VHDL代码书写规范

目　　次

1、目的 . . . 1
2、范围 . . . 1
3、定义 . . . 1
4、引用标准和参考资料 . . . 1
5、规范内容 . . . 1
5.1 VHDL编码风格 . . . 1
5.1.1 标识符（Identifiers)命名习惯 . . . 2
5.1.1.1标识符定义命名规定 . . . 2
5.1.1.2标识符大小写规定 . . . 2
5.1.1.3 信号名连贯缩写的规定 . . . 2
5.1.1.4 信号名缩写的大小写规定 . . . 2
5.1.1.5 信号名一致性规定 . . . 2
5.1.1.6 信号命名有关建议 . . . 3
5.1.2数据对象和类型 . . . 3
5.1.2.1类型使用规定 . . . 3
5.1.2.2 数据及数据类型使用建议 . . . 3
5.1.2.3 数据使用注意内容 . . . 4
5.1.3 信号和变量 . . . 4
5.1.3.1 信号不许赋初值。 . . . 4
5.1.3.2 变量使用建议 . . . 4
5.1.3.3 信号、变量使用注意内容 . . . 4
5.1.4 实体 . . . 4
5.1.4.1 实体、结构体使用规定 . . . 4
5.1.4.2 实体使用建议 . . . 6
5.1.4.3 实体使用注意内容 . . . 6
5.1.5 语句 . . . 7
5.1.5.1 VHDL各语句使用规定 . . . 7
5.1.5.2 VHDL 语句使用建议 . . . 10
5.1.5.3 VHDL 语句使用注意内容 . . . 11
5.1.6 运算符(operator) . . . 11
5.1.6.1 表达式书写规定 . . . 11
5.1.6.2比较运算符规定 . . . 11
5.1.7 function . . . 11
5.1.7.1 function 使用规定 . . . 11
5.1.7.2 function 使用建议 . . . 11
5.1.7.3 function 使用注意内容 . . . 11
5.1.8 procedure . . . 12
5.1.8.1 procedure 使用规定 . . . 12
5.1.8.2 procedure 使用注意内容 . . . 12
5.1.9 类属(generics) . . . 12
5.1.9.1 generic 使用注意内容 . . . 12
5.1.10 package . . . 12
5.1.10.1 package 使用建议 . . . 12
5.1.10.2package 使用注意内容 . . . 13
5.1.11 FSM（有限状态机） . . . 13
5.1.11.1 FSM 使用规定 . . . 13
5.1.12 Comments . . . 13

5.1.12.1 Comments 使用建议 . . . 13
5.1.13 TAB键间隔 . . . 13
5.1 代码模块划分 . . . 13
5.2 代码编写中容易出现的问题 . . . 14
5.2.1 资源共享问题 . . . 14
5.2.2组合逻辑描述的多种方式 . . . 14
5.2.3 考虑综合的执行时间 . . . 15
5.2.4 避免使用Latch . . . 15
5.2.5 多赋值语句案例：三态总线 . . . 15
6、附录 . . . 16
6.1 VHDL保留字 . . . 17
6.2 VHDL 编写范例 . . . 17
6.3 函数书写实例 . . . 19
6.4 程序包书写实例 . . . 20
6.5 参数化元件实例 . . . 21

1、目的

本规范的目的是提高书写代码的可读性、可修改性、可重用性，优化代码综合和仿真的结果，指导设计工程师使用VHDL规范代码和优化电路，规范化公司的ASIC/FPGA设计输入，从而做到：① 逻辑功能正确，②可快速仿真，③ 综合结果最优，④ 可读性较好。

2、范围

本规范涉及VHDL编码风格、规定，编码中应注意的问题，VHDL代码书写范例等。

本规范适用于所有的采用VHDL代码进行设计的项目。

3、定义

VHDL：Very high speed IC Hardware Description Language, 甚高速集成电路的硬件描述语言

FSM：Finite Status Machine,有限状态机

simulate：仿真，通过输入激励在计算机上验证设计是否正确。包括RTL仿真和门级仿真。

模拟：是指对一个物理器件的结构、功能或其他特性如延时特性等用抽象的语言或高级语言（如用C语言进行算法描述）所进行的建模。

4、引用标准和参考资料

下列标准包含的条文，通过在本标准中引用而构成本标准的条文。在标准出版时，所示版本均为有效。所有标准都会被修订，使用本标准的 各方应探讨使用下列标准最新版本的可能性。

VHDL For Programmable Logic	Mr,Kevin Shahill USA

5、规范内容

以下内容中，有关的保留字用黑体标识。对不作为审核的内容用“建议”字眼标识。

5.1 VHDL编码风格

本章节中提到的VHDL编码规则和建议适应于 VHDL的任何一级（RTL，behavioral, gate_level)，也适用于出于仿真，综合或二者结合的目的而设计的模块

