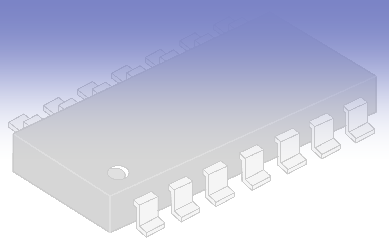
第 10 章

组合逻辑设计

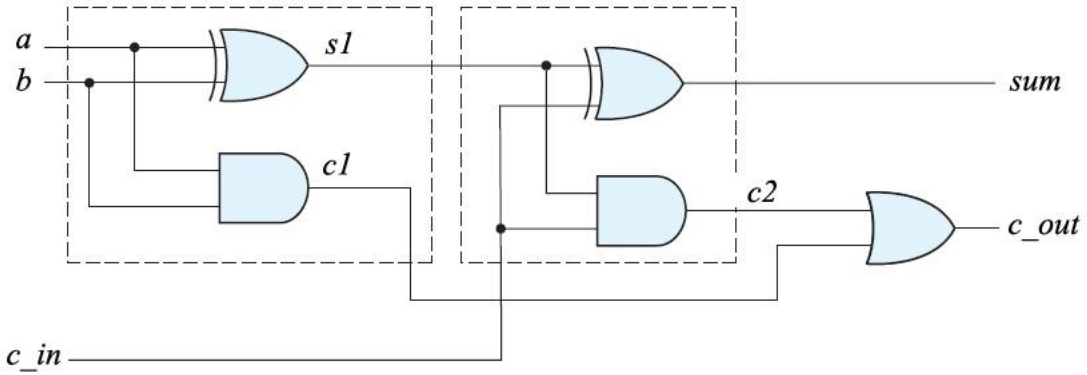
内容



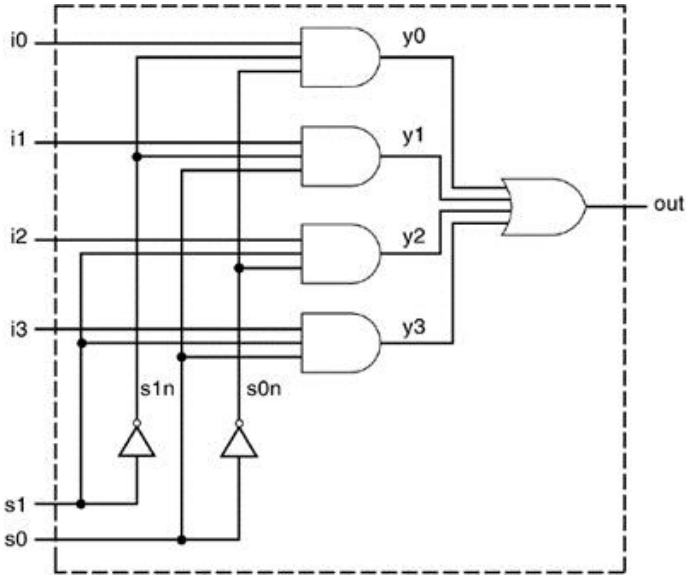
- 电路模块的设计
  - 多路复用器
  - 译码器
  - 编码器
  - 三态缓冲器
  - 比较器
  - 加法器
  - 乘法器

组合逻辑电路

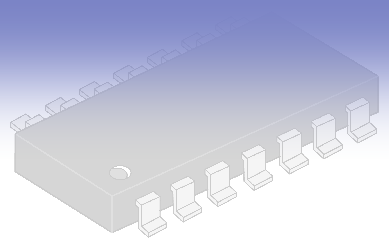
- 输出完全由当前输入决定
  - 一旦输入信号变化，输出随之改变
- 不包含触发器
  - 连续赋值语句 assign
    - 只要右边表达式中的操作数上有事件发生 —— 表达式立即被计算，新结果赋给左边的线网



- 带有电平敏感量列表always 语句块
  - 当敏感量列表中的变量上有事件发生
  - 执行语句块中的行为语句
  - 直到执行完最后一条语句后
  - 等待下一次事件发生



2 选 1 多路选择器（1）



功能

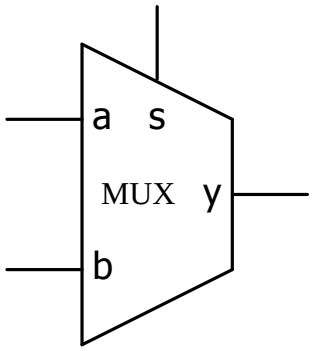
- 若 s 为 1，则输入 a 连接到输出 y
- 若 s 为 0，则输入 b 连接到输出 y
- 使用连续赋值语句描述——可以综合

```
module mux2to1(output y, input s, b, a);
    assign y = ( s ) ? b : a;
endmodule
```

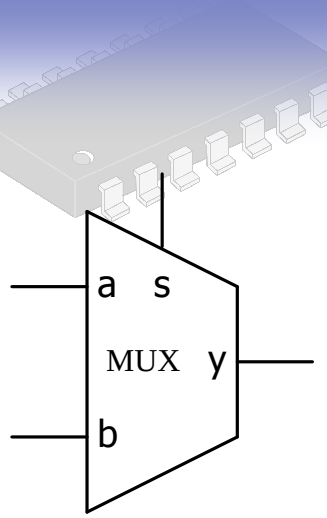
- 比较：采用用户定义原语描述——不可综合

```
primitive mux2x1_udp(output f, input s, I0, I1 );

// 定义 UDP 状态表
table
// input are in the same order as the input list
//   s    I0   I1   :    f    // 标识符用于增加可读性
     0    0    ?    :    0;
     0    1    ?    :    1;
     1    ?    0    :    0;
     1    ?    1    :    1;
     ?    0    0    :    0;
     ?    1    1    :    1;
endtable
endprimitive
```

- 2 to 1 多路选择器符号



2 选 1 多路选择器（2）

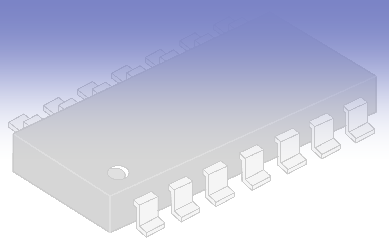


- 功能
  - 使用过程语句描述 —— 可以综合
    - 使用条件表达式
    - 使用条件语句

```
module mux2to1(output reg y, input s, b, a);
   always @(*)
      y = ( s ) ? b : a;
endmodule
```

```
module mux2to1(output reg y, input s, b, a);
   always @(*)
      if ( s ) y = b;
      else y = a;
endmodule
```

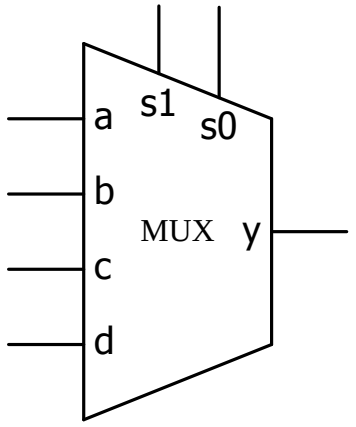
4 选 1 多路选择器（1）



- 功能
  - 由 s1和 s0 选择 a、b、c 和 d 连接到输出 y
  - 使用连续赋值语句描述 —— 可以综合

```
module mux4to1(output y, input s1, s0, d, c, b, a );
   assign y = (s1) ? ((s0) ? d : c) : ((s0) ? b : a);
endmodule
```

s1	s0	y
0	0	a
0	1	b
1	0	c
1	1	d



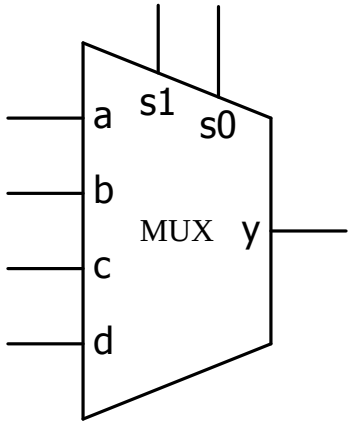
4 to 1 多路选择器符号

4 选 1 多路选择器（2）

s1	s0	y
0	0	a
0	1	b
1	0	c
1	1	d

- 功能
  - 使用过程语句描述 —— 可以综合

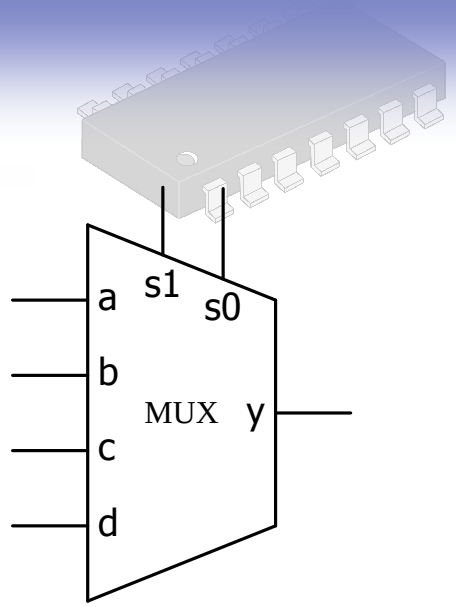
```
module mux4to1( output reg y,
                input s1, s0, d, c, b, a);
   always @(*)
      if ( s1 )
         if ( s0 ) y = d;
         else y = c;
      else
         if ( s0 ) y = b;
         else y = a;
endmodule
```



4 to 1 多路选择器符号

4 选 1 多路选择器（3）

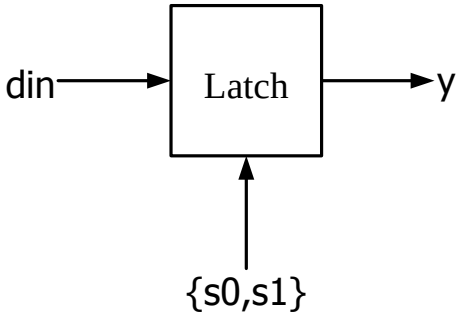
s1	s0	y
0	0	a
0	1	b
1	0	c
1	1	d

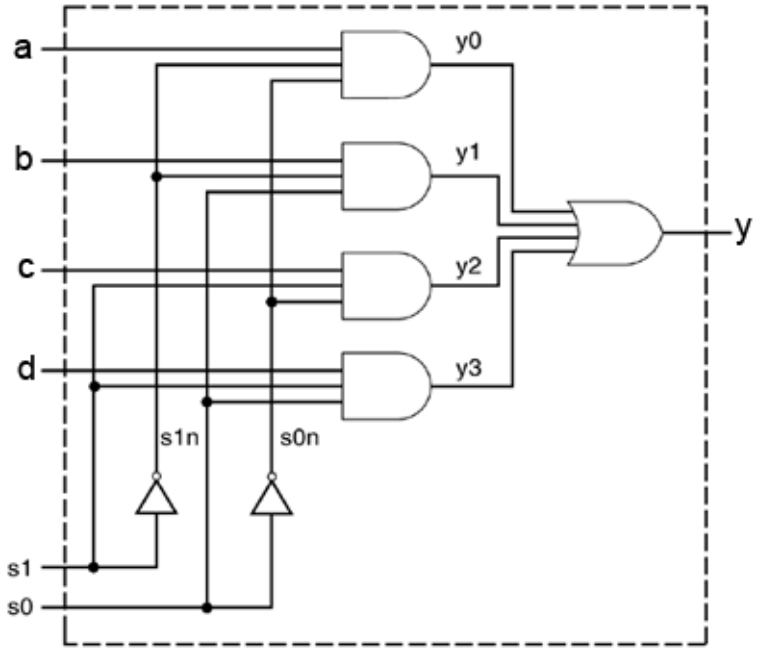


4 to 1 多路选择器符号

- 功能
  - 使用过程语句描述

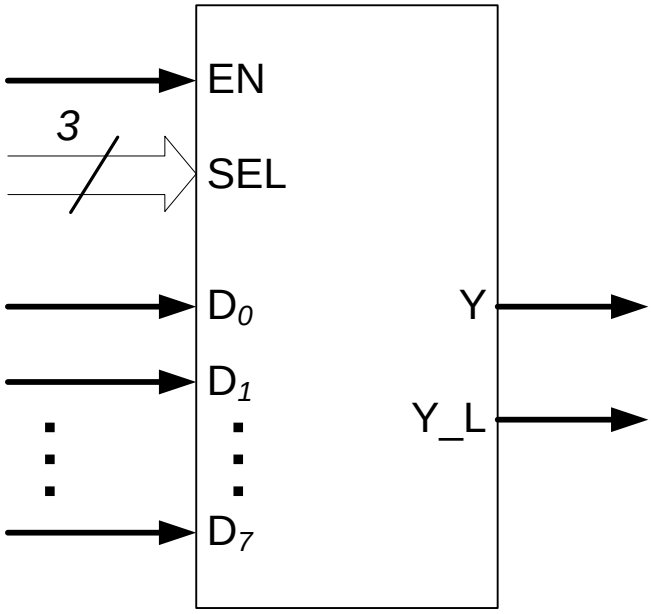
```
module mux4to1( output reg y,
                input s1, s0, d, c, b, a);
   always @(*)
      case ( {s1,s0} )
         2'b00:   y = a;
         2'b01:   y = b;
         2'b10:   y = c;
         2'b11:   y = d;
         default:y = 1'bx;
      endcase
endmodule
```





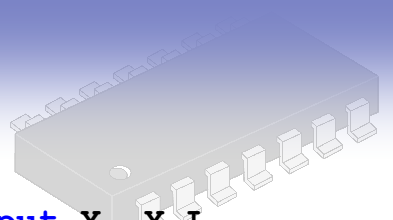
8 选 1 位多路复用器

输入				输出	
EN_L	A	B	C	Y	Y_L
1	X	X	X	0	1
0	0	0	0	D0	$\overline{D0}$
0	0	0	1	D1	$\overline{D1}$
0	0	1	0	D2	$\overline{D2}$
0	0	1	1	D3	$\overline{D3}$
0	1	0	0	D4	$\overline{D4}$
0	1	0	1	D5	$\overline{D5}$
0	1	1	0	D6	$\overline{D6}$
0	1	1	1	D7	$\overline{D7}$




```
module mux8to1( output Y, Y_L,
                input [7:0] D,
                input EN_L, A, B, C );
   reg f;
   always @(*) begin
      if (EN_L) f = 1'b0;
      else
         case({A,B,C})
            3'b000:   f = D[0];
            3'b001:   f = D[1];
            3'b010:   f = D[2];
            3'b011:   f = D[3];
            3'b100:   f = D[4];
            3'b101:   f = D[5];
            3'b110:   f = D[6];
            3'b111:   f = D[7];
            default: f = 1'b0;
         endcase
   end
   assign Y   =  f,
          Y_L = ~f;
endmodule
```

