规佳

VerilogHDL 代码

规范

版本：v1.0

上海规佳智能科技有限公司

2025 年 1 月 16 日

1 版本

版本：1.0

新建。

2 简介

此规范是约束一种 VerilogHDL 代码风格的规范，目的是为让编程人员能够快速编写以及仿真验证修改代码，令代码风格能够统一，方便维护。

代码需要做到简洁、清楚、清晰，用最少的文字表达相应的逻辑。若冗长：一，字打的多编写慢；二，文字多，重要信息被淹没，问题难定位；三，行数多，上下查找慢。

故该规范多数与文字摆放相关。

3 特征规范

规范1: 变量表达必须分离;

对于 reg 信号，尽可能的放在自己单独的 always 块中。除非其中变量为一组信号，变化完全一致，否则不可放在同一个 always 模块之中。

例如图 1 代码，不规范代码一眼看不清每个信号逻辑作用，几个信号混在一起，犹如乱麻，逻辑上交织不开，增加调试困难。而分离的代码能够明显将信号逻辑作用显示。阅读方便，若逻辑出错，修改起来，很是方便。

```
14  //不规范
15  always @(posedge clk )
16      if( rst  ) begin
17          cnt     <= 4'd0;
18          clk_div <= 1'b0;
19      end
20      else if(cnt < NUM_DIV / 2 - 1) begin
21          cnt     <= cnt + 1'b1;
22          clk_div <= clk_div;
23      end
24      else begin
25          cnt     <= 4'd0;
26          clk_div <= ~clk_div;
27      end
```

```
29  //规范
30  always @(posedge clk )
31  if( rst )                           cnt<= 4'd0;
32  else if(cnt == NUM_DIV / 2 -1 )     cnt<= cnt + 1'b1;
33  else                                cnt<= 4'd0;
34
35  always @(posedge clk )
36  if( rst_n)                          clk_div<= 1'b0;
37  else if(cnt == NUM_DIV / 2  -1 )    clk_div<= !clk_div;
```

图 1 代码示例图

另外例如图 2 代码，所有变量行为一致，可以放入一个 always 块中。

```
4   always@(posedge clk)
5   if( rst )           {en , cntNum, overTime }<='h0;
6   else if( bram_en & bram_we & bram_addr=='h2 )
7                       {en , cntNum, overTime }<={ bram_wdata[31] , bram_wdata[20:16] , bram_wdata[9:0] };
```

图 2 多个变量一个逻辑块示例图

分离原因:

分离编写，每个信号，逻辑清晰。何时赋值、何时变化，一目了然，书写方便，若仿真中，发现错误，定位修改，亦为方便。

当编写某信号时，不必考虑其它信号赋值，只需专一考虑，当前信号逻辑关系，在何种条件下，进行何种变化，或者需要如何变化，其条件是什么。

代码字数少，错误概率下降；代码行数低，滚动翻页查找信号容易。写作速度提升，成功率提升。

这一点非常关键，请按这样书写。

规范2: 左条件右赋值

在 always 块中，尽可能将条件和赋值放在一行，多个条件赋值行，尽量进行列对齐，这就形成左条件右赋值结构，如此表达，文字就如实际电路图一样。如图 1 所示。若一行太长，可将赋值换行等，如图 2。

```
always @(posedge clk )
if( rst )                          cnt<= 4'd0;
else if(cnt == NUM_DIV / 2 -1 )    cnt<= cnt + 1'b1;
else                               cnt<= 4'd0;

always @(posedge clk )
if( rst_n)                         clk_div<= 0;
else if(cnt == NUM_DIV / 8  -1 )   clk_div<= 1;
else if(cnt == NUM_DIV / 6  -1 )   clk_div<= 0;
else if(cnt == NUM_DIV / 4  -1 )   clk_div<= 1;
else if(cnt == NUM_DIV / 2  -1 )   clk_div<= 0;
```