5.1.1 标识符（Identifiers)命名习惯

标识符用于定义实体名、结构体名、信号和变量名等，选择有意义的命名对设计是十分重要的。命名包含信号或变量诸如出处、有效状态等基本含义，下面给出一些命名的规则，包括VHDL语言的保留字。

5.1.1.1标识符定义命名规定

- 标识符第一个字符必须是字母，最后一个字符不能是下划线，不许出现连续两个下划线。
- 基本标识符只能由字母、数字和 下划线组成。
- 标识符两词之间须用下划线连接。

  如： *Packet_addr,*　*Data_in,*　*Mem_wr,*　*Mem_ce*
- 标识符不得与保留字同名，VHDL保留字见附录6.1。

5.1.1.2标识符大小写规定

- 对常量、数据类型、实体名和结构体名采用全部大写；
- 对变量采用小写；
- 对信号采用第一个词首字符大写。
- 保留字一律小写。

5.1.1.3 信号名连贯缩写的规定

长的名字对书写和记忆会带来不便，甚至带来错误。采用缩写时应注意同一信号在模块中的一致性。一致性的缩写习惯有利于文件的阅读理解和交流。

部分缩写的统一规定为：

Addr　address；　Clk　clock；　Clr　clear；　Cnt　counter

En　enable；　Inc　increase；　Lch　latch；　Mem　memory

Pntr　pointer；　Pst　preset　；　Rst　reset；

Reg　register；　Rd　reader；　；　Wr　write

常用多个单词的缩写：

ROM　RAM　CPU　FIFO　ALU　CS　CE

自定义的缩写必须在文件头注释。

5.1.1.4 信号名缩写的大小写规定

- 单词的缩写若是信号名的第一个单词则首字符大写，如：Addr_in中的Addr 。若该单词缩写不是第一 个单词则小写， 如：Addr_en 中的en 。
- 多个单词的首字符缩写都大写，不管该缩写在标识符的什么位置， 如：RAM_addr ，Rd_CPU_en。

5.1.1.5 信号名一致性规定

同一信号在不同层次应保持一致性。

5.1.1.6 信号命名有关建议

- 建议用有意义而有效的名字，能简单包含该信号的全部或部分信息，如输入输出信息：Data_in （总线数据输入）、Din（单根数据线输入）、FIFO_out （FIFO数据总线输出）；如宽度信息：Cnt8_q （8位计数器输出信号的命名）。
- 建议添加有意义的后缀，使信号名更加明确，常用的后缀如下：

后缀	意义
_clk	时钟信号
_d	寄存器的数据输入信号
_q	寄存器的数据输出信号
_z	连到三态输出的信号
_L	_L后加数字，表示信号延迟时钟周期数，如_L1
_s	实体端口信号的反馈信号
_en	使能控制信号
_n	求反的信号
_xi	芯片原始输入信号
_xo	芯片原始输出信号
_xod	芯片的漏极开路输出
_xz	芯片的三态输出
_xbio	芯片的双向信号

说明：

1. 采用D触发器对信号进行延迟，延迟信号的命名在原信号名之后加后缀_L,若是在流水线设计中有级延迟，再分别加后缀_L1，_L2，..... 。 L 表示lock。

2. 模块内的反馈信号，在原信号名之后加后缀_s 。 “s” 表示 same。

5.1.2数据对象和类型

5.1.2.1类型使用规定

- VHDL是很强的类型语言，可综合的数据类型为标量类型（包含可枚举类型、整型、浮点型、物理类型）和组合类型（包含记录、数组），模拟模型的数据类型为存取类型、文件型。可综合的VHDL代码的编写不采用模拟类型、浮点型、物理类型。
- 不同基本类型的数据不能由另一类型赋值，不同类型间的赋值需使用运算符的重载。如：Cnt8_q 为STD_LOGIC_VECTOR 类型，若不对 ‘+’ 运算符重载，则 Cnt8_q <= Cnt8_q + 1 语句在综合中将出错。可通过对 + 运算符进行重载 即使用**use IEEE**.std_logic_arith.**all** 语句，则上句赋值语句是正确的 。
- 常量名和数据类型必须用大写标识符表示。

5.1.2.2 数据及数据类型使用建议

- 为改善代码的可读性，建议可把常用的常量和自定义的数据类型在程序包中定义。
- 建议使用别名来标识一组数据类型有利于代码的清晰。如：

  signal Addr ： STD_LOGIC_VECTOR(31 **downto** 0);

  alias Top_addr ： STD_LOGIC_VECTOR(3 **downto** 0) **is** Addr(31 **downto** 28);