8 选 1 位多路复用器测试平台

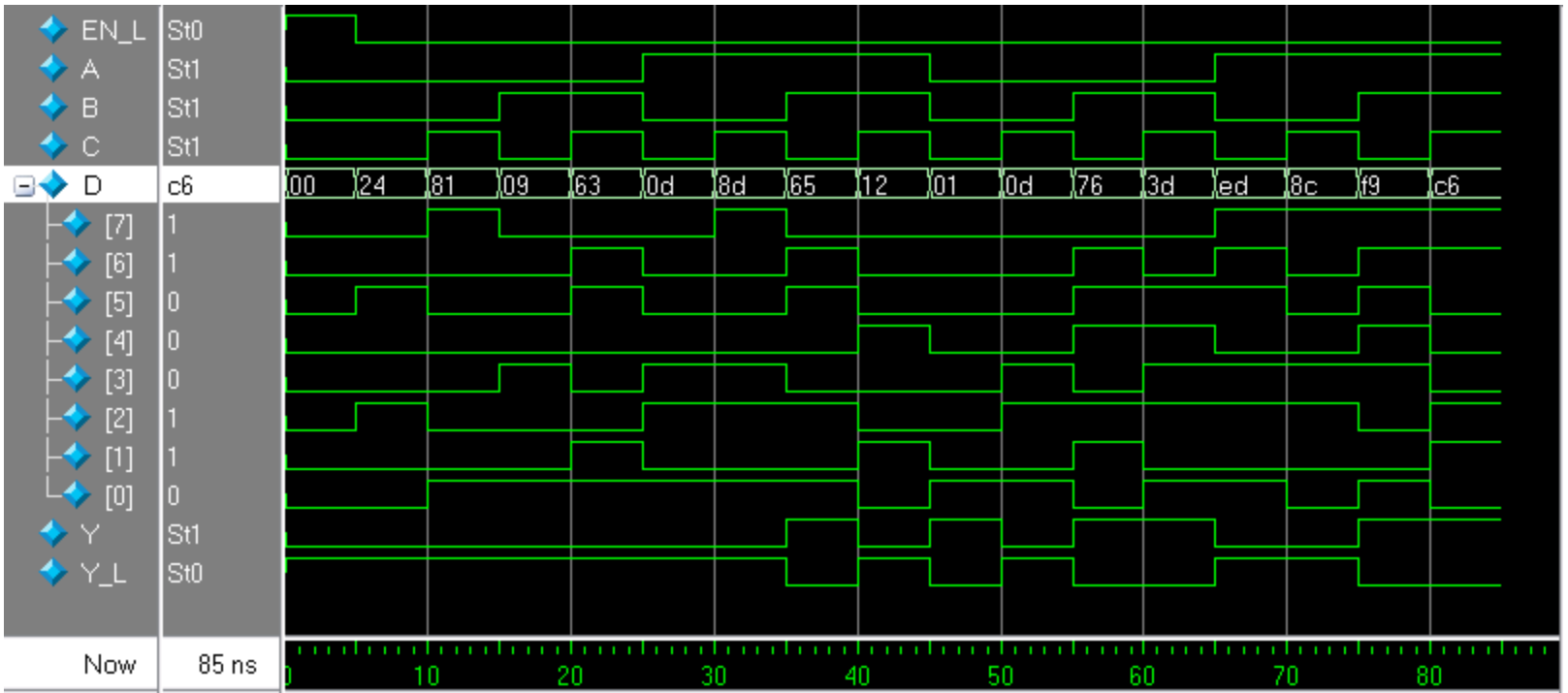


```
`timescale 1ns / 1ns
`include "mux8to1.v"
module mux8to1_tb();
    reg p_EN_L, p_A, p_B, p_C;
    reg [7: 0] p_D;
    wire p_Y, p_Y_L;

    mux8to1 u0(.Y(p_Y), .Y_L(p_Y_L),
               .D(p_D), .EN_L(p_EN_L), .A(p_A), .B(p_B), .C(p_C));

    integer k;
    initial begin
        p_EN_L = 1'b1;
        {p_A, p_B, p_C} = 3'b000;
        p_D = 8'b0;

        #5 p_EN_L = 1'b0;

        for ( k = 0; k < 16; k = k + 1)
        begin
            {p_A, p_B, p_C} = k % 8;
            p_D = { $random } % 256;
            #5;
        end
    end
    initial
        $monitor( "At time %4t, EN_L=%b, ABC=%b, D=%b, Y=%b, Y_L=%b",
                  $time, p_EN_L, {p_A, p_B, p_C}, p_D, p_Y, p_Y_L);
endmodule
```

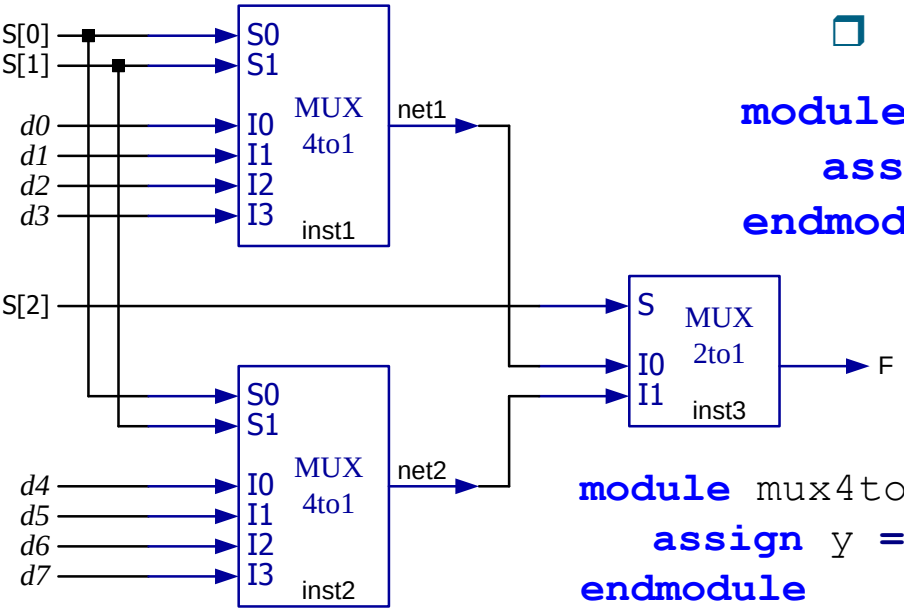
```
module mux8to1( output Y, Y_L,
                input [7:0] D,
                input EN_L, A, B, C );
    reg f;
    always @(*) begin
        if (EN_L) f = 1'b0;
        else
            case({A,B,C})
                3'b000:   f = D[0];
                3'b001:   f = D[1];
                3'b010:   f = D[2];
                3'b011:   f = D[3];
                3'b100:   f = D[4];
                3'b101:   f = D[5];
                3'b110:   f = D[6];
                3'b111:   f = D[7];
                default: f = 1'b0;
            endcase
    end
    assign Y   =  f,
           Y_L = ~f;
endmodule
```

8 to 1 MUX 仿真

```
# At time   0, EN_L=1, ABC=000, D=00000000, Y=0, Y_L=1
# At time   5, EN_L=0, ABC=000, D=00100100, Y=0, Y_L=1
# At time  10, EN_L=0, ABC=001, D=10000001, Y=0, Y_L=1
# At time  15, EN_L=0, ABC=010, D=00001001, Y=0, Y_L=1
# At time  20, EN_L=0, ABC=011, D=01100011, Y=0, Y_L=1
# At time  25, EN_L=0, ABC=100, D=00001101, Y=0, Y_L=1
# At time  30, EN_L=0, ABC=101, D=10001101, Y=0, Y_L=1
# At time  35, EN_L=0, ABC=110, D=01100101, Y=1, Y_L=0
# At time  40, EN_L=0, ABC=111, D=00010010, Y=0, Y_L=1
# At time  45, EN_L=0, ABC=000, D=00000001, Y=1, Y_L=0
# At time  50, EN_L=0, ABC=001, D=00001101, Y=0, Y_L=1
# At time  55, EN_L=0, ABC=010, D=01110110, Y=1, Y_L=0
# At time  60, EN_L=0, ABC=011, D=00111101, Y=1, Y_L=0
# At time  65, EN_L=0, ABC=100, D=11101101, Y=0, Y_L=1
# At time  70, EN_L=0, ABC=101, D=10001100, Y=0, Y_L=1
# At time  75, EN_L=0, ABC=110, D=11111001, Y=1, Y_L=0
# At time  80, EN_L=0, ABC=111, D=11000110, Y=1, Y_L=0
```



八选一多路复用器 —— 利用模块实例引用



- 由 2个4x1多路器和1个2x1多路器

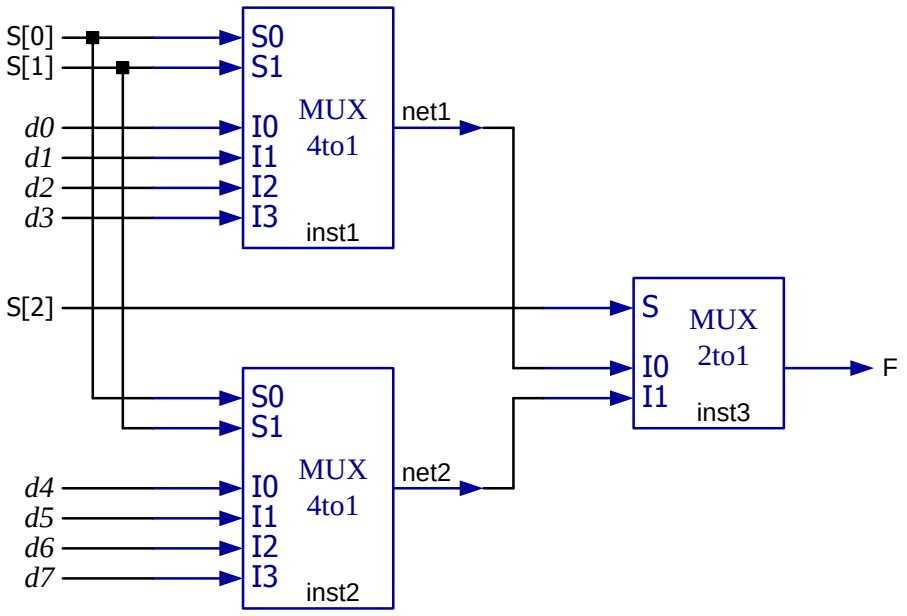
```
module mux2to1(output y, input s, b, a);
   assign y = ( s ) ? b : a;
endmodule
```

```
module mux4to1(output y, input s1, s0, d, c, b, a );
   assign y = (s1) ? ((s0) ? d : c) : ((s0) ? b : a);
endmodule
```

```
`include "mux4x1.v"
`include "mux2x1.v"
module mux8x1(output F, input [2:0] s, input [7:0] d );
   wire net1, net2;

   // 引用模块实例
   mux4x1 inst1( net1, s[1], s[0], d[3], d[2], d[1], d[0] );
   mux4x1 inst2( net2, s[1], s[0], d[7], d[6], d[5], d[4] );
   mux2x1 inst3( F, s[2], net2, net1 );
endmodule
```

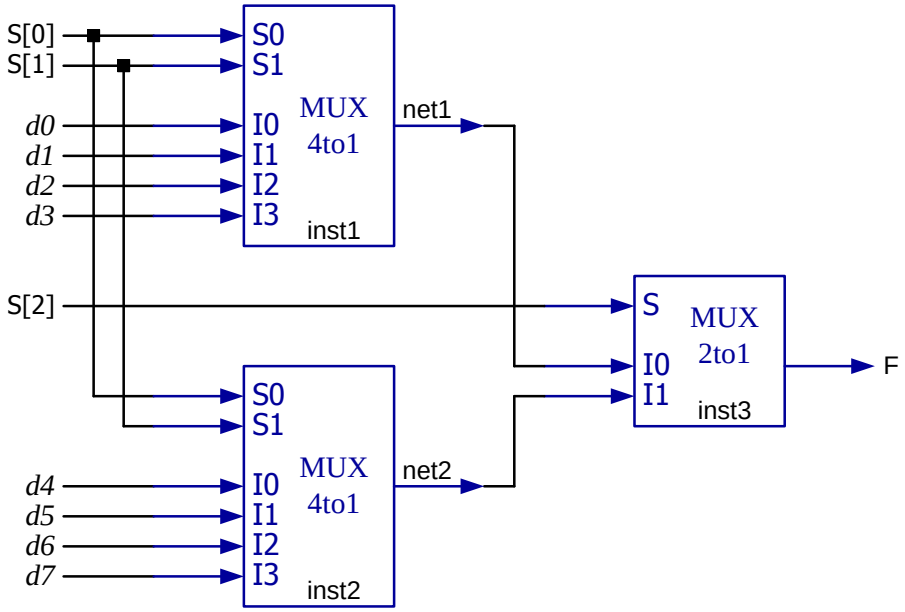
八选一多路复用器 —— 使用函数（1）

```
module mux8to1(output reg F, input [2:0] s, input [7:0] d );
    reg w1, w2;

    // mux 2 to 1
    function y_2x1( input s, b, a);
        y_2x1 = ( s ) ? b : a;
    endfunction

    // mux 4 to 1
    function automatic y_4x1( input [1:0] sel, input [3:0] data);
        case ( sel )
            2'b00: y_4x1 = data[0];
            2'b01: y_4x1 = data[1];
            2'b10: y_4x1 = data[2];
            2'b11: y_4x1 = data[3];
            default:y_4x1 = 1'bx;
        endcase
    endfunction

    // 调用函数
    always @(*) begin
        w1 = y_4x1(s[1:0], d[3:0]);
        w2 = y_4x1(s[1:0], d[7:4]);
        F  = y_2x1(s[2], w2, w1);
    end
endmodule
```



