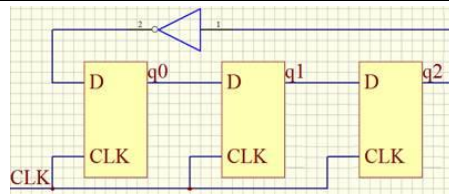


1、请画出以下程序综合后的电路图，并写 Testbench，画出仿真波形：

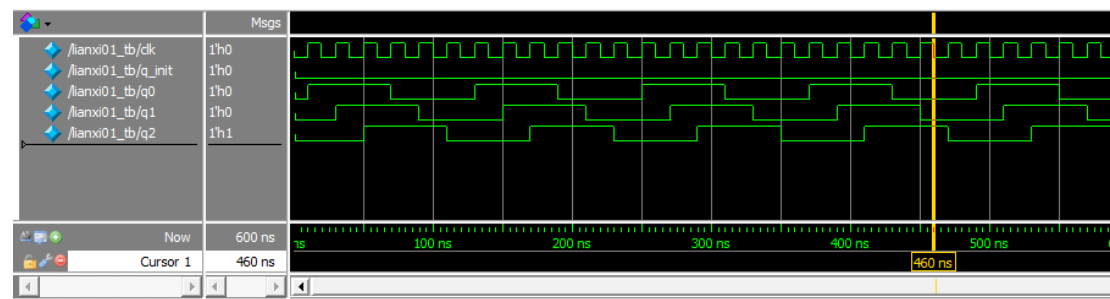
```
always@(posedge clk)
begin
    q0<=~q2;
    q1<= q0;
    q2<= q1;
end
```



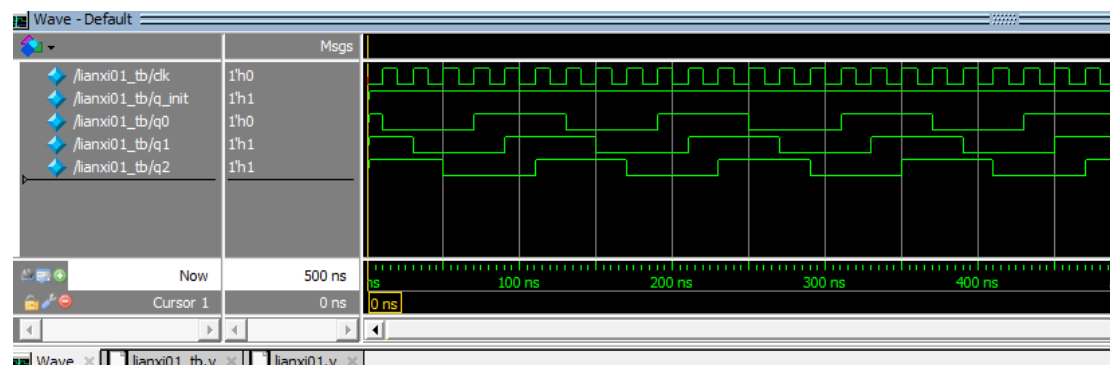
代码如下：

Module	Testbench
<pre>module lianxi01(clk,q_init,q0,q1,q2); input clk; input q_init; output q0,q1,q2; reg q0,q1,q2; always@(*)begin     q0 = q_init;     q1 = q_init;     q2 = q_init; end always@(posedge clk)begin     q0 &lt;= ~q2;     q1 &lt;= q0;     q2 &lt;= q1; end endmodule</pre>	<pre>`timescale 1ns/1ns module lianxi01_tb(); reg clk,q_init; parameter CYCLE = 20; lianxi01 uut(.clk(clk),.q_init(q_init),.q0(q0),.q1(q1),.q2(q2)); initial begin     clk = 0;     forever         #(CYCLE/2)             clk = ~clk; end initial begin     # 1     q_init = 0; end endmodule</pre>

将 q0,q1,q2 初始化为 0，然后开始仿真，得到的仿真波形图如下：



将 q0,q1,q2 初始化为 1，然后开始仿真，得到的仿真波形图如下：

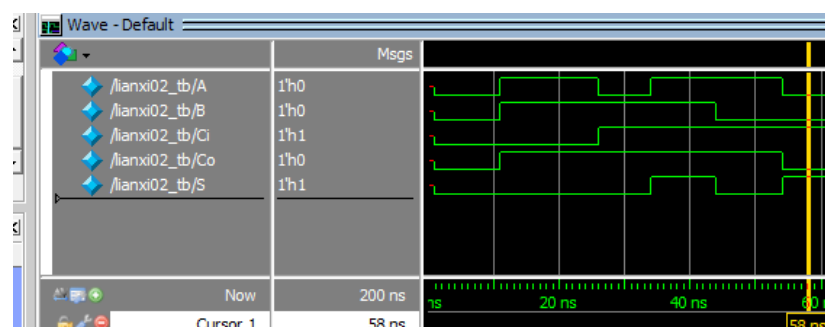


2. 根据 HA 模块程序，写出引用 HA 模块描述 FA 模块的 Verilog 程序及其 Testbench，画出仿真波形。

代码如下：

Module	Testbench
<pre> module HA(A,B,S,C);     input  A,B ;     output S,C ;     assign {C,S} = A + B; endmodule  module lianxi02(A,B,Ci,Co,S);     input  A,B,Ci;     output Co,S ;     reg Co,S ;     wire cout1,cout2,S1,FA; //定义中间变量      HA ha1(.A(A),.B(B),.S(FA),.C(cout1));     HA ha2(.A(FA),.B(Ci),.S(S1),.C(cout2));     always@(*)begin         Co = cout1   cout2;         S = S1;     end endmodule </pre>	<pre> module lianxi02_tb();     reg A,B,Ci;     wire Co,S;     lianxi02 lianxi(.A(A),.B(B),.Ci(Ci),.Co(Co),.S(S));     initial begin         #1         A = 0; B=0; Ci=0;         #10         A = 1; B=1; Ci=0;         #15         A = 0; B=1; Ci=1;         #8         A = 1; B=1; Ci=1;         #10         A = 1; B=0; Ci=1;         #10         A = 0; B=0; Ci=1;     end endmodule </pre>

