HDL 编码风格与编码指南

草案初稿

作者

徐欣

孙广富

Rev. 0.1

June 30，2002

目　　录

第一部分：说明

1.准则的重要程度分三个层次：

好的经验——表明这条规则是一般情况下比较好的经验，在大多数的情况下要遵循，在特殊情况下可以突破这一规则。

推荐——推荐这一规则，在遵循这一规则的条件下，一般不会出现问题；

强烈推荐——表示严格规定，除非出现特别特殊的情况，否则要严格遵守。

2.斜体部分一般表明不按照规则执行，会出现的问题和现象，或一些相关注释。

3．版本及修订工作

姓名	修订	日期	联系方式
徐欣，孙广富	规范的最初发布	2002-6-30	Dr.xuxin@163.net

第二部分：HDL 编码风格

1. 文件头和修订列表

作为好的源代码，其中必须包含所有需要的信息。因此源代码中要包含文件头和修订列表（以获得修改情况）。

1．1 文件头包含以下内容：

· 模块名
· 文件名
· 需要的库
· 模块描述
· 使用的仿真器——其运行平台和版本
· 使用的综合工具，其运行平台和版本
· 作者名字和 e-mail

1．2 修订列表包含以下内容：

· 修订版本号
· 改动的数据
· 修订者名字和 e-mail
· 改动的详细描述

下面是一个例子：

```
Example Header
-------------------------------------------------------------------------------
-- Module          : MAC (Multiply Accumulate Unit)
-- File            : mac.vhd
-- Library         : ieee,.....
-- Description     : It is a general Purpose Multiply Accumulate Unit capable of
-- Simulator       : Modelsim 5.2 / Windows 95
```

-- Synthesizer : Synplify / Windows95

-- Author / Designer : Harish Y S (harish@opencores.org)

Example Revision List

-- Revision Number : 1

-- Date of Change : 20th March 2000

-- Modifier : Harish Y S (harish@opencores.org)

-- Description : Initial Design

-- Revision Number : 2

-- Date of Change : dd mm yyyy

-- Modifier : XYZ (email)

-- Description : Modified the ????.to improve ????..

文件头的标准模式：

-- Title :

-- Project :

-- File :

-- Author : name <email>

-- Organization:

-- Created :

-- Last update :

-- Platform :

-- Simulators :

-- Synthesizers:

-- Targets :

-- Dependency :

-- Description:

-- Copyright (c) notice

-- Revisions :

-- Revision Number :

-- Version :

-- Date :

-- Modifier : name <email>

-- Desccription :