5.1.2.3 数据使用注意内容

可枚举类型的值为标识符或单个字母的字面量，是区分大小写的，如 Z 与z 将是两个不同的量。

5.1.3 信号和变量

5.1.3.1 信号不许赋初值。

5.1.3.2 变量使用建议

变量主要用在高层次的模拟模型建模及用于运算的用途，但变量的综合较难定义，对于编写可综合的VHDL模块，在没有把握综合结果情况下建议不使用。

5.1.3.3 信号、变量使用注意内容

- 在VHDL中，信号(**signal**)代表硬件连线，因此可以是逻辑门的输入输出，同时，信号也可表达存贮元件的状态。端口也是信号。
- 在进程（**process**）中，信号是在进程结束时被赋值。因此，在一个进程中，当一个信号被多个信号所赋值时，只有最后一个赋值语句起作用。如下例：

```
Sig_p： process(A，B，C）
    begin
        D <= A；----- ignored!!
        X <= C or D;
        D <= B；---- overrids !!
        Y <= C xor D;
    end;
```

上面实际的结果是 B赋值给D， （C xor B）结果赋值给X、Y。

- 变量不能表达连线或存贮元件。变量的赋值是直接的、非预设的。变量将保持其值直到对它重新赋值。如下例：

```
Ver_p：process(A，B，C）
        variable d ： STD_LOGIC;
    begin
        d ： = A;
        X <= C or d;
        d ： = B;
        Y <= C xor d;
    end process ;
```

实际结果是 X <= C **or** A，Y<= C **xor** B。

5.1.4 实体

5.1.4.1 实体、结构体使用规定

- **library IEEE**；**use IEEE**.std_logic_1164.**all**； 除**IEEE**大写外，其余小写。

- 实体名和结构体名必须用大写标识，实体名必须与文件名同名。自定义的其他标识符如信号名、变量名、标号等不得与实体名、结构体名同名。
- 实体端口数据模式不准使用**buffer** 模式

缓冲模式主要用在实体内部可读的端口，如计数器的输出 。为简化大型设计各模块间接口的配合，要求不要使用。需要反馈的信号可定义内部信号来解决。如计数器端口Count ，可内部定义信号 **signal** Cnt8_q ： STD_LOGIC_VECTOR(7 **downto** 0) ;

Cnt8_q 该信号可在内部反馈，最后通过 赋值语句：

Count <= Cnt8_q ; 来实现端口的定义。

- 实体端口数据类型规定

实体端口的数据类型采用IEEE std_logic_1164 标准支持的和提供的最适合于综合的数据类型STD_ULOGIC、STD_LOGIC和这些类型的数组。不采用**IEEE** 1076 /93 标准支持和提供的BIT 、BIT数组、INTEGER及其派生类型。这是为保证模拟模型和综合模型的一致性及减少转换时间和错误。

- 一个文件只对应一个实体。

实体是设计文件的基本单元，其书写规范要求如下：

- 一条语句占用一行，每行应限制在80个字符以内。
- 如果较长（超出80个字符）则要换行。
- 代码书写要有层次即层层缩进格式、清晰、美观。
- 要有必要的注释（25%）
- 实体开始处应注明文件名、功能描述、引用模块、设计者、设计时间及版权信息等。

如：

```
-- Filename ：
-- Author ：
-- Description ：
-- Called by ：Top module
-- Revision History ：99-08-01
--                  Revision 1.0
-- Email ： M@swip.com.cn
--Company ： swip Technologies .Inc
--Copyright(c) 1999, swip Technologies Inc, All right reserved

library IEEE;
use IEEE.std_logic_1164.all;
entity ENTITY_NAME
port(
    Port1 ： in STD_LOGIC;
    Port2 ： in STD_LOGIC;
```

```
    Port3  : out   STD_LOGIC;
    ..
    Portn : out  STD_LOGIC
);
end ENTITY_NAME ;
architecture BEHAVIOR  of  ENTITY_NAME  is
begin
        Statements;
end  BEHAVIOR ;
```

5.1.4.2 实体使用建议

- 实体名的命名建议能大致反映该实体的功能如：COUNTER8（8位宽的计数器模块），DECODER38（3-8线译码器模块）。
- 行为级的结构体名命名为 BEHAVIOR，结构级的结构体名命名为 STRUCTURE，若有多个结构体，用后缀A、B..... 命名。如：

```
architecture  BEHAVIOR of COUNTER8 is
begin
    .....
end  BEHAVIOR;
```

- 一个实体可以有多个结构体，对单个结构体的实体，文件要包含结构体和实体说明，便于查阅。对多个结构体的实体，建议把常用的结构体放在文件中，其余结构体用单独文件表示，使用时用**configuration** 语句进行配置。
- 结构体的描述分为行为级描述、数据流描述和结构化描述。若无特殊要求，建议采用行为级描述和数据流的描述，不采用结构化描述或**BOOLEAN** 数据流的描述。
- VHDL设计中，如果可以避免采用器件厂商的专用元件库（硬Core），则尽量不要使用，除非只有采用该库元件才能实现你设计的性能指标。这是因为要充分利用VHDL独立于工艺且易于维护的优点。