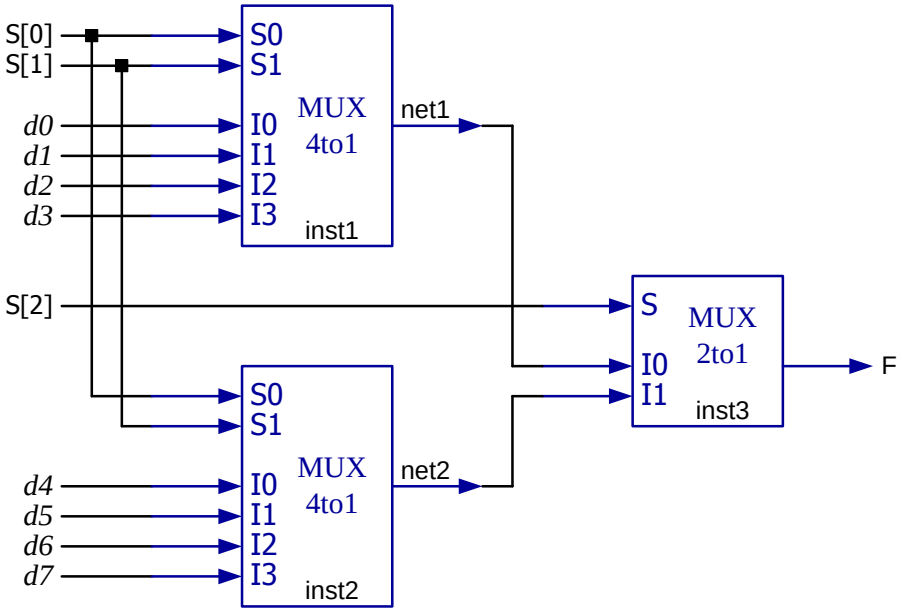
八选一多路复用器 —— 使用函数（2）

```
module mux8to1(output F, input [2:0] s, input [7:0] d );
    // mux 2 to 1
    function y_2x1( input s, b, a);
        y_2x1 = ( s ) ? b : a;
    endfunction

    // mux 4 to 1
    function automatic y_4x1( input [1:0] sel, input [3:0] data);
        case ( sel )
            2'b00:  y_4x1 = data[0];
            2'b01:  y_4x1 = data[1];
            2'b10:  y_4x1 = data[2];
            2'b11:  y_4x1 = data[3];
            default:y_4x1 = 1'bx;
        endcase
    endfunction

    wire w1, w2;

    // 调用函数
    assign w1 = y_4x1(s[1:0], d[3:0]),
           w2 = y_4x1(s[1:0], d[7:4]),
           F  = y_2x1(s[2], w2, w1);
endmodule
```



八选一多路复用器 —— 使用任务

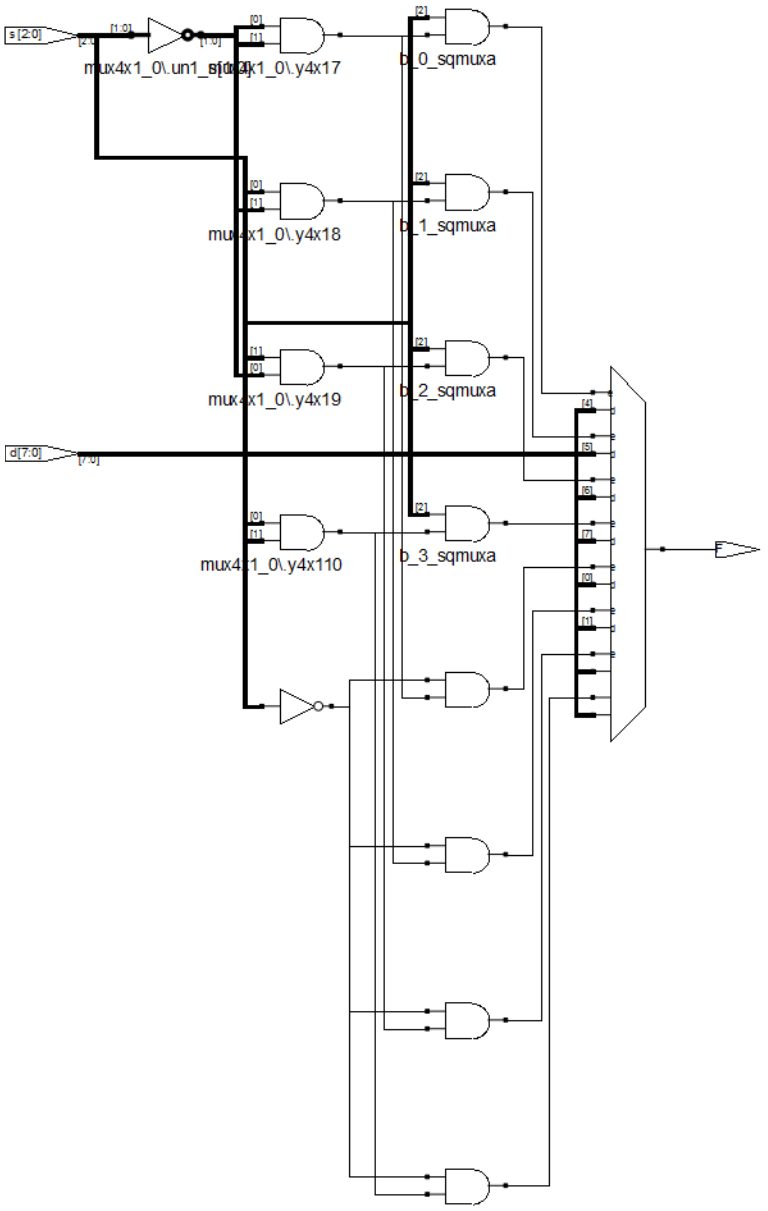
```
module mux8to1(output reg F, input [2:0] s, input [7:0] d );
    // mux 2 to 1
    task mux2x1( output y2x1, input s, b, a);
        y2x1 = ( s ) ? b : a;
    endtask

    // mux 4 to 1
    task automatic mux4x1( output y4x1, input [1:0] sel, input [3:0] din);
        case ( sel )
            2'b00:  y4x1 = din[0];
            2'b01:  y4x1 = din[1];
            2'b10:  y4x1 = din[2];
            2'b11:  y4x1 = din[3];
            default:y4x1 = 1'bx;
        endcase
    endtask

    reg w1, w2;

    // 调用任务
    always @* begin
        mux4x1( w1, s[1:0], d[3:0]);
        mux4x1( w2, s[1:0], d[7:4]);
        mux2x1( F, s[2], w2, w1);
    end
endmodule
```



八选一多路复用器

- 使用函数设计的综合结果
  - 工具：Synplify Premier E-2011.03-SP2

```
module mux8to1(output reg F, input [2:0] s, input [7:0] d );
    // mux 2 to 1
    task mux2x1( output y2x1, input s, b, a);
        y2x1 = ( s ) ? b : a;
    endtask

    // mux 4 to 1
    task automatic mux4x1( output y4x1,
                           input [1:0] sel, input [3:0] din);
        case ( sel )
            2'b00:  y4x1 = din[0];
            2'b01:  y4x1 = din[1];
            2'b10:  y4x1 = din[2];
            2'b11:  y4x1 = din[3];
            default:y4x1 = 1'bx;
        endcase
    endtask

    reg w1, w2;

    // 调用任务
    always @* begin
        mux4x1( w1, s[1:0], d[3:0]);
        mux4x1( w2, s[1:0], d[7:4]);
        mux2x1( F, s[2], w2, w1);
    end
endmodule
```



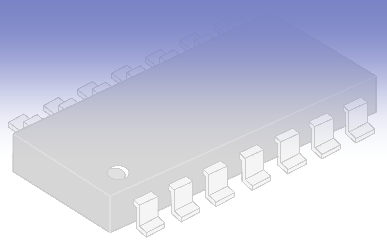
2-4 译码器

- 2到4译码器的真值表

输入		输出			
a1	a0	y3	y2	y1	y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

```
module decoder2x4(output reg [3:0] y,
                  input [1:0] a );
   always @(*)
      case (a)
         0:    y = 1;
         1:    y = 2;
         2:    y = 4;
         3:    y = 8;
         default:y = 4'bx;
      endcase
endmodule
```

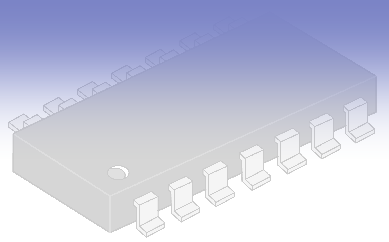
2-4 译码器——使用函数设计



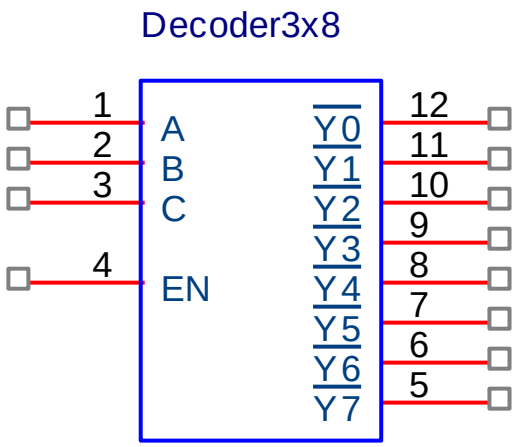
```
module decoder2x4(output [3:0] y,
                  input [1:0] a );

    function [3:0] d2x4(input [1:0] c);
        case (a)
            0:       d2x4 = 1;
            1:       d2x4 = 2;
            2:       d2x4 = 4;
            3:       d2x4 = 8;
            default:d2x4 = 4'bx;
        endcase
    endfunction

    assign y = d2x4(a);
endmodule
```

输入		输出			
a1	a0	y3	y2	y1	y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

3-8 译码器

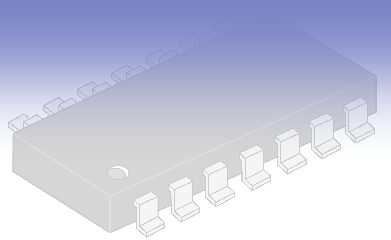


- 三个输入变量译成八个输出之一
  - 使能信号：EN，只有使能有效，译码器才工作
  - 输入：（C, B, A）
  - 输出：（ Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7 ）



EN	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	X	X	X	1	1	1	1	1	1	1	1
X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	0	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1

3-8 译码器设计方法1



```
module decoder3x8(output reg [7:0] YL, input EN, C, B, A );
   always @ (*)
   begin
      if ( EN )
         case ( { C, B, A } )
            0: YL = 8'b1111_1110;
            1: YL = 8'b1111_1101;
            2: YL = 8'b1111_1011;
            3: YL = 8'b1111_0111;
            4: YL = 8'b1110_1111;
            5: YL = 8'b1101_1111;
            6: YL = 8'b1011_1111;
            7: YL = 8'b0111_1111;
            default: YL = 8'HFF;
         endcase
      else
         YL = 8'b1111_1111;
   end
endmodule
```

- 使用 case 语句模型
  - 最常用的方法
  - 根据真值表
  - 可综合

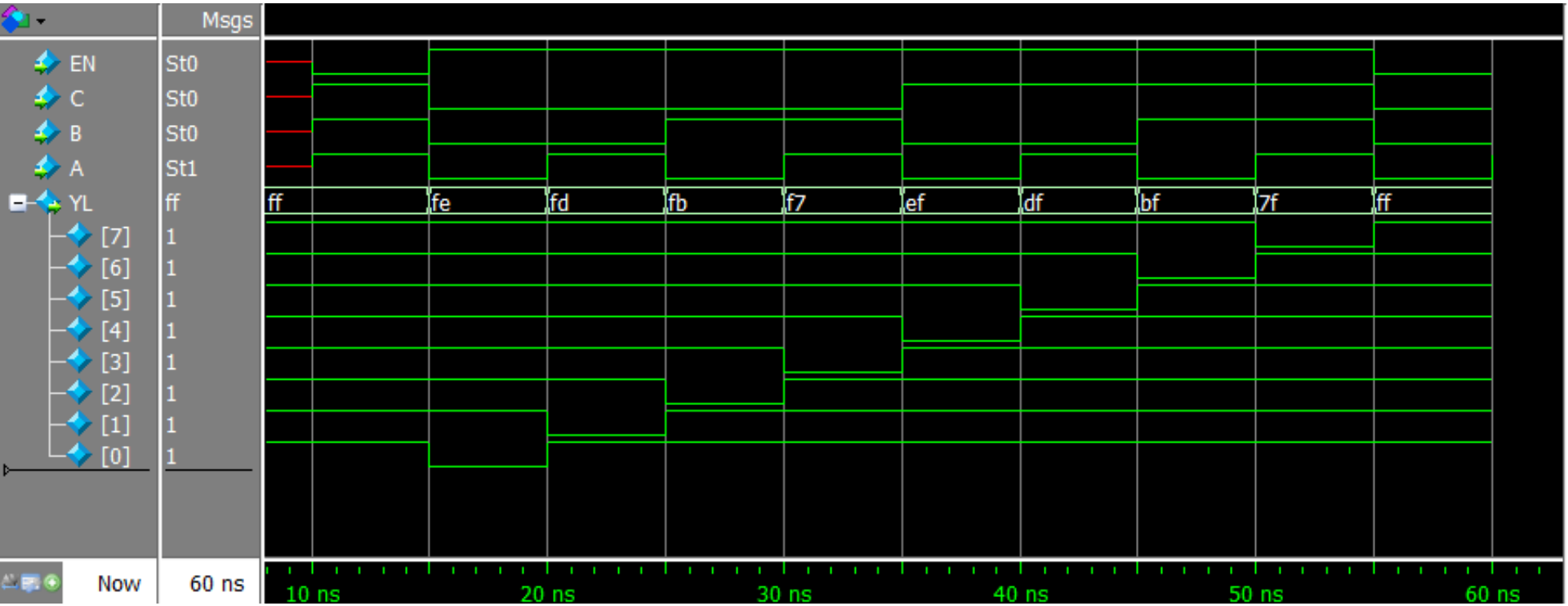
EN	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	X	X	X	1	1	1	1	1	1	1	1
X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	0	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1

方法一、测试平台

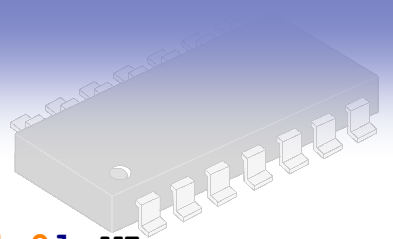
```
`timescale 1ns / 1ns
`include "decoder3x8_m1.v"
module decoder3x8_tb;
    parameter S0 = 7, S1 = 16;
    wire [ 7 : 0 ] data;
    reg p_en, p_c, p_b, p_a;
    decoder3x8 u0(.YL(data), .EN(p_en), .C(p_c), .B(p_b), .A(p_a));

    reg [3:0] en_code;
    initial begin
        { p_en,p_c,p_b,p_a } = 4'bx000;
        # 5 { p_en,p_c,p_b,p_a } = 4'bx;
        # 5;
        for ( en_code = S0; en_code < S1; en_code = en_code + 1)
        begin
            { p_en,p_c,p_b,p_a } = en_code;
            #5;
        end
    end
    initial
        $monitor( "At time %4t,  EN = %b, CBA = %b,  data = %b",
                  $time, p_en, {p_c,p_b,p_a}, data );
endmodule
```