结果如下图：



结果分析：在这个例子中，进行了五段时间的仿真，分别是：

- 1) A = 0; B=0; Ci=0: 当三者都为 0 时，输出 Co 和 S 也为 0，验证正确；
- 2) A = 1; B=1; Ci=0: 此时 HA1 输出的为 cout1=1,FA=0, HA2 输出的 cout2=0,S1=0, 则 Co 是 cout1 或 cout2，结果为 1，S=S1=0，验证正确；
- 3) A = 0; B=1; Ci=1: 此时 HA1 输出的为 cout1=0,FA=1, HA2 输出的 cout2=1,S1=0, 则 Co 是 cout1 或 cout2，结果为 1，S=S1=0，验证正确；
- 4) A = 1; B=1; Ci=1: 此时 HA1 输出的为 cout1=1,FA=0, HA2 输出的 cout2=0,S1=1, 则 Co 是 cout1 或 cout2，结果为 1，S=S1=1，验证正确；
- 5) A = 1; B=0; Ci=1: 此时 HA1 输出的为 cout1=0,FA=1, HA2 输出的 cout2=1,S1=0, 则 Co 是 cout1 或 cout2，结果为 1，S=S1=0，验证正确；
- 6) A = 0; B=0; Ci=1: 此时 HA1 输出的为 cout1=0,FA=0, HA2 输出的 cout2=0,S1=1, 则 Co 是 cout1 或 cout2，结果为 0，S=S1=1，验证正确；

3. 设计一奇偶校验位生成电路，输入八位总线信号 `bus`，输出奇校验位 `odd` 信号，偶校验位 `even` 信号，写 Testbench，并画出仿真波形。

代码如下：

Module	Testbench
<pre>module lianxi03(bus,even,odd); input    bus; output   even,odd; reg  even,odd; always@(*)begin     even = ^bus;     odd  = ^~bus; end endmodule</pre>	<pre>module lianxi03_tb(); reg[7:0] bus; wire even,odd; lianxi03 lianxi(.bus(bus),.even(even),.odd(odd)); initial begin     #1     bus = 8'b00000000;     #10     bus = 8'b10110101;     #10     bus = 8'b00110110;     #10     bus = 8'b11111111;     #10     bus = 8'b00000001; end endmodule</pre>

用到逻辑操作的知识点：

### 3. 逻辑操作符

半加器的例子中出现了两种逻辑操作符：  
逻辑与 “&” 和逻辑异或 “^”。

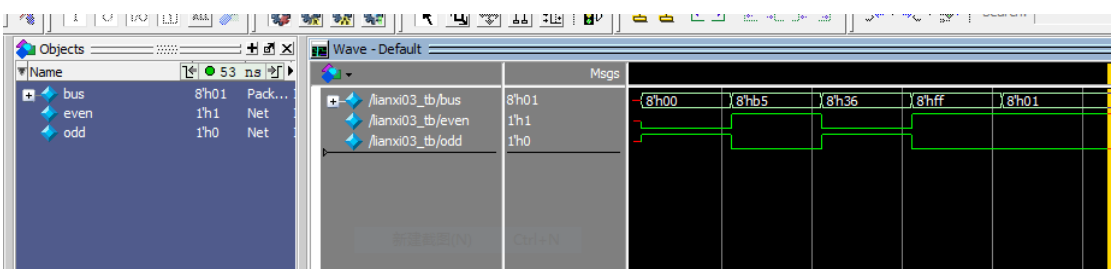
Verilog的逻辑操作符大部分与C语言一致，比如：  
逻辑按位与： &  
逻辑按位或： |  
逻辑按位异或： ^  
逻辑按位取非： ~  
但Verilog也有自己的扩展，比如  
逻辑按位 与非 ~&  
逻辑按位 或非 ~|  
逻辑按位 同或 ~^

这里“按位”的意思，是可以对多位逻辑信号按位进行逻辑运算

半加器的电路结构

```
assign SO = A ^ B;
assign CO = A & B;
```

仿真结果如下图：



结果说明：

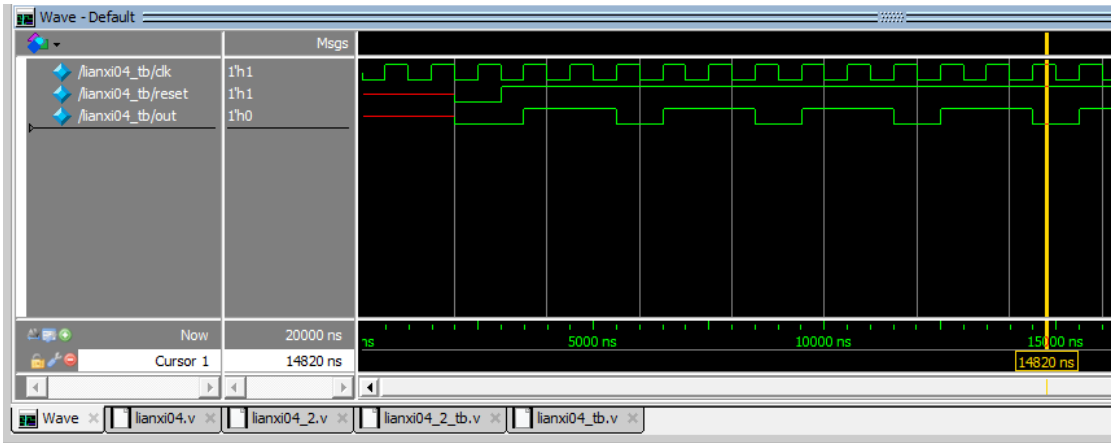
- 1) 当 `bus` 里含奇数个 1 时，`even`=1，`odd`=0;
- 2) 当 `bus` 里含偶数个 1 时，`even`=0，`odd`=1。

