海思高校合作——QA 培训资料

一、 RTL CODE 规范

1.标准的文件头

在每一个版块的开头一定要使用统一的文件头，其中包括作者名，模块名，创建日期，概要，更改记录，版权等必要信息。

统一使用以下的文件头：

```
// ********************************************************************
// COPYRIGHT(c)2005, Hislicon Technologies Co, Ltd
// All rights reserved.
//
// IP LIB INDEX :   IP lib index just sa UTOPIA_B
// IP Name      :   the top module_name of this ip, usually, is same
//                  as the small ip classified name just as UTOPIA
// File name    :   file_name of the file just as "tx_fifo.v"
// Module name  :   module_name of this file just as "TX_FIFO"
// Full name    :   complete Emglish nme of this abbreviated
//
// Author       :   Athor/ID
// Email        :   Author's email
// Data         :
// Version      :   V 1.0
//
//Abstract      :
// Called by    :   Father Module
//
// Modification history
// -----------------------------------------------------------------------------------------------
// //
// $Log$
//
// ********************************************************************************
```

2. 标准的 module 格式 (module 整体结构)

对于模块的书写采用统一的格式便于项目内部成员的理解和维护，我们用批处理建立了一个 MODULE 模块，其内容解释如下：

- 端口定义按照输入，输出，双向的顺序：
- 模块名、模块例化名统一，例化名前加大写 U_以区分 （ 多次例化另加标识 ），三者关系：

文件名 ： xxx .v　　(小写)

模块名 ： XXX （大写）
例化名 ： U_XXX (大写)

IP 内部所有的模块名都要加 IP 名或者 IP 名简称作前缀，如 USB_CTRL、USB_TX_FIFO。

```
// ******************************
//   DEFINE MODULE PORT   //
// ******************************
//
// module   MODULE_NAME   (
                          // INPUT
                          input_port_1,
                          …
                          input_port_m,

                          // OUTPUT
                          output_port_1,
                          …
                          output_port_m,
                         );

// ******************************
//   DEFINE PARAMETER   //
// ******************************
parameter…

// ******************************
// DEFINE INPUT
// ******************************
input              rst_n    ;     // reset, active low .
input              clk_*    ;     // clock signal , 50M .
input   [n:0]      a_din    ;     // *****
input   [k:0]      b_din    ;     // *****

// ******************************
// DEFINE OUTPUT   //
// ******************************
output   [m:0]     a_dout   ;     // *****
output   [i:0]     b_dout   ;     // *****

// ******************************
// OUTPUT ATRRIBUTE   //
// ******************************
// REGS
```

```
reg    [m:0]      a_dout    ;       // *****
//WIRES
wire   [i:0]      b_dout    ;       // *****


// ********************************
// INSTSNCE MODULE     //
// ********************************
MODULE_NAME_A   U_MODULE_NAME_A(
                                                .A          (A          ),
                                                .B          (B          ),
                                                .C          (C          ),
                                                ); …

// ********************************
//MAIN CODE    //
// ********************************
  … …
  … …
  … …
// ********************************    //
Endmodule
```

3.一致的排版

A. 一致的缩排

- 统一的缩排取 4 个空格宽度
- 输入输出信号的宽度定义与关键字之间，信号名与宽度之间要用空格分开；所有宽度定义对所有信号名对齐，代码风格统一如下：

```
input     [3:0]      input_a     ;      // *****
input                input_b     ;      // *****
…
output [128:0]    output_a ;
output [15:0]     output_b ;
output                output_c ;
```

B. 一致的 begin end 书写方式

always 中，一定要用 begin end 区分，格式和代码风格统一如下：

```
always @ (postedge clk or negedge rst_n）
begin
   if (rst_n==1’b0)
      syn_rst<= ‘DLY 1’b0;
   else
      begin
         if (a==b)
```

```
                syn_rst<= 'DLY 1'b1;
            else
                syn_rst<= 'DLY 1'b0;
        end
    end
```

if else 中仅有一个语句行时，不要使用 begin end; 如果有多个语句行时，begin end 和 if （）或 else （）空四个格。

格式如下：

```
if  (…)
…
else if (…)
else
**************************************************************************

if  (…)
…
else if (…)
  begin
    …
    …(
  end
else
```

