

# 数字逻辑设计

张春慨  
计算机科学与技术学院  
ckzhang@hit.edu.cn

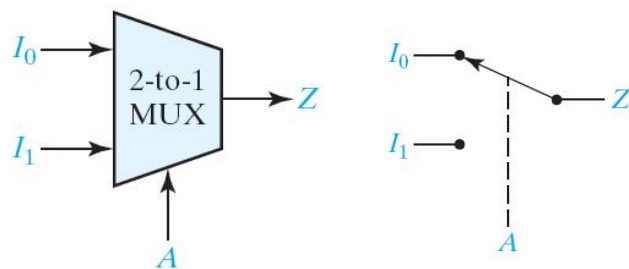
# 组合逻辑元件的Verilog实现

---

- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

# 2选1多路选择器

2选1多路选择器



$$Z = A'I_0 + AI_1$$

当输入 $I$ 的位宽为**8**，则称为2输入8位多路复用器

# 采用多重if语句实现2输入8位多路复用器

---

```
module Vrmux2in8b_b(  
    input EN_L, S,    //EN-L为低电平有效的使能信号， S为数据选择控制端  
    input [7:0] D0, D1,  
    output reg [7:0] Y  
);  
    always @(*) begin  
        if (~EN_L == 1'b0) Y = 8'b0;           //如果EN_L不为0则输出0  
        else if (S == 1'b0) Y = D0;           //若S为0输出D0寄存器值  
        else if (S == 1'b1) Y = D1;           //S为1输出D1寄存器值  
        else Y = 8'b0;  
    end  
endmodule
```

# 2输入8位多路复用器的数据流型Verilog模块

---

- 可以使用一系列条件操作符(?:)来提供所要求的功能。

```
module Vrmux2in8b_d(  
    input EN_L, S,  
    input [7:0] D0, D1,  
    output [7:0] Y  
);
```

```
    assign Y = (~EN_L == 1'b0) ? 8'b0 : (  
                (S == 1'd0) ? D0 :  
                ((S == 1'd1) ? D1 : 8'b0)); //? 前条件满足则取前一个值
```

```
endmodule
```

# 采用case语句的4输入8位多路复用器

- 一个选择输入值对应一个**case**语句

- 更易读也更好维护。

```
module Vrmux4in8b(
    input EN_L,
    input [1:0] S,
    input [7:0] A, B, C, D,
    output reg [7:0] Y);
always @ (*) begin
    if (~EN_L == 1'b0) Y = 8'b0;
    else case (S) //真值表做法: 2'd0指2bit, 十进制0
        2'd0: Y = A;
        2'd1: Y = B;
        2'd2: Y = C;
        2'd3: Y = D;
        default: Y = 8'b0;
    endcase
end
endmodule
```

# 组合逻辑元件的Verilog实现

---

- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

# 三态缓冲器 three-state buffer

## ■ 又称三态驱动器

➤ 4种物理上不同的三态缓冲器的逻辑符号

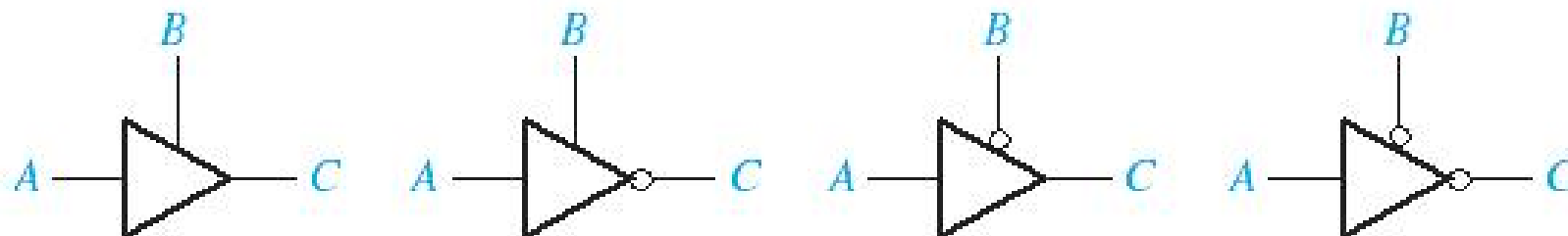
三态缓冲器还可用来增强输出驱动能力。

三态——

■ 0

■ 1

■ Z: 高阻态电阻很大, 相当于开路



B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

(a)

B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

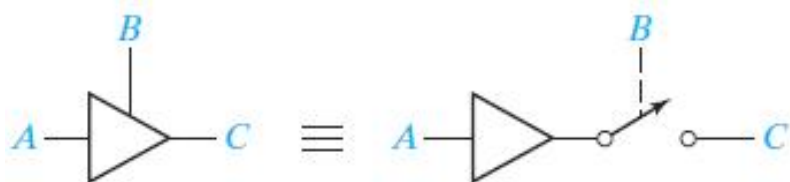
(b)