4. 利用一个 1MHz 的输入时钟 `clk`，设计一个带复位端且对输入时钟 `clk` 进行三分频模块，写 `Testbench`，并画出仿真波形。设计要求：复位信号为异步复位、低电平有效。

三分频（时钟为 1MHz，占空比为 2/3）代码如下：

Module	Testbench
<pre>module lianxi04(clk,reset,dout); input  clk,reset; output dout; reg  dout; reg[1:0] count; always@(posedge clk or negedge reset)begin if(reset == 1'b0) begin dout &lt;= 0; count &lt;= 2'b00; end else begin if(count == 0)begin dout &lt;= ~dout; count &lt;= count + 1'b1; end else if(count == 2)begin dout &lt;= ~dout; count &lt;= 2'b00; end else begin count &lt;= count + 1'b1; end end end end endmodule</pre>	<pre>`timescale 1ns/1ns module lianxi04_tb(); reg clk,reset; wire out; parameter CYCLE = 1000;  lianxi04 lianxi(.clk(clk),.reset(reset),.dout(out));  initial begin clk = 0; forever # (CYCLE/2) clk = ~clk; end  initial begin # (2*CYCLE) reset = 0; # CYCLE reset = 1; end end endmodule</pre>

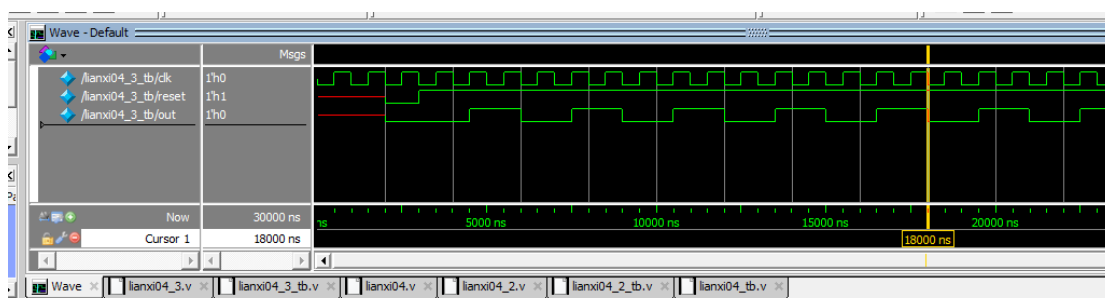
时钟为 1MHz，占空比为 2/3 的三分频仿真结果如下：



三分频（时钟为 1MHz，占空比为 50%）代码如下：

Module	Testbench
<pre> module lianxi04_3(input clk, input arst, output clk_div ); parameter N = 3; reg [2:0] cnt; reg clk_a; reg clk_b; wire clk_c;  always@(posedge clk or negedge arst) begin     if(arst == 1'b0)         cnt &lt;= 0;     else if(cnt == N-1)         cnt &lt;= 0;     else         cnt &lt;= cnt + 1; end  always@(posedge clk or negedge arst) begin     if(arst == 1'b0)         clk_a &lt;= 0;     else if(cnt == (N-1)/2    cnt == N-1)         clk_a &lt;= ~clk_a;     else         clk_a &lt;= clk_a; end  always@(negedge clk or negedge arst) begin     if(arst == 1'b0)         clk_b &lt;= 0;     else         clk_b &lt;= clk_a; end assign clk_c = clk_a   clk_b; //N[0]=1 奇数，否则偶数 assign clk_div = N[0] ? clk_c : clk_a; endmodule </pre>	<pre> `timescale 1ns/1ns module lianxi04_3_tb(); reg clk,reset; wire out; parameter CYCLE = 1000;  lianxi04_3 lianxi_3(     .clk(clk),     .arst(reset),     .clk_div(out) );  initial begin     clk = 0;     forever         # (CYCLE/2)         clk = ~clk; end  initial begin     # (2*CYCLE)     reset = 0;     # CYCLE     reset = 1; end endmodule </pre>

