文档类型	开发文档
保密级别	机

技 术 报 告

名　称：Verilog 代码风格规范

编　号：

版本号：

作　者 _____
项　目 _____
部　门 产品中心
日　期 2006-6-9

目　录

前　言

为了更好地规范团队成员在研发项目时 VERILOG 硬件描述语言的撰写，以达到代码规范化和标准化的目的，特制定本规范。

修订日期	版本	修订人	修订项目

1 总则

本规范规定了 IC 设计项目开发过程中 VERILOG HDL 源代码的编写总则、要求及模板文件。

本规范适用于信息安全团队及 IPTV 项目各 IC 产品在设计开发过程中源代码的编写。

2 职责

各模块设计成员负责根据本规范的要求编写 VERILOG HDL 源代码。

系统组成员负责本规范对各项目的 VERILOG HDL 源代码进行规范化格式审查及管理。

3 目的

制定本规范的目的：

3．1 便于项目组成员之间对源代码的理解、交流及相互检查；

3．2 便于设计者本人在项目开发之后或产品升级过程中利用源代码很快理解原有设计；

3．3 便于模块开发过程中不同版本源代码的管理；

3．4 便于模块仿真过程中很快发现问题的出处；

3．5 便于模块整合时各子模块的链接。

4 内容

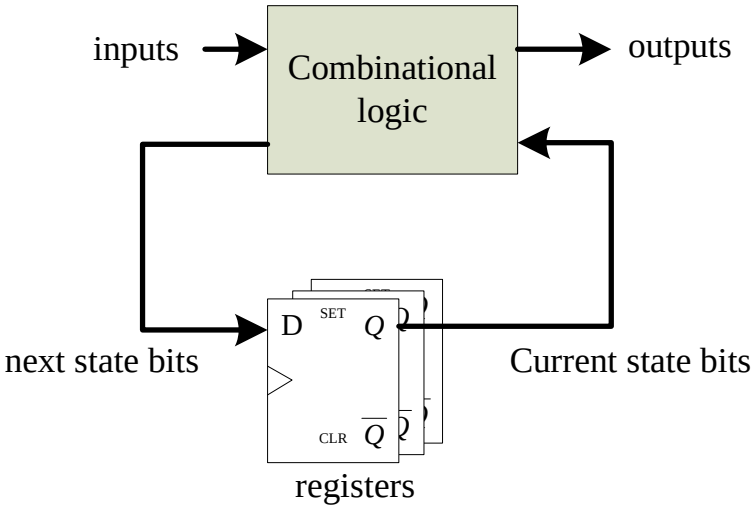
4.1 基本原则

4.1.1RTL级代码风格

RTL 是指 Register Transfer Level，即寄存器传输级，代码显式定义每一个 DFF，组合电路描述每个 DFF 之间的信号传输过程。当前的主流工具对 RTL 级的综合、优化及仿真非常成熟。

不建议采用行为级甚至更高级的语言来描述硬件，代码的可控性，可跟踪性及可移植性难以保证。

4.1.2组合时序电路分开原则



Sequential circuit model

图 4-1 数字逻辑电路模型

(a) current state bits=↑(next state bits);

(b) next state bits=f1(inputs,current state bits);

(c) outputs=f2(inputs,current state bits);

DFF 和组合逻辑描述分开，DFF 在 always 块中完成，组合逻辑通常采用 assign 语句完成。

例如图 4-1中的电路可以描述如下：

```
//------------------------------------------------------------------------
// 通用的数字电路模型描述方式
// 组合与时序电路分开描述
// 时序电路在 always 块中，
// 组合电路采用 assign 赋值语句
// 本代码中的组合电路部分的函数 f1,f2 均代表任意的与或表达式。
//------------------------------------------------------------------------
// 时序电路部分，异步复位
always   @( posedge Clk or negedge Resetn )
begin
if ( !Resetn )
    current_state_bits    <=  0       ;
else
    current_state_bits    <=  next_state_bits   ;
end
// 组合电路部分
```

```
assign   next_state_bits   =   f1(inputs, current_state_bits)  ;
assign   outputs =  f2(inputs, current_state_bits)  ;
```