B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z

(c)

B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

(d)





# 用Verilog实现三态输出

---

- Verilog为高阻态内设了位数据值“z”，很容易指定三态输出

```
module Vr74x541(  
    input G1_L, G2_L, //门控信号，低电平有效  
    input [7:0] A,  
    output [7:0] Y);  
  
    assign Y = (~G1_L & ~G2_L) ? A :  
        8'bzzzz_zzzz; //只有G1_L和G2_L都为0.输出A  
endmodule
```

```
module Vr74x245(  
    input G_L, DIR,  
    inout [7:0] A, B); //输出端口可作输入  
  
    assign A = (~G_L & ~DIR) ? B :  
        8'bzzzz_zzzz;  
    assign B = (~G_L & DIR) ? A :  
        8'bzzzz_zzzz; //满足条件则AB交换输出  
endmodule
```

# 组合逻辑元件的Verilog实现

---

- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

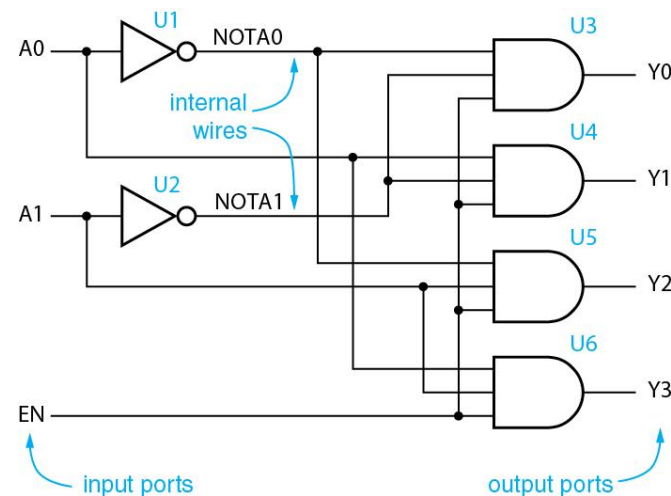
# 用Verilog实现2-4译码器——1

- Verilog设计译码器方法有很多
- 最原始的方法是编写等效于译码器逻辑电路的结构性程序。

```
module Vr2to4dec_s(A0, A1, EN, Y0, Y1, Y2, Y3);  
  input A0, A1, EN;  
  output Y0, Y1, Y2, Y3;  
  wire NOTA0, NOTA1;
```

```
  not U1 (NOTA0, A0);  
  not U2 (NOTA1, A1);  
  and U3 (Y0, NOTA0, NOTA1, EN);  
  and U4 (Y1, A0, NOTA1, EN);  
  and U5 (Y2, NOTA0, A1, EN);  
  and U6 (Y3, A0, A1, EN);  
endmodule
```

缺点：不易于理解和维护



Inputs			Outputs			
EN	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

# 用Verilog实现2-4译码器——2

- 使用Verilog的行为化描述。
- 采用了一个**always**语句，其敏感信号列表包括译码器所有输入。
- 将输出变量声明为**reg**类型，故而在过程块中对输出变量赋值。
- 用一个**if**语句来测试使能输入。
  - 如果**EN**为0，所有输出都设置为0；
  - 当**EN**有效时，译码器的功能。

```
module Vr2to4dec_b1(A0, A1, EN, Y0, Y1, Y2, Y3);
    input A0, A1, EN;
    output reg Y0, Y1, Y2, Y3;

    always @ (A0, A1, EN)
        if (EN==0) {Y3,Y2,Y1,Y0} = 4'b0000;
        else
            case ({A1,A0})
                2'b00: {Y3,Y2,Y1,Y0} = 4'b0001;
                2'b01: {Y3,Y2,Y1,Y0} = 4'b0010;
                2'b10: {Y3,Y2,Y1,Y0} = 4'b0100;
                2'b11: {Y3,Y2,Y1,Y0} = 4'b1000;
                default: {Y3,Y2,Y1,Y0} = 4'b0000;
            endcase
    endmodule
```

# 用Verilog实现2-4译码器——3

---

- 使用2-4二进制译码器的行为化风格Verilog模块

- 能够能更好地捕捉到译码器的行为特性。

```
module Vr2to4dec_b2(A0, A1, EN, Y0, Y1, Y2, Y3);  
  input A0, A1, EN;  
  output reg Y0, Y1, Y2, Y3;  
  reg [3:0] IY;  
  integer i;  
  always @(A0 or A1 or EN) begin  
    IY = 4'b0000;    // Default outputs all 0  
    if (EN==1)       // If enabled...  
      for (i=0; i<=3; i=i+1) // set out bit i where i equals {A1,A0}  
        if (i == {A1,A0}) IY[i] = 1;  
    {Y3,Y2,Y1,Y0} = IY; // Copy internal bit-vector variable to outputs  
  end  
endmodule
```

