

第5章

Verilog运算符与结构描述语句

5.1 运算操作符

5.1.1 按位逻辑操作符

表 5-1 按位逻辑操作符功能说明与用法示例

逻辑操作符	逻辑功能	A,B 逻辑操作结果	C,D 逻辑操作结果	C,E 逻辑操作结果
~	逻辑取反	$\sim A = 1'b1$	$\sim C = 4'b0011$	$\sim E = 6'b101001$
	逻辑或	$A B = 1'b1$	$C D = 4'b1111$	$C E = 6'b011110$
&	逻辑与	$A \& B = 1'b0$	$C \& D = 4'b1000$	$C \& E = 6'b000100$
^	逻辑异或	$A \wedge B = 1'b1$	$C \wedge D = 4'b0111$	$C \wedge E = 6'b011010$
~^ 或 ^~	逻辑同或	$A \sim \wedge B = 1'b0$	$C \sim \wedge D = 4'b1000$	$C \sim \wedge E = 6'b100101$
设: $A=1'b0$; $B=1'b1$; $C[3:0]=4'b1100$; $D[3:0]=4'b1011$; $E[5:0]=6'b010110$;				

5.1 运算操作符

5.1.2 逻辑运算操作符

&& 逻辑与
|| 逻辑或
! 逻辑非。

例如设 $A=4'b1001$, $B=4'b0001$; 则:

$A \& B = (1|0|0|1) \& (0|0|0|1) = 1 \& 1 = 1'b1$

$1 \& z = 1'bz$; $0 \& z = 1'b0$; $1 | z = 1'b1$; $0 | z = 1'bz$;

5.1 运算操作符

5.1.3 算术运算操作符

表 5-2 算术操作符功能及其示例

逻辑操作符	功能	说明	示例
+	加		$S = A + B = 8' b00011000$
-	减		$S = B - A = 8' b11111110$
*	乘		$S = A * B = 8' b10001111 = 2' H8F$
/	除	结果：小数抛弃	$S = A / 3 = 8' b00000100$
%	求余	除法求余数	$S = A \% 3 = 8' b00000001$

设示例数据是： $A[3:0] = 4' b1101$ ； $B[3:0] = 4' b1011$ ； 定义 s 为 $s[7:0]$

5.1 运算操作符

5.1.3 算术运算操作符

【例 5-1】

```
module test1 (A,B,C,D,RCD,RAB,RM1,RM2,S,C0,R1,R2);  
input [3:0] C,D ; input signed [3:0] A,B;  
output [3:0] RCD; output [3:0] RAB;  
output [7:0] RM1; output [7:0] RM2;  
output [3:0] S; output C0; output R1,R2;  
reg [3:0] S ; reg C0;  
reg [3:0] RCD ; reg [7:0] RM1 ;  
reg signed [3:0] RAB; reg signed [7:0] RM2;  
reg R1,R2;  
always@(A,B,C,D) begin  
    RCD <= C+D; RAB <= A+B;  
    RM1 <= C*D; RM2 <= A*B;  
    {C0,S} <= {1'b0,C} - {1'b0,D}; //注意并位操作  
    R1 <= (C>D); R2<=(A>B);  
end  
endmodule
```

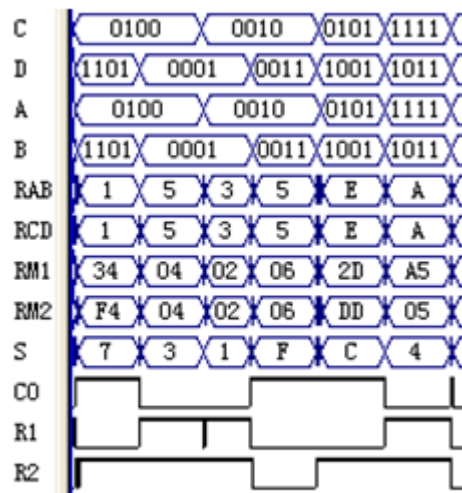


图 5-1 例 5-1 的仿真波形

5.1 运算操作符

5.1.4 关系运算操作符

表 5-3 等式操作符与示例

等式操作符	含义	等式操作示例
==	等于	(3==4) = 0; (A==4'b1011) = 1; (B==4'b1011) = 0;
!=	不等于	(D!=C) = 1; (3!=4) = 1;
===	全等	(D===C) = 1; (E===4'b0x10) = 0;
!==	不全等	(E!==4'b0x10) = 1;

设: A=5'b01011; B=4'b0010; C=4'b0z10; D=4'b0z10; E=3'bx10

表 5-4 不等式操作符与示例

等式操作符	含义	操作示例
>	大于	(A < B) = 0; (A > B) = 1;
<	小于	(A < 20) = 1; (A > 12) = 1;
<=	小于或等于	(A >= 14) = 0; (A <= 13) = 1;
>=	大于或等于	注, 以上示例中, 设 A=4'B1101; B=4'B0110

5.1 运算操作符

5.1.5 BCD码加法器设计示例

【例 5-2】

```
module BCD_ADDER (A,B,D) ;
  input [7:0] A,B;  output [8:0] D;
  wire [4:0] DT0, DT1 ; reg [8:0] D; reg S;
  always@ (DT0)
    begin if (DT0[4:0] >= 5'b01010 )
      //如果低位 BCD 码的和大于等于 10,则使和加上 6, 且有进位, 使进位标志 s 等于 1。
      begin D[3:0] = (DT0[3:0]+4'b0110); S=1'b1; end
      else begin D[3:0] = DT0[3:0] ; S=1'b0; end
    end //否则, 将低位值赋予低位 BCD 码 D[3:0]输出, 无进位, 使进位标志 s 等于 0。
  always@ (DT1)  begin
    if (DT1[4:0]>=5'b01010)
      begin D[7:4] = (DT1[3:0]+4'b0110); D[8]=1'b1; end
      else begin D[7:4] = DT1[3:0] ; D[8]=1'b0; end
    end
  assign DT0 = A[3:0] + B[3:0] ; //设没有来自低位的进位。
  assign DT1 = A[7:4] + B[7:4] + S; //S 是来自低位 BCD 码相加的进位。
endmodule
```