4.1.3复位

所有 DFF 必须加异步低电平有效复位信号，同步复位根据实际情况决定是否添加。

4.2 命名规则

4.2.1基本命名标准

1） 全局异步复位输入信号命名为 Resetn/Rstn，多复位域则命名为 ResetnXxx/Rstn，Xxx 代表复位域含义缩写且首字母大写；同步复位输入信号命名为 SReset；

2） 时钟输入信号：单一时钟域则命名为 Clk；多时钟域则命名为 Clkxxx，xxx 代表时钟域含义且首字母大写。

4.2.2命名准则

4.2.2.1 模块名命名规则

1. **模块名命名规则**：硬件系统负责人先划分好项目模块，并根据此规则命名模块名；采用层次化命名方法，层次之间用下划线“_”隔开；模块名称只允许分段第一个字母大写。下面以 USB 项目举例（下同）：Sie_Pkt 表示 Sie 模块中的 Pkt(Packet)子模块，处于 IP 模块层次的第三层（包含 IP 最顶层）

第二层模块前缀定义如下：

SIE module：Sie_

Main Controller module：Mctl_

GPIO&MultiBus module：Gpmb_

Cpu Interface module：Citf_

Mcu Interface module：Mitf_

Endpoint Buffer module：Epb_

MCU module：Mcu_

2. IP 等特殊模块命名规则。

例如： 顶层管脚及 Sie 模块的 UTMI 接口部分和 Gpmb 的 GPBUS 接口未按上面的规则命名，较为独立，如有约定俗成的命名则采用约定名，如为全新开发的 IP，可事先定义好 IP 模块名。

4.2.2.2 模块间连线模块端口命名规则：

1. 若连线来自上级模块则保留原名，若直接输出到上级模块则同上级模块端口名；
2. 同级模块之间连线的命名格式为 “源子模块”+“目的子模块”+“意义（参考后面的缩写表）”；若为多目的连线，则“源子模块”+“意义（参考后面的缩写表）”；各级模块的命名采用上述方法逐级递推。注意“源子模块”、“目的子模块”及“意义均可采用缩写。

例如：Sie 模块下的 Sie_Pkt 模块和 Sie_Sil 模块之间的数据连线可表示为 PktSilDat[7:0]，如果 Sie_Pkt 送出的数据给多个其它模块的输入，则命名为 PktDat[7:0]；而 Sie_Pkt 模块下的子模块 Pa 和子模块 Ms 之间的连线可表示为 PaMsStart。

3. 模块端口命名规则：

确定上级模块端口已命名，理清不同模块端口间关系，其命名原则如下：

（1） 若端口与上级模块端口直连，则采用上级模块端口名；

（2） 对任意一对一连接的端口，相关模块的对应端口名字均为 “源模块”+“目的模块”+“意义（参考后面的缩写表）”;

（3） 对于一对多连接的端口，相关模块的对应端口名字均为“源模块” +“意义（参考后面的缩写表）”。

（4） IP 等特殊模块端口命名规则较为独立，如有约定俗成的命名则采用约定名，如为全新开发的 IP，可事先定义好 IP 模块端口名。

例如：Sie 模块下的 Sie_Pkt 模块和 Sie_Sil 模块之间的相连的数据端口均可表示为 PktSilDat[7:0]，如果 Sie_Pkt 送出的数据给多个其它模块的输入，则 Sie_Pkt 及相关的接收模块对应端口均命名为 PktDat[7:0]；

USB 等 IP 的端口，采用 USBIP 定义的端口名。

4. 为了保证连线名与连线两端端口名的一致性，连线中尽量避免使用 Out,In 等标志方向的字符，而使用无方向性的字符如：Dat,Add 等。

5. 常数、参数全部大写。

6. 同一信号经过寄存器处理的命名规则(此规则为推荐级别):

