟的命名宜以 clk_开始，以时钟频率或时钟用途标识结束，非时钟相关信号名称不宜包含 clk。

示例：

```
--遵循示例
--时钟的命名宜与一般的端口名字相区别
entity clk_ent is
port(
      clk ：in std_logic；
      rst ：in std_logic；
      clk_10M ：out std_logic；
      clk_gps ：out std_logic )；
end clk_ent；
```

5.5.6 命名宜符合下列规则：

a） 信号名称不宜超过 20 个字符；

b） 变量名称宜以 V 开始；

c） 常量名称宜以 C 开始；

d） 实体名称宜以_ent 结束；

e） 实体名称约定文件宜是〈entity〉.vhd；

f） 结构体名称宜以_arc 结束；

g） 结构文件名称宜是〈architecture〉.vhd；

h） 包名称宜以_pack 结束；

i） 包声明文件名称宜是〈package〉.vhd；

j） 包文件名称宜是〈package〉_body.vhd；

k） 配置名称宜以_conf 结束；

l） 配置声明文件名称宜是〈configuration〉.vhd；

m） 部件名称宜以 comp 开始；

n） 类型名称宜以 T 开始；

o） 子类型名称宜以 ST 开始；
p） 函数名称宜以 F 开始；
q） 器件名称宜以 U 开始；
r） 块标号名称宜以_block 结束；
s） generate 标号名称宜以_gen 结束；
t） 进程标号名称宜以_proc 结束；
u） 初始信号名称宜以 P_开始；
v） 输入端口名称宜以_in 结束；
w） 输出端口名称宜以_out 结束；
x） 输入/出端口名称宜以_inout 结束；
y） 初始化模块标号名称以_init 结束；
z） 使能信号名称宜以_en 结束，非使能信号名称不宜使用_en；
aa） 低有效信号名称宜使用_n 结尾；
bb） 经过一个反相器或缓冲器的信号名称宜仅改变信号后缀名；
cc） 复位端口名称宜使用 rst 起始命名，非复位信号名称里不宜包含 rst。

5.5.7 跨越多个模块层次的信号应该使用同样的命名。

当一个信号跨越多个模块层次时，在不同的层次使用同样的命名有助于提高程序的可读性和可维护性。

示例：

```
--遵循示例
entity rs422_ent is
port(
      clk : in std_logic;
      rst : in std_logic;
      rx_in : in std_logic;
      rx_data_out : out std_logic_vector (15 downto 0);
      rx_syn_out : out std_logic );
end rs422_ent;
……
  rx_syn_out <= S_rx_synb and S_rx_syna;
--信号 clk、rst、rx_in、rx_data_out、rx_syn_out 跨越多个层次时采用相同的命名
rx_b_block : rec_ent
port map ( clk          => clk,
           rst          => rst,
           rx_in        => rx_in,
           rx_data_out  => rx_data_out(15 downto 8),
           rx_syn_out   => S_rx_synb );
rx_a_block : rec_ent
port map ( clk          => clk,
           rst          => rst,
           rx_in        => rx_in,
           rx_data_out  => rx_data_out(7 downto 0),
           rx_syn _out  => S_rx_syna );
……
```

5.6 运算类

5.6.1 信号应在赋值后使用。

信号在使用前应赋值，以保证其使用时具有确定的状态。

示例：

```
--违背示例
entity example_ent is
port(
      SEL_in : in std_logic;
DATA_out : out std_logic );
end example_ent;

architecture behave_arc of example_ent is
signal S_tmp : std_logic;
begin
process (SEL_in, S_tmp)
  begin
    if(SEL_in = '1') then
      DATA_out <= S_tmp;   -- S_tmp 信号在使用前没有赋值
    else
      DATA_out <= '0';
    end if;
  end process;
end behave_arc;
```

5.6.2 运算符两端信号或参数位宽应相等。

运算符两端信号或参数位宽应相等，具体要求如下：

a) 赋值语句、比较语句、运算语句左右信号或参数长度应该相等；

b) 在 case 语句中，case 表达式和 case 条件语句中的信号或参数的长度应该相等。

5.6.3 常量赋值表达式等号右边应为固定整型值或可计算的固定整型值。

设计中所有的常量应具有可计算性，且为固定的整型值，否则被声明或赋值的常量在编译期间无法参与计算。

示例：

```
--遵循示例
--常量赋值表达式可以计算出明确的数值
……
Signal S : integer;
constant Cint1 : integer := 1;
constant Cint2 : integer := Cint1 + 1;
……
begin
S <= Cint1 + Cint2;
……
```