4. 一致的信号命名风格

简洁，清晰，有效是基本的信号命名规则，详见命名规范。

全称	缩写	中文含义
acknowledge	ack	应答
adress	addr(ad)	地址
arbiter	arb	仲裁
check	chk	校验，如 CRC 校验
clock	clk	时钟
config	cfg	Configuration,装置
control	ctrl	控制
count	cnt	计数
data in	din(di)	数据输入
data out	dout(do)	数据输出
decode	de	译码
decrease	dec	减一
delay	dly	
disable	dis	不使能
error	err	错误（指示）
enable	en	使能

frame	frm	帧
generate	gen	生成，如 CRC 生成
grant	gnt	申请通过
increase	inc	加一
input	in(i)	
length	len	（帧、包）长
nmport	nm	网管相关
output	out(o)	
packet 不推荐 packet	pkt	与帧相同
priority	pri	优先级
pointer	ptr	指针
rd enable	ren	读使能
read	rd	读（操作）
ready	rdy	应答信号或准备好
receive	rx	（帧数据）接收
request	req	（服务、仲裁）请求
reset	rst	
segment	seg	
souce	scr	源（端口）
ststistics	stat	统计
timer	tmr	定时器
switcher	sf	Switch fabric
temporary	tmp	临时
transmit	tx	发送（帧数据）相关
Valid	vld(v)	有效、校验正确
wr enable	wen	写使能
write	wr	写操作

a. 端口、信号、变量名的所有字母小写：函数名、宏定义、参数定义用大写
b. 使用简称、缩略词（加上列表）
c. 基于含义命名（避免以数字命名的简单做法），含义可分段（最多分三段），每一小段之间加下划线"_",如 tx_data_val;命名长度一般限制在 20 个字符以内。
d. 低电平有效信号，加后缀"_n",如 rst_n
e. 无条件寄存的寄存信号在原信号上加 ff1、ff2… 如原信号 data_in, 寄存一拍 data_in_ff1, 寄存两拍 data_in_ff2
f. 不能用 "reg",作为最后的后缀名，因为综合工具会给寄存器自动加上_reg, 如果命名里就用_reg 作为后缀名则扰乱了网表的可读性。

5.统一的表达式书写

A. 括号的使用

如果一个表达式的分组情况不是很明显时，加上括号有助于理解。
例如下面的代码加上括号就清晰很多。

```
if （&a==1'b1&&!flag==1'b1‖b==1'b1）                    //
```

改为：

if （(&a==1'b1）&&（!flag==1'b1）‖（ b==1'b1））　　//

B．适当的使用空格

一般表达式在运算符的两侧要各留出一个空格，但定义比较长的表达式，去掉预先级高的运算符前的空格，使其与运算对象紧连在一起，可以更清晰的显示表达式结构。

还是上面的例子：

if （(&a==1'b1）&&（!flag==1'b1）‖（ b==1'b1））　　//

改为：

if （(&a == 1'b1）&&（!flag == 1'b1）‖（ b == 1'b1））　　//

"<=", "=="前后都要加空格。

C．　赋值要指明比特宽度

赋值或者条件判断时要注明比特宽度，注意表达式的位宽匹配。如：

```
        reg [4:0] signal_a;
错误：  1  signal_a <= 5;
        2  if(signal_a == 5)
        3  signal_a <= signal_b[3:0]+4;
正确：  1  signal_a <= 5d5
        2  if(signal_a == 5d5)
        3  signal_a <= {1'b0, signal_b[3:0]+5d4
```