同一信号经过寄存器打过一级之后命名加上“Reg”，打过两级之后加上“Reg2”，依次类推；同一信号前一级寄存器的输入命名为信号名加上“Pre”，前两级寄存器的输入命名为信号名加上“Pre2”，依次类推。

7. 对于低（下降沿）有效信号添加后缀“n”。

4.2.3参考示例

下图所示为一芯片的顶层框图，进行如下模块划分

1. 时分复用模块（Tdm module）包括：Tdm_Int，Tdm_Clk，Tdm_Buf，Tdm_MemCtl；
2. 外部存储控制器（Emc module）包括：Emc；
3. 发送分段器（TxSar module）包括：Tx_Memctl，Tx_Proc；
4. 接收重组器（RxSar module）包括：Rx_MemCtl，Rx_Proc；
5. UTOPIA 接口模块（Utopia module）包括：Ut_Ptx，Ut_Prx，Ut_Stx，Ut_TxFifo1，Ut_TxFifo2，Ut_RxFifo；
6. Cpu 接口模块（CpuItf module）包括：CpuItf；
7. 时钟恢复模块（ClkRec module）包括：ClkRec；
8. 时钟及复位模块（ClkRst module）包括：ClkRst；
9. 测试模块（JTAG module）包括：JTAG。

第二层模块前缀定义如下：

Tdm module：Tdm_

Emc module：Emc_

TxSar module：Tx_

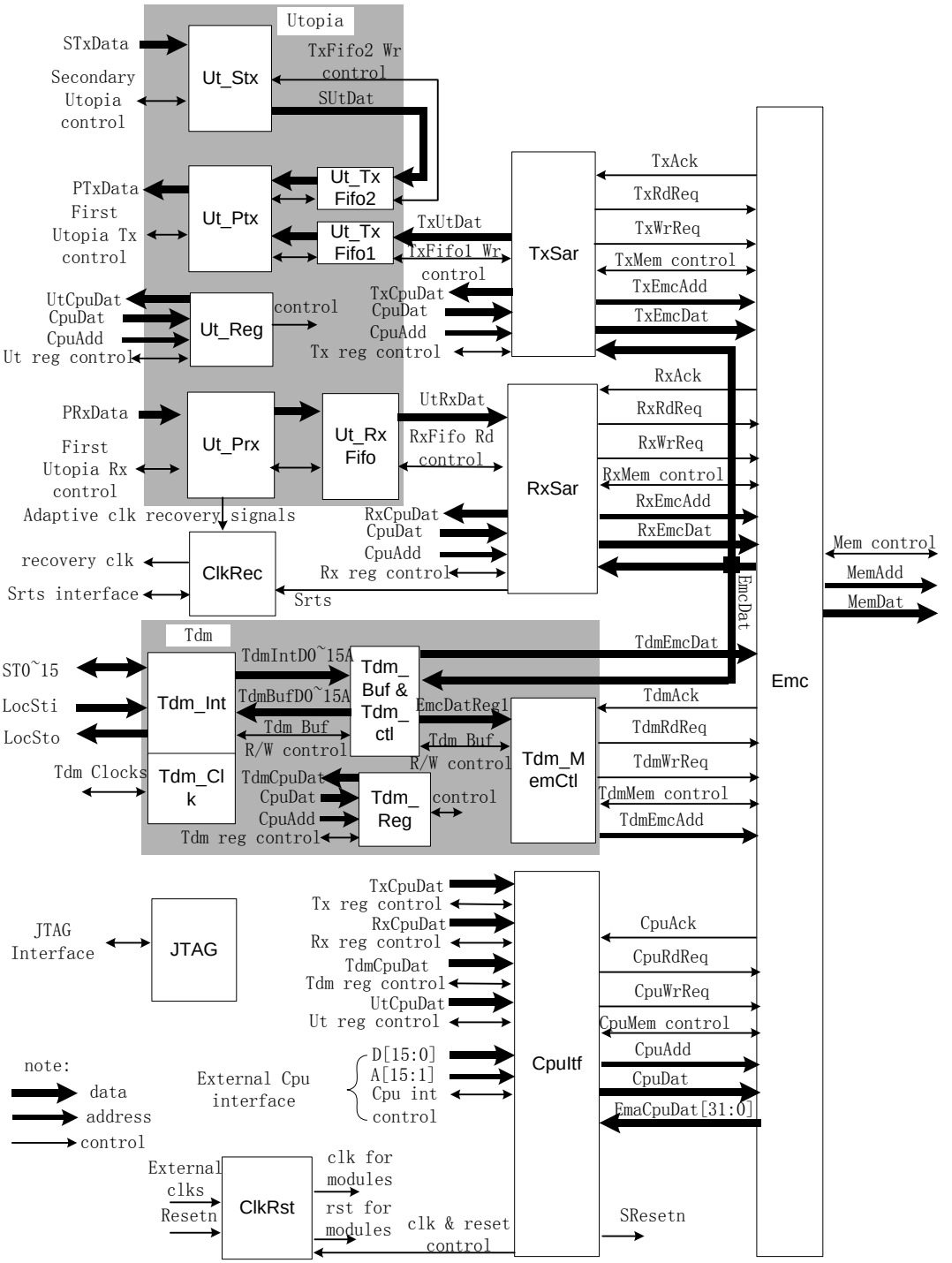
RxSar module：Rx_

Utopia module：Ut_

CpuInt module：Cpu_

ClkRec：Crec_

ClkRst module：Crs_



4.2.4常用缩写表

含义	缩写	含义	缩写	含义	缩写	含义	缩写
Ready	Rdy	Address	Add	Empty	Ept	Over	Ovr
Data	Dat	Full	Ful			Chip Select	Cs
Enable	En	Write	Wr	Read	Rd	Read write	Rw