5.6.4 不应在整数、实数和无符号整数之间使用隐式转换。

在整数、实数、无符号整数任何两种类型之间进行隐式转换，会影响程序的可移植性。

示例：

```
--违背示例
……
signal u1 : unsigned(3 downto 0);
signal int1 : integer range 0 to 15;
……
int1 <= u1; --将无符号数隐式转化为整数
……
```

```
--遵循示例
……
signal u1 : unsigned (3 downto 0);
signal int1 : integer range 0 to 15;
……
int1 <= to_integer(u1); --使用转换函数进行不同数据类型之间的转换
……
```

5.6.5 避免将有符号数和无符号数混用于同一运算式。

示例：

```
--违背示例
……
signal u1, u2 : unsigned (3 downto 0);
signal s1 : signed (3 downto 0);
……
u2 <= u1 + s1; --将有符号数和无符号数混于同一运算式中
……
```

5.6.6 同一模块中的单向信号不应有多重驱动。

采用多个信号驱动同一个信号，容易造成信号的冲突。

示例：

```
--违背示例
……
signal S_net_1, S_net_2, S_net_3, S_net_4 : std_logic;
……
  out <= S_net_1 and S_net_2;
  out <= not (S_net_3 nor S_net_4);
-- out 被多个信号驱动
……
```

5.6.7 相同的信号源应避免驱动同一个寄存器的多个异步信号端。

为在设计中避免竞争情况，应避免使用同一信号驱动一个寄存器的多个异步端口。

示例：

```
--违背示例
--相同的信号源不应驱动同一个寄存器的多个异步信号端
……
test_block：test_ent      -- rst 和 set 分别为 test_block 模块的复位端和置位端
port map( clk   =〉clk，
          rst   =〉rst_c，     --使用 rst_c 信号驱动复位端
          d_in  =〉d_in，
          set   =〉rst_c，     --使用 rst_c 信号驱动置位端
          d_out =〉d_out );
……
```

5.6.8 三态节点的使能端和输入端不宜被同一信号源驱动。

示例：

```
--违背示例
entity example_ent is
port(
      d_in : in std_logic;
      en_in : in std_logic;
      clk : in std_logic;
      rst : in std_logic;
      d_out : out std_logic );
end example_ent;
……
signal S_ctrl : std_logic;
signal S_tmp1 : std_logic;
S_ctrl 〈= d_in and S_tmp1;
  d_out 〈= S_ctrl   when S_ctrl = '1'else 'Z';
--输入信号和使能信号被相同的信号源(S_ctrl)驱动
……
```

5.6.9 不宜使用组合逻辑的输出控制三态门。

组合逻辑易由于竞争冒险产生毛刺，不宜使用组合逻辑输出作为使能控制三态门。

示例：

```
--违背示例
entity example_ent is
port(
      a_in : in std_logic;
      b_in : in std_logic;
      d_in : in std_logic;
      d_out : out std_logic );
end example_ent;
……
signal S_c : std_logic;
  S_c 〈= a_in xor b_in;
  d_out 〈= d_in when S_c = '0'else 'Z';   ---使用组合逻辑的输出控制三态门
……
```

5.7 循环控制类

5.7.1 用于控制循环终止条件的参数应为常量。

示例：

```
--遵循示例
……
SUM: for i in 1 to 3 loop      --用于控制循环终止条件的参数为常量
sum = sum + i;
end loop SUM;
```

5.7.2 不应在 for 循环体内修改循环变量。

示例：

```
--违背示例
……
SUM: for i in 1 to 3 loop
  sum = sum + i;
  i := i + 1;                  --在循环体内修改循环变量
end loop SUM;
……
```

5.7.3 while 语句条件表达式不应为常量。

示例：

```
--遵循示例
……
i := 1;
sum := 0;
while i <= 5 loop
sum = sum + i;
  i := i + 1;
end loop;
```

5.8 分支控制类

5.8.1 case 语句表达式不应为固定常量。

case 语句执行时，根据条件数值选择不同分支执行，如果条件表达式为常量则其余分支多余。

示例：

```
--违背示例
entity example_ent is
port(
      A_in, SEL_in : in std_logic;
      A_out : out std_logic );
end example_ent;
```

```
architecture behave_arc of example_ent is
  constant CTMP : std_logic := '1';
  constant CFS_COND : std_logic := '1';
begin
process (SEL_in, A_in)
  begin
  --case 语句的条件表达式不应为常量
    case (CFS_COND) is
      when CTMP => A_out <= A_in;
      when others => A_out <= 'X';
end case;
  end process;
end behave_arc;
```

5.8.2 case 语句的分支不应使用变量或者变量表达式。

示例：

```
--违背示例
case (Va) is
when "01" : Vb = Va;
when "10" : Vb = Va + 1;
when "Vc + Vd" : Vb = 1; --使用变量或者变量表达式作为 case 语句的分支
……
end case
```