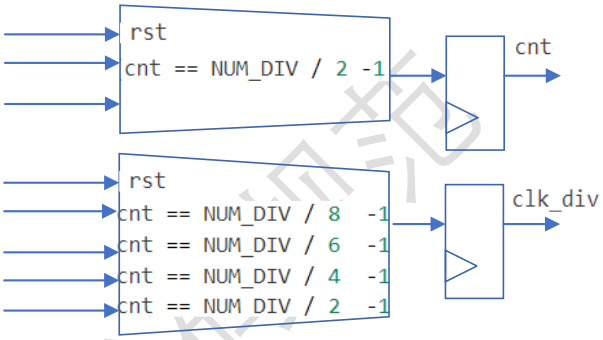

图 3 左条件右赋值示例图

若相关 EDA 工具不允许同行，则信号赋值部分换行后也应尽量往右侧缩进。

规范3: 少用 begin……end 块

如果一个逻辑块中只有一个信号，begin end 可以省去了。

原因：减少字数，让逻辑部分凸显；减少行数，减少上下翻动页面的操作，会节省很多时间；行数多了，不容易定位问题。如图 4 所示，不规范代码占用 11 行，而规范代码仅占 4 行，代码行数大大降低；另外，规范代码，一眼可分辨 axis_tvalid 信号何时变为 1，何时变为 0，而不规范代码不易分辨 axis_tvalid 信号行为；

```
55  //不规范
56  always @(posedge clk )begin
57      if( rst )begin
58          axis_tvalid<= 1'd0;
59      end
60      else if( axis_tvalid & axis_tready )begin
61          axis_tvalid<= 1'd0;
62      end
63      else if( tx )begin
64          axis_tvalid<= 1'd1;
65      end
66  end
```

```
69  //规范
70  always @(posedge clk )
71  if( rst )                                axis_tvalid<= 1'd0;
72  else if( axis_tvalid & axis_tready )axis_tvalid<= 1'd0;
73  else if( tx )                            axis_tvalid<= 1'd1;
```

图 4 有无 begin…end 对比示例图

若相关 EDA 工具不允许非必要 begin end 缺失，需要添加，但也尽量少用。

规范4: 使用规佳多段式状态机;

不得使用一段式状态机，而使用规佳多段式状态机。其特性：

1， 继承于 3 段式状态机；

2， 结合规范 1：就演变为多段式状态机；

3， 使用 One-Hot 编码，方便阅读。ST[IDLE]，就是一个寄存器，不是 ST==IDLE 的比较组合逻辑，FPGA 中大量寄存器资源，ASIC 可能存在扣寄存器问题。若寄存器资源紧缺，可通过综合工具设定状态机类型。

模板如下：

```
1  localparam  IDLE            = 0;
2  localparam  HEAD            = 1;
3  localparam  DATS            = 2;
4
5  (*keep="true"*)
6  reg [DATS:IDLE]    ST;
7  reg [DATS:IDLE]   nST;
8
9  //synthesis translate_off
10 reg [63:0] ST_STRING;
11 always @(*)
12   case(1'b1)
13     ST[IDLE]      : ST_STRING = "IDLE";
14     ST[HEAD]      : ST_STRING = "HEAD";
15     ST[DATS]      : ST_STRING = "DATS";
16     default       : ST_STRING = "    ";
17   endcase
18 //synthesis translate_on
19
20 always@(posedge clk)
21 if( rst )       ST <= 'd1;
22 else            ST <= nST;
```

```
24 always@*
25 begin
26   nST = 'd0;
27   case(1'b1)
28     ST[IDLE]:
29       if( axis_tvalid  )        nST[HEAD] = 1'b1;
30       else                      nST[IDLE] = 1'b1;
31     ST[HEAD]:
32       if( axis_tvalid & axis_tready & axis_tlast  )     nST[IDLE]  = 1'b1;
33       else if( axis_tvalid & axis_tready & pktOver  )   nST[DATS]  = 1'b1;
34       else                                              nST[HEAD]  = 1'b1;
35
36     ST[DATS]:
37       if( axis_tvalid & axis_tready & axis_tlast  )     nST[IDLE]  = 1'b1;
38       else                                              nST[DATS]  = 1'b1;
39
40     default:                                            nST[IDLE]  = 1'b1;
41   endcase
42 end
43
44 assign axis_tready =  ST[HEAD] | ST[DATS] ;
45
46 reg [7:0]  pcnt;
47
48 always@(posedge clk)
49 if( rst )                          pcnt <= 'd5;
50 else if( ST[IDLE] & nST[HEAD] )    pcnt <= 'h5;
51 else if( ST[HEAD] & axis_tvalid & axis_tready ) pcnt <= pcnt - 1;
52
53 assign pktOver = pcnt ==1 ;
```