Wide	Wid	Select	Sel	Single	Sng	Twice	Twi
Request	Req	Acknowledge	Ack	Valid	Vld	Pull down	Pd
Pull Up	Pu			End	End	Control	Ctl
Read End	Rdd	Write End	Wrd	Error	Err	Get	Get
		Parity	Par	Number	Num	Almost Full	Aful
Command	Cmd			Cell	Cel	Mark	Mark
Pointer	Ptr	Head	Hd	Payload	Pyld	Left	Lft
Right	Rgt			Reset	Rst	Halt	Hlt
Master	Mst	Slave	Slv	Count	Cnt	Last	Last
Almost Empty	Aept			Clear	Clr	Receive	Rcv
Trasmit	Tx	Trasmit data	Txd	Received data	Rxd	Difference	Dif
		Pulse	Pls	Value	Val	Busy	Busy
Start	Str	State	Sta				

4.3 VERILOG HDL源代码文件结构

VERILOG HDL 源代码文件结构包含以下几个部分

1） 文件头说明

2） 宏定义

3） 模块及端口定义

4） 变量、参数定义说明

5） 代码主体

6） 注释行

7） 独立的 Include.v 文件

4.3.1VERILOG HDL 代码文件文件头

在代码的开始部分必须按以下顺序加注文件头

1 公司的版权声明;这部分大家直接拷贝范例然后修改 COPYRIGHT (C) 后的年份时间即可。

2 项目控制信息：

2.1 项目名，更改一次即可。

2.2 本文件基于哪个版本修改的版本号，在模块 Release 1.0 之后的修改必须每次修改都做记录；不一定只是当前版本减 1 或者 0.1 或者 0.01，如范例是基于版本 2.0 修改的，而

