目录

[1. 用任意语言写1:100的质数（7.17） 2](#_Toc86266947)

[2. 奇数分频（三分频）（7.18） 3](#_Toc86266948)

[3. 偶数分频（二分频）（7.19） 5](#_Toc86266949)

[4. 画出可以检测10110串的状态转移图，要求每检测到一次该序列，输出两个周期的高电平信号，用低功耗方式（7.21） 6](#_Toc86266950)

[5. 用verilog实现1bit信号边沿检测功能，输出一个周期宽度的脉冲信号 8](#_Toc86266951)

[6. 异步复位，同步释放（7.22） 9](#_Toc86266952)

[7. 跨时钟域（快时钟到慢时钟，或者是慢时钟到时钟）(7.23) 10](#_Toc86266953)

[8. 序列信号发生器，在clk信号作用下周期性输出“0010110111” 12](#_Toc86266954)

[9. 描述带进位输入、输出的两位全加器。端口：A/B为加数，CIN为进位输入，S为和，COUT为进位输出（7.24） 13](#_Toc86266955)

[10. 写一个同步FIFO，给定深度和位宽(7.25) 14](#_Toc86266956)

[11. 异步FIFO测试点：虽然不会写代码，但是要知道思想(7.26) 16](#_Toc86266957)

[12. 握手（7.27） 17](#_Toc86266958)

[13. 写一个100MHz的时钟(7.28) 18](#_Toc86266959)

[14. 红绿灯（7.29） 19](#_Toc86266960)

[15. 贩卖机（7.30） 19](#_Toc86266961)

[16. 写断言和随机约束（7.31） 19](#_Toc86266962)

[17. 用Verilog实现glitch free时钟切换电路。输入sel，clka，clkb，sel为1输出clka，sel为0输出clkb 19](#_Toc86266963)

[18. CDC 跨时钟域 20](#_Toc86266964)

[19. 找出8bit中第一次出现1的个数verilog 20](#_Toc86266965)

[20. 串并转换 20](#_Toc86266966)

[21. 对下述代码，使用“加法器”和“选择器”画出最优的电路图（电路面积最小） 21](#_Toc86266967)

[22. 仲裁器 22](#_Toc86266968)

[23. 从100个数中找到最大的两个数 23](#_Toc86266969)

[24. Verilog实现8bit串行数据转并行数据 25](#_Toc86266970)

[27.脉冲检测 26](#_Toc86266971)

[28.10个10bit相加，需要寄存器多少bit？ 26](#_Toc86266972)

[29.每10个数求和并输出，要求资源最少 26](#_Toc86266973)

[30、4bit乘法器（门电路） 28](#_Toc86266974)

[31、格雷码和二进制之间的转换 28](#_Toc86266975)

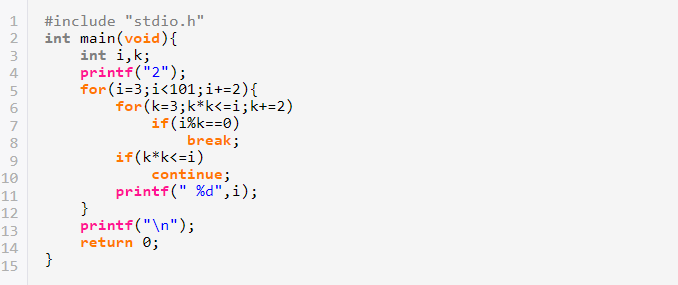
[32、用Verilog语言编写module，实现： 29](#_Toc86266976)

# 用任意语言写1:100的质数（7.17）

质数是只能被1和它本身整除

C语言:

先对一个数开方，得到k，开始便利遍历，如果这个数可以整除(k+1)内的数，则这个数不是质数，否则输出这个数



Matlab:

for i = 2:100 %外层循环，i的初值是2，终值为100

for j = 2:100 %内层循环，j的初值为2，终值为100

if (~mod(i,j)) %i除以j取余后再取反

break; %跳出循环

end

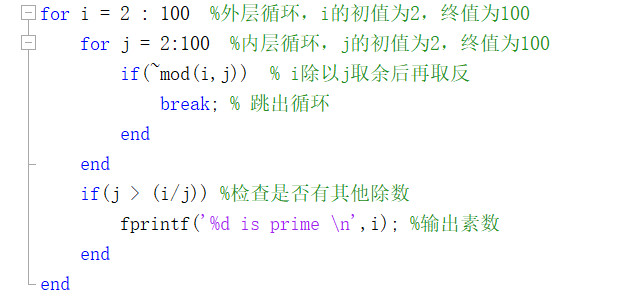
end

if(j > (i/j))

fprintf('%d is prime \n',i);