5.1 运算操作符

5.1.5 BCD码加法器设计示例

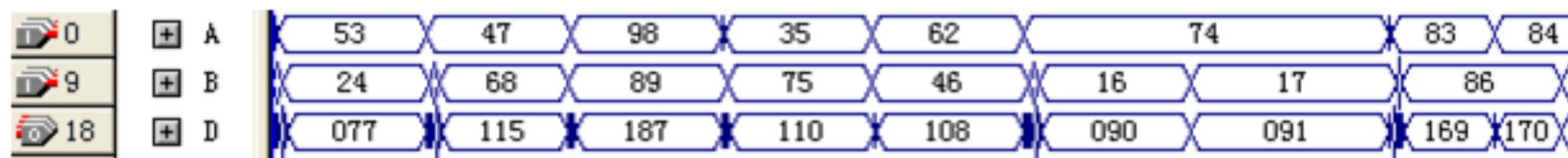
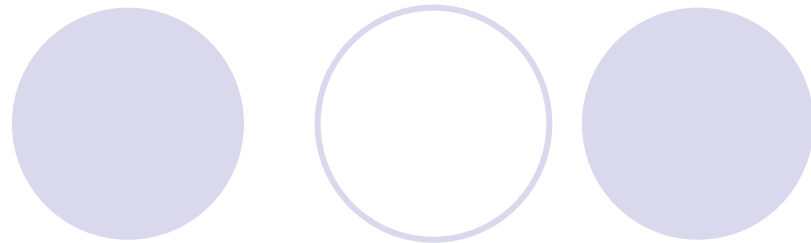


图 5-2 例 5-2 的仿真波形

5.1 运算操作符



5.1.6 缩位操作符

$\&$ (与) $\sim\&$ (与非)
 $|$ (或) $\sim|$ (或非)
 \wedge (异或) $\wedge\sim, \sim\wedge$ (同或)

5.1.7 并位操作符

$\{a1, b1, 4\{a2, b2\}\} = \{a1, b1, \{a2, b2\}, \{a2, b2\}, \{a2, b2\}, \{a2, b2\}\} = \{a1, b1, a2, b2, a2, b2, a2, b2, a2, b2\}$

5.1 运算操作符

5.1.8 移位操作符应用法

$V \gg n$ 或 $V \ll n$

$V \gg 1$ 的值是 $8'b01100100$;

$V \ll 3$ 的值是 $8'b01001000$

$V \ggg n$ 或 $V \lll n$

```
output signed[7:0] y;  
input signed[7:0] a;  
assign y = (a<<<2);  
若 a=10101011,则输出 y=10101100  
若 a=10001111,则输出 y=00111100
```

```
parameter C=8'sb10101011;  
parameter D=8'sb01001110;  
output [7:0] Y1,Y2;  
assign Y1=(C>>>2);//结果: Y1=11101010  
assign Y2=(D>>>2);//结果: Y2=00010011
```

5.1 运算操作符

5.1.9 使用移位操作符的设计示例

【例 5-3】	【例 5-4】
<pre>module MULT4B(R,A,B); parameter S=4; output[2*S:1] R ; input[S:1] A,B ; reg[2*S:1] R; integer i; always @(A or B) begin R = 0 ; for(i=1; i<=S; i=i+1) if(B[i]) R=R+(A<<(i-1)); end endmodule</pre>	<pre>module MULT4B (R,A,B); parameter S=4; output[2*S:1] R; input[S:1] A,B; reg[2*S:1] R,AT; reg[S:1] BT,CT; always @(A,B) begin R=0; AT = {{S{1'B0}},A}; BT = B; CT = S; for(CT=S; CT>0; CT=CT-1) begin if(BT[1]) R=R+AT; AT = AT<<1; BT = BT>>1; end end end endmodule</pre>



图 5-3 四位乘法器时序仿真图

5.1 运算操作符

5.1.9 使用移位操作符的设计示例

【例 5-5】	【例 5-6】
<pre>module SHIF4(DIN,CLK,RST,DOUT); input CLK,DIN,RST; output DOUT; reg [3:0] SHFT; always@(posedge CLK or posedge RST) if(RST) SHFT<=4'B0; else begin SHFT[3]<=DIN; SHFT[2:0] <= SHFT[3:1]; end assign DOUT=SHFT[0]; endmodule</pre>	<pre>module SHIF5 (DIN,CLK,RST,DOUT); input CLK,DIN,RST; output DOUT; reg [3:0] SHFT; always@(posedge CLK or posedge RST) if(RST) SHFT<=4'B0; else begin SHFT <= (SHFT >> 1); SHFT[3] <= DIN; end assign DOUT = SHFT[0]; endmodule</pre>

5.1 运算操作符

5.1.10 条件操作符

条件表达式 ? 表达式1 : 表达式2

【例 5-7】

```
module DFF2 (input CLK, input D, input RST , output reg Q );  
    always @(posedge CLK )    Q <= RST ? 1'b0 : D;  
endmodule
```

5.2 连续赋值语句

```
assign 目标变量名 = 驱动表达式;
```

```
assign [延时] 目标变量名 = 驱动表达式;
```

【例 5-8】

```
module MUX41a (A,B,C,D,S1,S0,Y);  
  input A,B,C,D,S1,S0;  
  output Y;  
  assign AT = S0 ? D : C ;  
  assign BT = S0 ? B : A ;  
  wire Y = (S1 ? AT : BT);  
endmodule
```

```
`timescale 10ns/100ps  
assign #6 R1 = A & B;
```