5.8.3 case 语句除所有条件分支能完全覆盖其表达式或状态机寄存器的所有取值外，应包含一个 when others 分支。

示例：

```
--遵循示例
case (Va & Vb & Vc) is
    when "100" =>
        base_addr_r(24 downto 0) <= VA + (code_r(7 downto 0) & "000000");
    when"000" =>
        base_addr_r(24 downto 0) <= VB + (code_r(7 downto 0) & "000000");
    when "101" =>
        base_addr_r(24 downto 0) <= VC + (code_r(7 downto 0) & "000000");
    when "001"=>
        base_addr_r(24 downto 0) <= VD + (code_r(7 downto 0) & "000000");
    when "010" =>
        base_addr_r(24 downto 0) <= VE + (code_r(7 downto 0) & "000000");
--case 语句包含一个 when others 分支
    when others=> base_addr_r(24 downto 0) <= base_addr_r(24 downto 0);
end case;
```

```
--遵循示例
type state is (s0, s1, s2, s3, s4, s5);
signal nstate, cstate : state;
……
process (cstate, ……)
begin
  nstate <= s0;
  case nstate is
    when s0 =>
    ……
    when s1 =>
    ……
    when s2 =>
    ……
    when s3 =>
    ……
    when s4 =>
    ……
    when s5 =>
    ……
-- case 语句包含一个 when others 分支
    when others=>
    ……
    end case;
end process;
```

5.8.4 组合逻辑中 if 语句应包含 else 分支。

组合逻辑中,if 语句如缺少 else 分支会产生锁存器。

示例:

```
--遵循示例
architecture behave_arc of example_ent is
begin
process (a, en)
  begin
    if (en = '1') then
      c <= a;
    else
      c <= 0;
    end if;
  end process;
end behave_arc;
```

```
--违背示例
architecture behave_arc of example_ent is
begin
process (a, en)
  begin
    if (en = '1') then      --if 语句缺少 else 分支
      c <= a;
    end if;
  end process;
end behave_arc;
```

5.8.5 不宜在进程中出现空语句。

示例：

```
--遵循示例
architecture behave_arc of example_ent is
begin
  process (a, en)
    begin
      if (en = '1') then
        c <= a;
      else
        c <= 0;
      end if;
    end process;
end behave_arc;
```

```
--违背示例
architecture behave_arc of example_ent is
begin
  process (a, en)
    begin
      if (en = '1') then
        c <= a;
      else
        NULL; --不宜在进程中出现空语句
      end if;
    end process;
end behave_arc;
```

5.8.6 多级 if else 语句应重视优先级顺序对性能的影响。

第一级 if 优先级最高，路径延时也最短。如果存在关键路径的信号，宜结合功能实现将其放在靠前的 if 语句内。

5.9 时钟类

5.9.1 时钟信号不应再汇聚。

时钟信号再汇聚，即时钟信号经过不同路径后进行组合逻辑处理产生一个信号作为时钟信号，再汇聚路径易导致时钟路径上发生时序冲突。

示例：

```
--违背示例
entity example_ent is
port(
      clk : in std_logic;
      rst_n : in std_logic;
      sel1 : in std_logic;
      sel2 : in std_logic;
      din : in std_logic;
      dout : out std_logic );
end example_ent;

architecture behave_arc of example_ent is
signal clk1 : std_logic;
signal clk2 : std_logic;
signal clk_sel : std_logic;
begin
   clk1 <= sel1 and clk;
   clk2 <= sel2 or clk;
   clk_sel <= clk1 and clk2;--此处的时钟信号在时钟网络上发生了再汇聚
process (rst_n, clk_sel)
  begin
    if(rst_n = '0')then
      dout <= '0';
    elsif clk_sel'event and clk_sel = '1' then
      dout <= din;
    end if;
  end process;
end behave_arc;
```

上述代码的电路示意图见图 1。

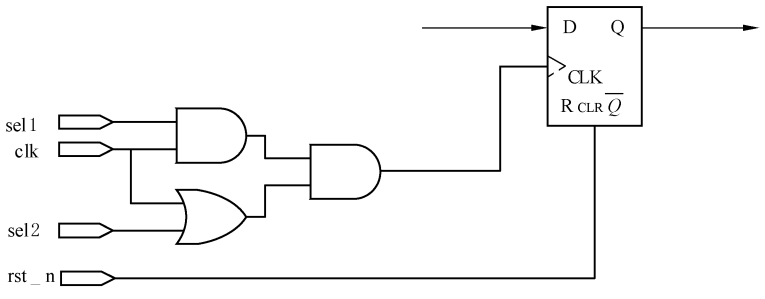

图 1 代码电路示意图

5.9.2 时钟信号不应连接到多路选择器的控制端。

示例：

```
--违背示例
entity example_ent is
port(
      clk : in std_logic;
      d_in : in std_logic;
      d_out : out std_logic );
end example_ent;
……
   d_out <= d_in when clk = '1'else '0'; --时钟信号连接到多路选择器的控制端
……
```