因为工具默认是 32 位宽，如果不注明位宽，工具检查会报 warning,而且这样增加了设计的严谨性。

6. 统一的语句书写－－条件判断结构书写方式

A. 条件的完整性

If else 搭配使用，对于缺省的条件要写"else；"；

If elsed 条件判别式要全面，比如 if（a == 1'b0）；

Case 中的缺省条件要写"default"；

B."if　else"结构：适用于复杂条件判断的语句

但对于复杂的条件判断，使用？：如果不仔细分析条件的每一条路径，就让读代码搞不清它是到底要做什么。例如：

C = (!Ic&&!rc)?0(Ic?rc:Ic)　//　　(?:)

改为：

```
always @(Ic or rc)          //                  if else
begin
    if ((Ic==0)&&(rc==0))
        c = 0;
    else if (Ic==1)
        c = rc;
    else
        c = Ic;
end
```

即使是简单的条件判断，我们也必须使用 IF-ELSE，当涉及复杂的条件判断，使用 IF-ELSE 结构以获得清晰的结构便于理解和维护。因此必须使用 IF-ELSE。

C."IF ELSE"结构 VS"CASE"结构

IF ELSE 结构综合的结构可能是与或非门构成的，也可能是一组多路选择器，而 case 结构综合结果一般会是多路选择器，但对于可以优化的 case 综合工具会综合出更简单的结构。

所有对于可以写出平行结构的条件，优先写成 case 结构，例如地址译码等，条件之间有重复和嵌套的情况则是写成 if else 结构。

D. Finite State Machine

不允许有模糊不清的状态机模式，所有的状态机必须清晰明了。

我们要求将状态机的时序部分和组合逻辑部分分开。

例如：

```
module state4 (
                clock
                reset
                out
                );
input           reset
input           clock;
output  [1:0]   out;
parameter [1:0]   stateA=2'b00;
parameter [1:0]   stateB=2'b01;
parameter [1:0]   stateC=2'b10;
parameter [1:0]   stateD=2'b11;
reg       [1:0]   state;
reg       [1:0]   nextstate;
reg       [1:0]   out;
always @ (posedge clock)
begin
    if (reset ==1, 0'b0)
        state <= stateA;
    else
        state <= nextstate;
end

always @ (state)
begin
    case (state)
        stateA: begin
           nextstate = stateB;
        end
        stateB: begin
           nextstate = stateC;
        end
```

```
            stateC: begin
               nextstate = stateD;
            end
            stateD: begin
               nextstate = stateA;
            end
        endcase
    end
    always@(postdge clock or negedge reset)
    begin
        if (reset==='b0)
            out <= 2'b0;
        else begin
            if (state==…)
                out <= …;
            else
                out <= …;
        end
    end
    endmodule
```

7. 统一格式的 always 程序块的书写

A. always 中的变量的赋值方式——阻塞与非阻塞赋值

当进行时序逻辑建模时，always 块中使用非阻塞赋值——NON_BLOCKING;

参加如下代码:

```
always @(posedge clk or negedge rst_n)
begin
    if (rst_n == 1'b0;
        myreg <= 1'b0;
    else
        myreg <= 'DLY1'b1;
end
```

always 块中使用的 NON_BLOCKING 赋值时在"<="前要加上＃ 'DLY，如上例；

当使用 always 语句进行组合逻辑建模时，always 块中使用阻塞赋值——BLOCKING；

参见如下代码：

```
    always @（addr）
    begin
        case (addr)

            2'b00 : cs0_n=1'b0;
    2'b01 : cs0_n=1'b1;
    2'b10 : cs0_n=1'b0;
    2'b11 : cs0_n=1'b1;
    default: cs0_n=1'b1;
```

```
    endcase
  end
```

- 如果要使用 always 语句同时进行时序与组合逻辑建模时，一定使用非阻塞赋值；例如：

```
//组合逻辑与时序逻辑在同一个 always 块中
 always@(posedge clk or negedge reset_n)
begin
   if(reset_n==1'b0)
      out<=1'b0;
   else
      begin
         case(count)
            2'b00 : out<= `DLY in_a;
            2'b01 : out<= `DLY in_b;
            2'b10 : out<= `DLY in_c;
            2'b11 : out<= `DLY in_c;
            default: out<= `DLY in_a;
      end
end
```