5.1.4.3 实体使用注意内容

- VHDL设计应是层级型的设计。VHDL设计实体由实体说明和结构体组合而成。实体是一个设计的基本单元模块。即顶层的设计模块由次一级的实体构成，每个次一级实体又可由再下一层次的实体构成。最低层模块可以是表达式或最基本的实体模块构成。这种设计方法就是Top-To-down 的设计方法。
- 实体端口模式为 **in**，**out**，**buffer**，**inout**。模式为**in** 的信号不能被驱动，模式**out**的信号不能用于反馈，同时必须仅被一个信号所驱动；缓冲模式的端口不能被多重驱动（除非用决断函数解决外）同时仅可以连接内部信号或另一个实体的缓冲模式的某个端口。
- VHDL设计各模块接口定义时要考虑模块间配合的方便，如实体端口的模式，端口的数据类型等。

5.1.5 语句

5.1.5.1 VHDL各语句使用规定

- **with-select-when** 语句书写规范规定

with- select - when 语句提供选择信号赋值，是根据选定信号的值对信号赋值。代码的书写规范为：

```
with selection_signal select
        Select_name <= value_a when value_1_of_selection_signal，
                       value_b when value_2_of_selection_signal，
                       value_c when value_3_of_selection_signal，
                       .....
                       value_x when last_value_of_selection_signal;
```

- **with- select - when** 语句的selection_signal的所有值必须具备完整性，若没写完整必须有一个 **others** 语句，如下三个写法，其综合的效果是一致的，因为S的元素不是已知的逻辑值，X将不被定义，但对RTL仿真而言，其结果是不一致的，这是因为RTL仿真支持多值元素。

```
with S select
        X <= A when “00”,
             B when “01”,
             C when “10”,
             D when others;

with S select
        X <= A when “00”,
             B when “01”,
             C when “10”,
             D when “11”,
             D when others;

with S select
        X <= A when “00”,
             B when “01”,
             C when “10”,
             D when “11”,
             “--” when others;
```

建议不使用第三种写方法。

- **with- select - when** 语句中对有相同的支项可合并书写 如X <= A when “00” | “10”。
- **when_else** 语句书写规范规定

when_else 语句提供为条件信号赋值，即一个信号根据条件被赋一值，代码书写规范为：

```
Signal_name <= value_a when condition1 else
               value_b when condition2 else
               value_c when condition3 else
                .....
               value_x;
```

- 当条件是表达式时，表达式须用（）括起来，使代码更为清晰。如：

  when （a = b **and** C= ‘1’）**else**

- **if** 必须有一个**else** 对应（除在如下面例子的情况下可不写**else**语句）。

  例：

```
process( Clk,Rst)
begin
    if ( Rst = '1') then
        Q <= '0';
    elsif ( Clk ‘event and Clk = ‘1’) then
        Q <= D;
    end if;
end process;
```

  当没有**else**语句时，将产生不希望的存储器。

- **case- when** 语句书写规范规定

  该语句用于规定一组根据给定选择信号的值而执行的语句，可用**with-select-when** 语句等效。

  代码的书写规范为：

```
case- selection_signal is
  when value1_of _selection signal =>
        Statements1;
  when value2_of _selection signal =>
        Statements2;
  ....
  when last_value_of _selection signal =>
        Statements x ;
  when others =>
        Statements x;
  end case;
```

- **case_when** 语句必须有 **when others** 支项。
- 若信号在**if -else** 或**case-when** 语句作非完全赋值，必须给定一个缺省值。
- **process** 显示敏感列表必须完整

对有Clk 的 **process** ，不同综合工具有不同的要求，有些只要写Clk和Rst就可，建议根据具体情况简化设计书写。

- 有clk的**process** 的敏感列表中，为方便修改，敏感列表书写规范如下：

```
Lab ： process（Clk, Rst，
                list1，list2,....）
begin
```

- 每个**process** 前须加个lable。
- 不同逻辑功能采用不同的**process** 进程块，把相同功能的放在同一进程中，如触发器组进程块：

```
D_p  : process(Clk,Rst,
               D1,D2,...,Dn)
    begin
      if  (Rst =  '1' ) then
                Q1 <= '0' ;
                Q2 <= '0' ;
                ...
                Qn <= Dn;
      elsif (Clk 'event and Clk = '1') then
                Q1 <= D1;
                Q2 <= D2;
                ...
                Qn <= Dn;
       end if;
    end process;
```

- **generate** 语句书写规范规定

在需要重复生成多个器件如多个器件的重复例化时，使用生成机构可简便代码书写。如下32位总线的三态缓冲器的例化：

```
Gen_lab1: for  I  in 0 to 31 generate
        inst_lab: threestate  port  map(
                        Din => Value(i),
                        Rd  => Rd,
                        Dout => Value_out(i)
                        );
end generate;
```