时钟为 1MHz，占空比为 50% 的三分频仿真结果如下：

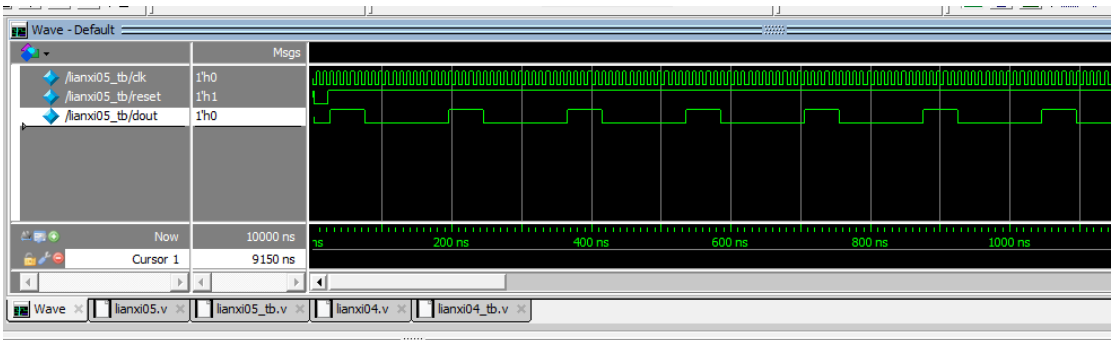


5. 用 verilog 语言实现一个输出脉冲，要求是 12 个时钟周期是低电平、5 个时钟周期是高电平，写 Testbench，并画出仿真波形。

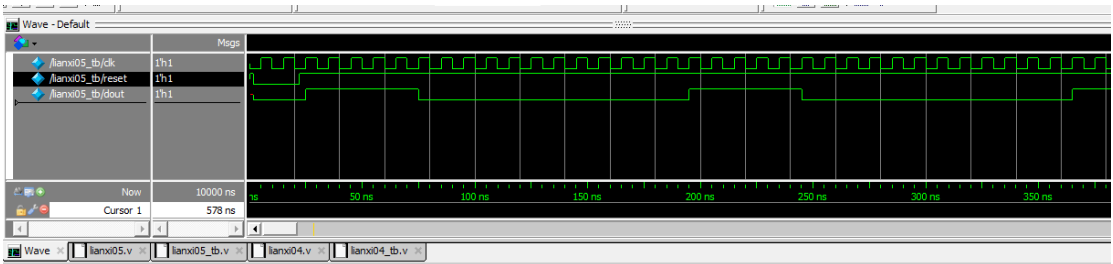
代码如下：

Module	Testbench
<pre>module lianxi05(clk,reset,dout); input clk,reset; output dout; reg  dout; reg[4:0] count1; always@(posedge clk or negedge reset)begin if(reset == 1'b0)begin dout&lt;=1'b0; count1 &lt;= 5'd0; end else begin if(count1 == 5'd0)begin dout &lt;= ~dout; count1 &lt;= count1 + 1'b1; end else if(count1 == 5'd5) begin dout &lt;= ~dout; count1 &lt;= count1 + 1'b1; end else if(count1 == 5'd16)begin dout &lt;= dout; count1 &lt;= 5'd0; end else begin count1 &lt;= count1 +1; end end end endmodule</pre>	<pre>`timescale 1ns/1ns module lianxi05_tb(); reg  clk,reset; wire dout; parameter CYCLE =10; parameter RST_TIME =2; lianxi05 lianxi_05(.clk(clk),.reset(reset),.dout(dout));  initial begin clk = 0; forever # (CYCLE/2) clk = ~clk; end  initial begin reset = 1; # 2 reset = 0; # (CYCLE * RST_TIME) reset = 1; end endmodule</pre>

仿真波形图如下：



将波形放大，方便验证：（由图可知，设计的模型符合设计要求）

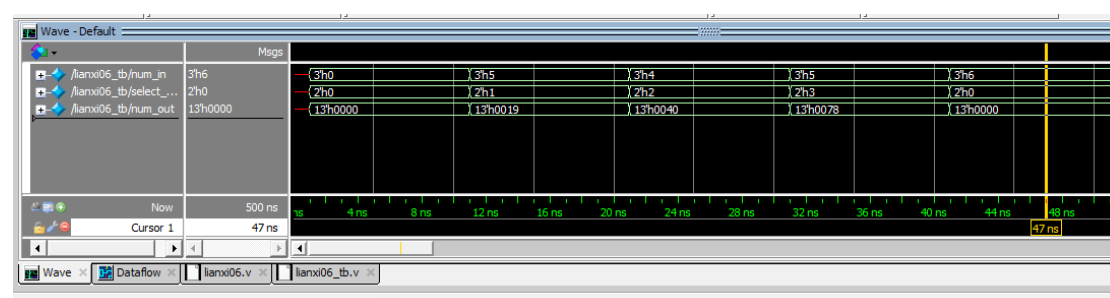


6. 设计一个带控制端的逻辑运算电路，采用调用 **function** 函数的方式，分别完成正整数的平方、立方和阶乘的运算。编写测试模块，并给出仿真波形。

设输入的数据为三位数（0-7 之间的正整数），代码如下：

Module	Testbench
<pre> module lianxi06(in,s,out); input[2:0] in; input[1:0] s; output[12:0] out; reg [12:0] out; always@(*)begin     case(s)         2'b00: out &lt;= 0;         2'b01: out &lt;= square(in);         2'b10: out &lt;= cube(in);         2'b11: out &lt;= factorial(in);         //default:out&lt;= 0;     endcase end function [12:0] square;     input[2:0] x;     assign square = x*x; endfunction function [12:0] cube;     input[2:0] y;     assign cube = y*y*y; endfunction function [12:0] factorial;     input[2:0] z;     reg [2:0] index;     begin         factorial = z?1:0;         for(index 2;index&lt;=z;index=index+1)             factorial = index*factorial;         end     endfunction endmodule </pre>	<pre> `timescale 1ns/1ns module lianxi06_tb(); reg [2:0] num_in; reg [1:0] select_mode; wire[12:0] num_out;  lianxi06 lianxi_06_6(.in(num_in),.s(select_mode),.out(num_out));  initial begin     #1     num_in = 3'b000;     select_mode = 2'b00;     # 10     num_in = 3'b101;     select_mode = 2'b01;     # 10     num_in = 3'b100;     select_mode = 2'b10;     # 10     num_in = 3'b101;     select_mode = 2'b11;     # 10     num_in = 3'b110;     select_mode = 2'b00; end endmodule </pre>

得到仿真的波形图如下：



结果分析：

当 num\_in=0, select\_mode=00 时，归 0 模式，num\_out=0，正确；

当 num\_in=5, select\_mode=01 时，平方模式，num\_out=h19=16\*1+9=25，正确；

当 num\_in=4, select\_mode=10 时，立方模式，num\_out=h40=16\*4+0=64，正确；

当 num\_in=5, select\_mode=11 时，阶乘模式，num\_out=h78=16\*7+8=125,正确；

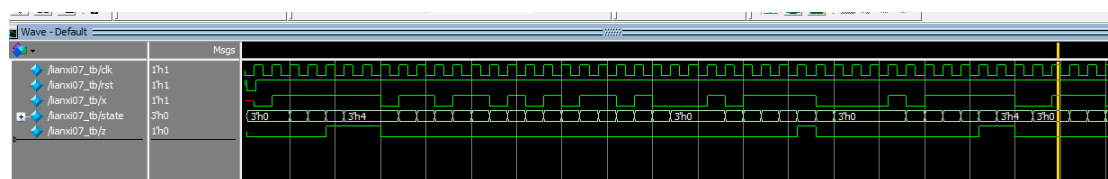
当 num\_in=6, select\_mode=00 时，归 0 模式，num\_out=0，正确；