2．联机注释

每一个重要的操作和定义后都要加上注释，描述操作和声明的使用。

3．命名规则

3．1 实体和结构

规则：· 实体名要确切描述其功能；

· 实体名只能用小写字母，不超过 10 个字符；

推荐：每个实体最好有一个 3-4 个字母的缩略名，可以将其应用在其内部的构造模块（component）和信号名中。

3．2 端口

规则：· 端口名应和信号相对应，以大写字母开头；

· 若端口是标准设备，可包含标准名，不超过 15 个字符；

· 端口声明后要有详细注释。

3．3 结构体

结构体定义系统行为，可从不同方面对其进行描述，结构体和实体是一致的，其名字要表明系统描述的方法。

规则：· 构造体名可用“ behavioural ”表示行为描述，“ structural ”表示结构描述，“ RTL ”表示寄存器描述等；

· 由综合出来的结构体要有“ _syn ”后缀，并且在开始和结束出要注明采用的技术；

· 在“ ARCHITECTURE ”语句前要有一行注释，说明其功能，并说明是否可综合，或仅可仿真。

推荐：当一个设计中包含多个文件时，通过加“ _arch ”后缀来加以区分。

3．4 元件 component

元件 component 在 VHDL 设计的层次结构中使用。

规则：其名称以包或实体的缩略名作开头；

可取有实际意义的单词，大小写可混用，最好不要超过 8 个字符。

3．5 配置

配置是用来说明逻辑模块和其构造体间的关系。

规则：配置名中要包含顶层设计名；以大写字母开头，不超过 15 个字符。

推荐：加“ _cfg ”后缀区分多个文件。

3．6 包、函数和过程

3．6．1 包

规则：· 包中要包含系统所需定义的所有常量，数据类型，模块，过程和函数；

· 包名以大写字母开头，不超过 15 个字符。

推荐：加“ _pkg ”后缀区分多个文件，为包定义一个 3-4 个字符的缩略名，加在其中的常量，过程和函数名中，用以区分不同包中的内容。

3．6．2 函数和过程

规则：· 以大写字母开头，不超过 10 个字母；

· 要体现其功能，用前缀“ l_ ”表示局部变量；

· 局部信号应有其特征域。

推荐：加入包的缩略名于其中。

3．7 常量和类属说明

规则：用大写字母，要明确描述常量的用法。

推荐：加入包的缩略名于其中。

3．8 枚举（enumeration），数据类型，记录和数组

规则：用大写字母，新数据类型要加后缀“ _typ ”。

3．9 信号和变量

3．9．1 信号

规则：· 第一个符号必须是字母，信号名要描述其功能，不超过 15 个字符；

· 头三个字母要显示说明驱动模块的类型，要把其驱动实体，模块或进程缩略名加在前面：如控制单元——“ ctl ”，算术逻辑运算单元——“ alu ”，乘法器——“ mac ”，数据地址发生器——“ dag ”；

· 如果信号只是在仅有时钟的进程中获得其值的，则加“ _q ”，若是总线则加“ _reg ” 信号定义语句后要有一行注释描述其功能；

· 信号名要表明信号的极性：高电平有效/正逻辑（P），低电平有效/负逻辑（N）

· 全局信号“ G ”，局部信号“ L ”；若是三态信号，加 Z；

· 后续字符要说明信号的内容。

例如：“alu{GBaugend ”——其驱动的模块为算术逻辑运算单元，高电平有效全局信号，是算术逻辑运算单元的其中一个操作数的总线信号。

“macNGWoverflow ”——其驱动的模块是乘法器，低电平有效全局单线信号，其功能是在乘法器溢出时修改状态寄存器的溢出标志位。

3．9．2 变量

规则：· 变量名要简单并能描述其功能；

· 变量名可包含各种格式的字母、数字和下划线；

· 变量名要确切的表示其行为。

3．10 进程和块

进程、块和配置可取有实际意义的单词，大小写可混用，最好不超过 8 个字符。

3．10．1 进程

规则：· 所有进程必须有进程名，用以描述其功能；

· 注释要包含以下内容：组合、时序进程，组合进程要定义所有敏感信号，时序进程要定义时钟和其边沿（上升沿或下降沿），时序进程还要定义复位信号——如果有的话，其有效与否与时钟有关。

3．10．2 块

3．11 测试工作台 test bench

由于测试工作台在设计流程种的重要地位，因此，对其有一些特殊的要求。

规则：· 其名称要与实体名一致，且加后缀“ _TB ”；

· 结构体、进程、变量和信号同样遵循上述规则；

· 内存组织和仿真生成由过程和函数来实现；

· 出错报告要提供下述信息：实体或模块名，信号或变量名，过程或函数名，当前时间点，错误号或错误名，可能的出错原因，出错位置（RTL, structural 或 behavioral 代码）；