- 生成机构必须有一个标号，如上的Gen_lab1
- **if -then** 用在生成机构中，不能有**else** 或**elsif** 语句，如下复杂的生成机构语句：

```
G1：for I in 0 to 3 generate
```

```
G2： for  j in 0 to 7 generate
        G3：if (I <1 )  then generate
                Ua : thrst  port map( Val(j),Rd,Val_out(j));
        end generate;
        G4：if (i = 1) then generate .....
```

- **port map** 语句书写规范规定

```
Uxx ： Module_name
 port map （
      port1  => port 1,
      port2  => port2,
       .....
      portn  => port n
);
```

- 为便于阅读，**port map** 采用 名字对应（=>）映射方法。
- **port map** 中总线到总线映射时 （X **downto** Y）要写全。
- 向量采用降序方法即 X **downto** Y 格式，向量有效位顺序的定义为从大数到小数。
- **port map** 的module （设计者自编写的**entity**）名用Uxx标识，cell（如厂家提供的库元件、RAM、Core等）名用Vxx 标识。

5.1.5.2 VHDL 语句使用建议

- 作为可综合的代码编写， ‘-’值建议不用。如下一个代码：

```
with  tmp select
   X  <=   A  when  “1---”,
           B   when “-1--”,
           C   when  “--1-”,
           D  when  “---1”,
            ‘0’   when others ;
```

该代码在RTL级仿真中不会出错，但在综合过程中可能编译出错，视综合工具而定。

- 由于不同综合工具支持能力问题，建议不采用**wait** 语句即不使用隐式敏感表。

5.1.5.3 VHDL 语句使用注意内容

- **when_else** 语句具有优先级，第一个**when** 条件级别最高，最后一个最低，可用顺序语句的**if-else** 替代。书写时必须考虑敏感路径。
- 当信号的值为不相关的值时，最好用选择信号赋值语句，如多路选择器；当信号的值为相关时，选用**when-else** 语句，如编写优先编码器。
- 注意**If --elsif ---elsif ---else**的优先级。最后一个**else** 优先级最低。必须把关键路径放在优先级高的语句中。

5.1.6 运算符(operator)

5.1.6.1 表达式书写规定

为便于理解，用（）表示逻辑运算符执行的优先级。如：

X <= (A **and** B) **and** (C **or** （**not** D）);

建议运算操作符两边都加上空格。如：

5.1.6.2比较运算符规定

向量比较时，比较的向量的位宽要相等，否责会引起warning或error。除非重载等值比较运算符（调用numeric_std 库）。

5.1.7 **function**

5.1.7.1 **function** 使用规定

- **function** 代码书写规范规定

function FUNCTION_NAME (参数1：参数类型，参数2：参数类型 .....）
return 返回类型 **is**
begin
顺序语句;
end ;

实例见附录6.3。

5.1.7.2 **function** 使用建议

- 函数主要用于类型的转换或重载运算符的定义，对于使用IEEE1164标准的面向综合的VHDL设计，采用std_logic 类型，不必考虑与**bit**类型的转换，可调用 numeric_std 标准程序包实现类型转换和 + 运算符的重载。建议少用厂家提供的函数或自定义的类型转换函数。
- 对多次重复的表达式可用一个函数来定义。

5.1.7.3 **function** 使用注意内容

- 函数参数只能是输入类型，不能被赋值修改。
- 只能有一个返回值。
- 定义函数必须为顺序语句，且其中不能定义新的信号，但可在函数说明域中说明新的变量，并在定义域中对其赋值。
- 函数在结构体说明域中或程序包中定义。见附录6.3。

5.1.8 **procedure**

5.1.8.1 **procedure** 使用规定

- **procedure** 书写规范规定

procedure PROCEDURE_NAME （**signal** 参数名： 模式 类型;
signal 参数名： 模式 类型;

...） **is**

begin

过程体；

end procedure ;

5.1.8.2 **procedure** 使用注意内容

- 过程用于数值运算、类型转换、运算符重载或设计元件的最高层设计结构。
- 过程参数缺省模式为 **in**。

5.1.9 类属(**generics**)

5.1.9.1 **generic** 使用注意内容

- 类属为传递给实体的具体元件的一些信息，如器件的上升下降沿的延时信息（对应类属为rise、fall）、用户定义的数据类型如负载信息（对应类属为load）及数据通道宽度、信号宽度等。
- 对大型设计，建议使用类属来构造参数化的元件。其调用的方法为：

  Uxx : 参数化的实体名 **generic map**（实参）

  port map (

  端口映射表；

  ）；

  见附录6.5。
- 若元件的类属在定义时已经指定默认值，在调用时，若不改变该参数值可不用定义实参的映射即**map**（实参）可不写。

5.1.10 **package**

5.1.10.1 **package** 使用建议

- 对大型设计，建议把全局的常量（如数据宽度等）、指令状态编码、元件组、函数和子程序组分别用元素包、器件包、函数包来构造。如通过调用元件程序包，实体的结构体说明区域中就不必再对调用的器件进行**component** 说明。
- 程序包通过**use** 语句使之可见。可通过保留字**all**使包中所有单元都可见。如：

  use work..yourpacketname.**all**;