图 5 规佳状态机示例图

第一段：各个状态定义段。用于定义各个状态。

第二段：状态信号定义段。声明定义状态变量。

第三段：调试信号赋值段。仿真调试时，可将该信号变为 ASIC 显示，不需要文字对照，即可获知当前何种状态。

第四段：状态信号赋值段。当前状态行为定义，一般不需要改变。

第五段：下一状态转移段。描述状态机转移行为。这一段将状态转移展现十分清晰，故一般写状态时，并不需要再画流程图之类，代码描述就是流程图。

第六段：各个信号赋值段。根据当前状态以及条件，对各个信号进行描述。

不必背诵段数，使用时，复制粘贴，修改状态名，编写状态转移部分，尽量一个模块一个状态机，各模块 ST 名称不需要改，所有状态机模块，都是使用一个 ST。

并不需要，区分摩尔，还是米勒，各个信号，根据状态，以及条件，分别编写，信号赋值，逻辑正确即可。如遇后面，时序不足，在最长路径，则可以将，有些信号，逻辑输出，改为寄

存器输出。

状态机犹如树干，各信号犹如华果，写状态机如画树，画好树干添华果。

规范5: 使用 for generate 例化多个模块

当一次调用多个相同模块时，应当使用 generate 例化多个模块。如此可一段代码例化多个实例，大大减少行数，减少错误。

如下图所示，一行代码可生成多个 DAC 模块的例化，并且可通过传递参数进行更改。一个 dac81416 有 16 个 DAC，如此例化，可以控制 16*CHIP_NUM 个 DAC。

```
genvar i;
generate
   for (i=0; i< CHIP_NUM; i=i+1)
   begin: DAC81416
     dac81416_Top dac81416_Top(

                .clk               (     clk_100m                )
               ,.rst               (     rst_100m                )

               ,.da_tvalid         (     daXA_tvalid       [16*i     +: 16    ]       )
               ,.da_tready         (     daXA_tready       [16*i     +: 16    ]       )
               ,.da_din            (     daXA_din          [16*16*i  +: 16*16 ]       )

               ,.da_init           (     da_init           [i]      )

               ,.dac81416_SDO      (     dac81416_SDO      [i]      )
               ,.dac81416_SCLK     (     dac81416_SCLK     [i]      )
               ,.dac81416_SDI      (     dac81416_SDI      [i]      )
               ,.dac81416_csn      (     dac81416_csn      [i]      )

               ,.dac81416_toggle   (     dac81416_toggle   [3*i +:3 ]      )
               ,.dac81416_ldacn    (     dac81416_ldacn    [i]      )
               ,.dac81416_rstn     (     dac81416_rstn     [i]      )
               ,.dac81416_clrn     (     dac81416_clrn     [i]      )
               ,.dac81416_almoutn  (     dac81416_almoutn  [i]      )
     );
   end
endgenerate
```

图 6 使用 for generate 例化多个模块示意图

规范6: 使用 Axis 接口 FIFO，或者使用空信号与数据同时有效的 FIFO。

不推荐使用标准 fifo，因为标准 fifo，当 empty 信号拉低说明有数据时，有效数据不能在 rddata 信号给出，需要 rd 读有效后，下一拍有效数据才能在 rddata 信号给出，来回占用 3 个周期，使用极为不便。