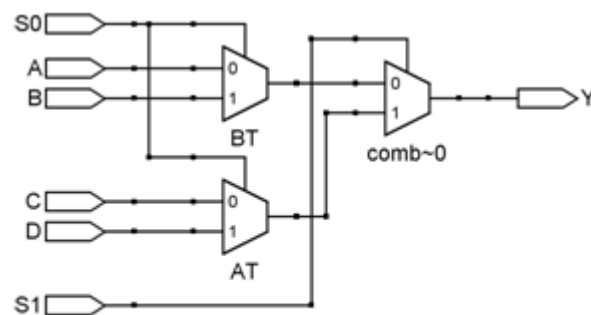


图 5-4 例 5-8 的 RTL 图

5.3 例化语句

5.3.1 半加器设计

【例 5-9】

```
module h_adder (A,B,SO,CO) ;  
    input A,B;  
    output SO,CO;  
    assign SO = A ^ B; //将变量 A 和 B 执行异或逻辑后将结果赋给输出信号 SO  
    assign CO = A & B; //将变量 A 和 B 执行与逻辑后将结果赋给输出信号 CO  
endmodule
```

5.3 例化语句

5.3.2 全加器设计

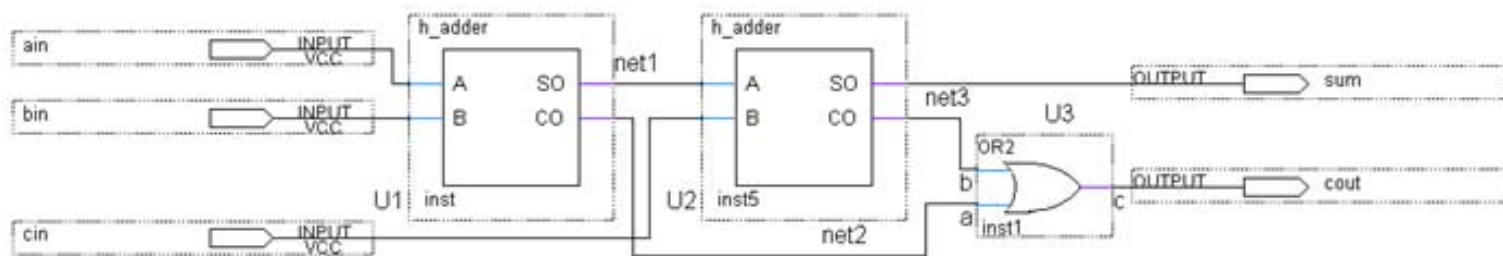
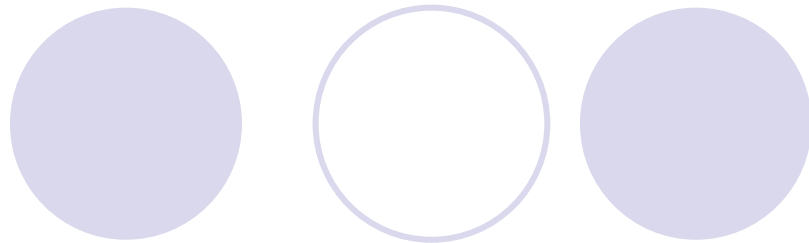


图 4-31 全加器 f_adder 电路图

【例 5-10】

```
module f_adder(ain,bin,cin,cout,sum);  
    output cout,sum;    input ain,bin,cin;  
    wire net1,net2,net3;  
    h_adder U1( ain, bin, net1, net2);  
    h_adder U2( .A(net1), .SO(sum), .B(cin), .CO(net3) );  
    or U3(cout, net2, net3);  
endmodule
```


5.3 例化语句



5.3.3 Verilog例化语句及其用法

1. 例化语句端口名关联法

<模块元件名> <例化元件名> (.例化元件端口 (例化元件外接端口名) , ...) ;

```
h_adder U2 (.A(net1), .SO(sum), .B(cin), .CO(net3));
```

```
h_adder U2 (.B(cin), .CO(net3), .A(net1), .SO(sum));
```

2. 例化语句位置关联法

5.4 参数传递语句应用

【例 5-11】	【例 5-12】
<pre>module MULT4B #(parameter S) (R,A,B); output[2*S:1] R ; input[S:1] A,B ; reg[2*S:1] R; integer i; ... //以下与例 5-3 相同</pre>	<pre>module MULTB (RP,AP,BP); output[15:0] RP ; input[7:0] AP,BP ; MULT4B #(.S(8)) U1(.R(RP), .A(AP), .B(BP)); endmodule</pre>

```
module SUB_E
#(parameter S1=4, parameter S2=5, parameter S3=2) (A,B,C);

SUB_E #(.S1(8), .S2(9), .S3(7)) U1(.C(CP), .A(AP), .B(BP) );
```

5.5 用库元件实现结构描述

【例 5-13】

```
module LOGICGATE (input A,B,C,S , output OUT);  
  wire a1,a2,a3,a4;  
  not u1 (a1,B);  
  and u2 (a2,A,a1);  
  or u3 (a3,C,B);  
  xor u4 (a4,a3,a2);  
  notif1 u5 (OUT,a4,S);  
endmodule
```

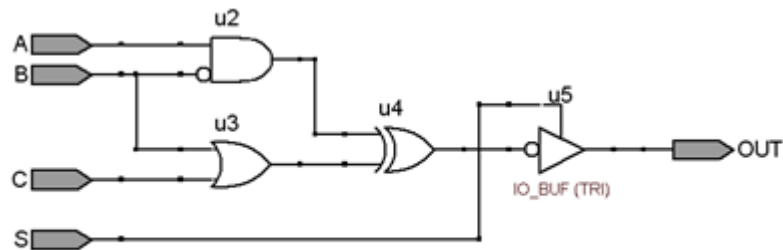


图 5-5 例 5-13 描述的逻辑电路

5.5 用库元件实现结构描述

基本门元件名 <门例化名> (<端口关联列表>)