B. always 中变量赋值的唯一性

- 组合 always 块一定要注意敏感量列表中的触发项完整且不冗余；如果不是这样，综合的电路会与实际设计不符合，会报 warning；
- 不要再多个 always 模块中对同一个 reg 型变量进行赋值；
- 更不能再同一个 always 中随一个变量双重赋值；

例如：

```
always@(posedge clk or posedge reset_n)
 begin
   if(reset_n==1'b0)
      out<=1'b0;
   else
      out<=    `DLY1'b1;        //out        1   0
      out<=    `DLY1'b0;
 end
```

- 推荐不要在一个 always 块里给多个变量赋值。如果将一组条件相同的变量写在一个 always 块中更有利于可读性的提高和功能的实现时候，可有例外情况，但请尽量多加注释，以增加可读性，并注意在组合 always 块中不要出现 LATCH（不如对状态机的组合 always 块及它对条件相似的多个变量赋值）；

C. always 中复位的书写

复位的条件表达式及命名要和 always 敏感列表中的描述相统一，并且一定要使用异步复位。所有的复位必须低有效。

例如：

```
  //
always@(posedge clk ot negedge rst_n) //
begin
   if(rst_n==1'b0)
```

```
        …
    else
        …
end
```

D. always 的注释

要在每一个 always 块之前加一段注释，增加可读性和便于调试。

```
//cm carry count which   …
always@(posedge clk_xc or negedge rst_n)
begin
    if(rst_n==1'b0)
        cm_carry_cnt<=1'b0;
    else
        cm_carry_cnt<=#`DLY1'b1;
end
```

8. 合理的注释

- 代码中应采用英文作详细的注释，注释量应达到代码总量的 50%以上。
- 指示应该与代码一致，修改程序的时候一定要修改相应的注释；
- 注释不应重复代码已经表明的内容，而是简介的点明程序的突出特征；
- 注释应该整个一个程序的线索和关键词，它连接整个程序中分散的信息并它帮助理解程序中不能表明的部分。

9. 重用化设计

层次结构与模块划分

- 层次设计的原理以简单为主——尽量避免不必要的层次；层次结构设计得好，在综合中就不需要太多的优化过程；
- 模块的划分根据层次设计来决定——模块化对于布线有很大帮助，模块化的设计中要尽量减少全局信号的使用；
- 通用的部分尽量提取出来作为一个共用模块，同时为了适应需求的更改也应提供用户定制模块入库的方式。