首选推荐使用 axis 接口 fifo，时序控制容易，甚至多个 fifo 可以首尾连接。

若不能使用 axis 接口 fifo，使用类似 first word passthough 模式（Xilinx），show ahead 模

式（Intel Altera）。

4 命名规范

规范7: 信号名与协议规范中名称一致

在有些协议规范中，定义了信号、状态或状态跳转条件名称，请尽量按照协议中名称命名。方便与协议对照，出现问题修改。例如 SATA 协议、FC 协议。

```
localparam    HR_IDLE           = 'd00;
localparam    HR_Reset          = 'd01;
localparam    HR_AwaitCOMINIT   = 'd02;
localparam    HR_AwaitNoCOMINIT = 'd03;
localparam    HR_Calibrate      = 'd04;
localparam    HR_COMWAKE        = 'd05;
localparam    HR_AwaitCOMWAKE   = 'd06;
localparam    HR_AwaitNoCOMWAKE = 'd07;
localparam    HR_AwaitAlign     = 'd08;
localparam    HR_SendAlign      = 'd09;
localparam    HR_Ready          = 'd10;
localparam    HR_Partial        = 'd11;
localparam    HR_Slumber        = 'd12;
localparam    HR_AdjustSpeed    = 'd13;
```

```
nST = 'h0;
case(1'b1)
  ST[HR_IDLE]:
    if( phy_ready  )                nST[HR_Reset]          = 1'b1;
    else                            nST[HR_IDLE]           = 1'b1;

  ST[HR_Reset]:
    if( txcomfinish )               nST[HR_AwaitCOMINIT]   = 1'b1;
    else                            nST[HR_Reset]          = 1'b1;
  ST[HR_AwaitCOMINIT]:
    if( rxcominitdet )              nST[HR_AwaitNoCOMINIT] = 1'b1;
    else if(  timeCountDone )       nST[HR_Reset]          = 1'b1;
    else                            nST[HR_AwaitCOMINIT]   = 1'b1;

  ST[HR_AwaitNoCOMINIT]:
    if( rxcominitdet )              nST[HR_AwaitNoCOMINIT] = 1'b1;
    else                            nST[HR_Calibrate]      = 1'b1;
```

图 7 信号名与协议名一致示例图

规范8: 多单词信号，使用大小写、或下划线区分。

多个单词组成的一个信号，需要使用大小写或下划线形式区分，例如 axisTvalid、axis_tvalid、AxisTvalid、Axis_Tvaild。

规范9: 信号名不得加入输入、输出等属性

信号名中不需要加入信号的输入、输出、wire、reg 等属性。例如一个名叫 cnt 的信号，不需要添加 cnt_o，cnt_i，cnt_w，cnt_r 指明 cnt 是输入信号？输出信号？wire 信号？reg 信号？

```
//不规范
module clkDiv#(
    parameter NUM_DIV = 6
    )(
    input          clk_i
    ,input         rst_i

    ,input         clk_en_i
    ,output reg    clk_div_o
);
    reg [3:0]   cnt_r;
    wire        timeOut_w = cnt_r==0 ;
```

```
//规范
module clkDiv#(
    parameter NUM_DIV = 6
    )(
    input          clk
    ,input         rst

    ,input         clk_en
    ,output reg    clk_div
);
    reg [3:0]   cnt;
    wire        timeOut = cnt==0 ;
```

图 8 信号名规范示例图

规范10: 信号命名要有意义，符合信号实际意义，尽量使用常用字命名

若信号名都是 a、b、c、d、temp 之类，代码很是难度，维护更加困难，故而信号起名字要起的有意义，尽量符合信号含义本身，如：

Cnt：计数器；

pCnt：packet cnt，数据包计数；

tCnt：time cnt，时间计数；

bCnt：bit cnt，bit 计数；

sel、arb：select，arbiter 仲裁选择；

5 模块规范

规范11： 使用 Verilog-2001 规范将定义模块

遵循 Verilog-2001 版规范，一次性在 model 中声明信号及输入输出类型，不需要全部将信号列入 model 中，再次声明输入输出类型。

尽量对齐，方便例化使用。将每行的逗号分隔符放在每行的前面，也是不错的选择。

```
//不规范
module clkDiv(  clk , rst , clk_div );
    parameter       NUM_DIV = 6;
    input           clk         ;
    input           rst         ;
    output          clk_div     ;

    reg             clk_div     ;