7. 设计一个串行数据检测器，要求是：连续 4 个或 4 个以上的 1 时输出为 1，其它输入情况下为 0。采用 FM 状态机编写该模块及测试模块，并存储仿真波形为 data\_detect.vcd 文件。画出状态图如下：（用笔画）

代码如下：

Module	Testbench
<pre> module lianxi07(x,z,clk,rst,state); input x,clk,rst; output z; output[2:0] state; reg[2:0] state; wire z; parameter IDLE=3'd0,A=3'd1,B=3'd2,C=3'd3,D=3'd4; assign z = ((state==C &amp;&amp; x==1)    (state==D &amp;&amp; x==1)) ? 1:0; always @(posedge clk or negedge rst) begin     if(rst==1'b0) state&lt;=IDLE;     else         casex(state)             IDLE: if(x==1) state&lt;=A;                     else state&lt;=IDLE;             A:    if(x==1) state&lt;=B;                     else state&lt;=IDLE;             B:    if(x==1) state&lt;=C;                     else state&lt;=IDLE;             C:    if(x==1) state&lt;=D;                     else state&lt;=IDLE;             D:    if(x==1) state&lt;=D;                     else state&lt;=IDLE;             default: state&lt;=IDLE;         endcase end endmodule </pre>	<pre> `timescale 1ns / 1ns module lianxi07_tb(); reg clk, rst,x; wire[2:0] state; wire z; always #10 clk=~clk; always @(posedge clk) begin     x&lt;={\$random}%2; end initial begin     clk=0;     rst=1;     #2 rst=0;     #10 rst=1;     #1000 \$stop; end initial begin     \$dumpfile("lianxi07_dumpfile.vcd");     \$dumpvars ; end lianxi07 lianxi_07(.x(x),.z(z),.clk(clk),.rst(rst),.state(state)); endmodule </pre>

仿真结果:



由波形可以看出,当出现 4 个或 4 个以上连 1 的时候,输出 1,否则输出 0,符合题目的要求。

保存的波形文件如下：于是生成 vcd 文件，此时可通过 vcd2wlf 转换为 wlf 格式，在 modelsim 中方便查看输入：

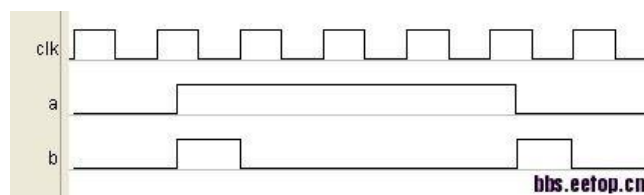
```
"vcd2wlf data detect.vcd data detect.wlf" + "vsim -view data detect.wlf"
```

比电脑 > 新加卷 (E:) > study > My\_Code > SOC > SOC\_Project

名称	修改日期	类型	大小
work	2020/10/28 14:48	文件夹	
data_detect.vcd	2020/10/28 14:48	VCD 文件	3 KB
lianxi01.v	2020/10/27 15:40	V 文件	1 KB



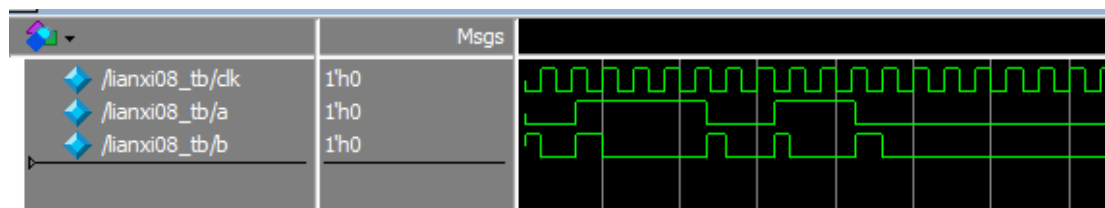
8. 根据时序图写 verilog 代码，已知时钟信号 **clk** 和输入信号 **a**，要获得如图所示的 **b**。编写该模块及测试模块，并存储仿真波形为 **b.vcd** 文件。



代码如下：

Module	Testbench
<pre> module lianxi08(clk,a,b); input clk,a; output b; reg b; always@(a)begin     b &lt;= 1'b1; end always@(posedge clk)begin     if(b==1'b1)begin         b&lt;=1'b0;     end end endmodule </pre>	<pre> `timescale 1ns/1ns module lianxi08_tb(); reg clk,a; wire b; lianxi08 lianxi_08(.clk(clk),.a(a),.b(b)); always #4 clk =~clk; initial begin     clk = 0;     a = 0;     #13     a = 1;     #34     a = 0;     #17     a = 1;     #21     a = 0; end initial begin     \$dumpfile("lianxi08_b.vcd");     \$dumpvars ; end endmodule </pre>

仿真结果如下：



由仿真结果可知，当 **a** 信号发生变化（包括上升沿和下降沿）的那一刻，**B** 的值立即变化为 **1**，等到时钟信号 **clk** 的下一个上升沿到来时，**B** 的值恢复为 **0**。由此验证了模型设计符合要求。

保存的 **VCD** 文件如下图：

电脑 > 新加卷 (E:) > study > My_Code > SOC > SOC_Project				
名称	修改日期	类型	大小	
lianxi08.vbak	2020/10/28 15:15	BAK 文件	1 KB	
lianxi08_b.vcd	2020/10/28 15:35	VCD 文件	3 KB	
lianxi08_tb.v	2020/10/28 15:28	V 文件	1 KB	