(输出, 输入 1, 输入 2, 输入 3, ……);

```
and U1 (out,in1,in2,in3);      //三输入与门, 例化名是 U1
and U2 (out,in1,in2);        //二输入与门, 例化名是 U2
```

```
bufif1 U1(out,in,enable);    //高电平使能的三态门
bufif2 U2(out,a,ctrl);       //低电平使能的三态门
```

```
not IC1 (out1,out2,in);      //1 输入 in, 2 输出 out1,out2
buf IC2 (out1,out2, out3,in); //1 输入 in, 3 输出 out1,out2, out3
```

5.6 用户自定义元件 (UDP)

```
primitive 元件名 (输出端口名, 输入端口名 1, 输入端口名 2,...)
  output 输出端口名;
  input 输入端口名 1, 输入端口名 2,...;
  reg 输出端口名;
  initial begin
    // 输出端口寄存器或时序逻辑内部寄存器赋初值 (0,1,X);
    ...
  end
  table
  // 输入 1   输入 2   ...   :   输出
  逻辑值     逻辑值     :   逻辑值;
  逻辑值     逻辑值     :   逻辑值;
  ...
  endtable
endprimitive
```

5.6 用户自定义元件 (UDP)

5.6.1 UDP组合元件设计

【例 5-14】自定义 UDP 元件 XOR2

```
primitive XOR2 (DOUT,X1,X2);  
  input X1,X2;  output DOUT;  
  table // X1  X2  : DOUT  
        0  0  :  0;  
        0  1  :  1;  
        1  0  :  1;  
        1  1  :  0;  
  endtable  
endprimitive
```

【例 5-15】例化 XOR2 和 and, 构成半加器

```
module H_ADDER (A,B,SO,CO) ;  
  input A,B;  
  output SO,CO;  
  XOR2 U1(SO,A,B); // 调用元件 XOR2  
  and U2(CO,A,B); // 调用元件 and  
endmodule
```

5.6 用户自定义元件 (UDP)

5.6.2 UDP时序元件设计

【例 5-16】 DFF UDP 元件

```
primitive primdff (Q, Clk, Data) ;      // DFF UDP
    output Q ;    input Data, Clk;    // 只有一个输出
    reg Q ;
    initial Q = 0;      // Q的初始化
    table
        // Clk Data Q (当前状态) Q(下一个状态)
        (01) 0 : ? : 0 ;      // 上升沿, Data 为 0
        (01) 1 : ? : 1 ;      // 上升沿, Data 为 1
        (0x) 1 : 1 : 1 ;      // 其他状态, 保持
        (0x) 0 : 0 : 0 ;
        (?0) ? : ? : - ;
        ? (??): ? : - ;
    endtable
endprimitive
```

5.6 用户自定义元件 (UDP)

5.6.2 UDP时序元件设计

表 5-5 UDP Table 中符号含义

符号	表示	说明
1	逻辑 1	
0	逻辑 0	
X	未知	允许在所有 UDP 的输入域以及时序 UDP 的当前状态域
?	指代 0、1 和 x	不允许在输出域
B	指代 0、1	允许在所有 UDP 的输入域以及时序 UDP 的当前状态域, 不允许在输出域
-	不改变	仅允许在时序 UDP 的输出域
(vw)	值从 v 到 w 变化	v 和 w 可以是 0、1、x、? 或 b 其中之一, 不允许在输出域
*	同(??)	任何输入值变化
R	同(01)	输入上升沿
F	同(10)	输入下降沿
P	指代(01)、(0x)和(x1)	输入可能的正边沿
N	指代(10)、(x0)和(x0)	输入可能的负边沿

5.6 用户自定义元件 (UDP)

5.6.2 UDP时序元件设计

【例 5-17】 D-Latch UDP 元件	【例 5-18】 DFF UDP 元件
<pre>primitive primlatch (q, clock, data); output q; input clock, data; reg q; table // clock data q q+ 0 1 : ? : 1 ; // 直通 0 0 : ? : 0 ; // 直通 1 ? : ? : - ; // 锁存 endtable endprimitive</pre>	<pre>primitive dff1 (q, clk, d); input clk, d; output q; reg q; initial q = 1'b1; table // clk d q q+ r 0 : ? : 0 ; // 上升沿 r 1 : ? : 1 ; // 上升沿 f ? : ? : - ; // 下降沿 ? * : ? : - ; // 其他状态 endtable endprimitive</pre>

5.6 用户自定义元件 (UDP)

5.6.2 UDP时序元件设计

【例 5-19】 双边沿 D 触发器元件

```
primitive prim_ddff (q, clk, d); // 双边沿 D 触发器 UDP
input clk, d; output q; reg q;
initial q = 1'b1; // q 初始化
table
// clk d q q+
r 0 : ? : 0 ; // 上升沿
r 1 : ? : 1 ; // 上升沿
f 0 : ? : 0 ; // 下降沿
f 1 : ? : 1 ; // 下降沿
? * : ? : - ; // 其他状态
endtable
endprimitive
```

例化程序 prim_ddff 如下:

```
module test_pddff(q, clk, d);
input clk, d; output q;
prim_ddff U1(q,clk,d); // 例化 UDP
endmodule
```

5.7 编译指示语句

5.7.1 宏定义命令语句

```
\define 宏名 (标志符) 宏内容 (字符串)
```

```
\define s A+B+C+D
```

```
assign DOUT= `s + E;
```

```
assign DOUT = A+B+C+D+E;
```

5.7 编译指示语句

5.7.2 文件包含语句 include

```
`include "文件名"
```

【例 5-20】

```
`include " h_adder.v "  
`include " or2a.v "  
    module f adder(input ain,bin,cin,output cout,sum );  
        wire e,d,f ;  
        h adder u1( ain, bin, e, d );  
        h adder u2(.a(e), .so(sum), .b(cin),.co(f) );  
        or2a u3(.a(d), .b(f), .c(cout) );  
    endmodule
```

5.7 编译指示语句

5.7.3 条件编译语句`ifdef`、`else`、`endif`

条件编译命令语句格式 1	条件编译命令语句格式 2
<pre>`ifdef 宏名 语句块 `endif</pre>	<pre>`ifdef 宏名 语句块 1 `else 语句块 2 `endif</pre>