end

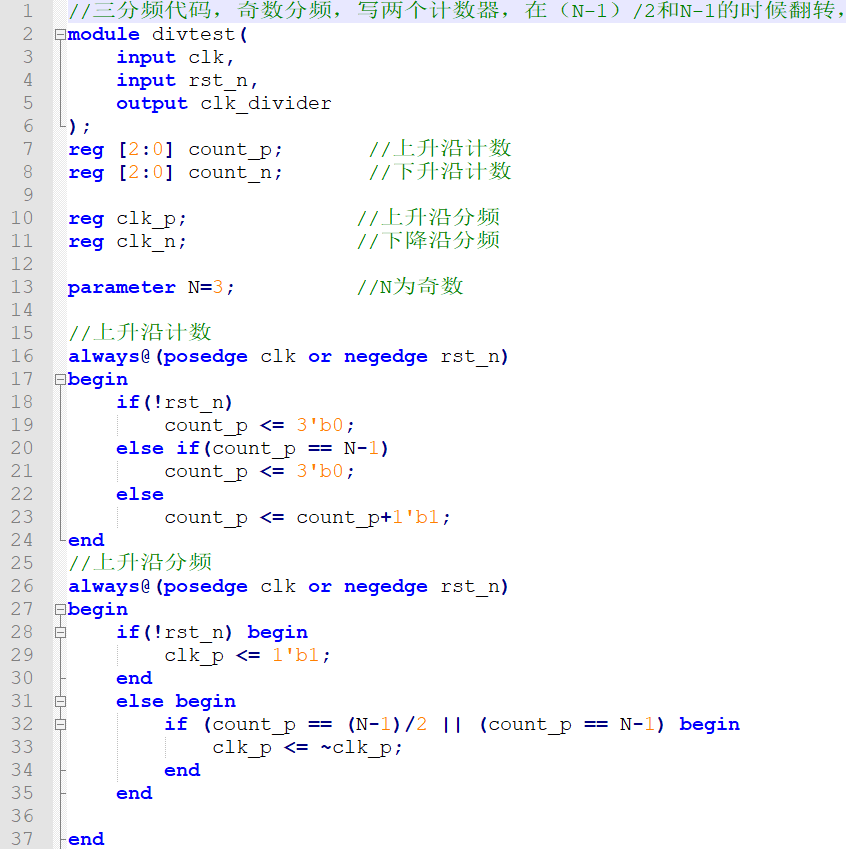
end

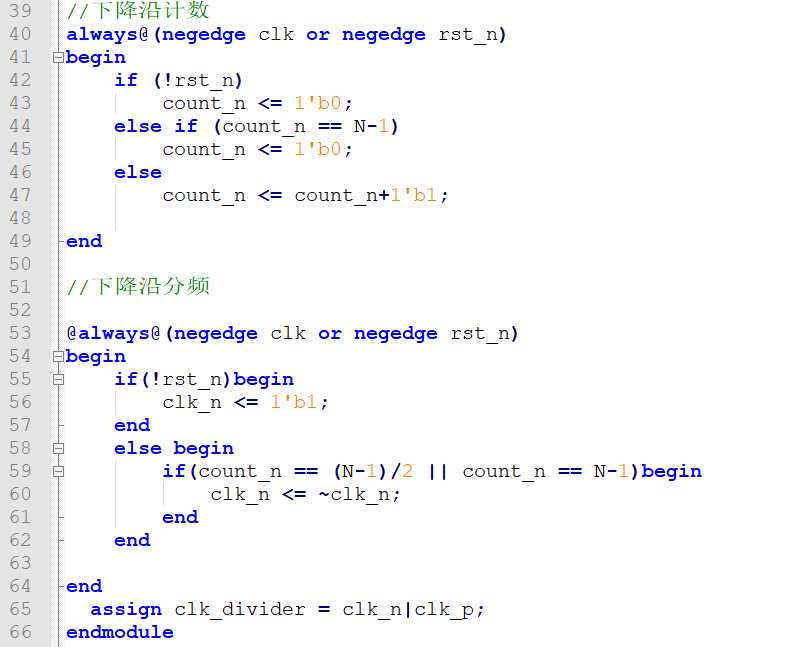


# 奇数分频（三分频）（7.18）

[偶数、奇数、小数分频器\_AD7533的博客-CSDN博客](https://blog.csdn.net/AD7533/article/details/109776437?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_baidulandingword-1&spm=1001.2101.3001.4242)

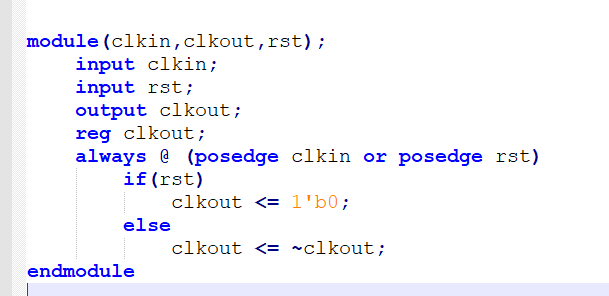
奇数分频：写两个计数器和分频器（分别是上升沿计数、下降沿计数，上升沿分频、下降沿分频），分频器在（N-1）/2和N-1的时候翻转，最后再将两个分频器或



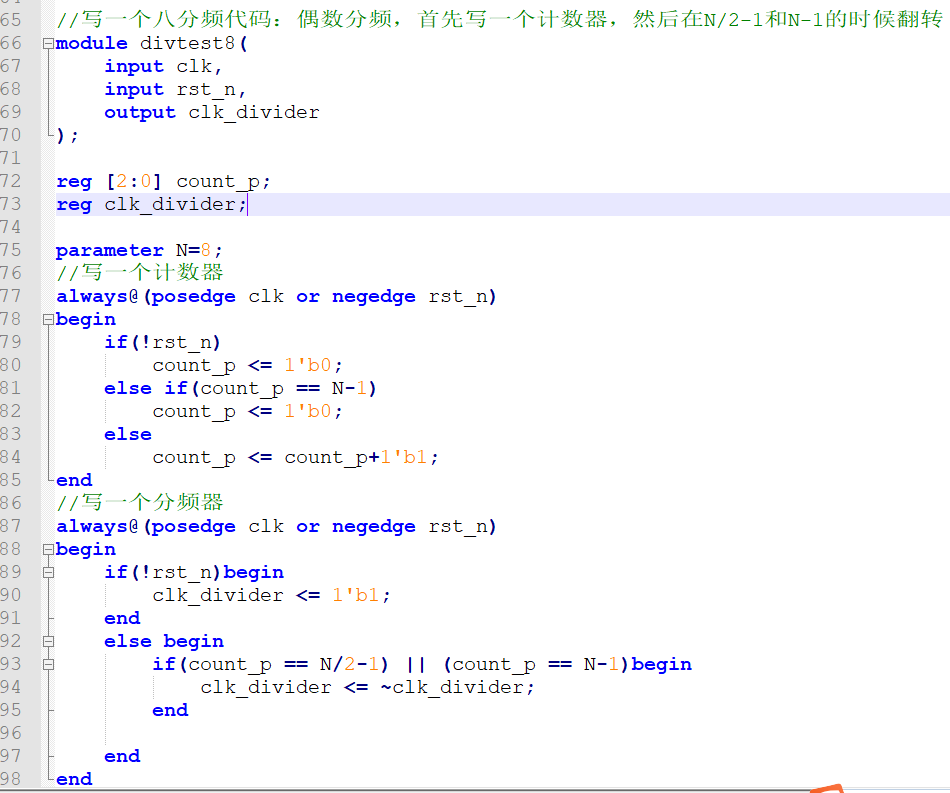


# 偶数分频（二分频）（7.19）

二分频:



首先定义参数，N分频就是N，偶数分频的时候写一个N位的计数器，再写一个分频器，在N/2-1和N-1的时候翻转



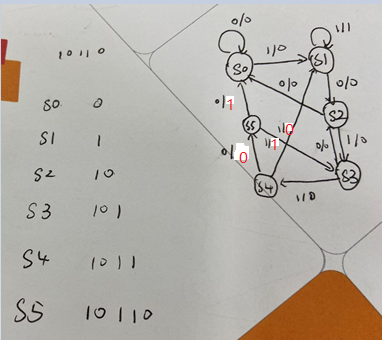
# 画出可以检测10110串的状态转移图，要求每检测到一次该序列，输出两个周期的高电平信号，用低功耗方式（7.21）

（一段式描述方法：整个电路用一个进程描述，状态转移条件的判断、状态输出和状态寄存器转移

二段式描述方法：第一个进程描述次态到现态转移，第二个进程描述状态转移条件的判断，组合逻辑输出

）

三段式状态机：第一个always块，描述对应当前状态的状态寄存器，后两个描述状态转移和输出



module FSM(

input clk,rst,

input data,

output flag

);

parameter s0=5’b00000;

parameter s1=5’b00001;

parameter s2=5’b00010;

parameter s3=5’b00101;

parameter s4=5’b01011;

parameter s5=5’b10110;

reg [4:0] state;

always @(posedge clk or negedge rst)begin

if(!rst) state<=s0;

else begin

case(state)

s0:

if(data) state<=s1;

else state<=s0;

s1:

if(data) state<=s1;

else state<=s2;

s2:

if(data) state<=s3;

else state<=s0;

s3:

if(data) state<=s4;

else state<=s2;

s4:

if(data) state<=s1;

else state<=s5;

s5:

if(data) state<=s1;

else state<=s3;