9. 编写代码完成 5 个 8bits 输入数据的比较排序，输入任意 8 个数据，输出数据为中间值（先比较排列），并编写 Testbench。

输入信号定义：clk：时钟输入；ngreset：复位信号；data0、data1……data4：输入数据 8bits

输出信号定义：middata：输出数据 8bits，中间值

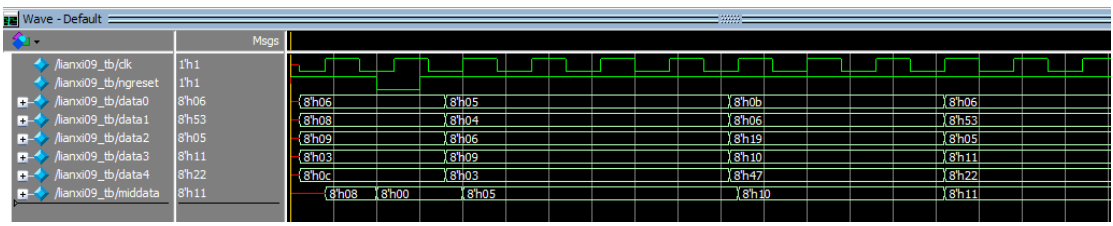
1)两个数比较采用子模块实现；2)两个数比较调用 task 方式；3)两个数比较定义为 function ；

代码如下：

Module	Testbench
<pre> module lianxi09_compare(a,b,out); input [7:0] a,b; output out; assign out = (a&gt;b)?1:0; endmodule  module lianxi09(clk,ngreset, data0,data1,data2,data3, data4,middata); input clk,ngreset; input [7:0] data0,data1,data2,data3,data4; output [7:0] middata; reg [7:0] middata; reg [2:0] com_0,com_1,com_2,com_3,com_4;  wire out0_1,out0_2,out0_3,out0_4; wire out1_0,out1_2,out1_3,out1_4; wire out2_0,out2_1,out2_3,out2_4; wire out3_0,out3_1,out3_2,out3_4; wire out4_0,out4_1,out4_2,out4_3;  always@(posedge clk or negedge ngreset)begin     if(ngreset == 1'b0)begin         com_0 = 3'b000;com_1 = 3'b000;         com_2 = 3'b000;com_3 = 3'b000;         com_4 = 3'b000;         middata &lt;= 0;     end     else begin         if (com_0 == 3'b010) middata &lt;= data0;         else if (com_1 == 3'b010) middata &lt;= data1;         else if (com_2 == 3'b010) middata &lt;= data2;         else if (com_3 == 3'b010) middata &lt;= data3;         else if (com_4 == 3'b010) middata &lt;= data4;     end end  lianxi09_compare c0_1(a(data0),b(data1),.out(out0_1)); lianxi09_compare c0_2(a(data0),b(data2),.out(out0_2)); lianxi09_compare c0_3(a(data0),b(data3),.out(out0_3)); lianxi09_compare c0_4(a(data0),b(data4),.out(out0_4));  lianxi09_compare c1_0(a(data1),b(data0),.out(out1_0)); lianxi09_compare c1_2(a(data1),b(data2),.out(out1_2)); lianxi09_compare c1_3(a(data1),b(data3),.out(out1_3)); lianxi09_compare c1_4(a(data1),b(data4),.out(out1_4));  lianxi09_compare c2_0(a(data2),b(data0),.out(out2_0)); lianxi09_compare c2_1(a(data2),b(data1),.out(out2_1)); lianxi09_compare c2_3(a(data2),b(data3),.out(out2_3)); lianxi09_compare c2_4(a(data2),b(data4),.out(out2_4));  lianxi09_compare c3_0(a(data3),b(data0),.out(out3_0)); lianxi09_compare c3_1(a(data3),b(data1),.out(out3_1)); lianxi09_compare c3_2(a(data3),b(data2),.out(out3_2)); lianxi09_compare c3_4(a(data3),b(data4),.out(out3_4)); </pre>	<pre> `timescale 1ns/1ns module lianxi09_tb(); reg clk,ngreset; reg [7:0] data0,data1; reg [7:0] data2,data3,data4; wire [7:0] middata;  lianxi09 lianxi_09(.clk(clk), .ngreset(ngreset), .data0(data0),.data1(data1), .data2(data2), .data3(data3),.data4(data4), .middata(middata));  always #4 clk =~clk; initial begin     ngreset = 1;     # 10     ngreset = 0;     # 5 ngreset = 1; end  initial begin     # 1     clk = 0;     data0=6;data1=8;     data2=9;data3=3;data4=12;     #17     data0=5;data1=4;     data2=6;data3=9;data4=3;      #33     data0=11;data1=6;     data2=25;data3=16;data4=71;      #25     data0=6;data1=83;     data2=5;data3=17;data4=34; end endmodule </pre>

<pre>lianxi09_compare c4_0(.a(data4),.b(data0),.out(out4_0)); lianxi09_compare c4_1(.a(data4),.b(data1),.out(out4_1)); lianxi09_compare c4_2(.a(data4),.b(data2),.out(out4_2)); lianxi09_compare c4_4(.a(data4),.b(data3),.out(out4_3));  always@(*)begin     com_0 = out0_1 + out0_2 + out0_3 + out0_4;     com_1 = out1_0 + out1_2 + out1_3 + out1_4;     com_2 = out2_0 + out2_1 + out2_3 + out2_4;     com_3 = out3_0 + out3_1 + out3_2 + out3_4;     com_4 = out4_0 + out4_1 + out4_2 + out4_3; end endmodule</pre>	
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