参数传递

- 需要传递参数的模块，在多次例化的时候统一都传递参数，不要例化同一个模块，有的传参数，有的不传。

模块划分的技巧：

将不同的时钟域分离开来；

按照不同的设计目标划分成块，分块式应在数据流方向上切分；

在同一模块中实现逻辑资源和算术资源的共享。

二. 关于 REVIEW

1. Review 目的

发现缺陷

降低成本
提高质量

2. 流程

1. 完成第一个字模块时，请提交该模块代码，进行规范检查评审。
2. Coding 期间 每两星期 提交依次代码和 review 报告。

Review 报告主要包括内容：

- Review 工作时，review 的代码模块
- 参与人
- 发现的缺陷和解决情况。

Review 建议：

（1） 制定 review 计划；（2） 每次 review 代码不超过 500 行。

ANNEX

CODE STYLE TEMPLATE

This a template of verilog code file, including file header and the main body of code in which some coding rules are demonstrated.

```
//*************************************************************
//
//     Copyright(c)2005, Hisilicon Technologies Co., Ltd
//     All rights reserved
//
//     IP LIB INDEX      :    IP lib index just as UTOPIA_B
//     IP Name           :    the top module_name of this ip, usually, is same as
                               the small ip classified name just as UTOPIA
//     File name         :    file_name of this file just as tx_fifo.v
//     Module name       :    module_name of this file just as TX_FIFO
//     Full name         :    complete English name of the abbreviated module_name
//     Author            :    Author
//     Email             :    Author's email
//     Data              :    2005/07/20
//     Version           :    current version, just this: v1.0, must same as the CVS version
//
//     Abstract          :
//
//     Called by         :    Father module just as TX_PROC
//
//     Modification history
//     ---------------------------------------------------------------------
// Version          Data(yyyy/mm/dd)     name
// Description
//
// $Log$
//
//*************************************************************

//*******************
//DEFINE(s)
```

```
//********************
//`define UDLY 1       //Unit delay, for non-blocking assignments in sequential logic

//********************
//DEFINE MODULE PORT
//********************
module MODULE_NAME(
                          //INPUT
                           rest_n            ,
                           clk_*             ,
                           a_din             ,
                           b_din             ,

                          //OUTPUT
                           a_dout            ,
                           b_dout

                               );

//********************
//DEFINE PARAMETER
//********************
//Parameter(s)

//********************
//DEFINE INPUT
//********************
input                rst_n          ;     //reset, active low .
input                clk_*          ;     //clock signal, 50M .
input   [n:0]        a_din          ;     //*****
input   [k:0]        b_din          ;     //*****

//********************
//DEFINE OUTPUT
//********************
output   [m:0]     a_dout        ;     //*****
output   [i:0]     b_dout        ;     //*****

//********************
//OUTPUT ATTRIBUTE
//********************
//REGS
reg     [m:0]      a_dout        ;     //*****
```

```
//WIRES
wire   [i:0]        b_dout       ;      //*****

//**********************
//INNER SIGNAL DECLARATION
//**********************
//REGS
reg    [3:0]        counter      ;      //*****

//WIRES
wire   [7:0]        temp1        ;      //*****

//**********************
//INSTANTCE MODULE
//**********************

//*****************************************************************
//instance of module MODULE_NAME_A filename:module_name_a.v
//*****************************************************************
MODULE_NAME_A   U_MUDULE_NAME_A(
                        .A                  (A                  ),
                        .B                  (B                  ),
                        .C                  (C                  )
                         );

//**********************
//MAIN CORE
//**********************

//Sequential logic style
always@(posedge clk_* or negedge rest_n)
begin : SEQ_BLOCK_NAME
    if (rst_n==1'b0)
        counter<=4'b0;
    else
        begin
            if (expression)
                counter <= #`DLY siginal_b;
            else;
        end
end // SEQ_BLOCK_NAME

//Combinational logic style
always@(signal_a or signal_b)
```

```
begin:COM_BLOCK-NAME
      case (expression)
            item1       :begin
                              signal_c=*****;
                         end
            item2       : //statement;
            default     ://statement;
      endcase
end // COM_BLOCK_NAME

//**********************
endmodule


      MACRO DEFINE TEMPLATE
//************************************************************
//
//     Copyright(c)2005, Hisilicon Technologies Co., Ltd
//     All rights reserved
//
//     IP LIB INDEX       :    IP lib index just as UTOPIA_B
//     IP Name            :    the top module_name of this ip, usually, is same as
                                the small ip classified name just as UTOPIA
//     File name          :    macro.v
//     Module name        :
//     Full name          :    complete English name of the abbreviated module_name
//     Author             :    Author
//     Email              :    Author's email
//     Data               :    2005/07/20
//     Version            :    current version, just this: v1.0, must same as the CVS version
//
//     Abstract           :
//
//     Called by          :    Father module .
//
//     Modification history
//     ---------------------------------------------------------------------
// Version          Data(yyyy/mm/dd)     name
// Description
//
// $Log$
//
//************************************************************
```

```
//*******************
//DEFINE(s)
//*******************
`define UDLY                1     //Unit delay, for non_blocking assignments in sequential logic
`define DATA_WIDTH          32    //AHB data width
`define ADDR_WIDTH          32    //AHB address width
```