5.7 编译指示语句

5.7.3 条件编译语句`ifdef`、`else`、`endif`

【例 5-21】

```
`define AND
module andd (out,A,B);
input[1:0] A,B;
output [1:0] out;
`ifdef AND
assign out=A&B;
`else assign out=A|B;
`endif
endmodule
```

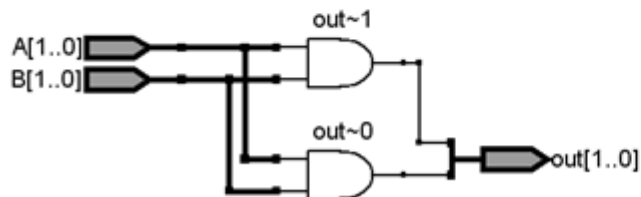


图 5-6 对应例 5-21 的 RTL 图

【例 5-22】

```
`define OR1
module andd (out,A,B);
input[1:0] A,B;
output [1:0] out;
`ifdef AND
assign out=A&B;
`else assign out=A|B;
`endif
endmodule
```

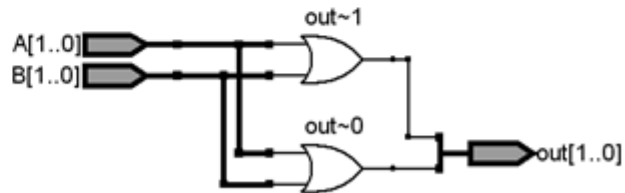


图 5-7 对应例 5-22 的 RTL 图

5.8 keep属性应用

5.7.3 条件编译语句`ifdef`、`else`、`endif`

【例 5-23】

```
module ff_adder(ain,bin,cin,cout,sum);  
    output cout,sum ;    input ain,bin,cin ;  
    (* synthesis, keep *) wire net1 ;  
    wire net2,net3 ;  
    ... //以下与了 5-10 相同
```

(* synthesis, keep *) 或 (* synthesis, probe_port, keep *)

5.8 keep属性应用

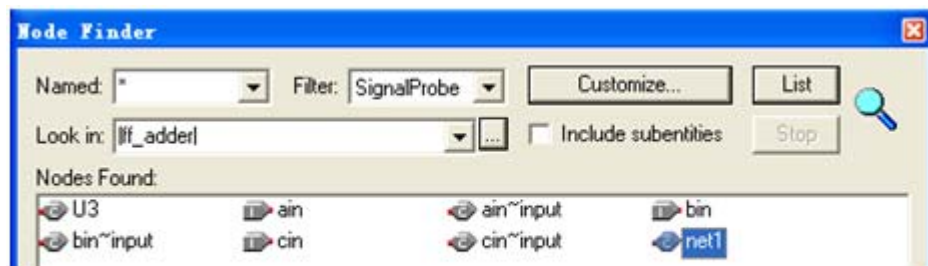


图 5-8 加入仿真测试信号 net1

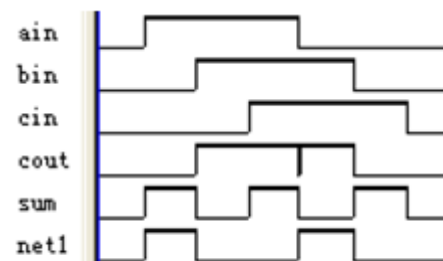


图 5-9 例 5-23 的仿真波形

```
(* synthesis, probe_port, keep *) wire net1;
```

```
(* synthesis, probe_port, keep *) reg [7:0] A ;
```