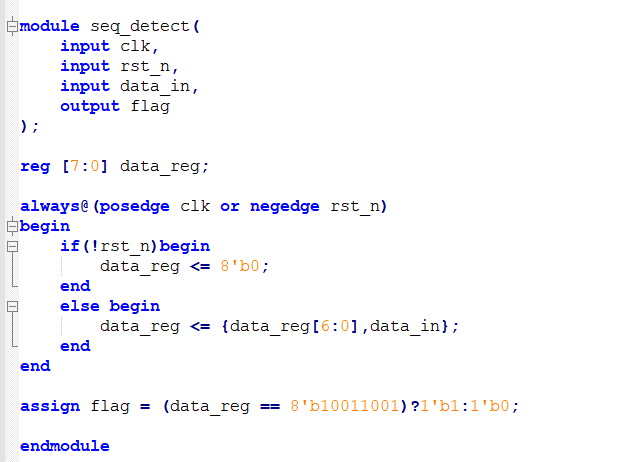
endcase

end

end

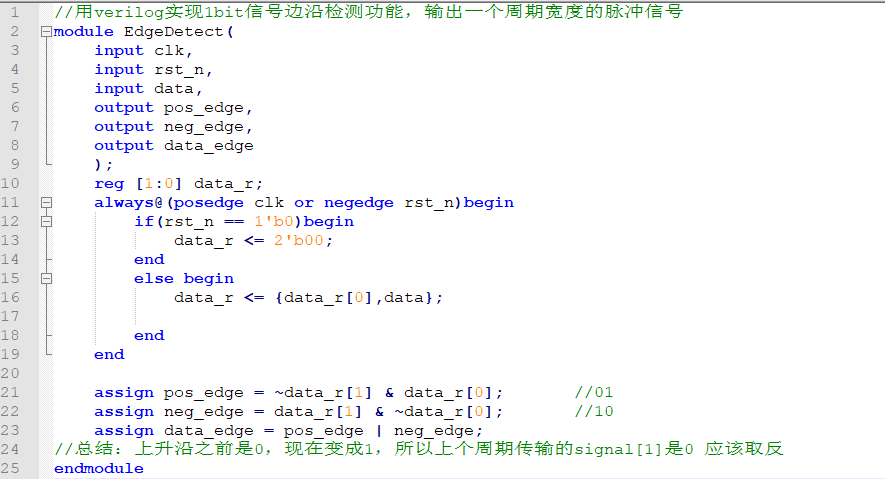
assign flag=(state==s5)?1’b1:1’b0;