3-8 译码器的仿真结果

```
# At time     0,   EN = x, CBA = 000,   data = 11111111
# At time     5,   EN = x, CBA = xxx,   data = 11111111
# At time    10,   EN = 0, CBA = 111,   data = 11111111
# At time    15,   EN = 1, CBA = 000,   data = 11111110
# At time    20,   EN = 1, CBA = 001,   data = 11111101
# At time    25,   EN = 1, CBA = 010,   data = 11111011
# At time    30,   EN = 1, CBA = 011,   data = 11110111
# At time    35,   EN = 1, CBA = 100,   data = 11101111
# At time    40,   EN = 1, CBA = 101,   data = 11011111
# At time    45,   EN = 1, CBA = 110,   data = 10111111
# At time    50,   EN = 1, CBA = 111,   data = 01111111
# At time    55,   EN = 0, CBA = 000,   data = 11111111
```



3-8 译码器设计方法2



- 使用 if 语句模型
  - 根据真值表
  - 也很常用
  - 可综合

EN	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	X	X	X	1	1	1	1	1	1	1	1
X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	0	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1

```
module decoder3x8m2( output reg [7:0] YL,
                     input EN,
                     input [2:0] DIN );
  always @ ( * )
    if ( EN ) begin
      if ( DIN == 0 ) YL = 8'b1111_1110; else
      if ( DIN == 1 )  YL = 8'b1111_1101; else
      if ( DIN == 2 )  YL = 8'b1111_1011; else
      if ( DIN == 3 )  YL = 8'b1111_0111; else
      if ( DIN == 4 )  YL = 8'b1110_1111; else
      if ( DIN == 5 )  YL = 8'b1101_1111; else
      if ( DIN == 6 )  YL = 8'b1011_1111; else
      if ( DIN == 7 )  YL = 8'b0111_1111; else
      YL = 8'hff;
    end
    else YL = 8'b1111_1111;
endmodule
```

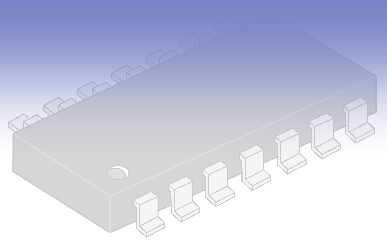
3-8 译码器设计方法3（1）

- 数据流描述
  - 由真值表得到布尔方程
  - 采用联系赋值语句

EN	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	X	X	X	1	1	1	1	1	1	1	1
X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	0	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1

$$
\begin{cases}
Y_0 = \overline{Y}_{0L} = E \bullet (\overline{C} \bullet \overline{B} \bullet \overline{A}) \\
Y_1 = \overline{Y}_{1L} = E \bullet (\overline{C} \bullet \overline{B} \bullet A) \\
Y_2 = \overline{Y}_{2L} = E \bullet (\overline{C} \bullet B \bullet \overline{A}) \\
Y_3 = \overline{Y}_{3L} = E \bullet (\overline{C} \bullet B \bullet A) \\
Y_4 = \overline{Y}_{4L} = E \bullet (C \bullet \overline{B} \bullet \overline{A}) \\
Y_5 = \overline{Y}_{5L} = E \bullet (C \bullet \overline{B} \bullet A) \\
Y_6 = \overline{Y}_{6L} = E \bullet (C \bullet B \bullet \overline{A}) \\
Y_7 = \overline{Y}_{7L} = E \bullet (C \bullet B \bullet A)
\end{cases}
$$

3-8 译码器设计方法3（2）



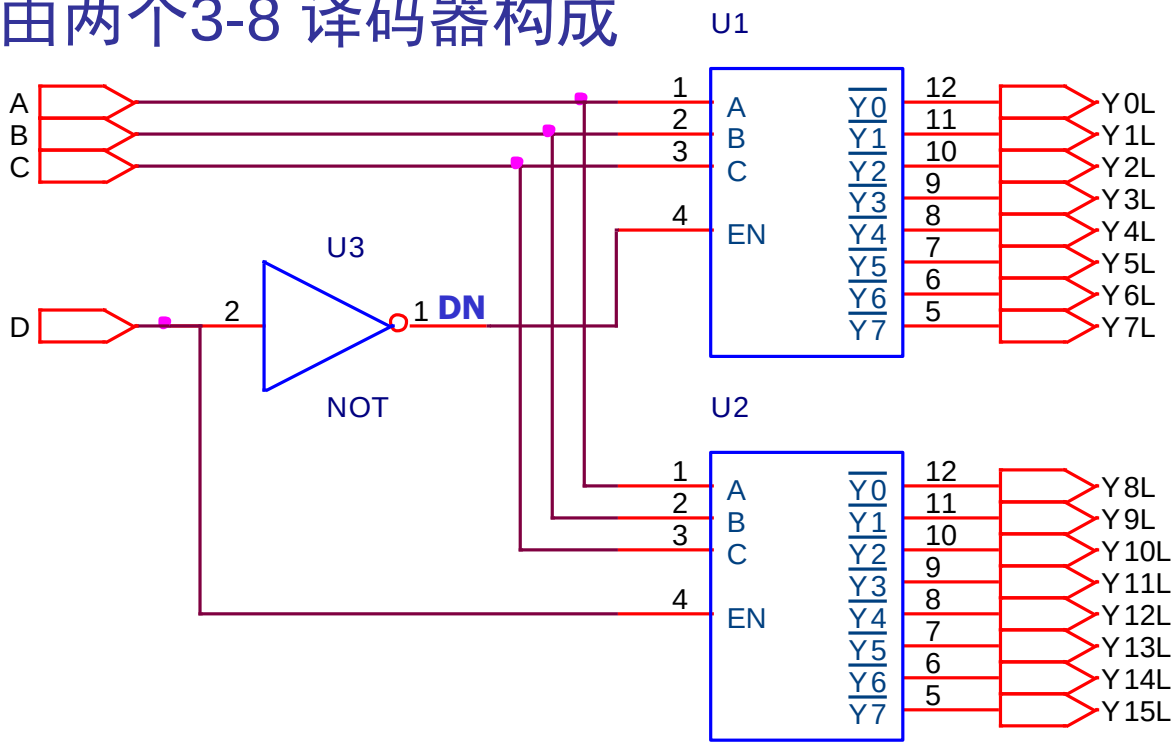
- 使用连续赋值语句 ——数据流描述

$$
\begin{cases}
Y_0 = \overline{Y}_{0L} = E \bullet (\overline{C} \bullet \overline{B} \bullet \overline{A}) \\
Y_1 = \overline{Y}_{1L} = E \bullet (\overline{C} \bullet \overline{B} \bullet A) \\
Y_2 = \overline{Y}_{2L} = E \bullet (\overline{C} \bullet B \bullet \overline{A}) \\
Y_3 = \overline{Y}_{3L} = E \bullet (\overline{C} \bullet B \bullet A) \\
Y_4 = \overline{Y}_{4L} = E \bullet (C \bullet \overline{B} \bullet \overline{A}) \\
Y_5 = \overline{Y}_{5L} = E \bullet (C \bullet \overline{B} \bullet A) \\
Y_6 = \overline{Y}_{6L} = E \bullet (C \bullet B \bullet \overline{A}) \\
Y_7 = \overline{Y}_{7L} = E \bullet (C \bullet B \bullet A)
\end{cases}
$$

```
`timescale 1ns / 1ns
module decoder3x8m3( output [7:0] YL,
                     input EN, C, B, A );
        wire [7 : 0 ] YH;

        assign YH[7] = EN&(  C &  B &  A),
               YH[6] = EN&(  C &  B & ~A),
               YH[5] = EN&(  C & ~B &  A),
               YH[4] = EN&(  C & ~B & ~A),
               YH[3] = EN&( ~C &  B &  A),
               YH[2] = EN&( ~C &  B & ~A),
               YH[1] = EN&( ~C & ~B &  A),
               YH[0] = EN&( ~C & ~B & ~A);

        assign YL = ~YH;
endmodule
```

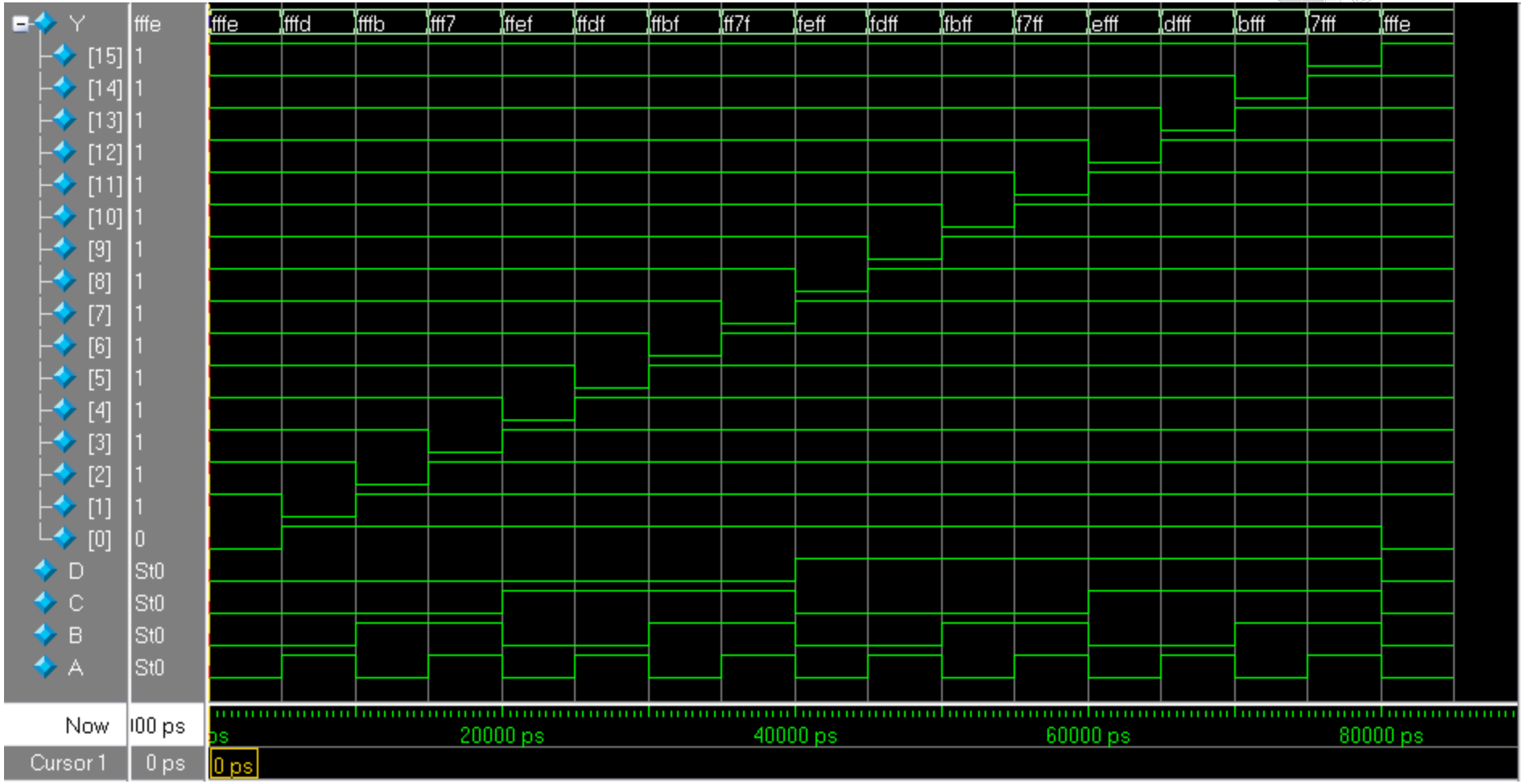
设计4-16译码器

- 由两个3-8 译码器构成

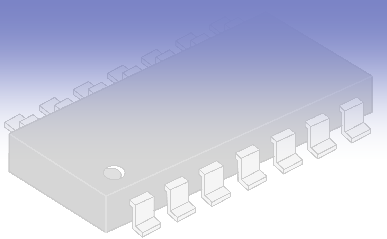



```
`include "decoder3x8.v"
module decoder4x16( output [15:0] y, input d, c, b, a );
    wire dbar;
    assign dbar = ~d;
    decoder3x8 u1( .YL( y[7:0]), .EN(dbar), .C(c), .B(b), .A(a) ),
               u2( .YL(y[15:8]), .EN(   d), .C(c), .B(b), .A(a) );
endmodule
```

4-16译码器结构建模的仿真结果

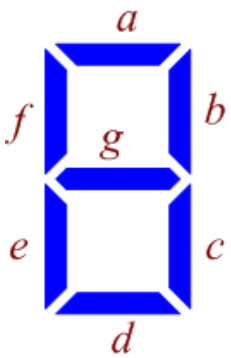


七段译码器



- 功能
  - 将 4 位 BCD 码译成 10 个输出之一
  - 输入变量
    - 输入数值： 4位 BCD 码（D, C, B, A）
    - 消隐输入：BI_L，用于关闭显示输出
      - 如，不显示一个数字的前端和尾部的 0
  - 输出变量：七个线段（ a, b, c, d, e, f, g ）组合的一个子集，共16种

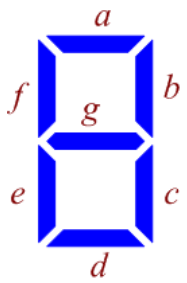
0123456789



输入					显示数字	输出						
BI_L	D	C	B	A		a	b	c	d	e	f	g
0	X	X	X	X	关闭	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
1	0	0	1	0	2	1	1	0	1	1	0	1
1	0	0	1	1	3	1	1	1	1	0	0	1
1	0	1	0	0	4	0	1	1	0	0	1	1
1	0	1	0	1	5	1	0	1	1	0	1	1
1	0	1	1	0	6	1	0	1	1	1	1	1
1	0	1	1	1	7	1	1	1	0	0	0	0
1	1	0	0	0	8	1	1	1	1	1	1	1
1	1	0	0	1	9	1	1	1	1	0	1	1

