**verilog编程规范**

1.规范的重要性

当前数字电路设计越来越复杂，一个项目需要的人越来越多，当几十号同事设计完成同一个项目时候，大家需要互相检视对方代码，如果没有一个统一的编程规范，那么是不可想象的，大家的风格都不一样，如果不统一的话，后续维护、重用等会有很大的困难，即使是自己写的代码，几个月后再看也会变的很陌生，也会看不懂（您可能不相信，不过笔者和同事交流发现大家都是这样的，时间长不看就忘记了），所以编程规范的重要性显而易见。

另外养成良好的编程规范，对于个人的工作习惯、思路等都有非常大的好处。

可以让新人尽快融入项目中，让大家更容易看懂您的代码。

2.概述

本规范基于 Verilog HDL 语言进行描述，主要采用顺序结构进行描述，包括项目层次、输入输出、信号命名、核心代码等进行描述。

3.项目组织形式

当前项目越来越复杂，项目的文件需要统一的存放在一个统一的文件夹下。根据各自功能不同，分门别类的存放。以项目 XX 为例，其文件存储方式如下所示。

XX

|--doc

|--ip

|--rtl

|--sim

|--flow

doc：存放项目相关的文档，包括该项目用到的 datasheet，芯片规格书

(specification、设计方案)等等。

ip：存放商业上购买的成熟 IP 以及项目中一些成熟的可复用的代码。一般的。

rtl：存放项目的 rtl 代码。这是项目的核心，文件名与 module 名称应当一致，建议按照模块的层次/子系统分开存放。

sim：存放项目的仿真代码。一般在 sim 中建立一级子项目，使得 sim 文件

恰恰在项目的两级子目录下。仿真文件的寻址需要上朔两级从根目录开始，这样

当文件发生变动时，文件的寻址不需要多么大的改动。&#0;

flow：存放后端的一些动作，包括 DC 综合、STA 时序分析、FV 形式验证、等后端动作。

4.设计风格

Verilog 文件主要有以下几个部分组成：文件声明，IO 声明，宏定义，wire&reg 定义，parameter 定义，module 具体实现。

1）文件声明

每一个 Verilog 文件的开头，都必须有一段声明的文字。包括文件的版权，作者，创建日期以及内容介绍等，如下表所示。

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Copyright (C) xx Coporation

// File name: xx.v

// Author: xxxxxxxxxx

// Date: 2012-01-01

// Version: 1.0

// Abstract: xxxxxxx

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

如果对该文件进行了修改，请在开头声明中添加以下语句，如下表所示。

// Version: 1.1

// Date: 2012-01-01

// Abstract: XXX

// Author: xxxxxxxxxx

本文规定：

① 不建议使用 include 包含文件；

② 一个.V 只包括一个 module；

2）输入输出定义

输入输出有 Verilog 95 和 2001 两种格式，本文采用 Verilog 2001 语法格式。下面是一个例子，包括 module 名字、输入输出、信号名字、输出类型、注释。

module divider7\_fsm (

//input

input sys\_clk , // system clock;

input sys\_rst\_n , // system reset, low is active;

//output &#0;

output reg clk\_divide\_7 // output divide 7 clk

);

规范：

① 一行只定义一个信号；

② 信号需要对齐；

③ 同一组的的信号放一起；

3）Prameter 定义

一个 module 中的 Prameter 声明，不建议随处乱放。

规范：

① 将 Prameter 定义放在紧跟着 module 的输入输出定义之后；

② Prameter 等常量命名全部使用大写。

③ 一行只定义一个信号；

// Prameter

parameter S0 = 7'b0000000;

parameter S1 = 7'b0000001;

parameter S2 = 7'b0000010;

parameter S3 = 7'b0000100;

parameter S4 = 7'b0001000;

parameter S5 = 7'b0010000;

parameter S6 = 7'b0100000;

4）WIRE/REG 定义

一个 module 中的变量声明需要集中放在一起，不建议随处乱放。

因此，本文规定：

① 将 reg 与 wire 的定义放在紧跟着 prameter 之后；

② 建议具有相同功能的信号集中放在一起；

③ 信号需要对齐，reg 和位宽需要空 2 格，位宽和信号名字至少空四格；

④ 位宽使用降序描述，[6:0]；

⑤ 时钟使用前缀 clk，复位使用后缀 rst；

⑥ 不能使用 Verilog 保留字作为信号名字；

⑦ 一行只定义一个信号&#0;