5.9.3 不宜使用组合逻辑产生的时钟。

组合逻辑容易产生毛刺，作为时钟控制信号时引起触发器的错误翻转。

示例：

```
--违背示例
entity example_ent is
port(
      a_in : in std_logic;
      b_in : in std_logic;
      d_in : in std_logic;
      d_out : out std_logic );
end example_ent;

architecture behave_arc of example_ent is
signal clk : in std_logic;
begin
    clk <= a_in and b_in; --组合逻辑产生时钟
process (clk)
  begin
    if clk'event and clk = '1' then --使用组合逻辑产生的时钟
       d_out <= d_in;
    end if;
  end process;
end behave_arc;
```

5.9.4 避免使用门控时钟。

使用门控时钟易因毛刺引起误触发，产生逻辑错误。

示例：

```
--违背示例
entity example_ent is
port(
      clk : in std_logic;
      ……
      clk_out : out std_logic );
end example_ent;
……
signal s1 : std_logic;
……
clk_out <= clk and s1; --输出时钟由输入时钟、控制信号经与门产生，容易产生毛刺
……
```

5.9.5 同一进程中应只使用一个时钟。

示例：

```
--违背示例
process(clk_1,clk_2) --在一个进程中使用多个时钟
  begin
    if rising_edge(clk_1)then
      a <= b;
    end if;
    if rising_edge(clk_2)then
      c <= d;
    end if;
  end process;
```

5.9.6 对跨时钟域信号应采取同步处理措施。

异步信号未经同步使用，可能产生亚稳态，引起错误。

示例：

```
--违背示例
--示例中 a_in 为外部输入信号与时钟 clk 异步
entity example_ent is
port(
      a_in : in std_logic; -- a_in 为外部的异步信号
      clk : in std_logic;
      d_out : out std_logic_vector (1 downto 0) );
end example_ent;

architecture behave_arc of example_ent is
signal S_cnt : std_logic_vector(1 downto 0);
……
begin
process (clk)
```

```
  begin
    if clk'event and clk = '1' then
      if(a_in = '1') then --a_in 为外部的异步信号,此处未经同步化直接使用
        S_cnt <= S_cnt + 1;
      end if;
    end if;
  end process;

  d_out <= S_cnt;
……
end behave_arc;
```

```
--遵循示例
entity example_ent is
port(
     a_in : in std_logic;
     clk : in std_logic;
     d_out : out std_logic_vector (1 downto 0) );
end example_ent;

architecture behave_arc of example_ent is
signal S_cnt : std_logic_vector(1 downto 0);
signal a_reg1 : std_logic;
signal a_reg2 : std_logic;
……
begin
process (clk) --同步输入信号 a_in
  begin
    if clk'event and clk = '1' then
      a_reg1 <= a_in;
      a_reg2 <= a_reg1;
    end if;
  end process;

process (clk)
  begin
    if clk'event and clk = '1' then
      if(a_reg2 = '1') then --使用同步后的信号
        S_cnt <= S_cnt + 1;
      end if;
    end if;
  end process;

  d_out <= S_cnt;
……
end behave_arc;
```

5.9.7 全局时钟宜使用全局时钟布线资源。

示例：

```
--遵循示例
--本举例是以某公司芯片设计为例
……
entity example_ent is
port(
……
      clk_in : in std_logic;
……
);
end example_ent;
……
BUFG_inst : BUFG
port map(O => clk, --将输入时钟连接至全局时钟网络后再使用
          I => clk_in );
……
```