endmodule

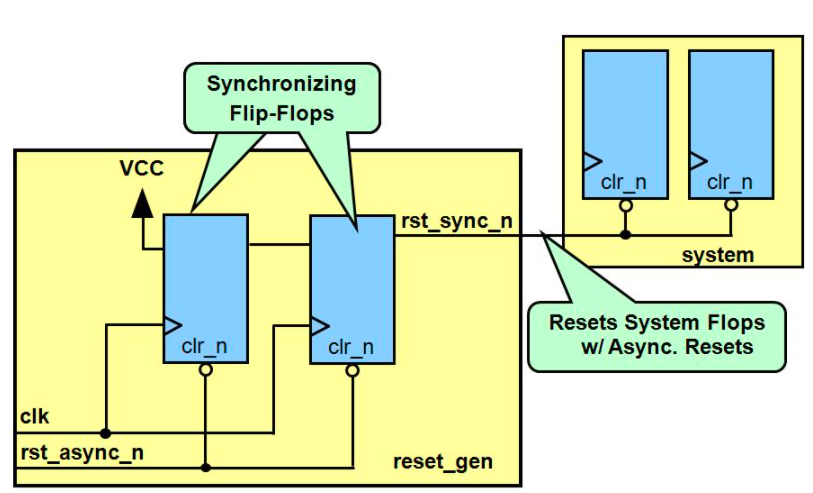


# 用verilog实现1bit信号边沿检测功能，输出一个周期宽度的脉冲信号

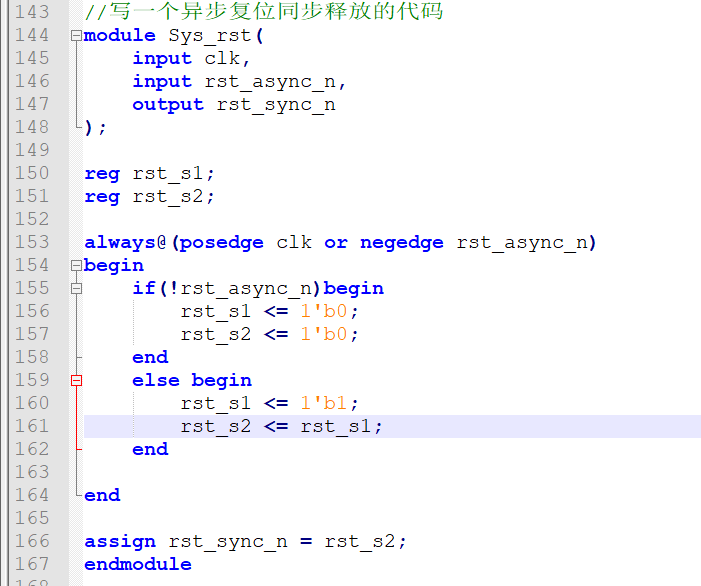
重点在上升沿，下降沿，上升沿或下降沿



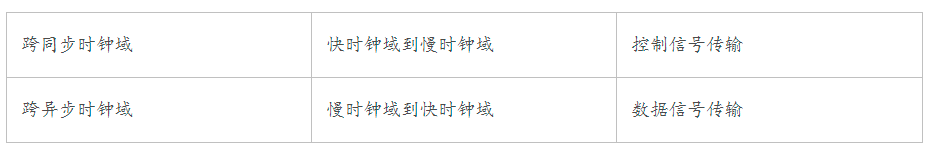
# 异步复位，同步释放（7.22）



思路：两级触发器同步，且第一级触发器输入为高电平，第二级触发器的输出为第一级触发器的输入，且释放的为第二级触发器的输出



# 跨时钟域（快时钟到慢时钟，或者是慢时钟到时钟）(7.23)



跨同步时钟域下，快时钟域到慢时钟域 的控制信号传输；

跨同步时钟域下，慢时钟域到块时钟域 的控制信号传输；

跨同步时钟域下，快时钟域到慢时钟域 的数据信号传输；

跨同步时钟域下，慢时钟域到块时钟域 的数据信号传输；

跨异步时钟域下，快时钟域到慢时钟域 的控制信号传输；

跨异步时钟域下，慢时钟域到块时钟域 的控制信号传输；

跨异步时钟域下，快时钟域到慢时钟域 的数据信号传输；

跨异步时钟域下，慢时钟域到块时钟域 的数据信号传输；

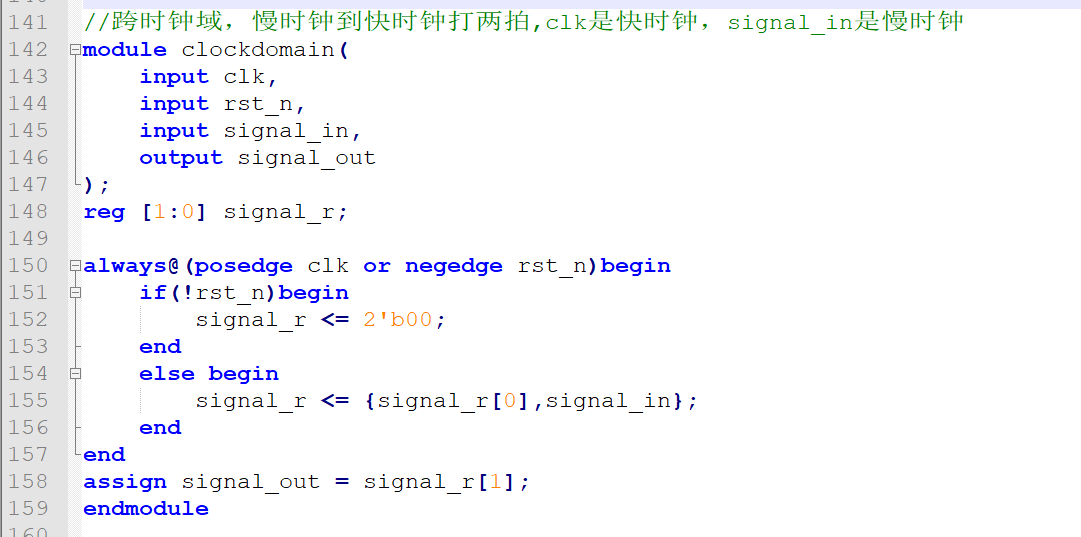
单bit（慢时钟域到快时钟域）：用快时钟打两拍，直接采一拍大概率也是没问题的，两拍的主要目的是消除亚稳态；

单bit（快时钟域到慢时钟域）：握手、异步FIFO、异步双口RAM；快时钟域的信号脉宽较窄，慢时钟域不一定能采到，可以通过握手机制让窄脉冲展宽，慢时钟域采集到信号后再“告诉”快时钟域已经采集到信号，确保能采集到；

多bit：异步FIFO、异步双口RAM、握手、格雷码；

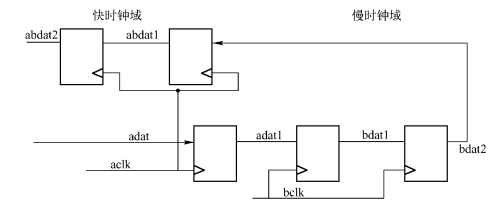
多bit中，强烈推荐异步FIFO，我在实际工程中使用多次，简单方便。

1. (1) 慢时钟域到快时钟域信号，一般情况下都是打两拍（单比特传输）也叫两级寄存器同步



(2) 快时钟域到慢时钟域

* 将快的时钟域中的信号进行展宽
* 对展宽的信号在慢的时钟域2级同步
* 在慢时钟域中产生反馈信号来对快的展宽信号进行拉低处理



握手/反馈机制

module(

input aclk,bclk,rst,

iuput abat,

output reg bdat2,

output reg abdat2

);

reg adat1;

reg bdat1;

reg abdat1;

always@(posedge aclk or negedge rst)

if(!rst) adat1<= 0;

else adat1<=adat;

end

always@(posedge bclk or negedge rst)

if(!rst) {bdat2,bdat1}=2’b00;

else {bdat2,bdat1}={bdat1,adat1};

end

always@(posedge aclk or negedge rst)

if(!rst) {abdat2,abdat1}=2’b00;

else {abdat2,abdat1}={abdat1, abdat2};

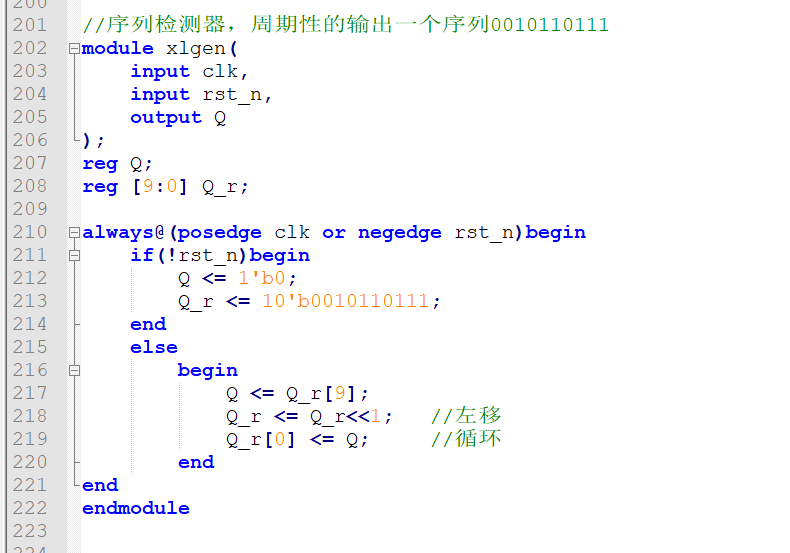
end

endmodule

* 数据变化速率比采样速率低、或者比采样速率略快时，可以使用握手信号进行。
* 无论是快到慢，还是慢到快，FIFO通吃。
* FIFO的设计需要注意FIFO空满信号产生问题、格雷码的应用问题、深度选择问题等。
* 说到格雷码的应用问题，也许会想到能不能先把数据变成格雷码，然后再通过双D触发器同步过去呢？这明显是不能的啊，你的数据不像FIFO的地址产生那样，是具有相邻性的，也就是只差一个1；因此不能把数据变成格雷码，再传输。

# 序列信号发生器，在clk信号作用下周期性输出“0010110111”

思路：几位数字就是几位的寄存器，写一个10位的寄存器，先输出最高位，左移一位，并将输出赋给低位进行循环

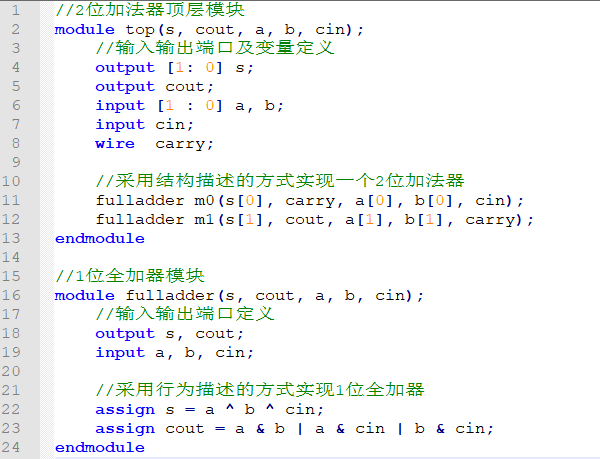


# 描述带进位输入、输出的两位全加器。端口：A/B为加数，CIN为进位输入，S为和，COUT为进位输出（7.24）

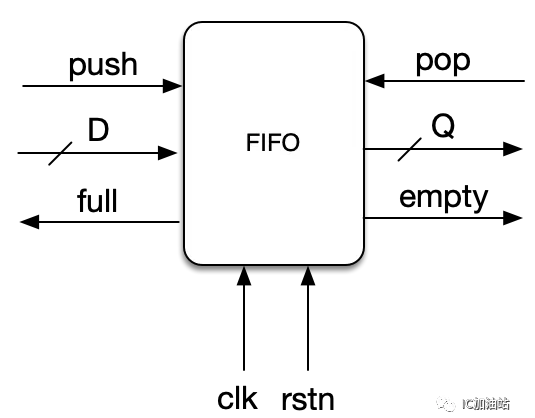
先写一位全加器，然后再写个顶层模块，调用两个一位全加器，其中第一个加法器的cout作为第二个加法器的cin

S = a^b^cin;

Cout = a&b|a&cin|b&cin;