5.9 SignalProbe使用方法

1. 按常规流程完成设计仿真和硬件测试
2. 设置SignalProbe Pins

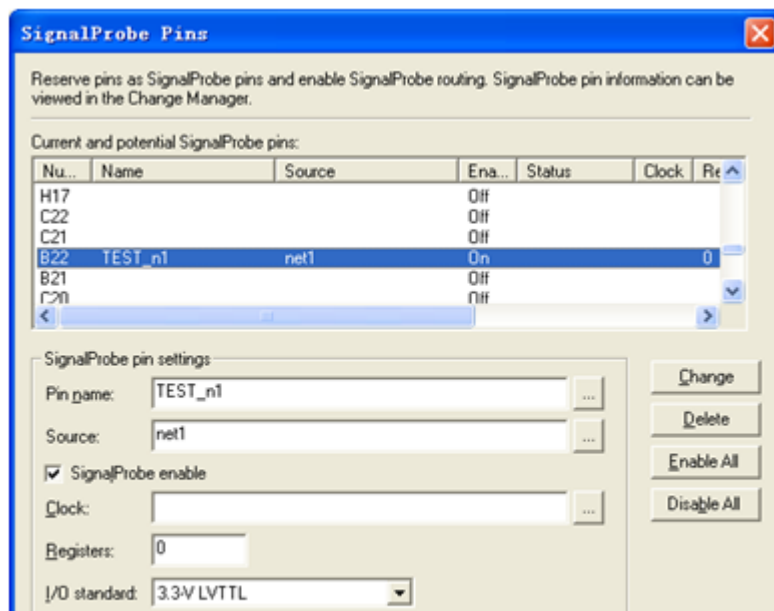


图 5-10 在 SignalProbe 对话框设置探测信号 net1

5.9 SignalProbe使用方法

3. 编译SignalProbe Pins测试信息并下载测试

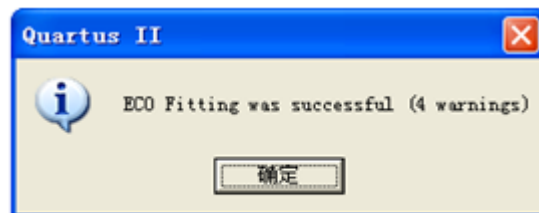


图 5-11 ECO 文件编译成功

习题

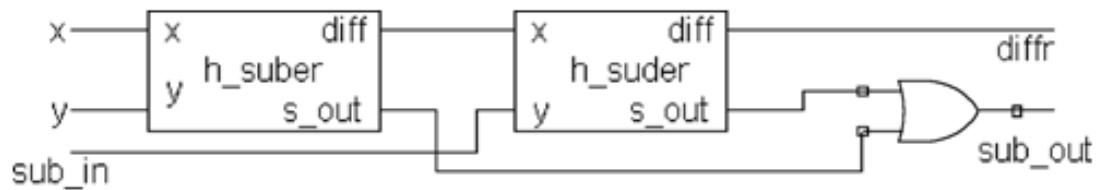


图 5-12 全减器模块图

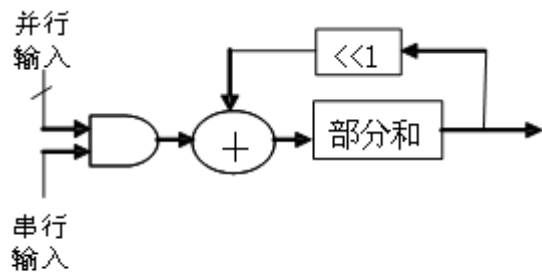


图5-13 最基本的硬件乘法器

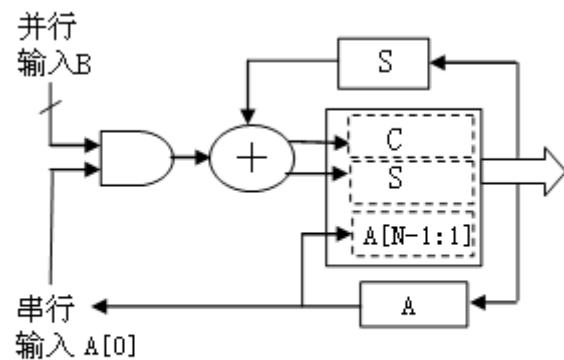


图5-14 改进后的硬件乘法器

习题

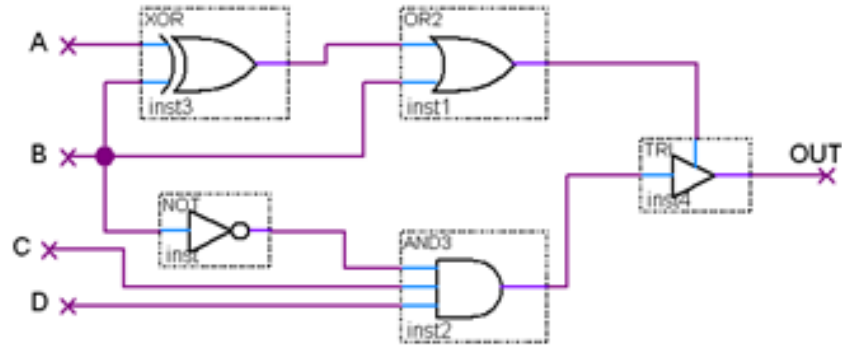


图 5-15 习题 5-15 逻辑电路图

实验与设计

5-1 高速硬件除法器设计实验

【例5-24】

```
module DIV16 (input CLK, input [15:0] A,B, output reg [15:0] QU,RE);
    reg [15:0] AT,BT,P,Q; integer i;
    always @(posedge CLK) begin
        AT = A ; BT = B; P = 16'H0000; Q = 16'H0000 ;
        for (i=15; i>=0; i=i-1)
            begin P={P[14:0], AT[15]}; AT={AT[14:0],1'B0} ; P=P-BT;
                if (P[15]==1) begin Q[i]=0; P = P+BT ; end
                else Q[i]=1 ; end
            end
        always @( * ) begin QU = Q; RE = P ; end
    endmodule
```