当前版本已经是版本 2.2。

2.3 本文件当前版本号，在模块 Release 1.0 之后的修改必须每次修改都做记录。

2.4 本文件模块名。

2.5 文件创建者及时间。

2.6 文件当前版本修改者及时间，在模块 Release 1.0 之后的修改必须每次修改都做记录。

2.7 版本修改历史记录，格式如范例：修改时间，修改者姓名，修改内容说明;，在模块 Release 1.0 之后的修改必须每次修改都做记录。

2.8 如有算法运算模块，最好给出其规模、级数等指标。

代码文件头范例如下：

```
//  ================ ZTEIC DESIGN CO.LTD ===============
//
//        Information contained in this Confidential and Proprietary work
//                  has been owned by ZTEIC DESIGN CO.LTD.
//        This Software may be used only under Licensing Agreement
//                  from ZTEIC DESIGN CO.LTD.
//
//
//                  COPYRIGHT (C) 2006 ZTEIC DESIGN CO.LTD.
//                        ALL RIGHT RESERVED
//
//
//        The entire notice above must be displayed on all authorized copies.
//      Copies may be made only under extended consent from ZTEIC DESIGN CO.LTD.
//    ------------------------------------------------------------------------------------------
//    Project and Control Information
//    ------------------------------------------------------------------------------------------
//    Project Name            :Zi1208
//    Last Version            :2.0
//    This Version            :2.2
//    This module name            :Core_Ctl_Fsm
//    This File generated by      :Kangxiaogang 2003/11/06
//    This File updated by        :XieHua 2006/01/13
//    Revision History            :
//          2004.3.23     LiuJun        correct `IC_RET EXEC2's bug,dec SP-->keep SP.
//          2004.3.28     LiuJun        correct `IC_RET EXEC1's bug,add dec SP
//                                      and change code style for avoiding latch generation
//          2004.4.5      LiuJun        add MOVC access control,including add port ban_movc
```

```
//          2004.4.13    LiuJun      correct          `IC_CJNE_XX's          bug,when
op_a==op_b,CY=0
//          2004.4.22    LiuJun      keep s_adr_mux when MOV A,XX
//          2004.4.29    LiuJun      add input port suspnd_core for sim_board
//          2004.5.12    LiuJun      replace SIU's s_ti/ri with 7816's ScdInt_i
//          2004.5.21    LiuJun      add Instruction Flag
//          2004/5/27    LiuJun      change design to fix re-enter EX interrupt service
subroutine
//                                   when low level trigger
//          2004.6.7     LiuJun      delete suspnd_core port
//          2004.6.10    LiuJun      fix scd's int s_regs_wr_en's bug
//          2004.8.9     LiuJun      change for DIV A B command,add 2 clock
//                                   cycles to reg inputs and outputs
//          2004.8.17    LiuJun      change 3 commands for aludata_i,each adds one
more state
//          2004.9.6     LiuJun      correct cjne_a_d's bug and djnz's bug
//          2005-5-25    XieHua      add rdx_mux signal
//          2005-6-30    XieHua      replace ban_movc with CoreMpu_MovcFlag
//          2005-8-22    XieHua      add one more state before startup to delay one
clock to start
//          2005-10-18   Luanchanghai     add the sixth interruption and define SIU
//          2006-1-13    XieHua      add dual-dptr support;
//                                   add mov @dptr,A one more state for 4 byte wr
//    --=================================================--
```

4.3.2VERILOG HDL 代码文件宏定义

在文件头之后如需要定义宏，则用`define 语句添加，每行定义一个宏，尽量添加注释。

4.3.3VERILOG HDL 代码文件模块名及端口信号

1 一个文件只能有一对 module/endmodule，且 module 定义名必须与文件名一致，但文件名含版本号。

2 所有端口信号在定义、声明时都必须是每行只有一个，方便添加注释。

3 声明模块端口信号的时候根据信号的方向按顺序分成输入、双向、输出三大部分；

4 模块中所有输入、输出和双向端口信号均必须定义；

5 端口信号定义时应与声明时顺序一致；

6 端口信号定义、声明时信号名的第一个字母应在同一列（建议利用 TAB8 键实现模块信号、关键词等的列对齐）；

4.3.4VERILOG HDL 代码文件信号、变量及参数

1 变量与参数分隔的原则；(强制)

2 端口信号变量与内部信号变量分隔的原则；(强制)