```

```
//规范
module clkDiv#(
    parameter NUM_DIV = 6
    )(
    input           clk
    ,input          rst

    ,output reg     clk_div
);
```

图 9 模块定义规范示例图

规范12： 信号分组，空行隔开

尽量将不同种类的信号，进行分组，用空行分隔，必要时添加注释。

```
module gjSPI_Master #(
        parameter CLK_DIV   = 8'd125     //spi clk div
        ,parameter ADDR_LEN  = 7
        ,parameter NUMB_LEN  = 3
        ,parameter DATA_LEN  = 16
    )(

        input                       clk
        ,input                      rst

        ,input                      cmd_tvalid          //rcv cmd
        ,output                     cmd_tready
        ,input      [NUMB_LEN-1:0]  cmd_number
        ,input                      cmd_type            // 1:read 0:write
        ,input      [ADDR_LEN-1:0]  cmd_addr
        ,input      [DATA_LEN-1:0]  cmd_din
        ,output reg                 cmd_din_tready

        ,output reg [DATA_LEN-1:0]  cmd_dout            //back data
        ,output reg                 cmd_dout_tvalid

        ,input                      spi_SDO             //spi interface
        ,output reg                 spi_SCLK
        ,output                     spi_SDI
        ,output reg                 spi_csn
);
```

图 10 数据分组示例图

规范13： 一个模块一个文件

一个文件中仅包含一个模块。

规范14： 文件头部或尾部添加文件说明

文件说明可以添加在文件头部或者尾部。

```
`timescale 1ns / 1ns
//////////////////////////////////////////////////////////////////////////////////
// Company:   guiJia
// Engineer:  kejie ma
//
// Create Date: 2021/01/27 09:33:28
// Design Name:
// Module Name: clockReset
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

图 11 模块说明示例图

规范15: 模块例化时，模块信号名尽量与外部信号名保持一致，文字分开，不可紧凑。

在模块例化时，最好保证信号的信号名与外部信号名一致，方便查找对信号查找。

```
//不规范
clkDiv#(
    .NUM_DIV  (     NUM_DIV )
    )clkDiv(
     .clk_i          (    clk_w          )
    ,.rst_i          (    rst_w          )

    ,.clk_en_i       (    clk_en_w       )
    ,.clk_div_o      (    clk_div_w      )
);
```

```
//规范
clkDiv#(
    .NUM_DIV  (     NUM_DIV )
    )clkDiv(
     .clk            (    clk            )
    ,.rst            (    rst            )

    ,.clk_en         (    clk_en         )
    ,.clk_div        (    clk_div        )
);
```

图 12 模块例化信号名示例图

6 表达规范

规范16: 尽量使用位运算

例如 if(rst ==1)，可以写成 if(rst)；例如：

if(a ==1 && b ==0 && c ==1)

if(a & ~b & c)

应写成后者，为减少字数，突出电路，突出逻辑。

切记 VerilogHDL 描述的是心中的电路。

规范17: 使用 always@*表达敏感信号列表

always@(a or b or c)容易遗漏信号，并带来书写麻烦，建议使用 always@*或 always@(*)代替。

规范18: 组合逻辑用阻塞赋值，寄存器用非阻塞赋值

always@*中的 block 用“=”赋值，always@(posedge clk)中用“<=”赋值。

规范19: 组合逻辑块中避免 Latch 产生

在 always@*中，变量若没有提前设定初值，if 中没有 else，或 case 中没有 default，综合时，就会产生 Latch，影响时序分析，影响电路功能。需要避免。

若需要 Latch，则需单独说明。

7 复位规范

规范20: 复位模块使用异步复位同步撤销。

为保证 recoverytime 和 removaltime 两个时间约束，需要使用异步复位、同步撤销的复位电路。复位电路代码如下图所示。