5.9.8 不宜使用行波时钟。

行波时钟在行波链上各触发器时钟之间会产生较大的时钟偏移，降低系统的实际速度，并且有可能超出最坏情况下的建立时间、保持时间，设计中不宜使用。

示例：

--违背示例

行波时钟使用电路的违背示例的示意图见图 2。

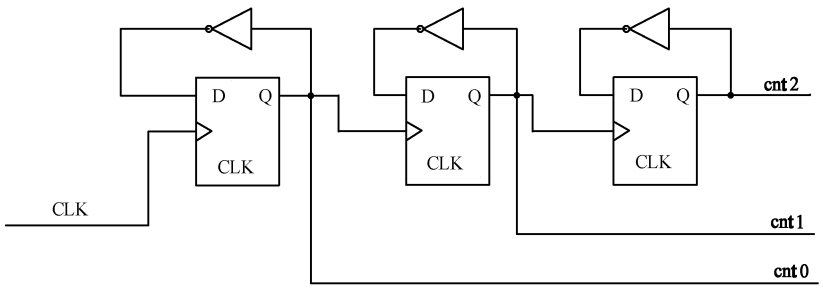

图 2 行波时钟使用电路示意图

--使用前级触发器的输出信号作为后级触发器的时钟信号

5.9.9 在设计中宜使用较少的时钟域，最好只使用一个时钟域。

多时钟域易引起亚稳态、时序分析困难等问题，在设计中宜减少时钟域，最好只使用一个时钟域。

5.9.10 时钟信号源不宜驱动不同时钟沿触发的寄存器。

设计中使用时钟信号的两个变化沿，不利于时序分析，不利于时序同步。

示例：

```
--违背示例
entity top_ent is
port(
      clk : in std_logic;
      rst_n : in std_logic;
      D_in : in std_logic;
      D_out : out std_logic );
end top_ent;

architecture behave_arc of top_ent is
signal S_Dout1_r, S_Dout_temp_r : std_logic;
begin
process (clk, rst_n)
  begin
    if rst_n= '0' then
      S_Dout_temp_r<= 0;
    elsif clk'event and clk='1'then ---此处使用时钟信号上升沿
      S_Dout_temp_r <= D_in;
    end if;
  end process;
process (clk, rst_n)
  begin
    if rst_n= '0' then
      D_out <= 0;
    elsif clk'event and clk='0'then ---此处使用时钟信号下降沿
      D_out <= S_Dout_temp_r;
    end if;
  end process;
end behave_arc;
```

5.9.11 避免在时钟路径上插入反相器或缓冲器。

在时钟路径上插入反相器或缓存器会增加时钟信号的偏移，引起时序错误，并使时序余量变小。

示例：

```
--违背示例
entity top_ent is
port(
      clk : in std_logic;
      reset : in std_logic;
      D_in : in std_logic;
      D_out : out std_logic );
end top_ent;

architecture behave_arc of top_ent is
……
```

```
signal clock_1 : std_Logic;

clock_1 <= not clk;--在时钟路径上使用反相器
INBUF_0_inst : INBUF
port map(PAD => clock_1,
         Y => clock_2 ); --在时钟路径上使用缓存器
process (reset, clock_2)
……
end process;
end behave_arc;
```

5.10 复位及初始化类

5.10.1 不应将寄存器的输出反馈到异步复位(置位)端。

同一个寄存器的输出信号不应经过组合逻辑后再反馈回异步复位(置位)端。

示例:

```
--违背示例
……
U5:GTECH_FD2
Port map (D => in2_in,
          CP => clk,
          CD => rst,
          Q => out1_out );
--触发器 U5 的输出经组合逻辑后又反馈回 U5 复位端。
rst <= in1_i AND out1_out;
……
```

5.10.2 同一寄存器的复位与置位信号不应来自同一个信号。

使用同一个信号作为寄存器的置位与复位信号会导致竞争与冒险,从而导致数据错误。

5.10.3 异步复位释放时应采用同步释放的方式。

在异步复位释放时应采用同步释放的方式,避免异步复位释放时的亚稳态。

示例:

```
--违背示例
……
Process (clk, rst_n) --复位信号 rst_n 与时钟 clk 为异步关系
begin
  if (rst_n = '0') then -- rst_n 释放时寄存器 S_test 存在亚稳态风险
    S_test <= "00";
……
```

```
--遵循示例
……
process (clk, rst_n)
begin
  if(rst_n = '0') then
    rst_n_syn0 <= '0';
    rst_n_syn1 <= '0';
  elsif clk'event and clk = '1' then --对异步复位信号进行同步释放处理
    rst_n_syn0 <= 1;
    rst_n_syn1 <= rst_n_syn0;
  end if;
end process;
……
process (clk, rst_n_syn1) --使用可以同步释放的复位信号进行复位
begin
  if(rst_n_syn1 = '0') then
    S_test <= "00";
……
```

5.10.4 寄存器复位端口不应悬空。

示例：

```
--违背示例
……
example_ent_inst : example_ent
port map (clk => clk,
          out_1 => out_1 ); --复位端口被悬空
……
entity example_ent is
port(
      clk : in std_logic;
      rst : in std_logic;
      out_1 : out std_logic );
end example_ent;
……
```

5.10.5 应使用复位/置位操作对寄存器进行初始化。

声明初始化方式对某些器件型号，综合工具是无法综合的。对寄存器的初始化不宜采用声明初始化方式，应使用复位/置位操作初始化。如存在信号无法通过复位/置位操作初始化，且确认使用的器件可以声明初始化时，可采用声明初始化方式。

示例：

```
--违背示例
……
signal S_test : std_logic_vector(1 downto 0):= "10";--声明初始化可能无法综合
……
```

```
--遵循示例
……
process (clk, rst_n)
begin
  if(rst_n = '0') then
  S_test <= "10"; --使用复位操作对寄存器进行初始化
……
```