3 不同功能块信号变量分隔的原则；

4 先 wire 变量后 reg 变量的原则；

所有信号必须显式定义，禁止采用缺省定义，包括端口信号，内部寄存器，内部连线等。定义方式采用分类定义的方式，模板如下：

```
// inputs
input           Clk       ;     //   From Chip Pin
input           Resetn          ;     //   From Chip Pin
// outputs
output          CrecRxVClk  ;     //   To Tdm
// ------------------------------------------------------
// wire declaretions
// ------------------------------------------------------
// (1)Module input signals
// ------------------------
wire       Clk             ;     // System Clock
wire       Resetn          ;     // Hard Reset Asychronous
// (2)Module internal signals
// ----------------------------
wire [11:0]     DIVXNRSum       ;
// (3) Module output signal
// ------------------------
wire       CrecRxVClk        ;
// ------------------------------------------------------------
// Register declarations
// ------------------------------------------------------------
// (1)Module internal signal
// -------------------------
reg   [13:0]     DIVXReg           ;
// (2)Module outputs signal
// -------------------------
```

4.3.5VERILOG HDL 代码文件设计主体

1 按模块电路子功能块加以注释行及多个空行分隔代码；

2 对于子模块引用类子功能块代码，一行一般只能引用一个信号，信号的引用顺序要和子模块的信号声明顺序一致，子模块端口名和应用模块 net 名都要有且尽量相同，模块调用的 Instance 名尽量和模块名接近；

例如：

```
//3)  instance of J0Cal
//   =================
      J0Cal     uJ0Cal(
.Clk            ( Clk_Rsa ),
```

```
    .Resetn         ( Resetn ),
    .SReset         ( SReset ),
    .StartJ0        ( StartJ0 ),
    .N_0            ( N_0 ),
    .EndJ0          ( EndJ0 ),
    .J0             ( J0 )
    );
```

3　对于状态机类子功能块代码，首先是简单的 State 时序逻辑电路 always 语句描述，其次是 next_state 组合逻辑电路 always+case 语句描述，每个状态的输出控制信号尽量不要在 next_state 组合逻辑电路中出现，而是单独描述；always 语句描述组合电路时一定要注意敏感列表的完整性。状态机须采用简单的核心状态机，

例如：

```
    //   2.1) HState
    //   ==============
    always   @(posedge Clk or negedge Resetn)
    begin
            if (!Resetn)
                    HState   <=   2'b0      ;
            else
                    HState   <=   NextHState;
    end
    always  @(HState  or  EndSetH1  or  HCommand  or  HABCommand  or
HAECommand or EndHCal or EndJ0)
    begin
    case (HState)
    2'b00:
        if (HCommand|HABCommand|HAECommand)
            NextHState   =    2'b01;
        else
            NextHState   =    2'b00;
    2'b01:
        if (EndHCal)
            NextHState   =    2'b10;
        else if(EndSetH1)
    //          if (EndSetH1)
            NextHState   =    2'b10;
        else
            NextHState   =    2'b01;
    2'b10:
```

```
                NextHState   =    2'b11;
        2'b11:
            if (EndJ0)
                NextHState   =    2'b00;
            else
                NextHState   =    2'b11;
        endcase
        end

        assign    EndHState    =EndJ0;
assign    EndCtl=EndHState&(Command==4'b0010)|EndABState|EndAEState;
assign    StartHCal =(HCommand|HABCommand|HAECommand)&~FastMulMode;
```

4　对于其他子功能块代码，应以时序子电路及其相关组合电路分组，每个分组必须将组合电路部分与时序电路部分分开描述，并分别加以注释；组合电路采用 assign 语句。