  其中yourpacketname是你的packet名。

5.1.10.2**package** 使用注意内容

- 程序包包括程序包说明和可选包体。程序包说明用来声明包中的 类型、元件、函数和子程序；包体则用来存放说明中的函数和子程序。
- 不含有子程序和函数的程序包不需要包体。（见附录6.4）
- 程序包中的类型、常量、元件、函数和其他说明对其他设计单元是可见的。

5.1.11 FSM（有限状态机）

5.1.11.1 FSM 使用规定

- VHDL 的FSM为双进程的有限状态机，即组合逻辑进程定义次态的取值，时序逻辑进程定义时钟上升沿来时次态成为现态。
- VHDL的FSM 不必为状态分配，而用行为级的枚举类型定义状态，根据需要在综合时选择状态的分配方式如 one-hot 或二进制编码。
- 状态机在上电时必须明确进入一个初始状态。
- 必须包括对所有状态都处理，不能出现无法处理的状态，不能使状态机进入死循环。

5.1.12 Comments

5.1.12.1 Comments 使用建议

- 对更新的内容尽量要做注释。
- 模块端口信号要做简要的功能描述。
- 语法块做简要介绍。

5.1.13 TAB键间隔

对TAB键的间隔，我们建议采用4个字符，这与许多软件的缺省设置是一致的。并且，VHDL语言中大多数保留字也是4个字符。

5.1 代码模块划分

模块设计的好坏直接影响着系统的设计好坏，模块设计的不好，会给后面的设计流程带来许多麻烦。设计模块的基本原则是：

1.有利于模块的可重用性

模块设计得好，可节省大量的重复工作，并且为以后的设计带来方便。

2.在组合电路设计中应当没有层次

可提高代码的可读性，另外一方面是综合的时候方便，并且时序较易满足。

3.每个模块输出尽量采用寄存器输出形式

这样设计是有利于时序的满足。

4.模块的按功能进行划分，划分要合理

5.模块大小应适中，不能太大，也不能太小，一般为2000门左右。具体情况，应当依据综合工具的性能而定。

6.模块的层次应当至少有三级

可将一个设计划分为三个层次：TOP、MID、功能CORE

•TOP

包括实例化的MID和输入输出定义（如果用综合工具插入管脚则可不要此层次）；

•MID

由两部分组成：1）时钟产生电路，如分频电路和倍频电路；2）功能CORE的实例化。

•功能CORE

包括各种功能电路的设计。一个复杂的功能可以分成多个子功能来实现，即再划分子层。

5.2 代码编写中容易出现的问题

5.2.1 资源共享问题

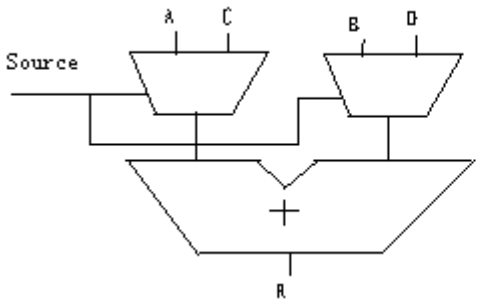
资源共享的主要思想是通过数据缓冲或多路选择的方法来共享数据通路的工作单元。在VHDL设计中资源共享必须与敏感路径问题进行综合考虑后采用适当的设计方法。

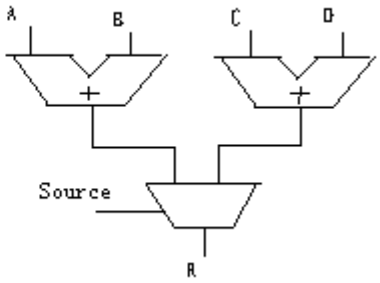
如：

```
R <= ( A + B) when (Source = '1') else
       ( C + D) ;
```

综合工具可综合成以下两种情况，当在综合工具选 中Resource Sharing 时，将资源共享一个加法器。这时，A、B、C或D到R是关键路径。如果关键路径是Source 到R，则不要要求资源共享。需要资源共享时可改成如下写法（建议采用该方式）：

```
R <= A when (Source = '1') else
     C;
S <= B when (Source = '1') else
     D;
F <= R + S;
```





5.2.2组合逻辑描述的多种方式

对组合逻辑的描述有多种方式，其综合结果是等效的。以4bit的与门为例：

*C <= A **and** B;*

等效于

*C（3） <= A（3） **and** B（3）;*

*C（2） <= A（2） **and** B（2）;*

*C（1） <= A（1） **and** B（1）;*

*C（0） <= A（0） **and** B（0）;*

等效于

```
for I in 3 downto 0 loop
  C（i）< = A（i） and B（i）;
end loop;
```

可以选择简洁的写法.