5.10.6 在同一时刻宜使用同一个复位域对所有模块进行复位。

在同一时刻内存在多个复位域可能混淆设计的初始状态,引起信号的冲突。

5.10.7 宜将系统复位信号设置为全局网络。

将系统复位信号设置为全局时钟网络,可以保证复位信号延迟最小,使所有触发器同时进行全局复位。

示例:

```
--遵循示例
……
BUFG_inst : BUFG --将复位信号连接至全局时钟网络
port map (O => g_rst_n,
          I => rst_n);
……
process (clk, g_rst_n) --使用经过全局时钟网络后的复位信号
begin
  if (g_rst_n = '0') then
……
```

5.10.8 不宜用时钟信号做复位(置位)操作。

在同步电路中,不宜将时钟信号作为复位(置位)信号使用。

5.10.9 同步复位时,外部复位信号宜使用两级级联寄存器同步。

同步复位时,为满足复位及复位释放时的时序要求,宜使用与目标寄存器同一时钟域相同时钟沿的两级级联寄存器同步外部复位信号。

示例:

```
--遵循示例
……
process (clk) --复位信号 rst_n 在时钟域 clk 下两级同步
begin
  if clk'event and clk = '1' then
    rst_n_syn0 <= rst_n;
    rst_n_syn1 <= rst_n_syn0;
  end if;
end process;
……
process (clk) --使用同步之后的复位信号进行同步复位
begin
  if clk'event and clk <= '1' then
    if(rst_n_syn1 = '0') then
      S_test <= "00";
……
```

5.10.10　同一个复位(置位)信号宜使用唯一的有效电平。

在设计中使用复位(置位)信号的两个有效电平,不能保证各模块同时复位(置位)。

示例:

--遵循示例

使用同一电平复位的示意图见图 3。

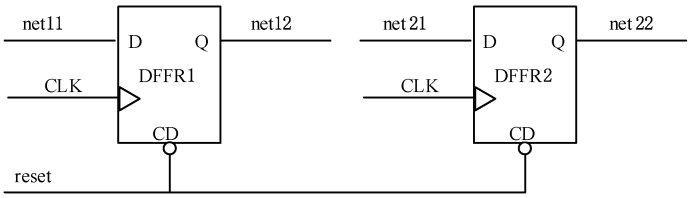

图 3　使用统一电平复位

--违背示例

使用不同电平复位的示意图见图 4。

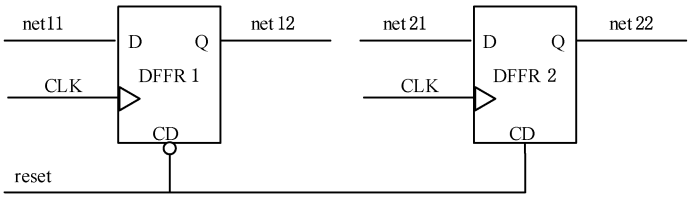

图 4　使用不同电平复位

5.10.11　组合逻辑作为复位信号时应该被同步。

组合逻辑作为复位信号时应该被同步,避免组合逻辑产生毛刺导致误复位。

示例:

```
--遵循示例
……
rst_n <= rst_a and rst_b; --复位信号由组合逻辑产生

process (clk)
begin
  if clk'event and clk = '1' then --对复位信号进行同步化处理,消除毛刺
    rst_n_syn0 <= rst_n;
    rst_n_syn1 <= rst_n_syn0;
  end if;
end process;
process (clk,rst_n_syn1) --使用同步处理后的复位信号进行复位
begin
  if(rst_n_syn1 = '0') then
……
```

5.11 状态机类

5.11.1 对状态机中的无效状态应进行合理处理。

为避免状态机死锁，对于状态机中无效的状态应进行合理的处理。

示例：

```
--违背示例
--程序中状态机进入无效状态后置空操作，可能导致状态机死锁
singal current_state_r, next_state_r : std_logic_vector (1 downto 0);
constant Cst0 : std_logic_vector (1 downto 0) := "00" ;
constant Cst1 : std_logic_vector (1 downto 0) := "01" ;
constant Cst2 : std_logic_vector (1 downto 0) := "10" ;
constant Cst3 : std_logic_vector (1 downto 0) := "11" ;
……
case current_state_r is
  when Cst0 =>
……
  when Cst1 =>
……
  when Cst2 =>
……
--当程序的状态机进入无效状态后执行 when others 分支，由于程序
--中未对无效状态进行任何处理，状态机一旦进入无效状态则无法再返回到正常的状
--态，导致状态机死锁。
  when others => null;
end case
```

```
--遵循示例
singal current_state_r, next_state_r : std_logic_vector (1 downto 0);
constant Cst0 : std_logic_vector (1 downto 0) := "00" ;
constant Cst1 : std_logic_vector (1 downto 0) := "01" ;
constant Cst2 : std_logic_vector (1 downto 0) := "10" ;
constant Cst3 : std_logic_vector (1 downto 0) := "11" ;
……
case current_state_r is
  when Cst0 =>
……
  when Cst1 =>
……
  when Cst2 =>
……
--进入无效状态后返回到初始状态。
  when others => next_state_r <= Cst0;
end case
```