```
185  module rstBlock#(
186          parameter DL    = 2
187    )(
188          input           areset
189          ,input          clk
190
191          ,output reg     rst
192          ,output reg     rstn
193  );
194
195  reg [DL-1 :1 ] rstDelay;
196
197  always@(posedge clk or posedge areset)
198  if( areset )    rstDelay<={DL-1{1'b1}};
199  else            rstDelay<=rstDelay<<1;
200
201  always@(posedge clk or posedge areset)
202  if( areset )    rst<=0;
203  else            rst<=rstDelay[DL-1];
204
205  always@(posedge clk or posedge areset)
206  if( areset )    rstn<=0;
207  else            rstn<=!rstDelay[DL-1];
208
209  endmodule
```

图 13 复位模块示例图

规范21: 模块内部使用同步复位，顶层复位模块提供高、低同步复位信号。

按照 Xilinx 官方建议，为避免异步复位信号，扇出过大，占用时钟树资源，提倡同步高电平复位，甚至不需要复位。不用复位较难，故模块内部使用同步复位。

在有些设计中，无法保证全部都是高电平复位，故在复位模块设计时，同时给出该时钟的同步高低电平复位：clk_100m；rst_100m；rst_n_100m；

```
module rstTop(
        input           areset
        ,input          clk_100m
        ,input          clk_125m
        ,input          clk_200m

        ,output         rst_100m
        ,output         rst_125m
        ,output         rst_200m

        ,output         rstn_100m
        ,output         rstn_125m
        ,output         rstn_200m
);
wire  aresets   = { rst_125m , rst_100m , areset    };
wire  clks      = { clk_200m , clk_125m , clk_100m  };
wire  rsts      = { rst_200m , rst_125m , rst_100m  };
wire  rstns     = { rstn_200m, rstn_125m, rstn_100m };

   genvar i;
   generate
      for (i=0; i< CHIP_NUM; i=i+1)
      begin: RST

          rstBlock#(
              .DL             (    2    )
          )rstBlock(
              .areset    (    areset    )
              ,.clk      (    clks    [i]   )

              ,.rst      (    rsts    [i]   )
              ,.rstn     (    rstns   [i]   )
          );
      end
   endgenerate
```

```
wire        clk_100m   ;
wire        clk_125m   ;
wire        clk_200m   ;

wire        rst_100m   ;
wire        rst_125m   ;
wire        rst_200m   ;

wire        rstn_100m  ;
wire        rstn_125m  ;
wire        rstn_200m  ;

rstTop rstTop(
        .areset         (    areset         )
        ,.clk_100m      (    clk_100m       )
        ,.clk_125m      (    clk_125m       )
        ,.clk_200m      (    clk_200m       )

        ,.rst_100m      (    rst_100m       )
        ,.rst_125m      (    rst_125m       )
        ,.rst_200m      (    rst_200m       )