5.2.3 考虑综合的执行时间

通常会推荐将模块划分得越小越好，事实上要从实际的设计目标、面积和时序要求出发。好的时序规划和合适的约束条件要比电路的大小对综合时间的影响要大。要依照设计的目标来划分模块，对该模块综合约束的scripts也可以集中在该特性上。要选择合适的约束条件，过分的约束将导致漫长的综合时间。最好在设计阶段就做好时序规划，通过综合的约束scripts来满足时序规划。这样就能获得既满足性能的结果，又使得综合时间最省。

5.2.4 避免使用Latch

使用Latch必须有所记录，不希望使用Latch时，应该对将条件赋值语句写全，如在if语句最后加一个**else**，**case** 语句加**others** 。

不完整的**if** 和**case** 语句导致不必要的latch的产生，下面的语句中，DataOut会被综合成锁存器。如果不希望在电路中使用锁存器，它就是错误。

```
process（Cond)
begin
    if (Cond = '1') then
        Data_out <= Data_in；
    end;
end process;
```

5.2.5 多赋值语句案例：三态总线

一根总线上挂多个三态电路时必须用多个进程来表示，如：

```
Tri_p1: process (Sel_a,A)
begin
        if (Sel_a = '1') then
                T <= A;
        else
                T <= 'Z';
        end if;
end process;

Tri_p2: process (Sel_b,B)
begin
        if (Sel_b = '1') then
                T <= B;
        else
```

```
                T <= 'Z';
        end if;
    end process;
```

为什么不能在一个**process**中进行处理呢？如下所示：

```
    Error : process (Sel_a,A,Sel_b,B)
    begin
        if (Sel_a = '1') then
            T <= A;
        else
            T <= 'Z';
        end if;
        if (Sel_b = '1') then
            T <= B;
        else
            T <= 'Z';
        end if;
    end process;
```

这是因为：上述两个**if**语句彼此没有优先级，又由于是对同一个信号（信号T）进行处理，则后一个处理会覆盖前一个处理。正确的做法是将这两个**if** 语句放在两个**process**中进行。

注意：只有三态电路才可以在多个**process**中出现，其它非三态电路若是在多个**process**中出现的话，有的综合工具会报告“短路”错误即多驱动问题，但在语法检查时不一定报错。

6、附录

6.1 VHDL保留字

VHDL语言的保留字如下：

absaccess	***after***	***alias***	***all***	***and***	***architecture***	***array***	***assert***	***attribute***		
begin	***block***	***buffer***	***bus***	***case***	***component***	***configuration***		***constant***		
disconnect		***downto***	***else***	***elsif***	***end***	***entity***	***exit***	***file***	***for***	***function***
generate		***generic***	***group***	***guarded***		***if***	***impure***	***in***	***inertial***	***inout***
is	***label***	***library***	***linkage***	***literal***	***loop***	***map***	***mod***	***nand***	***new***	***next***	***nor***
not	***null***	***of***	***on***	***open***	***or***	***others***	***out***	***package***		***port***	
postponed		***procedure***		***process***	***pure***	***range***	***record***	***register***	***reject***	***rem***	
report	***return***	***rol***	***ror***	***select***	***severity***	***signal***	***shared***	***sla***	***sll***	***sra***	***srl***
subtype	***then***	***to***	***transport***		***type***	***unaffected***		***units***	***until***	***use***	
variable	***wait***	***when***	***while***	***with***	***xnor***	***xor***				

另外，对不同的厂家，也有相应的保留字要求，可参见相应厂家提供的资料。

6.2 VHDL 编写范例

```
-- Filename  : Div5.vhd
-- Author    : zhouzhijian
-- Description  : Five division
-- Called by  : Top module
-- Revision History  : 99-08-01

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity DIV5 is
    port(
            Rst             : in STD_LOGIC;
            Mclk            : in STD_LOGIC;
            Div5_clk        : out STD_LOGIC
    );
end DIV5;
architecture BEHAVIOR of DIV5 is
    signal Cnt3_q    : STD_LOGIC_VECTOR(2 downto 0);
    signal Cnt3_d    : STD_LOGIC_VECTOR(2 downto 0);
    signal Div0      : STD_LOGIC;
    signal Div1      : STD_LOGIC;
begin
    Cnt_pd : process(Cnt3_q)
    begin
            if (Cnt3_q = "100") then
                    Cnt3_d <= "000";
            else
                    Cnt3_d <= Cnt3_q + 1;
            end if;
    end process;

    Cnt_pq : process(Rst,Mclk)
```

```
    begin
        if (Rst = '1') then
            Cnt3_q <= "000";
        elsif (Mclk = '1' and Mclk'event) then
            Cnt3_q <= Cnt3_d;
        end if;
    end process  ;

    Div0_P : process(Rst,Mclk)
    begin
        if (Rst = '1') then
            Div0 <= '1';
        elsif( Mclk = '1' and Mclk'event ) then
            if (Cnt3_q = "100") then
                Div0 <= '1';
            elsif (Cnt3_q = "001") then
                Div0 <= '0';
            else
                Div0 <= Div0;
            end if;
         end if;
    end process ;

    Div1_P : process(Rst,Mclk)
    begin
        if (Rst = '1') then
            Div1 <= '0';
        elsif (Mclk = '0' and Mclk'event) then
            Div1 <= Div0;
        end if;
    end process  ;

    Div_clk <= Div0 or Div1;
end BEHAVIOR;
```