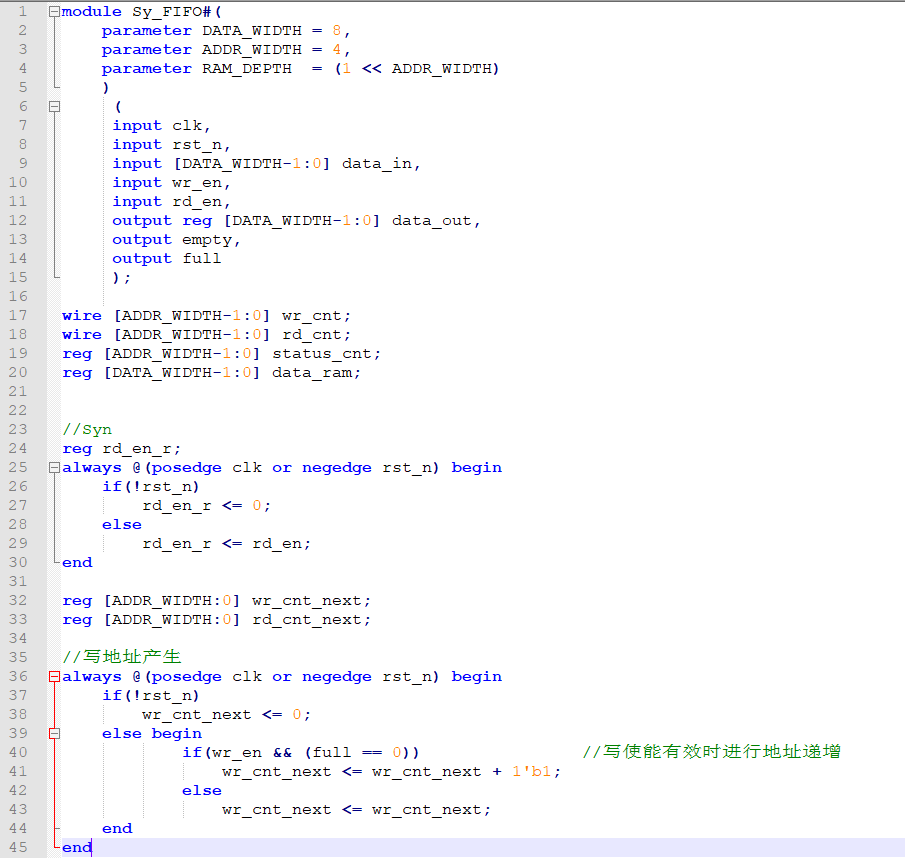


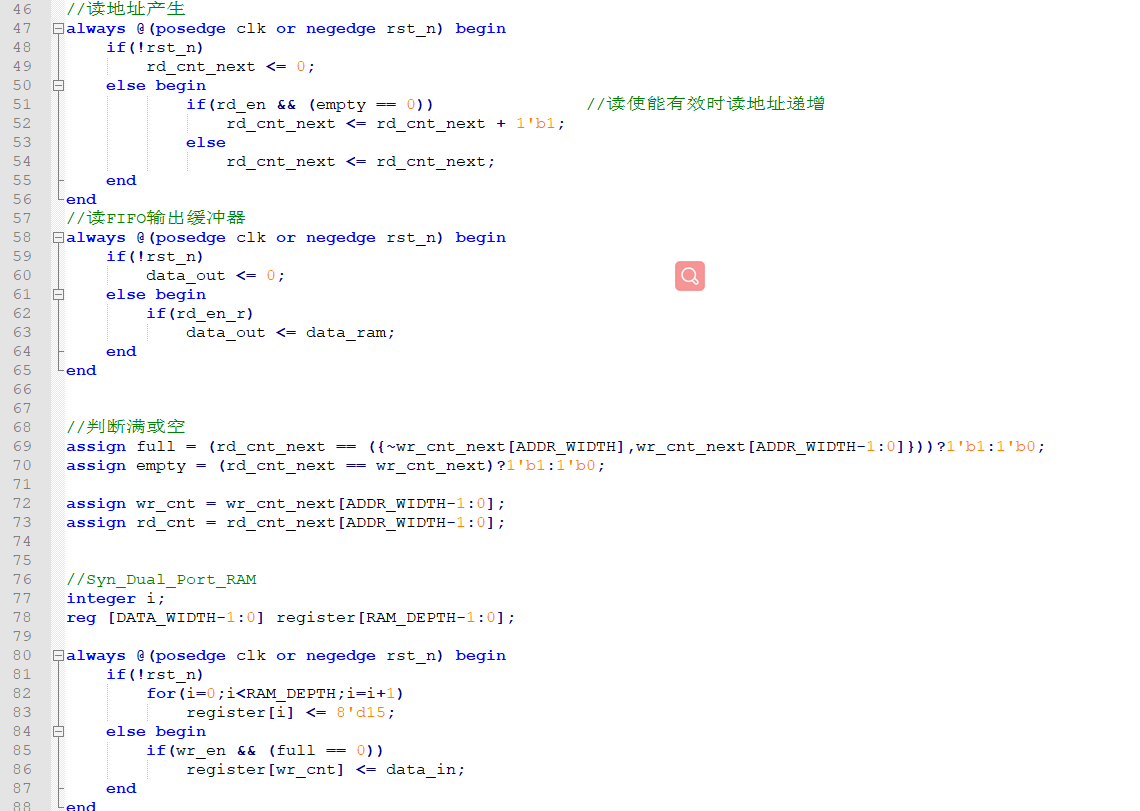
# 写一个同步FIFO，给定深度和位宽(7.25)