//reg define

reg [6:0] curr\_st ; // FSM current state

reg [6:0] next\_st ; // FSM next state

reg clk\_divide\_7 ; // generated clock，divide by 7

//wire define

wire [6:0] aa ;

5）信号命名

信号命名每个人都有不同的喜好，本文规定：

① 信号命名需要体现其意义，比如 fifo\_wr 代表 FIFO 读写使能；

② 可以使用“\_”隔开信号，比如 sys\_clk；

③ 内部信号不要使用大写，也不要使用大小写混合，建议全部使用小写；

④ 顶层 PAD 信号名字建议全部使用大写，比如 CPU\_WR 代表 CPU 写使能管

脚；

⑤ 模块名字使用小写；

⑥ 低电平有效的信号，使用\_n 作为信号后缀；

⑦ 异步信号，使用\_a 作为信号后缀；

⑧ 纯延迟打拍信号使用\_dly 作为后缀；

6）Always 块描述方式

规范：

① if 需要空四格；

② 一个 always 需要配一个 begin 和 end；

③ always 前面需要有注释；

④ 信号和逻辑运算符有空格；

⑤ beign 建议和 always 放在同一行；

⑥ 一个 always 和下一个 always 空一行即可，不要空多上；

⑦ 时钟复位触发描述使用 posedge sys\_clk 和 negedge sys\_rst\_n

⑧ 一个 always 快只包含一个时钟和复位；

⑨ 时序逻辑使用非阻塞赋值；&#0;

⑩ 信号逻辑处理的位宽需要一致，即使+1，也要和信号保持一致；

// gen a 1S counter , 1S = 50000000 \* 20ns

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (sys\_rst\_n ==1'b0)

clk\_cnt <= 26'b0;

else if ( clk\_cnt == 26'd50000000 )

clk\_cnt <= 26'b0;

else

clk\_cnt <= clk\_cnt + 26'b1;

end

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

xxx

7）Assign 块描述方式

规范：

① assign 模式的逻辑不能太复杂，否则易读性不好；

② assign 前面需要有注释；

③ 组合逻辑使用阻塞赋值；

// wfq\_sch\_en enable is active 1 second

assign wfq\_sch\_en = ( clk\_cnt == 26'b0 ) ? 1'b1:1'b0 ;

8）空格和 TAB

由于不同的解释器对于 TAB 翻译不一致，所以建议不要使用 TAB，全部使用

空格。

规范：

① 不使用 TAB，全部使用空格；

9）注释

注释的描写可以增加文件的可读性，易于维护。

规范：

① 注释描述需要清晰、简洁；&#0;

② 注释描述不要废话，冗余；

③ 注释描述需要使用“//”；

④ 注释描述需要对齐；

⑤ 核心代码和信号定义之间需要增加注释；

reg [6:0] curr\_st ; // FSM current state

reg [6:0] next\_st ; // FSM next state

reg clk\_divide\_7 ; // generated clock，divide by 7

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Main Program

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// gen a 1S counter , 1S = 50000000 \* 20ns

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (sys\_rst\_n ==1'b0)

clk\_cnt <= 26'b0;

else if ( clk\_cnt == 26'd50000000 )

clk\_cnt <= 26'b0;

else

clk\_cnt <= clk\_cnt + 26'b1;

end

10）例化

规范：

① moudle 例化使用 U\_XX\_0、U\_XX\_1 标示；

② 例化使用大写；

11）状态机

规范：

① 状态机使用三段式描述；

② 尽量避免使用状态机，减少电路复杂性；

5.可综合性

规范：

① 不使用不可综合语句；

② 不使用 repeat 等循环语句；&#0;

③ 避免使用异步反馈环电路；

④ RTL 级别代码里面不使用 initial 语句，仿真代码除外；

⑤ 避免产生 Latch 锁存器；

产生 Latch 例子:

① 条件没有补全

always @(\*) begin

if (condition)

latch\_i = aa;

end

② 组合逻辑赋值给自己

always @(\*) begin

if (condition)

aa = aa;

end

6.其他

规范：

① 代码写的越简单越好；

② 避免使用复杂和少见的语法，这种语法综合器优化力度较低；&#0;