仿真结果如下图：



结果分析：

- 1) 当 data0=6;data1=8; data2=9;data3=3;data4=12 时：中间值为 8, middata=8’h08，结果正确；
- 2) 遇到复位信号（异步复位，低电平有效），复位为 0，middata=8’h00，正确；
- 3) 当输入的值发生了变化，data0=5;data1=4; data2=6;data3=9;data4=3;时，要等到时钟信号 clk 的上升沿到来时，middata 的值发生变化，为 middata=8’h05，输出的正确；
- 4) 当输入的值发生了变化，data0=11;data1=6; data2=25;data3=16;data4=71;时，要等到时钟信号 clk 的上升沿到来时，middata 的值发生变化，为 middata=8’h10=16，输出的正确；
- 5) 当输入的值发生了变化，data0=6;data1=83; data2=5;data3=17;data4=34;时，要等到时钟信号 clk 的上升沿到来时，middata 的值发生变化，为 middata=8’h11=17，输出的正确；
- 6) 所以设计的模块符合要求。

采用调用 task 的方式和 function 的方式，主体代码和上面的相同，不同点如下：

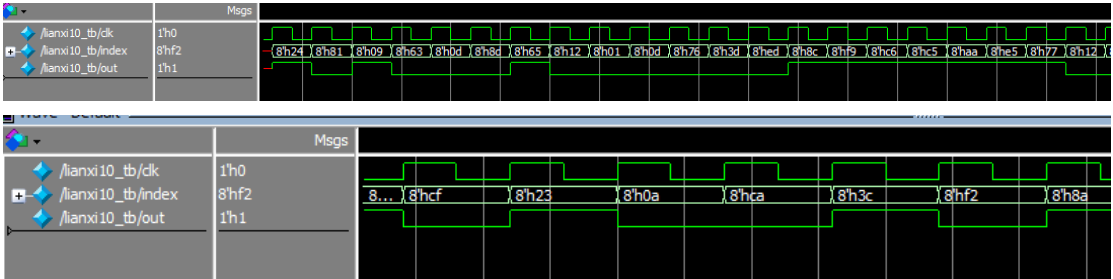
采用调用 task 的方式	采用 function 的方式
<pre>task compare_2; input A,B; output out; assign out = (A&gt;B)?1:0; endtask</pre>	<pre>function out; input A,B; assign out = (A&gt;B)?1:0; endfunction</pre>
调用方式: compare_2(data0,data1,out1_2);	调用方式: out(data0,data2);

10.设计一个序列发生器，根据输入数据（8bits）判别输出 1bits 的数据，如输入数据属于 0~127 则输出为 0、输入数据属于 128~255 则输出为 1。编写主模块及 Testbench。

代码如下：

Module	Testbench
<pre>module lianxi10(clk,index,out); input  clk; output[7:0]  index; output out; reg[7:0] index; reg out;  always@(posedge clk)begin     index &lt;= {\$random}%256;     if (index &lt;=127) out&lt;=0;     else out&lt;=1; end endmodule</pre>	<pre>`timescale 1ns/1ns module lianxi10_tb(); reg clk; wire[7:0] index; wire out;  lianxi10 lianxi_10(.clk(clk),.index(index),.out(out)); always #4 clk =~clk;  initial begin     clk = 0; end endmodule</pre>

仿真的结果如下：



在 module 中设计：当时钟信号 clk 的上升沿来到时，随机产生一个 8 位的数字，赋值给变量 index，然后对其值进行判断，如果在 0-127 之间，out 输出为 0，如果在 128-255 之间，out 输出为 1。

由以上的仿真结果可以验证：设计的模型符合要求。

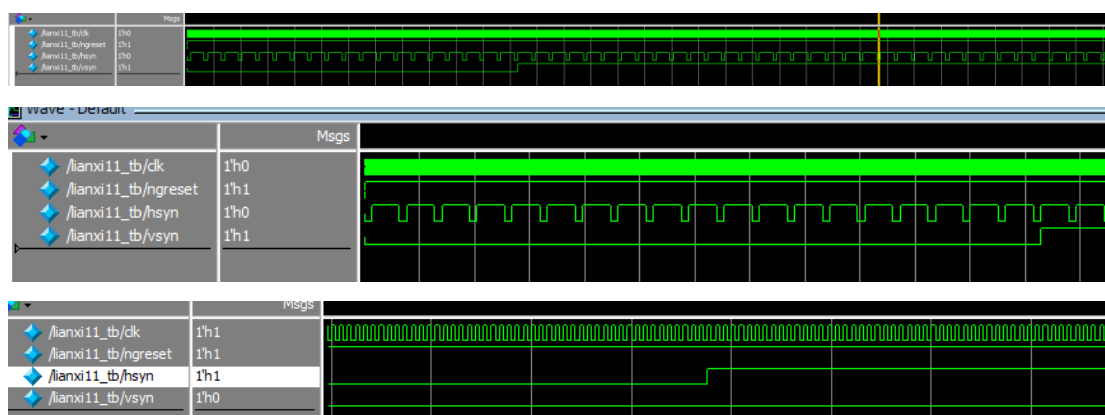
11. 输入时钟 clk 及复位信号 nreset, 输出行同步 hsyn、场同步 vsyn(场频 30fps); 要求写成可以合成的 RTL code, 编写 Testbench。

- 1) 其中 clk 频率为: 12MHz (83ns);
- 2) hsyn 信号=行消隐 (160 点) + 行有效 (640 点);
- 3) vsyn 信号=场消隐 (20 行) + 场有效 (480 行);
- 4) 1s = 时钟周期\*hsyn 总点数\*vsyn 总行数\*30fps。

代码如下:

Module	Testbench
<pre> module lianxi11(clk,nreset,hsyn,vsyn); input  clk,nreset; output hsyn,vsyn;  reg hsyn,vsyn;  reg[9:0] counter1; reg[9:0] counter2;  always@(posedge clk or nreset)begin     if(nreset == 1'b0)begin         hsyn &lt;= 1'b0;         vsyn &lt;= 1'b0;         counter1 &lt;= 1'b0;         counter2 &lt;= 1'b0;     end     else begin         if(counter1 == 799)begin             counter1 &lt;= 1'b0;         end         else counter1 &lt;= counter1 + 1'b1;     end end  always@(*)begin     if(counter1&lt;160) hsyn = 1'b0;     else hsyn = 1'b1; end  always@(posedge hsyn or negedge nreset)begin     if(nreset == 1'b0)begin         hsyn &lt;= 1'b0;         vsyn &lt;= 1'b0;         counter1 &lt;= 1'b0;         counter2 &lt;= 1'b0;     end     else begin         if(counter2 == 499)begin             counter2 &lt;= 1'b0;         end         else counter2 &lt;= counter2 + 1'b1;     end end  always@(*)begin     if(counter2&lt;20) vsyn = 1'b0;     else vsyn = 1'b1; end endmodule </pre>	<pre> `timescale 1ns/1ns module lianxi11_tb(); reg clk,nreset; wire hsyn,vsyn;  lianxi11 lianxi_11(.clk(clk),.nreset(nreset), .hsyn(hsyn),.vsyn(vsyn));  always #41.5 clk = ~clk;  initial begin     clk=0;     nreset =1;     # 5;     nreset =0;     # 3;     nreset =1; end endmodule </pre>

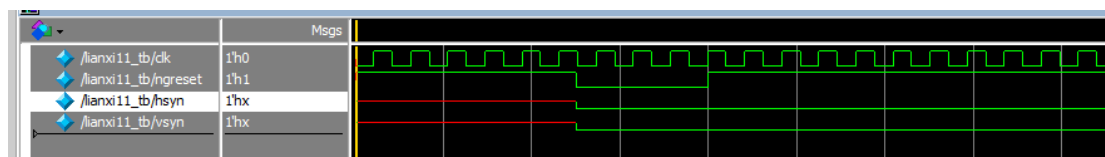
仿真结果如下图：



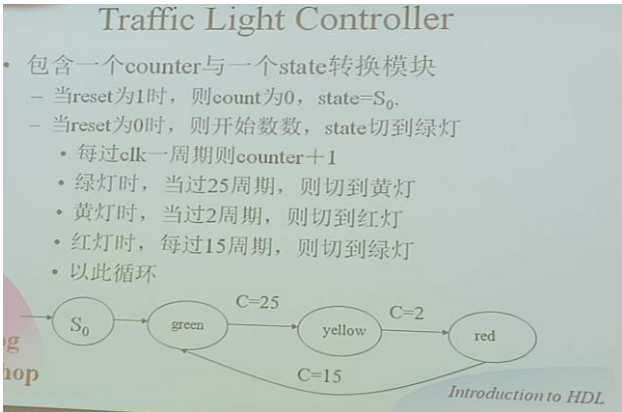
由图中可知：

- 1) 每当时钟信号为 160 个时，hsyn 信号变成 1，再过 640 个时钟信号后，hsyn 信号变成 0；
- 2) 当 hsyn 信号有 20 个时，vsyn 信号从 0 变成 1，再过 480 个 hsyn 信号时，vsyn 信号有从 0 变成了 1。

测试复位信号：（由图可知，当复位信号下降沿到来且为 0 时，hsyn 和 vsyn 信号的置为 0，符合设计的要求）



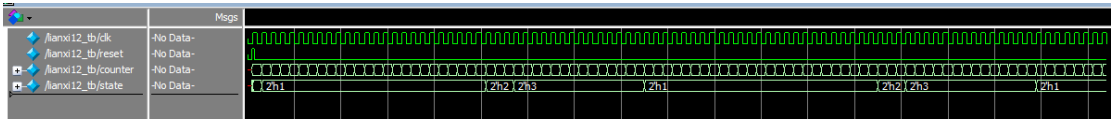
12. Traffic light controller 交通灯：要求



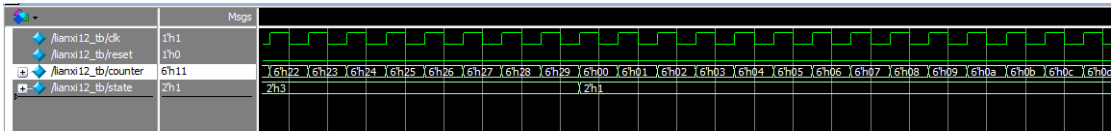
代码如下：

Module	Testbench
<pre>module lianxi12(clk,reset,count,state); input  clk ; input  reset; output [5:0] count; output [1:0] state; reg [5:0] count; reg [1:0] state; parameter S0=0,S1_GREEN=1,S2_YELLOW=2,S3_RED=3; always@(*)begin     if (count&lt;25) state = S1_GREEN;     else if ((count&gt;=25)&amp;(count&lt;=27))state =S2_YELLOW;     else if ((count&gt;=28)&amp;(count&lt;42))state=S3_RED; end always@(posedge clk or posedge reset)begin     if(reset == 1'b1)begin         count &lt;= 0;         state &lt;= S0;     end     else begin         if(count==41)count &lt;= 0;         else count &lt;= count + 1;     end end end endmodule</pre>	<pre>`timescale 1ns/1ns module lianxi12_tb(); reg clk,reset; wire[5:0] counter; wire[1:0] state;  lianxi12 lianxi_12(.clk(clk),.reset(reset),         .count(counter),.state(state));  always #5 clk =~clk;  initial begin     clk = 0;     reset = 0;     # 4     reset = 1;     # 3     reset = 0; end endmodule</pre>

仿真结果：



一个周期后重新计数，以及交通灯的状态从 3（红灯）变成 1（绿灯）



复位 reset 为 1 时，交通灯的状态变成 0。