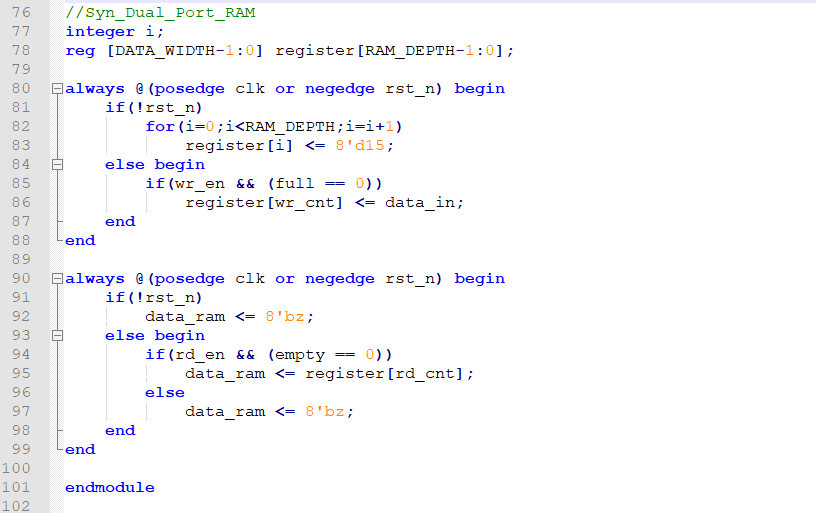


用Verilog实现一个同步FIFO，深度16，数据位宽8bit

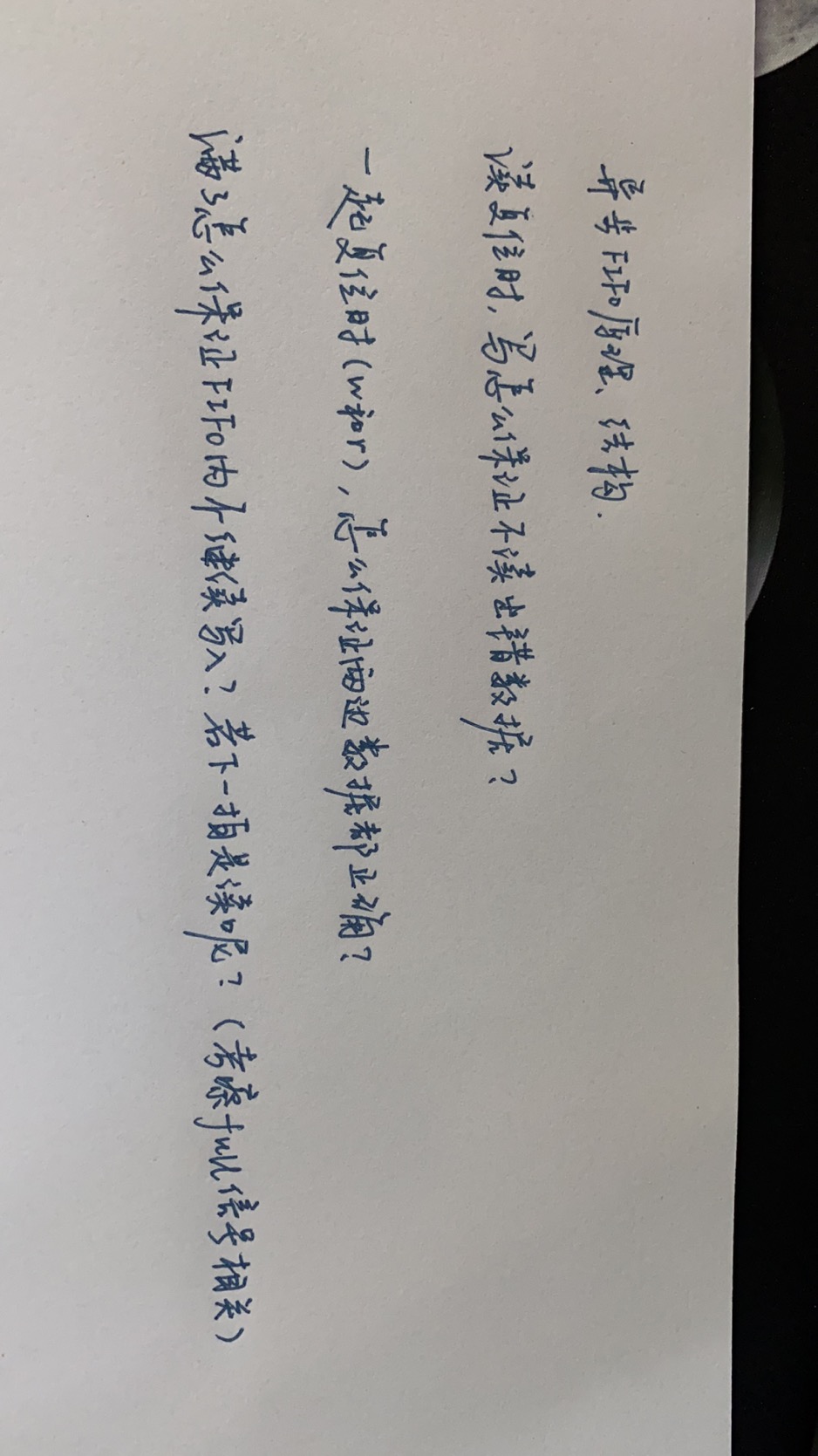
<https://mp.weixin.qq.com/s/_EkdpKjphnnlOzCp6uZnyQ>





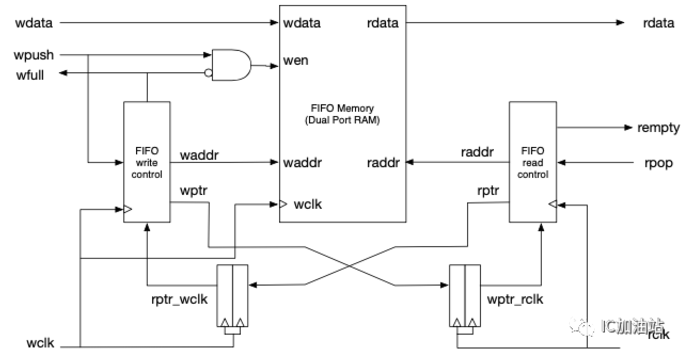


# 异步FIFO测试点：虽然不会写代码，但是要知道思想(7.26)



1、异步FIFO原理、结构

FIFO是一个存储管道，先进先出，同步FIFO是写入端和读出端同一个时钟域，异步FIFO中Push和Pop分别在不同的时钟域。



2、读复位时，写怎么保证不读出错数据

复位之后，只能进行写操作，因为什么都没有，读数据会读出错误的值

在复位的时候，FIFO空有效、满无效，禁止读数据，只能往里面写数据

3、一起复位时（w和r），怎么保证两边数据都正确

4、满了怎么保证FIFO内不继续写入？若下一拍是读呢

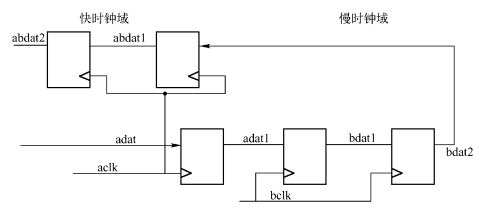
产生读空信号，目的是也就是防止继续读从而读出错误的数据；

怎么区分在读地址和写地址相同的时候是读空还是写满呢？

地址深度拓宽1位当做标志位，回卷一次标志位取反，读写地址全相等的时候，表示是读空；除了标志位外，剩余的地址为全部相等，那么就表示是写满。

Grey 码

# 握手（7.27）



module fast\_low(

input aclk,bclk,rst,

input adat,

output bdat2,abdat2

);

reg adat1;

reg bdat1;

reg abdat1;

always @ (posedge aclk or negedge rst)begin

if(!rst) adat1<=0;

else adat1<= adat;

end

always @ (posedge bclk or negedge rst)begin

if(!rst) {bdat2, bdat1} <=2’b00;

else {bdat2, bdat1} <= {bdat1,adat1};

end

always @ (posedge aclk or negedge rst)begin

if(!rst) {abdat2, abdat1} <=2’b00;

else {abdat2, abdat1} <= {abdat1,bdat2};

end

endmodule

# 写一个100MHz的时钟(7.28)

parameter DELY=10;

always (#DELY/2) clk=~clk;