例如：

```
        //   4) ELen
        //   ==============
        always   @(posedge Clk or negedge Resetn)
        begin
                if (!Resetn)
                  begin
                        ELenRegL   <= 8'b0      ;
                        ELenRegH   <= 4'b0      ;
                  end
                else
                  begin
                        ELenRegL   <=   NextELenL;
                        ELenRegH   <=   NextELenH;
                  end
        end
        assign    NextELenL    =ELenSelL&RsaReg_Wr ? RsaReg_Data_w
                :ELenRegL;
        assign    ELenSelL     =RsaReg_Sel&(RsaReg_Addr==`ELENLREG);
        assign    NextELenH    =ELenSelH&RsaReg_Wr ? RsaReg_Data_w[3:0]
                :ELenRegH;
        assign    ELenSelH     =RsaReg_Sel&(RsaReg_Addr==`ELENHREG);
        assign    ELenReg      = { ELenRegH, ELenRegL };
                    assign    ELen          =ELenReg;
```

9 位宽对齐。运算符号代码特殊处理。

4.3.6VERILOG HDL 代码文件注释行

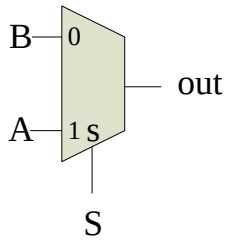
1 Verilog HDL 源代码各部分要有“//---------”和空行隔离；
2 源代码的功能描述的主体前应有本 module 的功能简介 overview；
3 各端口信号、变量、参数、子模块引用等尽量多添加注释说明；
4 不允许使用“/* …. */”的注释方法，提倡使用“//”；
5 不允许用中文注释。
6 在描述状态机的时候，每一个状态的功能，翻转条件，翻转后进入的状态，和产生的控制信号要作简单的介绍；
7 每个代码文件的结束的时候要用“//--======= END =====--”进行声明；

4.3.7VERILOG HDL 代码文件独立Include.v

不允许在 Verilog 模块设计代码文件中添加`Include 语句说明，必须单独编写独立的 XXX_Include.v 文件，将本模块包含的所有 Verilog 源代码用`Include 语句说明。不能在模块中定义`timescale，而是在 include.v 中添加。

5 VERILOG HDL代码范例

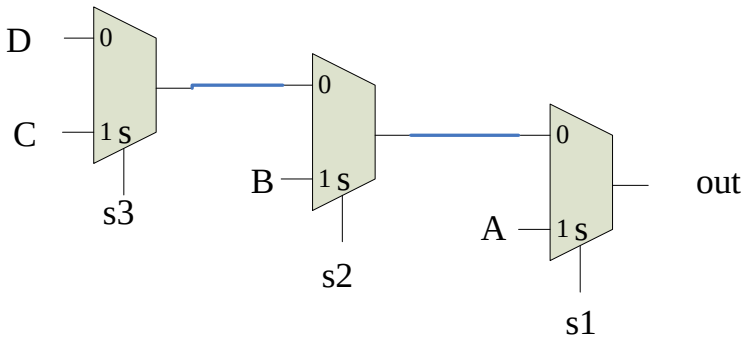
5.1 复用器表达方式



2 input MUX

图 5-1 MUX

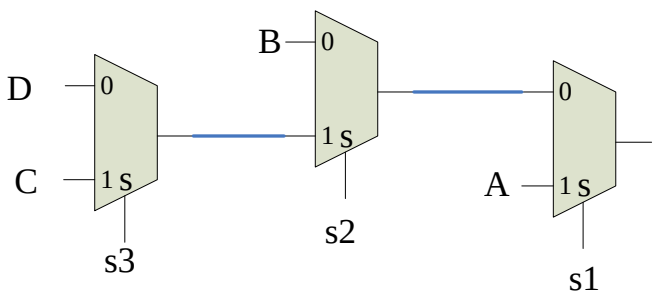
```
assign    out    =    S    ?    A         :    B;
或
assign    out    =    !S   ?    B         :    A;
```



cascaded MUX

图 5-2 多级 MUX 例 1

```
assign    out    =    s1    ?    A    :
                      s2    ?    B    :
                      s3    ?    C    :
                                 D    ;
```



cascaded MUX

图 5-3 多级 MUX 例 2

```
assign    out    =    s1    ?    A    :
                      !s2   ?    B    :
                      s3    ?    C    :
                                 D    ;
```