3．12 文件和目录结构

现在在目前的集成开发环境中自动管理

3．13 其它

· 尽可能使用类书参数说明（Generic）。

· 尽量多定义常数，这样可以增加代码的可读性。

· 写代码之前，要先画出系统框图，对自己要做的模块有一个清楚的认识，这样可以减少你写代码时间，提高效率。

第三部分：HDL 编码指导

1．复位

1．1 作用

复位使初始状态可预测，防止出现禁用状态。

1．2 推荐：

·FPGA 和 CPLD 的复位信号采用异步低电平有效信号，连接到其全局复位输入端，使用专用路径通道。

FPGA 和 CPLD 有固定时间延迟线，连接到所有资源上。

·对于目标器件为 ASIC 的 core，异步时钟只能局部使用，在顶层设计上要与时钟同步，这可以防止过长的延时。

·复位时，所有双向端口要处于输入状态。

1．3 强烈推荐：

复位信号必须连接到 FPGA 和 CPLD 的全局复位管脚。

这是由于这些管脚提供较低的抖动。

2．时钟

2．1 好的习惯：

在 core 中尽可使用小的时钟域。

2．2 推荐：

·信号穿过时钟的两半个周期时，要在前后分别取样；

防止出现半稳定状态。

·不要用时钟或复位信号作数据或使能信号，也不能用数据信号作为时钟或复位信号；

HDL 综合时会出现时序验证问题。

· 不要使用门时钟（don't use gated clock）。

2．3 强烈推荐：

时钟信号必须连接到全局时钟管脚上。

3．总线

推荐：

· 总线要从 0 位开始；

有些工具不支持不从 0 位开始的总线。

· 从高位到低位；

这样可以避免在不同设计层上产生误解。

4．通用规则

4．1 强烈推荐：

不要使用内部三态信号，否则增加功耗。

这样使后端的调整更困难。

4．2 同步设计和时序优化

4．2．1 强烈推荐：

· 只使用同步设计；

这样可以避免在综合、时序验证和仿真中的出现的一些问题。

· 不要使用延时单元；

· 所有块的外部 IO 必须注册；

这样可以避免较长的路径延时

4．2．2 推荐：

· 避免使用锁存器；

这样会产生综合和时序验证问题。

· 避免使用负延触发的双稳态多谐振荡器（flip flop）。

同样会产生综合和时序验证问题。

4．2．3 好的习惯：

块内部 IO 要例化。

这是设计问题，在大部分情况下推荐使用

5．verilog 编码指导原则

5．1 一般规则

5．1．1 强烈推荐：

在时钟驱动的同步进程中不要使用 block 结构，block 结构应用于异步进程种。

Synopsys 希望使用这种格式，有确定的仿真响应。

5．1．2 推荐：

·尽量使用无路径的“ include ”命令行；

HDL 应当与环境无关。

·避免使用“ ifdef ”命令，尽量用一个全局定义文件做所有的定义；

否则*容易产生版本和编辑问题*

5．1．3 好的习惯：

·尽量在一个文件中只用一个模块，文件名要和模块名相同；

·尽量在例化中使用名称符号，不要用位置符号；

有利于调试和增加代码的易读性。

·在不同的层级上使用统一的信号名；

容易跟踪信号，网表调试也容易。

·比较总线时要有相同的宽度。

否则其它位的值不可预测。

5．2 仿真和调试

5．2．1 强烈推荐：

全部的系统仿真任务都应在 Synopsys 命令“ synopsys translate on/off ”之中。

5．2．2 好的习惯：

·在全局定义文件中，在开始的时间标度命令中写“ timescale 1n/10p ”；

不同的“timescale”会导致仿真问题——竞争和过长的路径

· 尽量在“ display ”命令中使用“ %m ”（显示实例名）。

6．VHDL 代码指导原则

6．1 一般规则

6．1．1 强烈推荐：

· 外部端口用 std_logic 类型；

· 不要赋未知值“ x ”或检查验证无效的“ - ”；

这些值在仿真和综合时会产生不可预测的行为。

· 不要使用信号和变量的默认值（或初始值），用复位脉冲初始化信号和变量。

会在仿真和综合时出现不匹配。

6．1．2 好的习惯：

· 在整个 VHDL 工程中不要混用编码准则（i.e. VHDL 87 and VHDL 93）；

· 尽量在一个 VHDL 文件中做一个设计，文件名要和结构名一致；

· 尽量在模块例化中使用名称符号，不要用位置符号；

有利于调试和增加代码的易读性。

例如：

```
wb_if: wb
        PORT MAP (
        CLK     => CLK_i,
        RST_I => RST_I_i,
        ACK_O => ACK_O_i,
        ADR_I => ADR_I_i,
        CYC_I => CYC_I_i,
        DAT_I => DAT_I_i,
        DAT_O => DAT_O_i,
        RTY_O => RTY_O_i,
        STB_I => STB_I_i,
```

```
    WE_I    => WE_I_i);
```