# 红绿灯（7.29）

# 贩卖机（7.30）

# 写断言和随机约束（7.31）

# 用Verilog实现glitch free时钟切换电路。输入sel，clka，clkb，sel为1输出clka，sel为0输出clkb

module change\_clk\_source(

input clka,clkb,rst,

input sel,

output clkout

);

reg out1;

reg out2;

always@(posedge clka or negedge rst)begin

if(!rst) out1<=0;

else begin

if(sel) out1<=1;

else out1<=0;

end

end

always@(posedge clkb or negedge rst)begin

if(!rst) out2<=0;

else begin

if(!sel) out2<=1;

else out2<=0;

end

end

assign clkout=(out1&clka)|(out2&clkb);

endmodule

# CDC 跨时钟域

当我们要同步aclk时钟域的一个单周期的pulse到bclk时钟域时，我们期望bdata是什么样呢？答案就是，**bclk时钟域也是单周期的一个pulse。**

# 找出8bit中第一次出现1的个数verilog

二分频

module find\_ones(

input clk,rst,

input [7:0]x,

output [2:0]y

);

wire [3:0] data\_4;

wire [1:0] data\_2;

assign y[2]=|x[7:4];

assign data\_4=y[2]? x[7:4]: x[3:0];

assign y[1]=| data\_4[3:2];

assign data\_2=y[1]? data\_4[3:2]: data\_4[1:0];

assign y[0]= data\_2[1];

endmodule

# 串并转换

(1)lsb优先

(2)msb优先

module Deserialize(

input clk,rst,

input data\_i,

output reg [7:0]data\_o

);

//lsb优先

/\*reg [2:0] cnt;

always @ (posedge clk or negedge clk)begin

if(!rst)begin

data\_o<=0;

cnt<=0;

end

else begin

cnt<=cnt+1;

data\_o[cnt]<= data\_i;

end

end

//msb优先

\*/

reg [2:0] cnt;

always @ (posedge clk or negedge clk)begin

if(!rst)begin

data\_o<=0;

cnt<=0;

end

else begin

cnt<=cnt+1;

data\_o[7-cnt]<= data\_i;

end

end

endmodule

# 对下述代码，使用“加法器”和“选择器”画出最优的电路图（电路面积最小）

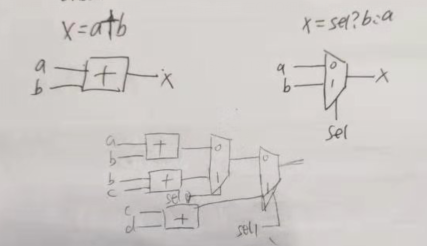
case (sel[2:0])

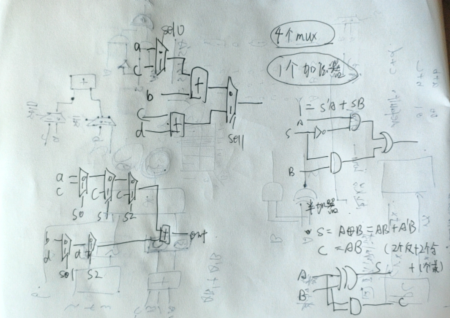
3’b000:sum=a+b;

3’b001:sum=b+c;

default:sum=c+d;

endcase





第一种解法是一个加法器和5个mux，第二种解法是2两个加法器两个mux，相当于比较3个mux和一个加法器的面积，一个加法器是2个反，2个与，一个或门。三个mux是3个反，6个与，三个或门。所以我觉得第二种的面积更小

# 仲裁器

定义就是当有两个或两个以上的模块需要占用同一个资源的时候，我们需要由仲裁器arbiter来决定哪一个模块来占有这个资源。

verilog的case自带"break"

<https://mp.weixin.qq.com/s/82o9iAIw1LiDsjBNmiBVDQ>

request和它的2进制补码按位与

module prior\_arb #(

parameter REQ\_WIDTH=16

) (

input [REQ\_WIDTH-1:0] req,

output [REQ\_WIDTH-1:0] gnt

);

assign gnt=req&(~(req-1));

end

# 从100个数中找到最大的两个数

Generate+模块复用

module top(

input clk,rst,

input [7:0] data,

output reg [7:0] max1,

output reg [7:0] max2

);

reg [6:0] cnt;

reg [7:0] out1;

reg [7:0] out2;

always@(posedge clk or negedge rst)begin

if(!rst) cnt<=0;

else begin

if(cnt==7’d99) cnt<=0;

else cnt<=cnt+1;

end

end

always@(posedge clk or negedge rst)begin

if(!rst) begin

out1<=0;out2<=0;

end

else begin

if(cnt==0)begin

out1<=data;out2<=data;

end

else begin

if(data>=out1)begin

if(data>=out2)begin

out2<=data;

out1<=out2;

end

else begin

out2<=out2;

out1<=data;

end

end

else begin

out1<=out1;out2<=out2;

end

end

end

end

always@(posedge clk or negedge rst)begin

if(!rst)begin

max1<=0;max2<=0;

end

else begin

if(cnt==0)begin

max1<=out1;

max2<=out2;