5.11.2 应对状态机综合属性进行适当的设置，确保对无效状态的处理综合有效。

应对状态机综合属性进行适当的设置，确保状态机综合后包含对无效状态的处理，可通过在综合设置中通过勾选安全模式或代码中增加综合属性约束等方式进行设置。

示例：

```
--遵循示例
--使用 Synplify 综合工具时，可以在代码中增加综合属性设置
type state is (s0, s1, s2, s3, s4, s5);
signal nstate, cstate : state;
attribute syn_encoding of cstate : signal is "safe, gray";
```

5.11.3 状态机应初始化为合法状态。

应使用全局复位信号使状态机初始化为已知的合法状态。

示例：

```
--遵循示例
process (rst_n, clk)
begin
  if rst_n = '0' then
    current_state_r <= IDLE; --初始化为 IDLE 状态
  elsif rising_edge(clk) then
  ……
  end if;
end process;
```

5.11.4 宜使用三段式状态机。

状态机宜使用三段式描述，使用三段式描述状态机可提高可读性和规范性。

示例：

```
--遵循示例
……
--第一个进程，同步时序进程，描述次态寄存器迁移到现态寄存器
process (rst_n, clk)
begin
  if rst_n = '0' then
    current_state_r <= IDLE;
  elsif rising_edge(clk) then
    current_state_r <= next_state_r;
  end if;
end process;
……
--第二个进程，组合逻辑进程，描述状态转移条件判断
process (current_state_r, AD_start,…)
begin
   next_state_r <= IDLE;
   case current_state_r is
     when IDLE =>
```

```
        if AD_start= '1' then
          next_state_r <= STATE_1;
        else
          next_state_r <= IDLE;
        end if;
      when STATE_1=>
        next_state_r <= STATE_2;
        ……
       when others=>
        next_state_r <= IDLE;
      end case;
end process;
--第三个进程,同步时序进程,描述次态寄存器输出
process (rst_n, clk)
begin
  if rst_n = '0' then
    AD_CS <= '1';
    AD_RD <= '1';
    AD_DONE <= '0';
    ……
  elsif rising_edge(clk) then
    case next_state_r is
    when IDLE =>
      AD_CS <= '1';
      AD_RD <= '1';
      AD_DONE <= '0';
      ……
      when STATE_1 =>
      AD_CS <= '0';
      ……
    end case;
  end if;
end process;
```

5.11.5 宜使用枚举类型来声明状态机。

示例：

```
--遵循示例
--枚举类型定义状态机
type state_values is (IDLE, Conv_Start, Config_Start, Write_Parameter);
signal state, next_state : state_values;
```

5.11.6 状态机中的状态数量及编码方式要求：

a) 有限状态机中状态的数量过大会产生较复杂的组合逻辑电路,使时序路径上的传输延时增大,状态机的数量一般应少于 40；

b) 二进制编码需要的寄存器较少，但对状态机译码时，需要比较多的组合逻辑，在面积重要时，宜采用二进制编码；

c) 独热编码需要较多的寄存器，但状态机之间需要的组合逻辑资源比较少，能够提高逻辑的速度，在速度重要时，宜采用独热码编码；

d) 格雷编码需要较少的寄存器，但形成的组合逻辑比较复杂，格雷码每个相邻的状态的切换只有一个比特的信号跳变，速度较快，在组合逻辑资源较多，且状态跳转顺序确定时，宜采用格雷编码。

示例：

```
--遵循示例
--采用二进制编码
type state_values IS (s0, s1, s2, s3, s4);
SIGNAL nstate, cstate : state_values;
attribute syn_encoding of cstate : signal is "safe,sequential"
```

```
--遵循示例
--采用独热码编码
type state_values IS (s0, s1, s2, s3, s4);
SIGNAL nstate,cstate : state_values;
attribute syn_encoding of cstate : signal is "safe, onehot";
```

```
--遵循示例
--采用格雷码编码
type state_values IS (s0, s1, s2, s3, s4);
SIGNAL nstate, cstate : state_values;
attribute syn_encoding of cstate : signal is "safe, gray";
```