建议用双进程来实现带有复杂的同步复位、置位的寄存器信号设计，如上计数器的写法。对Div0也可类似地书写。

6.3 函数书写实例

以下是一个使用多表决函数的全加器

```
-- Filename  : FullAdd.vhd
-- Author    : suwenbiao
-- Description  : A example of function
-- Called by  : Top module
-- Revision History   : 99-08-01

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity FULLADD is
port(
    A                   : in STD_LOGIC;
    B                   : in STD_LOGIC;
    Carry_in            : in STD_LOGIC;
    Sum                 : out STD_LOGIC;
    Carry_out           : out STD_LOGIC
);
end FULLADD;

architecture  BEHAVIOR of  FULLADD is
    function  Majority (A,B,C : STD_LOGIC)
    return   STD_LOGIC is
    begin
            return (( A and b) or (A and C) or ( B and C ));
    end Majority;
begin
    Sum <= A  xor B  xor  Carry_in;
    Carry_out <= Majority(A,B,Carry_in);
end  BEHAVIOR;
```

6.4 程序包书写实例

```
-- Filename  : My_pkg.vhd
-- Author    : suwenbiao
-- Description  : A example of  Package
-- Called by  : Top module
```

```
-- Revision History  : 00-03-18

library IEEE;
use IEEE.std_logic_1164.all;
package MY_PKG is
    -- Defined constant

    constant NUM64K : integer := 65636;
    constant EXT_RAM_ADD_WIDTH : integer := 16;
    constant HW_NUM : integer := 4;
    constant CELL_OUTPUT_CHANNEL_WIDTH : integer := 5;
    constant INPUT_CHANNEL_WIDTH : integer := ADDRESS_BUS_WIDTH;
    constant QUEUE_WIDTH : integer := 5;
    constant WORD_LENGTH_WIDTH : integer := 6;

--Define subtype
constant QUEUE_OVERFLOAT_WIDTH : integer := 16;

subtype QUEUE_OVERFLOAT_VECTOR is
    STD_LOGIC_VECTOR(QUEUE_OVERFLOAT_WIDTH-1 downto 0);

--Defined Reg group
component Rddf1
port(
    Clk     : in STD_LOGIC;
    Rst     : in STD_LOGIC;
    D       : in STD_LOGIC;
    Q       :out STD_LOGIC
    );
end component;
component Rreg1
port(
    Clk             : in STD_LOGIC;
    Rst             : in STD_LOGIC;
    Load            : in STD_LOGIC;
    D               : in STD_LOGIC;
    Q               :out STD_LOGIC
    );
end component;
component Rreg
    generic(Size    : integer  := 2);
```

```
port(
    Clk             : in STD_LOGIC;
    Rst             : in STD_LOGIC;
    Load            : in STD_LOGIC;
    D               : in STD_LOGIC_VECTOR( Size - 1 downto 0);
    Q               :out STD_LOGIC _VECTOR( Size - 1 downto 0)
    );
end component;
end Reg_pkg;
```

6.5 参数化元件实例

```
-- Filename  ：Reg_group.vhd
-- Author   ：suwenbiao
-- Description  ：A example of  Reg  Group with generic size
-- Called by  ：Top module
-- Revision History   ：00-03-18

library IEEE;
use IEEE.std_logic_1164.all
entity REG_GROUP is
    generic (Size : integer : = 2);
    port(
            Clk     : in STD_LOGIC;
            Rst     : in STD_LOGIC;
            Load    : in STD_LOGIC;
            D       :in STD_LOGIC_VECTOR(Size - 1 downto 0);
            Q       : out STD_LOGIC_VECTOR(Size - 1 downto 0)
    );
end REG_GROUP;
architecture  BEHAVIOR  of REG_GROUP is
begin
    R_p: process(Clk,Rst)
    begin
            if (Rst = '1') then
                    Q <= (others => '0') ;
            elsif(Clk 'event  and Clk = '1') then
```

```
                Q <= D;
            end if;
    end process;
end BEHAVIOR;
```

调用语句：

```
U1： REG_GROUP  map(4)
port map (
    Clk             =>      Clk,
    Rst             =>      Rst,
    Load            =>      Load,
    D               =>      D,
    Q               =>      Q
);
```

则该寄存器数据宽度为4。