七段译码器设计

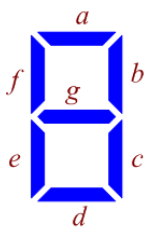
- 典型设计 —— 使用 case 语句模型



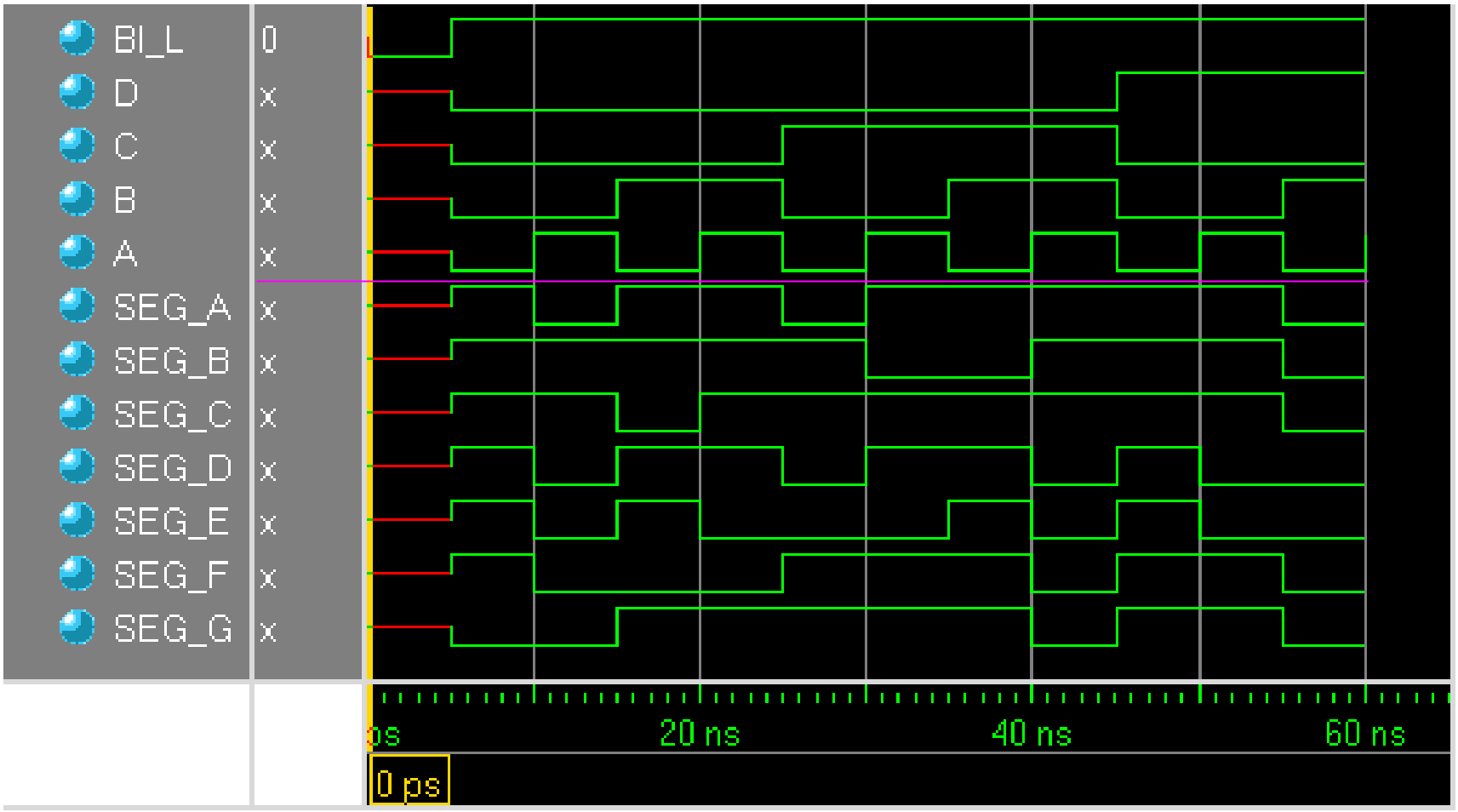
输入					显示数字	输出						
BI_L	D	C	B	A		a	b	c	d	e	f	g
0	X	X	X	X	关闭	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
1	0	0	1	0	2	1	1	0	1	1	0	1
1	0	0	1	1	3	1	1	1	1	0	0	1
1	0	1	0	0	4	0	1	1	0	0	1	1
1	0	1	0	1	5	1	0	1	1	0	1	1
1	0	1	1	0	6	1	0	1	1	1	1	1
1	0	1	1	1	7	1	1	1	0	0	0	0
1	1	0	0	0	8	1	1	1	1	1	1	1
1	1	0	0	1	9	1	1	1	1	0	1	1

```
module dec7seg( output reg [0:6] seg,
                input [3:0] code, input BI_L );
   always @( * ) begin
      if ( BI_L )
         case(code)
            0:       seg = 7'b111_1110;
            1:       seg = 7'b011_0000;
            2:       seg = 7'b110_1101;
            3:       seg = 7'b111_1001;
            4:       seg = 7'b011_0011;
            5:       seg = 7'b101_1011;
            6:       seg = 7'b101_1111;
            7:       seg = 7'b111_0000;
            8:       seg = 7'b111_1111;
            9:       seg = 7'b111_0011;
           default: seg = 7'b000_0000;
         endcase
      else seg = 7'b000_0000;
   end
endmodule
```

七段译码器仿真结果

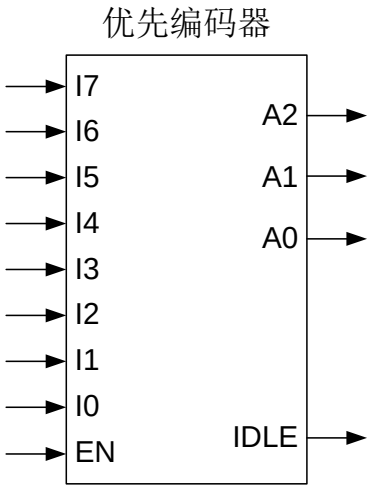


0 1 2 3 4 5 6 7 8 9



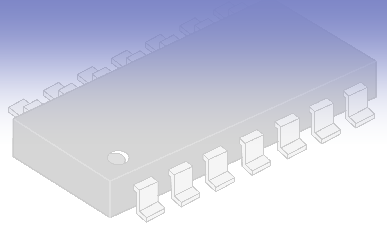
优先编码器（Priority encoder）

- 保证在某一时刻只有一个输入有效
  - 对输入线指定优先级（priority）—— 如：硬件中断
  - 当同时出现多个请求时，编码器产生最高优先级的请求编号
- 例、通用 8 输入优先编码器
  - 输入端口 9 个：8 个数据输入，1 个使能输入
  - 输出端口 4 个：3 个编码输出，1 个有效标志（IDLE）



优先级	输入								输出			
	EN	I7	I6	I5	I4	I3	I2	I1	I0	A2	A1	A0	IDLE
无	0	X	X	X	X	X	X	X	X	X	X	X	1
0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	X	0	0	1	0
2	1	0	0	0	0	0	1	X	X	0	1	0	0
3	1	0	0	0	0	1	X	X	X	0	1	1	0
4	1	0	0	0	1	X	X	X	X	1	0	0	0
5	1	0	0	1	X	X	X	X	X	1	0	1	0
6	1	0	1	X	X	X	X	X	X	1	1	0	0
7	1	1	X	X	X	X	X	X	X	1	1	1	0

8 输入优先编码器的设计



```
module encoder8x3( output reg idle, output reg [2:0] code,
                   input [7:0] data, input en );
    always @(*) begin
        if ( !en ) // en == 1'b0
        begin
            code = 3'bx;
            idle = 1'b1;
        end
        else begin
            if ( data[7]   == 1'b1) code = 7; else
            if ( data[7:6] == 2'b01) code = 6; else
            if ( data[7:5] == 3'b001) code = 5; else
            if ( data[7:4] == 4'b0001) code = 4; else
            if ( data[7:3] == 5'b0000_1) code = 3; else
            if ( data[7:2] == 6'b0000_01) code = 2; else
            if ( data[7:1] == 7'b0000_001) code = 1; else
            if ( data[7:0] == 8'b0000_0001) code = 0; else code = 3'bx;

            idle = 1'b0;
        end
    end
```

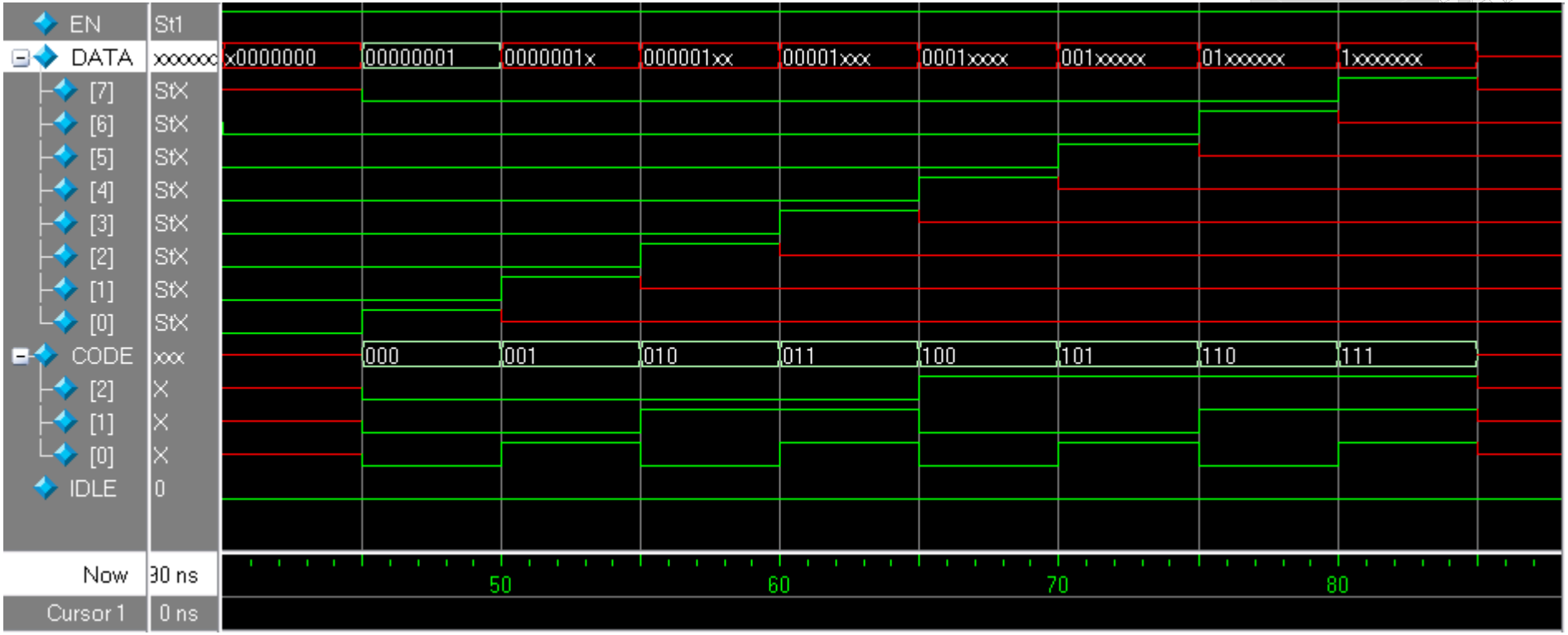
优先级	输入									输出			
	EN	I7	I6	I5	I4	I3	I2	I1	I0	A2	A1	A0	IDLE
无	0	X	X	X	X	X	X	X	X	X	X	X	1
0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	X	0	0	1	0
2	1	0	0	0	0	0	1	X	X	0	1	0	0
3	1	0	0	0	0	1	X	X	X	0	1	1	0
4	1	0	0	0	1	X	X	X	X	1	0	0	0
5	1	0	0	1	X	X	X	X	X	1	0	1	0
6	1	0	1	X	X	X	X	X	X	1	1	0	0
7	1	1	X	X	X	X	X	X	X	1	1	1	0

编码器模块的测试

```
`timescale 1ns / 1ns
`include "encoder8x3.v"
module encoder8x3_tb;
    parameter S0 = 0, S1 = 16;

    wire [2:0] p_code;
    wire p_idle;
    reg [7:0] p_data;
    reg p_en;
    reg [15:0] p_din0, p_din1;

    encoder8x3 u0(.idle(p_idle), .code(p_code), .data(p_data), .en(p_en));

    integer k;
    initial begin
        p_en = 1'b0;
        p_data = 8'b1111_1111;
        p_din0 = 16'bxxxx_xxxx_0000_0001;

        #5 p_en = 1'b1;
        p_data = p_din0[15:8];
        for ( k = S0; k < S1; k=k + 1) begin
            #5;
            p_din1 = {p_din0[14:0], p_din0[15]};
            p_din0 = p_din1;
            p_data = p_din0[15:8];
        end
    end
    initial
        $monitor( "At time %4t, EN= %b, data= %b, code= %b, IDLE=%b",
                  $time, u0.en, u0.data, u0.code, u0.idle);
endmodule
```

优先级	输入								输出			
	EN	I7	I6	I5	I4	I3	I2	I1	I0	A2	A1	A0	IDLE
无	0	X	X	X	X	X	X	X	X	X	X	X	1
0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	X	0	0	1	0
2	1	0	0	0	0	0	1	X	X	0	1	0	0
3	1	0	0	0	0	1	X	X	X	0	1	1	0
4	1	0	0	0	1	X	X	X	X	1	0	0	0
5	1	0	0	1	X	X	X	X	X	1	0	1	0
6	1	0	1	X	X	X	X	X	X	1	1	0	0
7	1	1	X	X	X	X	X	X	X	1	1	1	0