5.12 综合/约束类

5.12.1 在用于综合的代码中应避免使用不可综合的代码。

示例：

```
--空字符串常量不可综合
……
architecture DCVHDL_2159_arch of DCVHDL_2159 is
--违背示例
constant N : string := ""; -- 错误
--遵循示例
constant B : integer := 4; -- 正确
begin
……
end DCVHDL_2159_arch;
```

```
--非整型 generic 不可综合
--违背示例
……
entity DCVHDL_2024 is
generic(delay : string := "temp"); --错误
port(
      B : in std_logic;
      A : out std_logic_vector (1 downto 0));
end DCVHDL_2024;
--遵循示例
……
entity ex175 is
generic (n : integer := 8); ---正确
port(
      A : in std_logic_vector (n-2 downto 0);
      D : in std_logic;
      C : out std_logic;
      B : out std_logic_vector (n-1 downto 0));
end ex175;
```

5.12.2 在进行约束设置时,应避免不同的约束间出现矛盾。

顶层时序约束和模块时序约束应保持一致。这些时序约束一般包括:时钟约束、I/O 延迟约束、伪路径约束、多周期路径约束、最大/最小延迟约束等。

示例:

```
--违背示例
attribute syn_preserve : boolean;
attribute syn_preserve of current_state : signal is true --保护信号 current_state 不被优化
attribute syn_encoding of current_state : signal is "safe, gray";--状态机安全模式设置
--不同约束之间出现矛盾,导致状态机安全模式设置约束失效
```

5.12.3 应避免使用 buffer 类型的端口。

示例:

```
--违背示例
entity example_ent is
port(
      in_1 : in std_logic;
……
      out_1 : buffer std_logic ); --应避免使用 buffer 类型
end example_ent;
```

5.12.4 应在约束文件中对设计的每个时钟进行定义并添加周期约束。

应在约束文件中对设计的每个时钟进行定义,在对时钟进行定义的同时应设置其时钟周期。

示例：

```
--违背示例
--该示例的约束文件仅对 clk_1 进行了定义和周期约束，没有对 clk_2 进行定义和周期约束，工具无法对 clk2 时序路径的符合性进行确认
--约束文件
create_clock －name clk_1 －period 20 [get_pins clk_1]
--设计文件
entity clk_ent is
port(
      clk_1 ：in std_logic；
      clk_2 ：in std_logic；
      …… )；
end clk_ent；
architecture behave_arc of clk_ent is
……
process (clk_1,……)
begin
  ……
end process；
process (clk_2, ……)
begin
  ……
end process；
end behave_arc；
```

5.13 注释类

5.13.1 文件开头宜添加注释。

注释应使用英文书写，注释内容包括研制单位、作者、创建日期、工程名、模块名、版本号、所属项目、测试程序名、目标器件、软件工具及版本号、功能简介、模块结构、修改记录等。

示例：

```
--遵循示例
IEEE STD 1076—2008 VHDL file：CPU_TOP.vhd
Description：XX-3 CPU TOP control
Company：FPGA Panel，XXX
Author：mary
Project name：CPU_ctrl
Version：V1.00
Date：2015-2-20
FPGA type：actel APA300
Development tools：libero V9.0
Revision history：
Revision 1 2015-3-9 10：33：59 mary
Delete "rd_reg" signal
Revision 2 2015-4-8 11：07：23 mary
```

```
Add synthesize restriction: synthesis syn_encoding = "safe"
Add state default restriction
Copyright(C) 2015,XXX Institute,All rights reserved
……
```

5.13.2 端口定义的输入输出信号宜添加注释说明,注释宜靠近信号定义。

5.13.3 内部信号、变量宜添加注释说明,注释宜靠近信号定义。

5.14 编码格式类

5.14.1 宜使用空行进行适当的分隔,具体要求如下:

a) 代码书写应分节,各节之间宜采用一或多个空行分隔;

b) 功能集中或相关性较强的变量或信号的声明宜放在一起,用空行或注释分开。

5.14.2 程序编写时应注意对齐和缩进,具体要求如下:

a) 行首用空格键对语句进行对齐和缩进;

b) 行尾不宜有多余空格;

c) 同一个层次的所有语句左端对齐,不同层次之间的语句宜使用空格键缩进;

d) 嵌套语句宜逐层缩进对齐;

e) 端口、信号、变量定义应对齐;

f) 模块例化时,端口映射应对齐。

5.14.3 应使用空格进行适当的分隔,具体要求如下:

a) 不同变量、变量与符号、变量与括号之间宜保留一个空格;

b) 赋值符号两边宜保留一个空格;

c) 运算符两边宜保留一个空格。

5.14.4 其他要求如下:

a) 一行代码字符数不宜超过 72 个;

b) 同一行代码中不宜有多个 HDL 语句;

c) 调用模块进行端口映射时,一行代码只应映射一个信号;

d) 文件的引用应使用相对路径而不是绝对路径。

参 考 文 献

[1] IEEE Std 1076—2008 IEEE VHDL 语言标准参考手册(IEEE Standard VHDL Language Reference Manual)