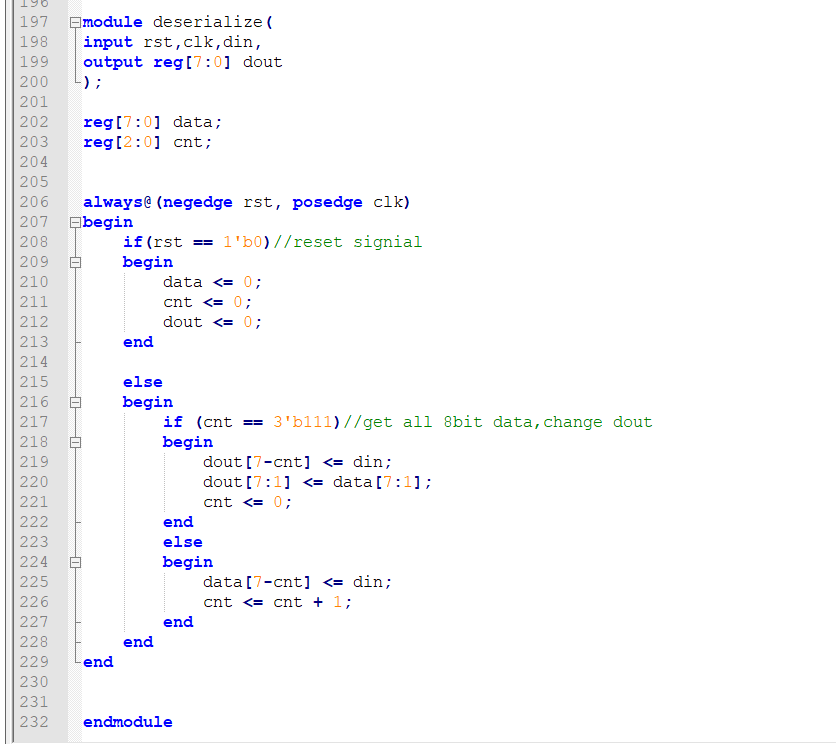
end

end

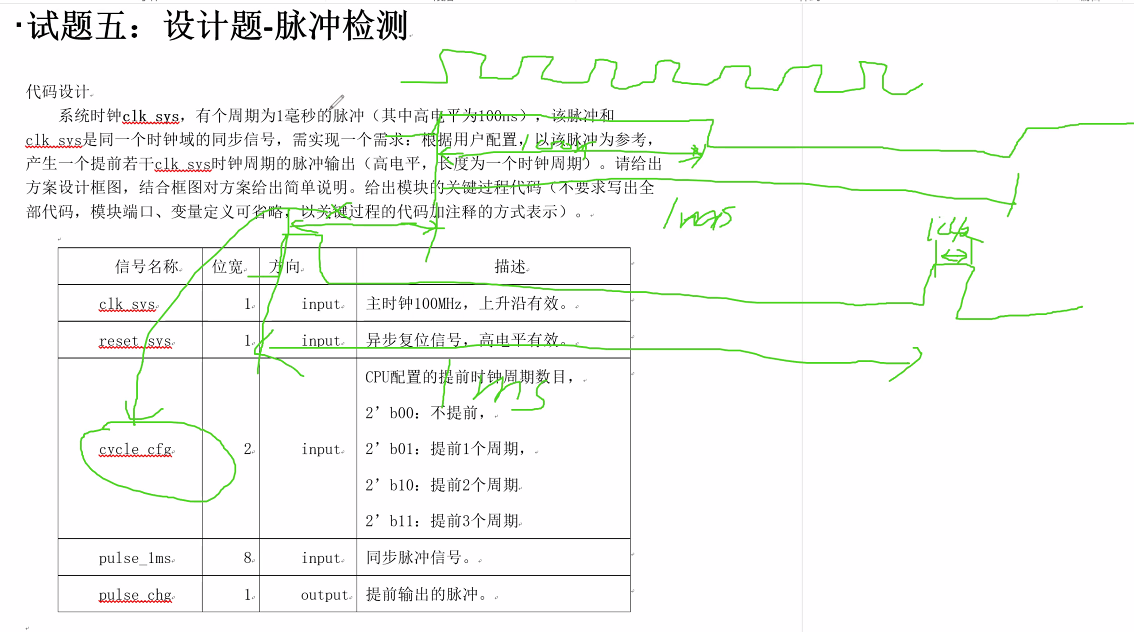
end

endmodule

# Verilog实现8bit串行数据转并行数据



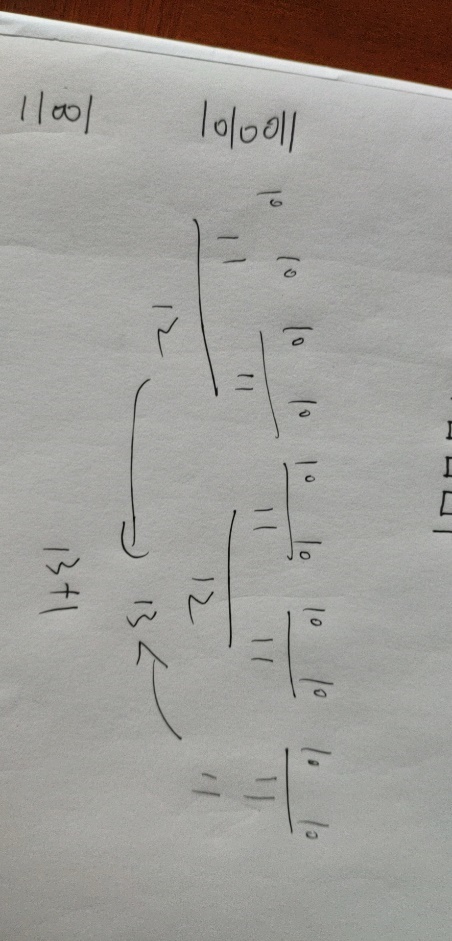
# 27.脉冲检测



提前一个周期：脉冲上升沿到来的时候，开始计数，计10^5个数，根据cycle\_cfg为0时，输入脉冲为输出脉冲，cycle\_cfg为1时，cnt为999998-7时，输出为1，其余输出为0，cycle\_cfg为2时，cnt为999997-6时，输出为1，其余输出为0，cycle\_cfg为2时，cnt为999996-5时，输出为1，其余输出为0。

# 28.10个10bit相加，需要寄存器多少bit？

14bit



# 29.每10个数求和并输出，要求资源最少

module top(

input clk,rst,

input [9:0] data,

output reg [13:0] out

);

reg [3:0]cnt;

reg flag;

always@(posedge clk or negedge rst)begin

if(!rst) begin

cnt<=0;

flag<=0;

end

else begin

if(cnt==4’d9)begin

cnt<=0;

flag<=1;

end

else begin

cnt<=cnt+1;

flag<=0;

end

end

end

reg [13:0] out1;

always@(posedge clk or negedge rst)begin

if(!rst) out1<=0;

else begin

if(cnt==0) out1<=data;

else out1<=out1+data;

end

end

always@(posedge clk or negedge rst)begin

if(!rst) out<=0;

else begin

if(flag) out<=out1;

else out<=out;

end

end

endmodule

# 30、4bit乘法器（门电路）

（1）查表法

（2）[数字电路基础知识——乘法器的设计（二）（ 串行、流水线、有符号数八位乘法器）\_摆渡沧桑-CSDN博客\_有符号数乘法器](https://blog.csdn.net/vivid117/article/details/101427302?ops_request_misc=&request_id=&biz_id=102&utm_term=IC%E4%B9%98%E6%B3%95%E5%99%A8%E7%94%B5%E8%B7%AF&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-3-101427302.first_rank_v2_pc_rank_v29&spm=1018.2226.3001.4187)

always@(a,b)begin

c=0;

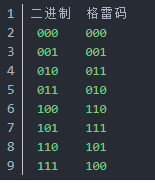
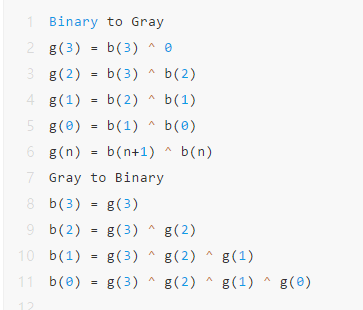
for(i=1;i<=size;i=i+1) begin

c=c+(b(i)?(a<<(i-1)):0);

end

end

# 31、格雷码和二进制之间的转换



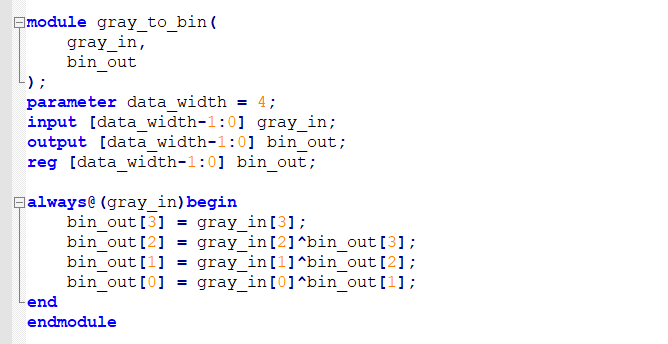
（1）二进制转格雷码

原数右移一位再与原数异或

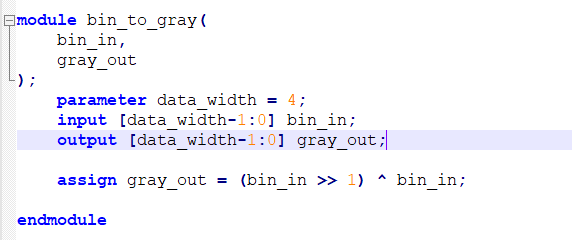
（2）格雷码转二进制

二进制最高位取格雷码最高位，然后循环做二进制高位与低一位的格雷码异或

格雷码转二进制



二进制转格雷码



# 32、用Verilog语言编写module，实现：

A、输入数据位宽32bit，非连续，带有字节有效信号的非对齐数据和数据有效信号，输入数据包长度为1-256字节的任意长度，

B、经过处理，按照输出64bit位宽输出的对齐数据，带有字节有效和数据有效输出；

C、同步电路；

D、在15分钟内完成；