编码器仿真结果

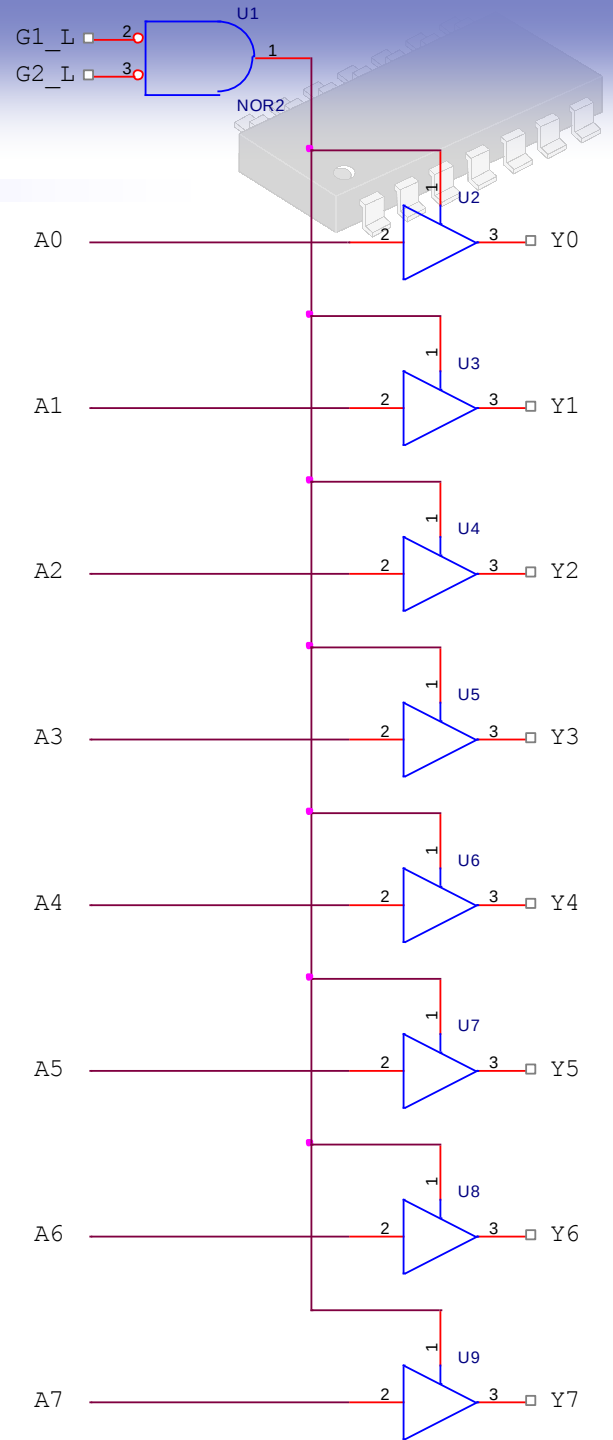



```
# At time   0, EN= 0, data= 11111111, code= xxx, IDLE=1
# At time   5, EN= 1, data= xxxxxxxx, code= xxx, IDLE=0
# At time  10, EN= 1, data= xxxxxxx0, code= xxx, IDLE=0
# At time  15, EN= 1, data= xxxxxx00, code= xxx, IDLE=0
# At time  20, EN= 1, data= xxxxx000, code= xxx, IDLE=0
# At time  25, EN= 1, data= xxxx0000, code= xxx, IDLE=0
# At time  30, EN= 1, data= xxx00000, code= xxx, IDLE=0
# At time  35, EN= 1, data= xx000000, code= xxx, IDLE=0
# At time  40, EN= 1, data= x0000000, code= xxx, IDLE=0
```

```
# At time  45, EN= 1, data= 00000001, code= 000, IDLE=0
# At time  50, EN= 1, data= 0000001x, code= 001, IDLE=0
# At time  55, EN= 1, data= 000001xx, code= 010, IDLE=0
# At time  60, EN= 1, data= 00001xxx, code= 011, IDLE=0
# At time  65, EN= 1, data= 0001xxxx, code= 100, IDLE=0
# At time  70, EN= 1, data= 001xxxxx, code= 101, IDLE=0
# At time  75, EN= 1, data= 01xxxxxx, code= 110, IDLE=0
# At time  80, EN= 1, data= 1xxxxxxx, code= 111, IDLE=0
# At time  85, EN= 1, data= xxxxxxxx, code= xxx, IDLE=0
```

8三态缓冲器（74x541）

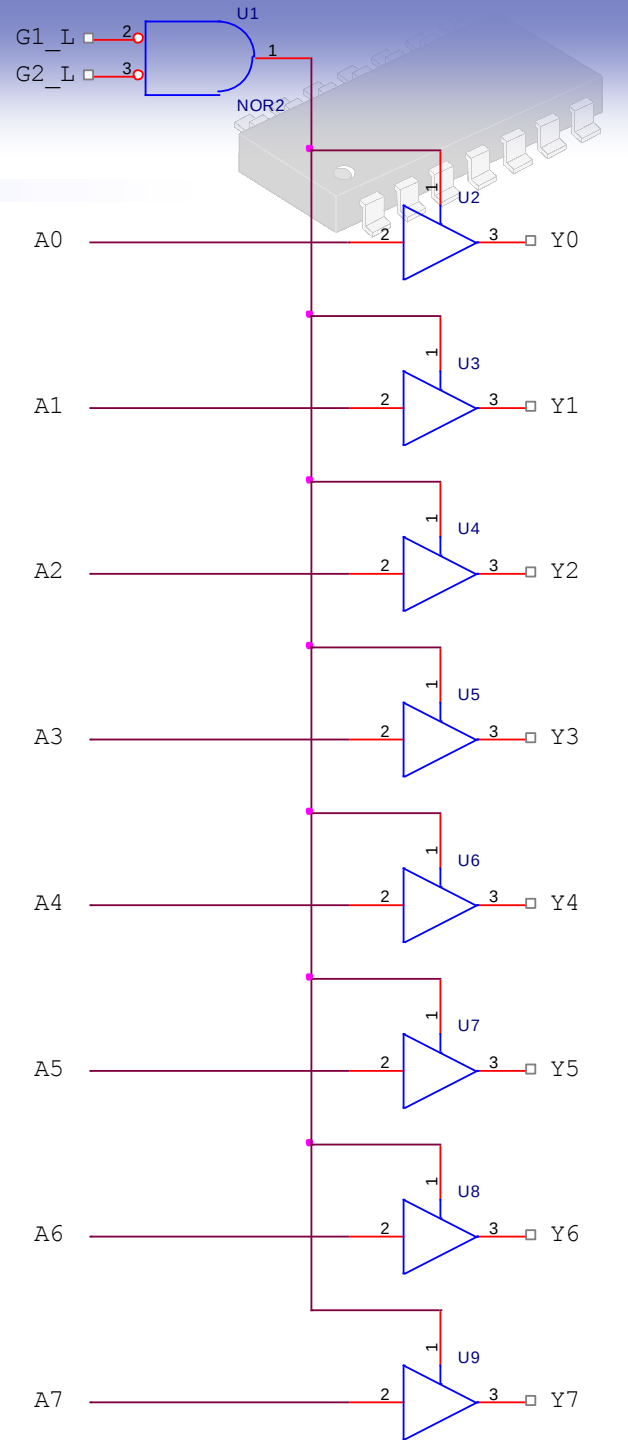
- 三态
  - 变量有三种可能的不同状态：逻辑 0、逻辑 1 和高阻
- 典型使用
  - 用于控制多个数据源到单个数据线的操作
  - 驱动总线
- 如：
  - 8 位微处理器系统，数据总线为 8 位宽
  - 使用 8 个三态缓冲器，外围设备每次在总线上放置 8 位数据



8三态缓冲器的设计

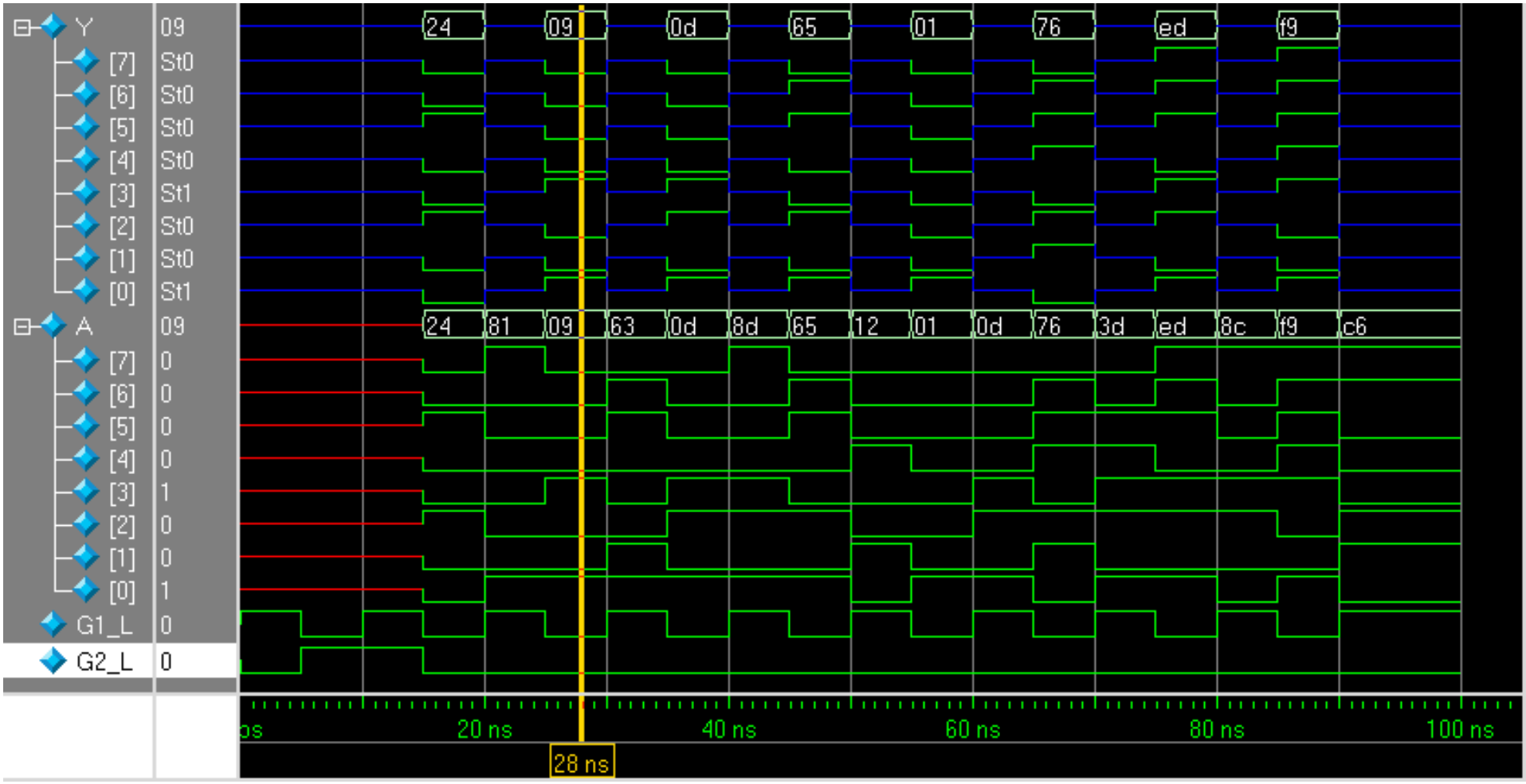
- 模块的设计
  - 结构设计 —— 利用 Verilog 原语

```
module tristate541 (Y, A, G1_L, G2_L);
   parameter SIZE = 8;

   output [SIZE-1:0] Y;
   input  [SIZE-1:0] A;
   input  G1_L, G2_L;

   wire GL;

   nor u0(GL, G1_L, G2_L);

   genvar k;
   generate for ( k=0; k<SIZE; k=k+1 )
   begin : LOOP
        bufif1 g1( Y[k], A[k], GL );
   end
   endgenerate
endmodule
```



8三态缓冲器模块的仿真波形

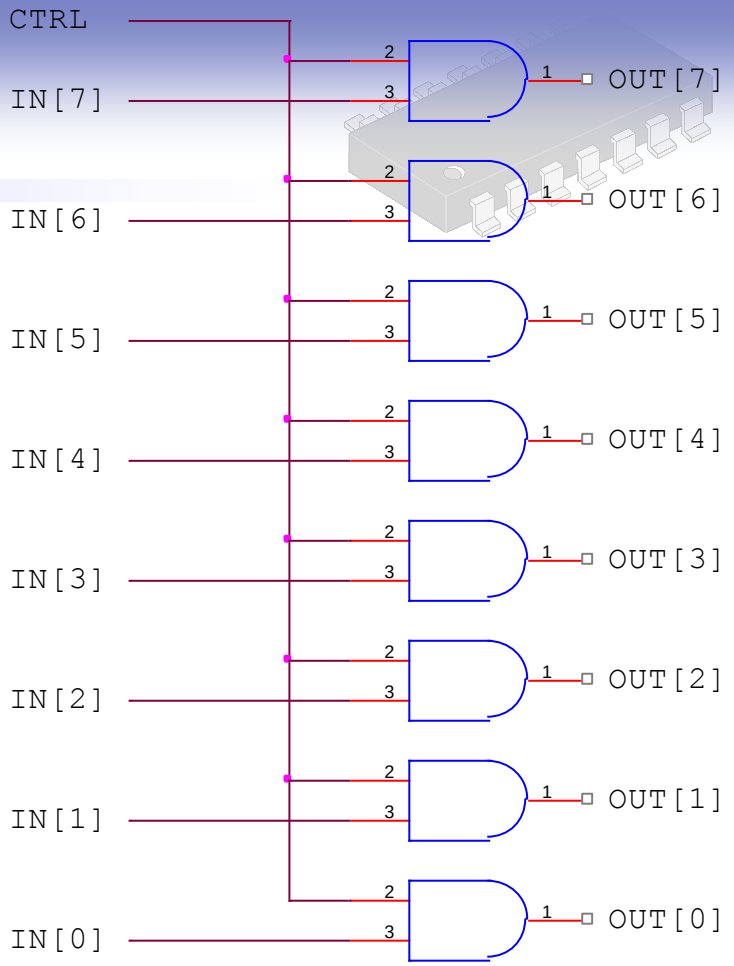
- 仿真分析

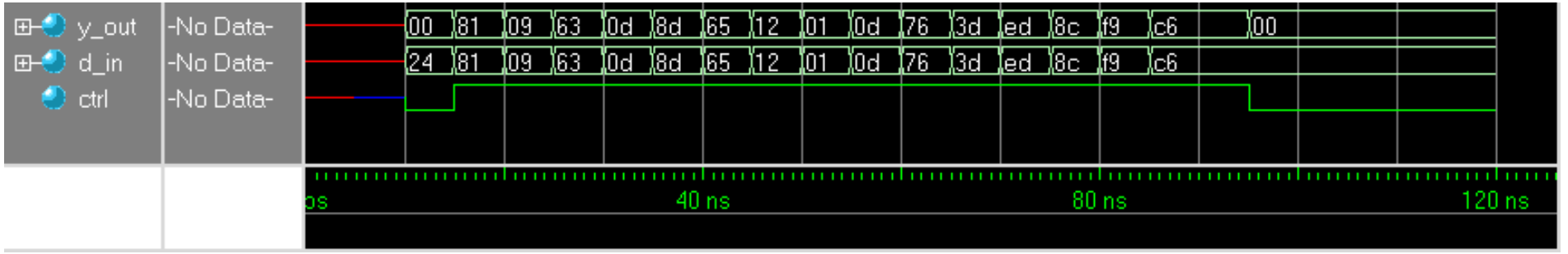


8 位数据通道控制器

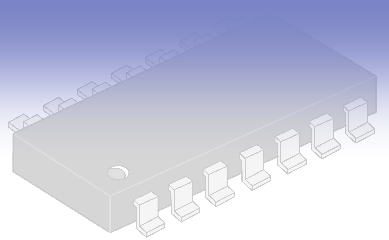
- 使用与门

```
`timescale 1ns / 100ps
module channel_sw_8b( output [7:0] y_out,
                      input[7:0] d_in,
                      input ctrl );
   assign y_out = (ctrl == 1'b1) ? d_in : 8'h00;
Endmodule
```