实验与设计

5-2 不同类型的移位寄存器设计实验

5-3 基于Verilog代码的频率计设计

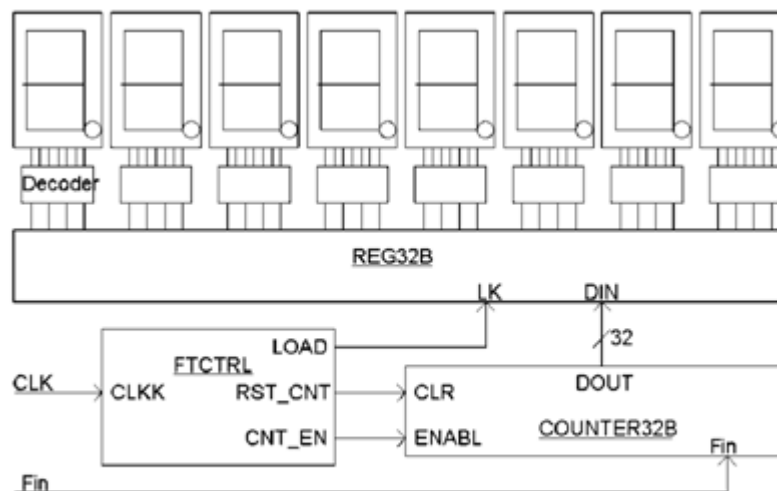


图 5-16 频率计电路框图

实验与设计

5-3 基于Verilog代码的频率计设计

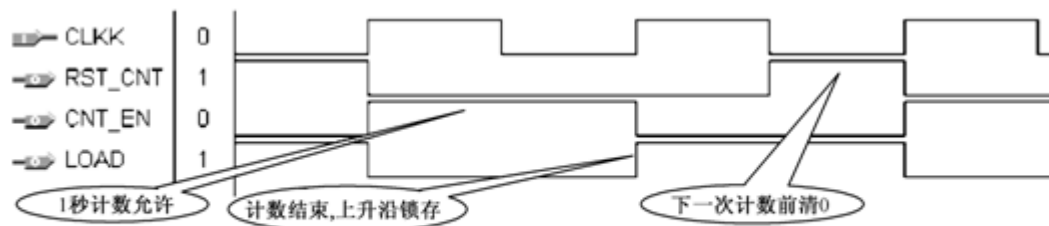


图 5-17 频率计测频控制器 FTCTRL 测控时序图

【例 5-25】

```
module FTCTRL (CLKK, CNT_EN, RST_CNT, LOAD);
    input CLKK;          output CNT_EN, RST_CNT, LOAD;
    wire CNT_EN, LOAD;   reg RST_CNT, Div2CLK;
    always @(posedge CLKK) Div2CLK <= ~Div2CLK ;
    always @(CLKK or Div2CLK) begin
        if (CLKK==1'b0 & Div2CLK==1'b0) RST_CNT <= 1'b1 ;
        else RST_CNT <= 1'b0 ;          end
    assign LOAD = ~Div2CLK ;          assign CNT_EN = Div2CLK ;
endmodule
```

实验与设计

5-4. 8位加法器设计实验

5-5 VGA彩条信号显示控制电路设计

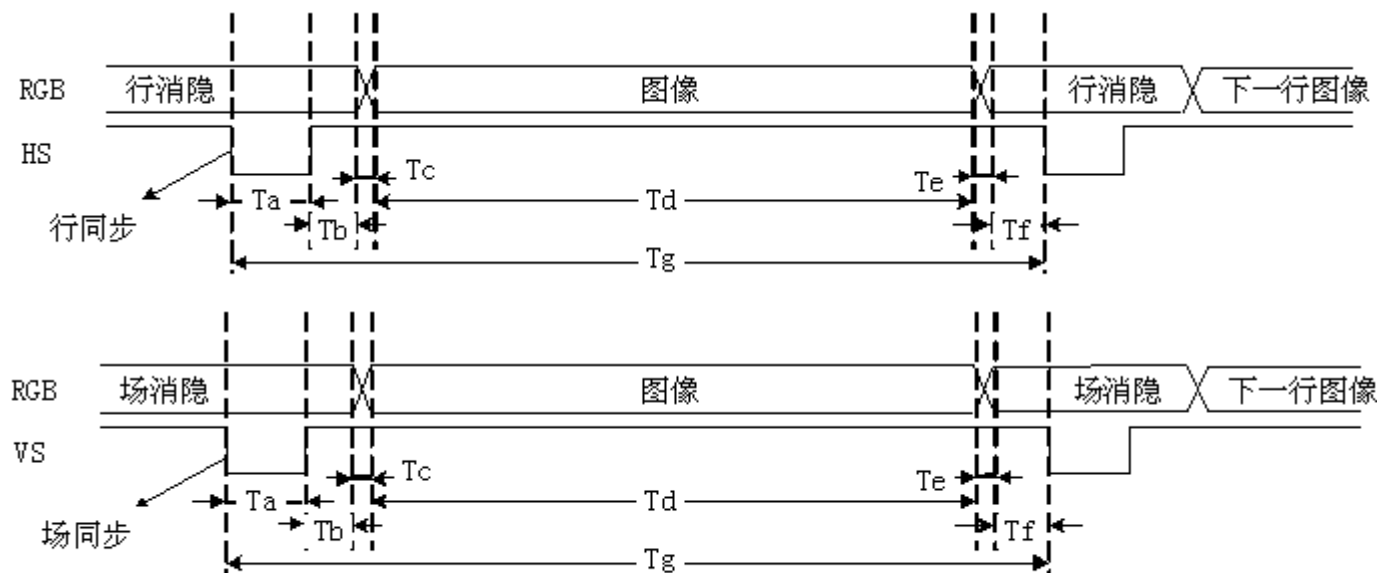


图 5-18 VGA 行扫描、场扫描时序示意图

实验与设计

5-5 VGA彩条信号显示控制电路设计

表 5-6 行扫描时序要求：(单位：像素，即输出一个像素 Pixel 的时间间隔)

		行同步头				行图像		行周期
对应位置	Tf	Ta	Tb	Tc	Td	Te	Tg	
时间(Pixels)	8	96	40	8	640	8	800	

表 5-7 场扫描时序要求：(单元：行，即输出一行 Line 的时间间隔)

		行同步头				行图像		行周期
对应位置	Tf	Ta	Tb	Tc	Td	Te	Tg	
时间(Lines)	2	2	25	8	480	8	525	

实验与设计

5-5 VGA彩条信号显示控制电路设计

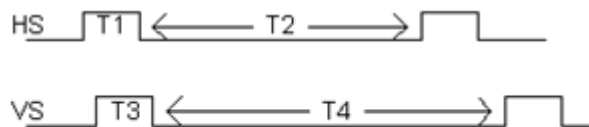


图 5-19 HS 和 VS 的时序图

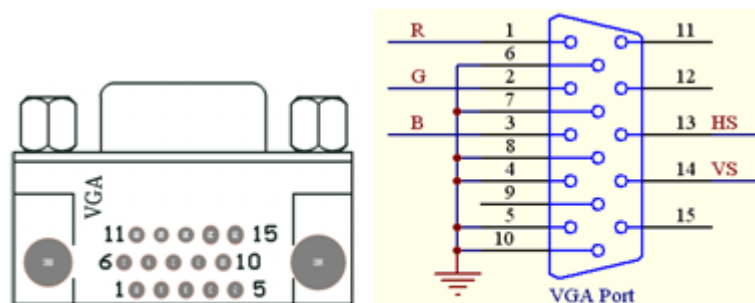


图 5-20 VGA 接口电路图，左接口从上往下看

表 5-8 颜色编码:

颜色	黑	蓝	红	品	绿	青	黄	白
R	0	0	0	0	1	1	1	1
G	0	0	1	1	0	0	1	1
B	0	1	0	1	0	1	0	1

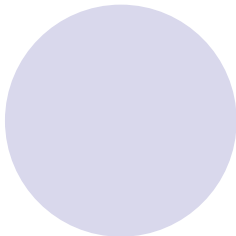
表 5-9 彩条信号发生器 3 种显示模式,

1	横彩条	1: 白黄青绿品红蓝黑	2: 黑蓝红品绿青黄白
2	竖彩条	1: 白黄青绿品红蓝黑	2: 黑蓝红品绿青黄白
3	棋盘格	1: 棋盘格显示模式 1	2: 棋盘格显示模式 2

【例 5-26】

```
module VGA_COLOR_LINE (CLK, MD, HS, VS, R, G, B); //VGA 显示器 彩条 发生器
    input CLK, input MD;          output HS, VS, R, G, B;
        wire R,G,B,VS,HS;          //红, 绿, 蓝信号, 和场同步, 行同步信号
        wire FCLK, CCLK;    reg HS1, VS1;    reg[1:0] MMD;    reg[4:0] FS;
    reg[4:0] CC;                    //行同步, 横彩条生成
    reg[8:0] LL;                    //场同步, 竖彩条生成
    reg[3:1] GRBX,GRBY,GRBP;        // X 横彩条, Y 竖彩条
    wire[3:1] GRB;
    assign GRB[2] = (GRBP[2] ^ MD) & HS1 & VS1 ;
    assign GRB[3] = (GRBP[3] ^ MD) & HS1 & VS1 ;
    assign GRB[1] = (GRBP[1] ^ MD) & HS1 & VS1 ;
    always @(posedge MD) begin
        if (MMD==2'b10) MMD<=2'b00; else MMD<=MMD+1 ; end //3 种模式
    always @(MMD) begin
        if (MMD == 2'b00) GRBP <= GRBX ; // 选择横彩条
        else if (MMD == 2'b01) GRBP <= GRBY ; // 选择竖彩条
        else if (MMD == 2'b10) GRBP <= GRBX ^ GRBY ; //产生棋盘格
        else GRBP <= 3'b000 ; end
    always @(posedge CLK ) begin // 20MHz 21 分频
        if (FS==20) FS<=0; else FS<=(FS+1) ; end
    always @(posedge FCLK) begin
        if (CC==29) CC<=0; else CC<=CC+1 ; end
```

接下页



```
always @(posedge CCLK) begin
    if (LL==481) LL<=0; else LL<=LL+1 ;    end
always @(CC or LL) begin
    if (CC > 23) HS1<=1'b0; else HS1<=1'b1 ; //行同步
    if (LL > 479) VS1<=1'b0; else VS1<=1'b1 ;    end //场同步
always @(CC or LL) begin
    if (CC < 3) GRBX <= 3'b111 ;           // 横彩条
    else if (CC < 6) GRBX <= 3'b110 ;
    else if (CC < 9) GRBX <= 3'b101 ;
    else if (CC < 12) GRBX <= 3'b100 ;
    else if (CC < 15) GRBX <= 3'b011 ;
    else if (CC < 18) GRBX <= 3'b010 ;
    else if (CC < 21) GRBX <= 3'b001 ;
    else GRBX <= 3'b000 ;
    if (LL < 60) GRBY <= 3'b111 ; // 竖彩条
    else if (LL < 120) GRBY <= 3'b110 ;
    else if (LL < 180) GRBY <= 3'b101 ;
    else if (LL < 240) GRBY <= 3'b100 ;
    else if (LL < 300) GRBY <= 3'b011 ;
    else if (LL < 360) GRBY <= 3'b010 ;
    else if (LL < 420) GRBY <= 3'b001 ;
    else GRBY <= 0 ;    end
assign HS = HS1 ;    assign FCLK = FS[3] ;
assign HS = HS1 ;    assign VS = VS1 ; assign R = GRB[2] ;
assign G = GRB[3] ; assign B = GRB[1] ; assign CCLK = CC[4] ;
endmodule
```

实验与设计

5-6 移位相加型8位硬件乘法器设计

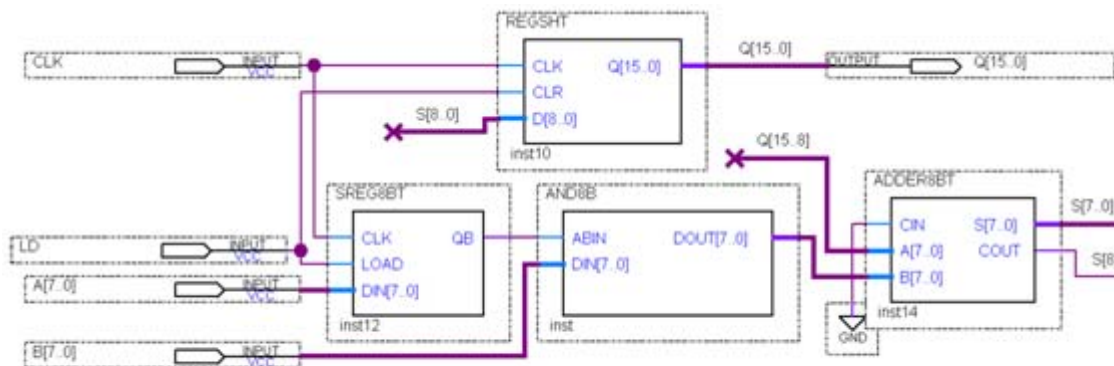


图 5-21 8 位乘法器逻辑原理图

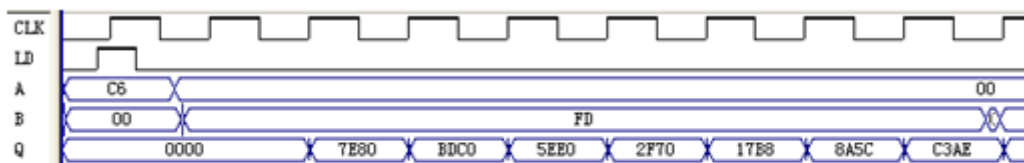


图 5-22 8 位移位相加乘法器运算逻辑波形图