        ,.rstn_100m     (    rstn_100m      )
        ,.rstn_125m     (    rstn_125m      )
        ,.rstn_200m     (    rstn_200m      )
);
```

图 14 顶层模块复位示例图

规范22： 注意各时钟复位撤销次序

如上图所示，可以在 aresets 中更改复位撤销次序，由此可见，areset 将控制所有复位，撤销时，rst_100m 先撤销，rst_125m 后撤销，最后是 rst_200m 撤销。

8 多时钟规范

规范23： 单位宽信号跨时钟域需要使用双寄存器同步，或者使用异步 FIFO 同步

当信号跨时钟域时，不可避免的会出现亚稳态。亚稳态主要是寄存器特性导致。MOS 版的寄存器如下图所示，主要由反相器和传输门组成。信号能够锁存，主要是非门作用，而非门结构及特性如下图所示。

假定非门特性为 y=f(x)；f(0.4)=0.6； f(0.6)=0.3；。

若 clock 上升沿到来时，采集到了中间电平，如 0.4v，那么 f(0.4)=0.6； f(f(0.4))=0.4；f(0.4)=0.6； f(f(0.4))=0.4；若非门特性非常理论对称，则输出也是中间电平；实际上并不对称有些差异，可想而知，电平会经过两个非门无限次迭代，最终收敛稳定。非门上下 mos 管越不对称，收敛越快。所以 ASIC 会将 2 个做的不对称。

若只有 1 级寄存器，输出的 Q，还处于迭代中，无法满足后面的逻辑时序；经过 2 级迭代，会令收敛时间大幅降低，可以满足后续逻辑时序。故跨时钟必须要两级寄存器同步。

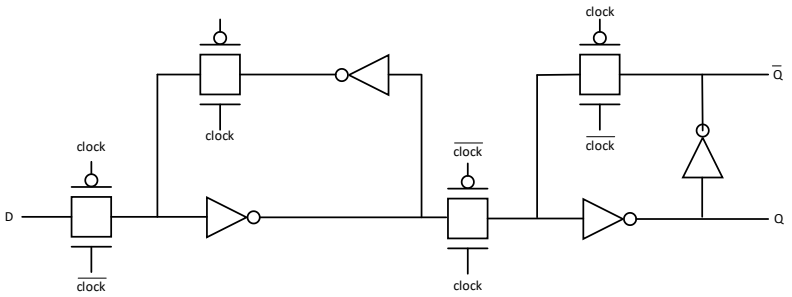

图 15 寄存器原理图

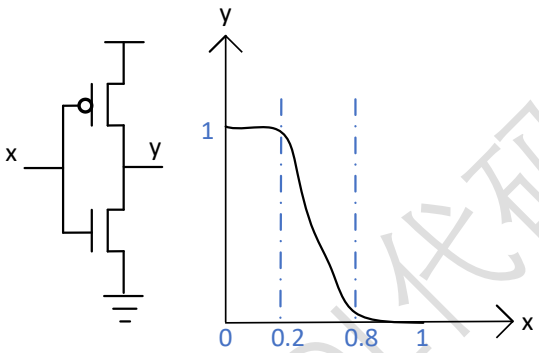

图 16 非门原理图

双所存，可以保证单位宽信号无误跨时钟域传输。

规范24： 使用异步 FIFO 进行多位宽信号跨时钟域

多位宽数据跨时钟域传递时，应使用异步 FIFO，尽量不要跨时钟域使用，因为添加约束时，需要将这些路径 faultpass 掉。当信号多时，将是一个复杂的工作。

所以多位宽信号使用异步 FIFO 进行跨时钟域。

另外，若跨时钟域寄存器太多，可以在架构上考虑，将前面的总线信号，进行跨时钟域传输。

规范25： 使用时钟使能模拟低频时钟

有些时候，某些模块时钟需求较低，而内部工作时钟较高，若使用较低时钟，会占用一部分时钟资源，还牵扯数据跨时钟域问题，这时候可以考虑由内部高频工作时钟，产生低频时钟使能，而低频模块工作时，寄存器工作在高频使能，但是由时钟使能控制更新。如下图串口发送逻辑代码。

```
module gjUartTx(
        input             rst
        ,input            clk
        ,input            clk_div

        ,input            tx_tvalid
        ,output           tx_tready
        ,input     [7:0]  tx_tdata

        ,output           uartTx
        ,output reg       uartTx_en
);

reg     [9:0]  txData;
always@(posedge clk )
if( rst )                           txData<= 10'h3ff;
else if( tx_tvalid & !uartTx_en  )  txData<={1'b1, tx_tdata, 1'b1 } ;
else if( clk_div )                  txData<={1'b1 ,txData [9:1] };

assign uartTx = txData[0] ;

reg     [9:0]  shift;
always@(posedge clk )
if( rst )                           shift<= 10'h0;
else if( tx_tvalid & !uartTx_en  )  shift<= 10'h3f;
else if( clk_div )                  shift<={1'b0 ,shift [9:1] };

assign uartTx_en = shift[0] ;

assign tx_tready = shift[1:0] ==1 & clk_div ;

endmodule
```

图 17 使用时钟使能模拟低频时钟