二进制比较器（Binary Comparator）



- 功能的描述
  - 比较 2 个二进制数，并指示它们之间的关系的电路
  - 两个操作数的比较，3 种情况：
    - A = B
    - A > B
    - A < B
- 1位比较器的的真值表

A	B	A=B	A>B	A<B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

1位比较器的逻辑表达式：

$$A\_EQ\_B = A' \cdot B' + A \cdot B = (A \oplus B)'$$

$$A\_GT\_B = A \cdot B'$$

$$A\_LT\_B = A' \cdot B$$

1位比较器的模块设计

- 数据流描述
  - 使用连续赋值语句

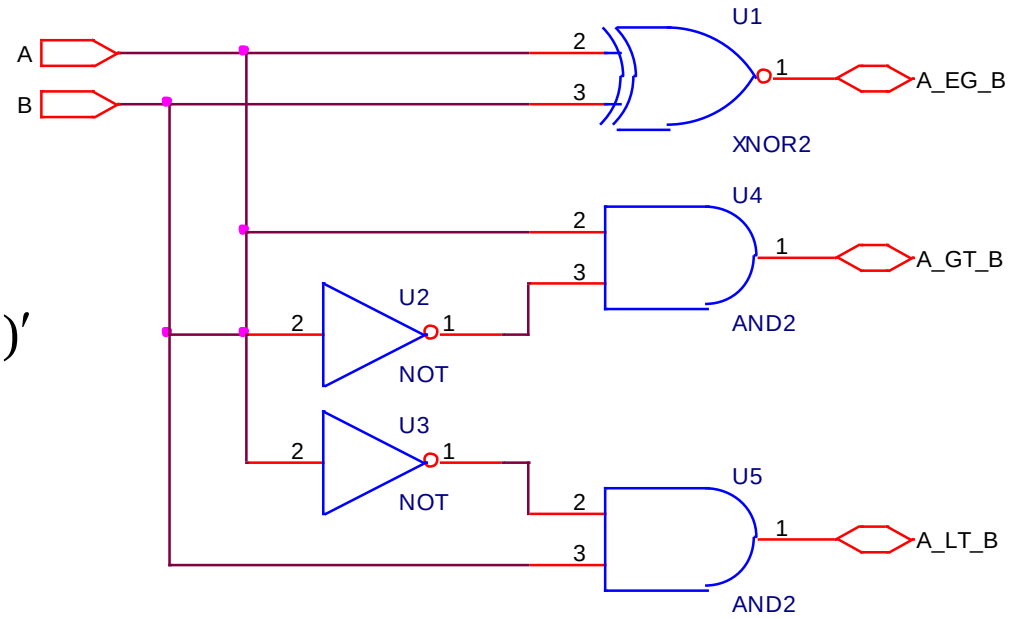
```
module binary_comp_1b ( output A_EQ_B, A_GT_B, A_LT_B,
                        input A, B);
        assign A_EQ_B = ~( A ^ B ),
               A_GT_B =    A & ~B,
               A_LT_B =   ~A &  B;
endmodule
```

1位比较器的逻辑表达式：

$$A\_EQ\_B = A' \bullet B' + A \bullet B = (A \oplus B)'$$

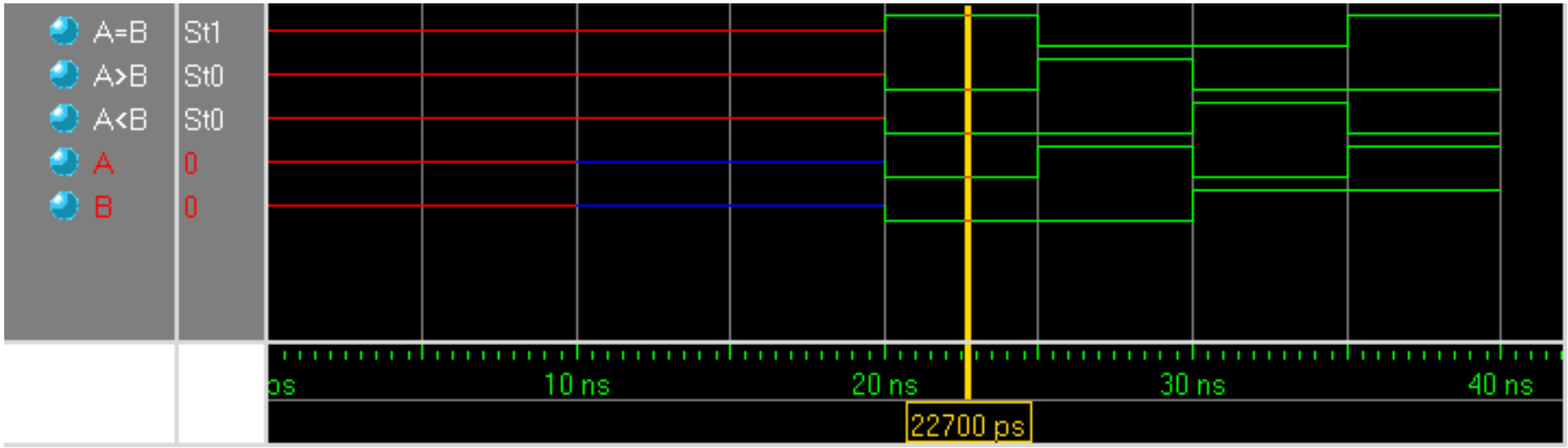
$$A\_GT\_B = A \bullet B'$$

$$A\_LT\_B = A' \bullet B$$



1位比较器的仿真测试

```
# At     0, A=x, B=x, <--> [A=B]= x, [A>B]= x, [A<B]= x
# At   100, A=z, B=z, <--> [A=B]= x, [A>B]= x, [A<B]= x
# At   200, A=0, B=0, <--> [A=B]= 1, [A>B]= 0, [A<B]= 0
# At   250, A=1, B=0, <--> [A=B]= 0, [A>B]= 1, [A<B]= 0
# At   300, A=0, B=1, <--> [A=B]= 0, [A>B]= 0, [A<B]= 1
# At   350, A=1, B=1, <--> [A=B]= 1, [A>B]= 0, [A<B]= 0
```

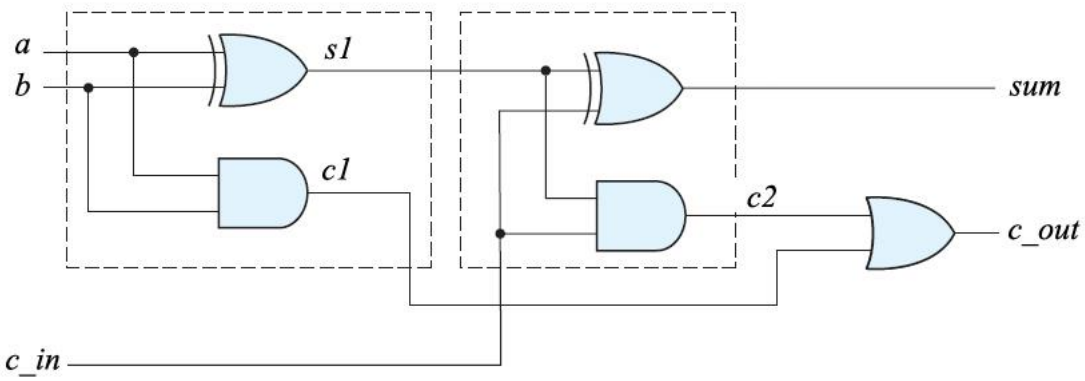


1 位全加器

- 1位全加器的数学表达式

$$\text{sum} = (a \oplus b \oplus \text{cin})$$

$$\text{cout} = (a \bullet b) + \text{cin} \bullet (a \oplus b)$$

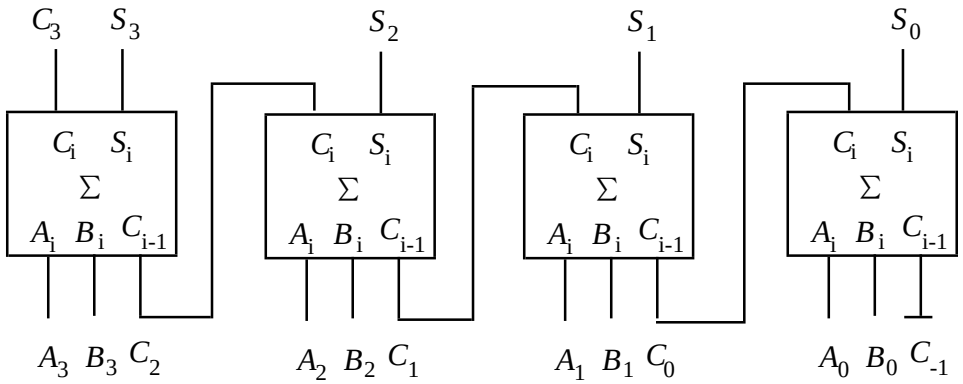



```
// Define a 1-bit full adder
module fulladd(output cout, sum, input a, b, cin);
    // Internal nets
    wire s1, c1, s2;

    // Instantiate logic gate primitives
    xor u10(s1, a, b);
    and u11(c1, a, b);

    xor u20(sum, s1, cin);
    and u21(s2,  s1, cin);

    or u30(cout, s2, c1);
endmodule
```

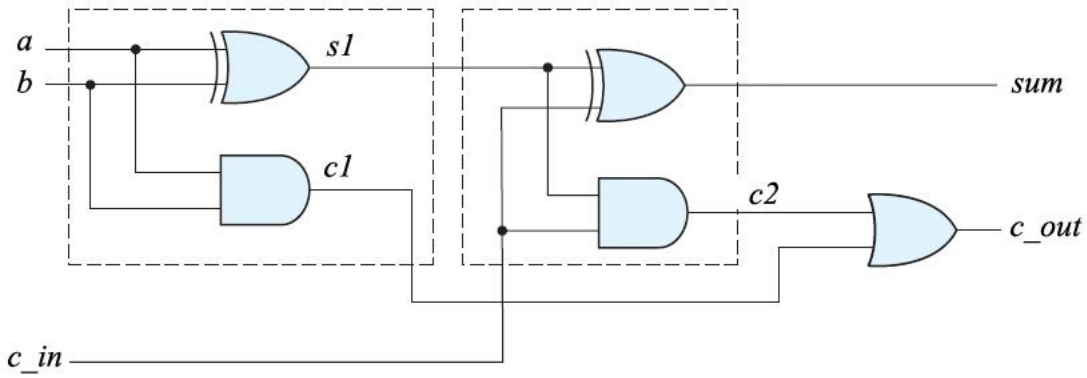
4位串行进位加法器

4位串行进位加法器

- 两个4位相加数 $A_3A_2A_1A_0$ 和 $B_3B_2B_1B_0$ 的各位同时送到相应全加器的输入端
- 进位数串行传送，最低位全加器的 C_{i-1} 端应接0
- 缺点：
  - 速度比较慢，进位信号是串行传递
  - 进位输出C3要经过四位全加器传递之后才能形成



超前进位加法器 —— 各级进位同时送到全加器进位输入端



全加器输出 S_i 和 C_i 逻辑表达式：

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$
$$C_i = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_{i-1} \qquad (1)$$

定义 $G_i = A_i \cdot B_i$，称为产生变量。当 $A_i = B_i = 1$ 时，$A_i \cdot B_i = 1$，得 $C_i = 1$，产生进位。

定义 $P_i = A_i \oplus B_i$，称为传输变量。当 $A_i \oplus B_i = 1$ 时，$A_i \cdot B_i = 0$，得 $C_i = C_{i-1}$，

即低位的进位传送到高位的进位输入端。

G_i 和 P_i 都只与被加数 A_i 和加数 B_i 有关，而与进位信号无关。

超前进位加法器 —— 各级进位同时送到全加器进位输入端

将 $G_i = A_i \cdot B_i$ 和 $P_i = A_i \oplus B_i$ 代入式：$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_{i-1} \\ C_i &= A_i \cdot B_i + (A_i \oplus B_i) \cdot C_{i-1} \end{aligned}$，得：

$$S_i = P_i \oplus C_{i-1} \tag{2a}$$

$$C_i = G_i + P_i C_{i-1} \tag{2b}$$

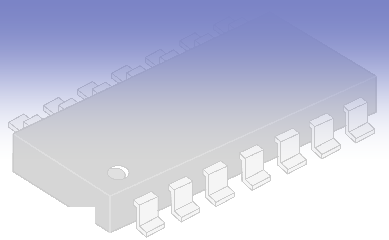
由式（2b）得各位进位信号的逻辑表达式如下：

$$\begin{aligned} C_0 &= G_0 + P_0 C_{-1} \\ C_1 &= G_1 + P_1 C_0 = G_1 + P_1 G_0 + P_1 P_0 C_{-1} \\ C_2 &= G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1} \\ C_3 &= G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1} \end{aligned} \tag{3}$$

由式（3）可以看出：各位的进位信号都只与 G_i、P_i 和 C_{-1} 有关，而 C_{-1} 是最低位的进位信号；

各位的进位都只与被加数 A_i 和加数 B_i 有关，它们是可以并行产生的，从而可实现快速进位。

4 位超前进位加法器



```
//dataflow 4-bit carry lookahead adder
module adder_cla4 ( output cout, output [3:0] sum, input  [3:0] a, b,
input cin );
    // define internal wires
    wire [3:0] gen, pro;
    genvar k;

    generate for (k=0; k<4; k=k+1) begin: GLOOP
        assign gen[k] = a[k] & b[k];
        assign pro[k] = a[k] ^ b[k];
    end
    endgenerate

    wire g3, g2, g1, g0, p3, p2, p1, p0, c3, c2, c1, c0;

    assign {g3, g2, g1, g0} = gen,
           {p3, p2, p1, p0} = pro;

    //obtain the carry equations
    assign c0 = g0 | (p0 & cin),
           c1 = g1 | (p1 & g0) | (p1 & p0 & cin),
           c2 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & cin),
           c3 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0)
                | (p3 & p2 & p1 & p0 & cin);
    //obtain the sum equations
    assign sum = {p3, p2, p1, p0} ^ { c2, c1, c0, cin};
    //obtain cout
    assign cout = c3;
endmodule
```