6．1．3 好的习惯：

·在不同的层级上使用统一的信号名；

容易跟踪信号，网表调试也容易。

·尽量用配置（configuration）映射实体、结构体和模块；

改变不同的结构体只要简单改动一个文件就可以了，这在仿真上很有用，能从高层上改变低层结构体。

·尽量在分开的库中编译每个块；

·使用常量和类属说明定义缓冲大小，总线宽度和其它单元参数。

这可增强可读性和代码复用性。

6．1．4 推荐：

·在一个最小化的包中定义模块和常量。

6．1．5 强烈推荐：

·不要在代码中使用 buffer 类型的端口读取输出数据；要使用 out 类型，再增加另外变量或信号，以获取输出值。

这是因为 buffer 类型的端口不能连接到其他类型的端口上，因此 buffer 类型就会在整个设计的端口中传播下去。

例如：

```
PROCESS (CLK, RST_n)
variable out_var : std_logic;
BEGIN    -- PROCESS
  IF RST_n = '0' THEN
      Outsignal <= '0';
      out_var <'0';
      outsign2 <= '0';
  ELSIF CLK'event AND CLK = '1' THEN
    Outsign2 <= out_var;    -- the same as Outsignal
    out_var := input1 and input2;
    Outsignal <= input1 and input2;
  END IF;
```

```
END PROCESS;
```

6．2 可综合编码

6．2．1 好的习惯：

尽量使用 FSM，一个在时序逻辑中，一个在组合逻辑中。

这可增加可读性和预测组合逻辑的大小

6．2．2 推荐：

尽量在一个单独的时钟进程中写时钟使能，而不要在两个不同的进程中使用，一个是时钟驱动的，一个是组合逻辑，如下例所示。

这是因为有些综合工具检查 CE 操作，若存在，就将其映射到触发器的 CE 端，否则，CE 管脚不被使用，隐含使用外部逻辑。这是 fpga 设计的一般习惯。

```
PROCESS (CLK, RST_n)
BEGIN    -- PROCESS
  IF RST_n = '0' THEN
      Outsignal <= '0';
  ELSIF CLK'event AND CLK = '1' THEN
      IF (CE = '1') THEN
          Outsignal <= '1';
      END IF;
  END IF;
END PROCESS;
```

6．2．3 强烈推荐：

·对变量要先读后写；

如果先写后读，就会产生长的组合逻辑和锁存器（或寄存器）。这是因为变量值是立即获取的。

```
PROCESS (CLK, RST_n)
Variable out_var : std_logic;
BEGIN    -- PROCESS
  IF RST_n = '0' THEN
      out_var <'0';
```

```
        outsign2 <= '0';
    ELSIF CLK'event AND CLK = '1' THEN
        Outsign2 <= out_var;    -- read
        out_var := input1 and input2; -- write
    END IF;
END PROCESS;
```

· 在组合逻辑进程中，其敏感向量标中要包含所有要读取得信号；

这是为了防止出现不必要的锁存器。

· 避免使用长的 if-then-else 语句，而使用 case 语句来代替；

防止出现较大的优先编码器，使得代码比较容易读懂。

6．3 以仿真和调试为目的的编码

6．3．1 好的习惯：

尽量使用两部分的 test bench，一部分作为数据产生和检验，另一部分作为时序总线接口协议的产生和检验。

这是为了从总线握手中分离数据（结果检验），为了使操作简单——改变总线握手协议而同时保持内部逻辑不变。