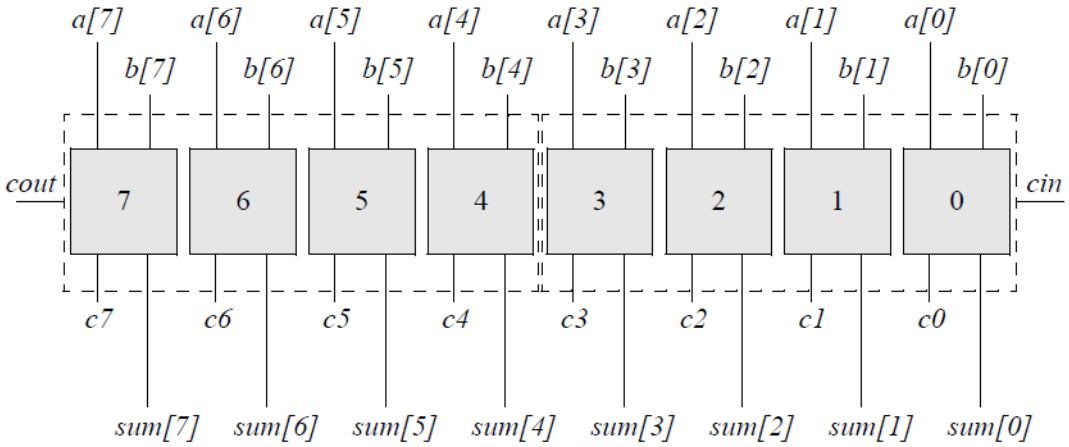
$$G_i = A_i \cdot B_i$$
$$P_i = A_i \oplus B_i$$

$$C_0 = G_0 + P_0 C_{-1}$$
$$C_1 = G_1 + P_1 C_0 = G_1 + P_1 G_0 + P_1 P_0 C_{-1}$$
$$C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1}$$
$$C_3 = G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}$$

$$S_i = P_i \oplus C_{i-1}$$

8 位超前进位加法器

- 将8位加法器分成两组
  - 高4位加法器：sum[7:4] = a[7:4] + b[7:4]+ c3
  - 低4位加法器：sum[3:0] = a[3:0] + b[3:0]+ cin



$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

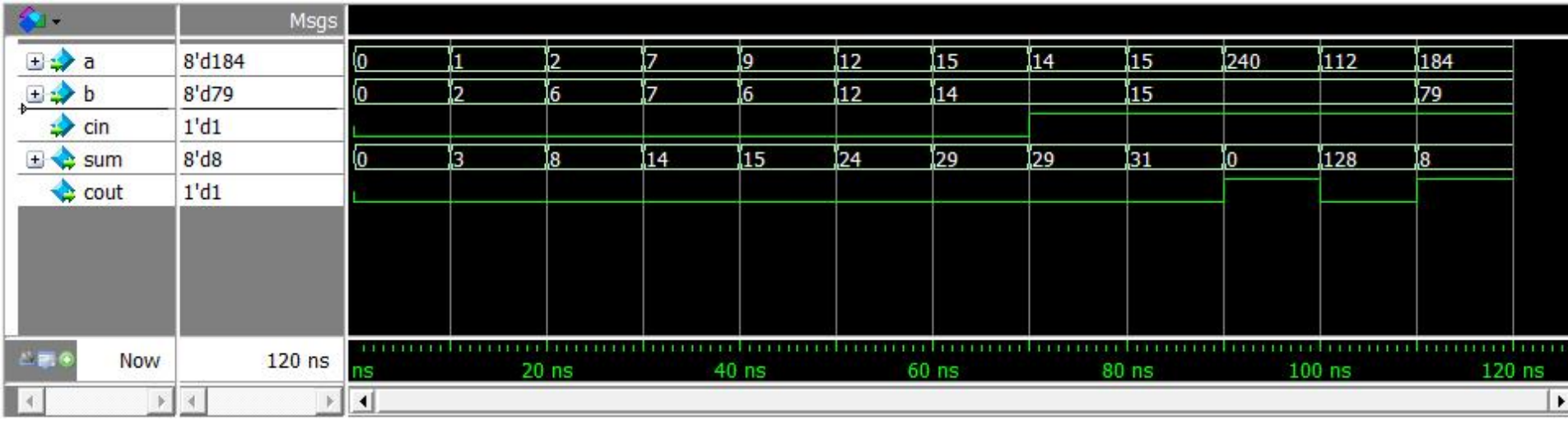
$$C_i = A_i \bullet B_i + (A_i \oplus B_i) \bullet C_{i-1}$$

```
module adder_cla8a( output cout, output [7:0] sum, input  [7:0] a, b, input cin );
    wire c3, c7;
    wire [3:0] s0, s1, a0, a1, b0, b1;

    assign a0 = a[3:0], a1 = a[7:4], b0 = b[3:0], b1 = b[7:4];

    adder_cla4 u0( .cout(c3), .sum(s0), .a(a0), .b(b0), .cin(cin) );
    adder_cla4 u1( .cout(c7), .sum(s1), .a(a1), .b(b1), .cin(c3)  );

    assign {cout, sum } = {c7, s1, s0};
endmodule
```

8 位超前进位加法器测试模块

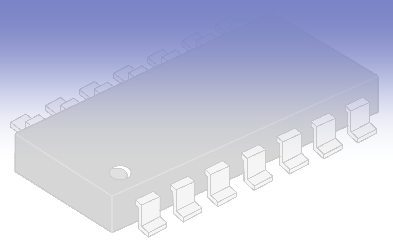
```
`timescale 1ns/1ns
`include "adder_cla8a.v"
//test bench for dataflow 8-bit carry lookahead adder
module adder_cla8a_tb;
    reg [7:0] a, b;
    reg cin;
    wire [7:0] sum;
    wire cout;
    //display signals
    initial
        $monitor ("a = %d, b = %d, cin = %b, cout = %b, sum = %d", a, b, cin, cout, sum);

    //apply stimulus
    initial begin
         #0 a = 8'b0000_0000;   b = 8'b0000_0000;   cin = 1'b0; //cout = 0, sum = 0000_0000
        #10 a = 8'b0000_0001;   b = 8'b0000_0010;   cin = 1'b0; //cout = 0, sum = 0000_0011
        #10 a = 8'b0000_0010;   b = 8'b0000_0110;   cin = 1'b0; //cout = 0, sum = 0000_1000
        #10 a = 8'b0000_0111;   b = 8'b0000_0111;   cin = 1'b0; //cout = 0, sum = 0000_1110
        #10 a = 8'b0000_1001;   b = 8'b0000_0110;   cin = 1'b0; //cout = 0, sum = 0000_1111
        #10 a = 8'b0000_1100;   b = 8'b0000_1100;   cin = 1'b0; //cout = 0, sum = 0001_1000
        #10 a = 8'b0000_1111;   b = 8'b0000_1110;   cin = 1'b0; //cout = 0, sum = 0001_1101
        #10 a = 8'b0000_1110;   b = 8'b0000_1110;   cin = 1'b1; //cout = 0, sum = 0001_1101
        #10 a = 8'b0000_1111;   b = 8'b0000_1111;   cin = 1'b1; //cout = 0, sum = 0001_1111
        #10 a = 8'b1111_0000;   b = 8'b0000_1111;   cin = 1'b1; //cout = 1, sum = 0000_0000
        #10 a = 8'b0111_0000;   b = 8'b0000_1111;   cin = 1'b1; //cout = 0, sum = 1000_0000
        #10 a = 8'b1011_1000;   b = 8'b0100_1111;   cin = 1'b1; //cout = 1, sum = 0000_1000
        #10 $stop;
    end

    adder_cla8a inst1 ( //instantiate the module
        .a(a),
        .b(b),
        .cin(cin),
        .sum(sum),
        .cout(cout)
    );
endmodule
```

8 位超前进位加法器仿真结果




```
# a =   0, b =   0, cin = 0, cout = 0, sum =   0
# a =   1, b =   2, cin = 0, cout = 0, sum =   3
# a =   2, b =   6, cin = 0, cout = 0, sum =   8
# a =   7, b =   7, cin = 0, cout = 0, sum =  14
# a =   9, b =   6, cin = 0, cout = 0, sum =  15
# a =  12, b =  12, cin = 0, cout = 0, sum =  24
# a =  15, b =  14, cin = 0, cout = 0, sum =  29
# a =  14, b =  14, cin = 1, cout = 0, sum =  29
# a =  15, b =  15, cin = 1, cout = 0, sum =  31
# a = 240, b =  15, cin = 1, cout = 1, sum =   0
# a = 112, b =  15, cin = 1, cout = 0, sum = 128
# a = 184, b =  79, cin = 1, cout = 1, sum =   8
```

8 位乘法器



- 二进制乘法
  - 0×0=0，0×1=0，1×0=0，1×1=1
  - 两个 4 位二进制数的相乘过程

	被乘数				X_3	X_2	X_1	X_0
×	乘　数				Y_3	Y_2	Y_1	Y_0
					Y_0X_3	Y_0X_2	Y_0X_1	Y_0X_0
				Y_1X_3	Y_1X_2	Y_1X_1	Y_1X_0	
			Y_2X_3	Y_2X_2	Y_2X_1	Y_2X_0		
		Y_3X_3	Y_3X_2	Y_3X_1	Y_3X_0			
	乘　积	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

  - 两个 4 位二进制数的相乘结果是 1 个 8 位二进制数
    - $S_7S_6S_5S_4S_3S_2S_1S_0$
  - 8 位二进制数的相乘的 Verilog 模块

```
module mult_8b( output reg [15:0] product,
                input [7:0] a, b);
   always @(*) product = a * b;
endmodule
```

18 位乘法器

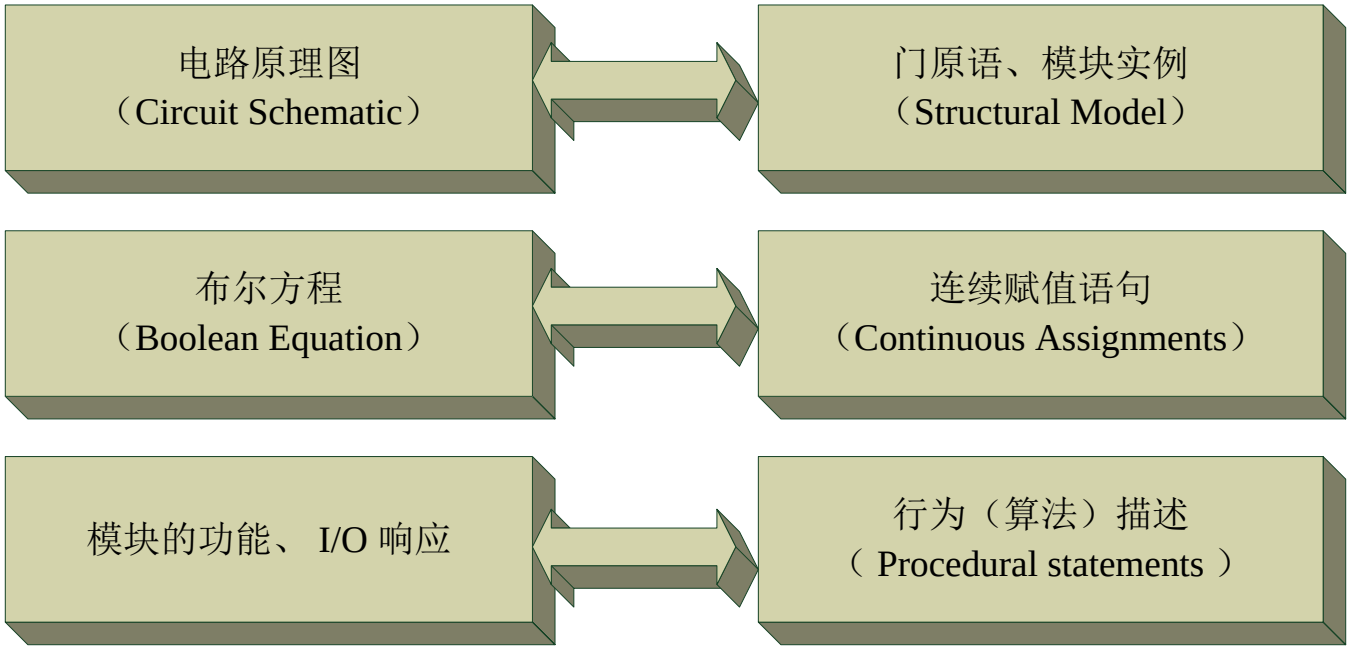
- 基于 Xilinx FPGA 设计
  - 《Spartan-3E Libraries Guide for HDL Designs —— ISE 10.1》
    - Spartan-3E —— FPGA 系列
    - ISE 10.1 —— 开发系统版本
  - 使用原语：MULT18X18
    - 采用组合逻辑形式实现 18X18 有符号乘法器
    - 输入、输出和输出均采用二进制补码
- Verilog 实例模板

Primitive: 18 x 18 Signed Multiplier

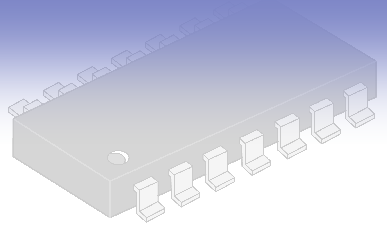

```
module mult18x18s( output [35:0] f, input [17:0] a, b );
    // Verilog Instantiation Template
    // MULT18X18: 18 x 18 signed asynchronous multiplier
    // Virtex-II/II-Pro, Spartan-3
    // Xilinx HDL Libraries Guide, version 10.1.2
    MULT18X18 MULT18X18_inst (
        .P(f), // 36-bit multiplier output
        .A(a), // 18-bit multiplier input
        .B(b)  // 18-bit multiplier input
    );
    // End of MULT18X18_inst instantiation
endmodule
```

电路模块的设计

组合逻辑功能的 Verilog 模块描述



总结



- 采用分层的设计思想
- 将大型的设计划分成若干小型的模块组件
- 每个小型的模块完成基本的逻辑功能
- 由一组小型的模块构成大型的逻辑运算的部件
- 基本的算术逻辑运算可以使用行为描述方法