# 用Verilog实现2-4译码器——4

---

- 最简练的行为化模型: 在将输出位初始化为全0之后, 其只需要索引 {A1, A0} 将IY为设置为1

```
module Vr2to4dec_b3(A0, A1, EN, Y0, Y1, Y2, Y3);
```

```
    input A0, A1, EN;
```

```
    output reg Y0, Y1, Y2, Y3;
```

```
    reg [3:0] IY;
```

```
    always @ (*) begin
```

```
        IY = 4'b0000;           // Default outputs all 0
```

```
        if (EN==1) IY[{A1,A0}] = 1; // Set selected output bit if enabled
```

```
        {Y3,Y2,Y1,Y0} = IY;      // Copy internal variable to output
```

```
    end
```

```
endmodule
```

# 2-4译码器的测试平台

- 设计一个测试平台来确保设计的正确性。
- 对于只有三个输入的译码器，就有八个不同的输入组合，这个测试平台采用一个变量*i*来逐一检查这些组合。

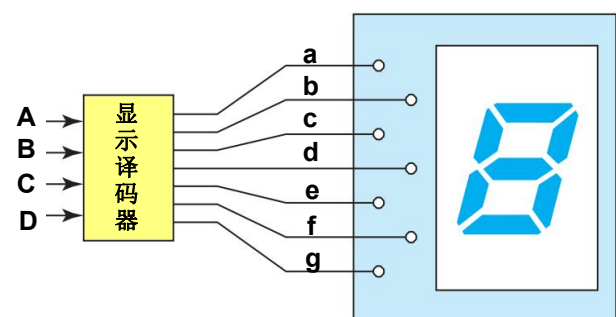
```
`timescale 1 ns / 100 ps
module Vr2to4dec_tb () ;
    reg A0s, A1s, ENs;
    wire Y0s, Y1s, Y2s, Y3s;
    integer i, errors;
    reg [3:0] expectY;

    Vr2to4dec_d UUT ( .A0(A0s),.A1(A1s),.EN(ENs), // Instantiate unit under test (UUT)
        .Y0(Y0s),.Y1(Y1s),.Y2(Y2s),.Y3(Y3s) );

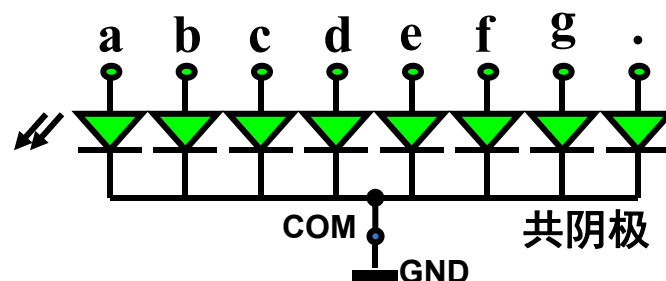
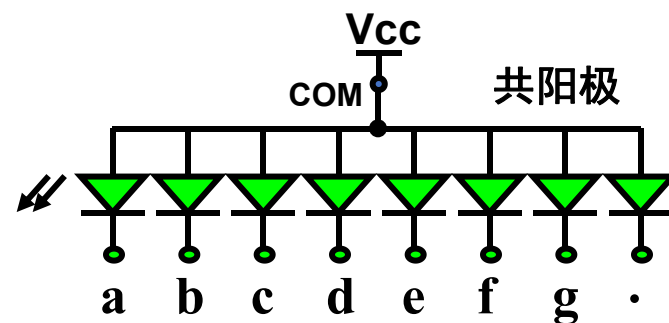
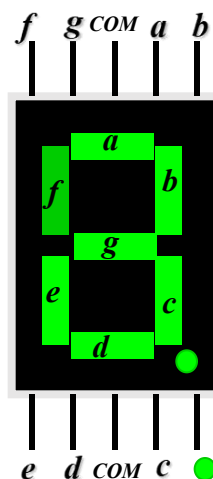
    initial begin
        errors = 0;
        for (i=0; i<=7; i=i+1) begin
            {ENs, A1s, A0s} = i;                // Apply test input combination
            #10 ;
            expectY = 4'b0000;                  // Expect no outputs asserted if EN = 0
            if (ENs==1) expectY[{A1s,A0s}] = 1'b1; // Else output {A1,A0} should be asserted
            if ({Y3s,Y2s,Y1s,Y0s} != expectY) begin
                $display("Error: EN A1A0 = %b %b%b, Y3Y2Y1Y0 = %b%b%b%b",
                    ENs, A1s, A0s, Y3s, Y2s, Y1s, Y0s);
                errors = errors + 1;
            end
        end
        $display("Test complete, %d errors",errors);
    end
endmodule
```

# 七段显示译码器

显示译码器：与显示器件（如数码管）配合，将输入代码转换为十进制码或特定编码，并在显示器件上显示相应的字形



七段数码管



8421BCD码驱动的共阴极七段数码管显示译码器功能表

输入				译码输出						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1




# 用Verilog实现七段译码器

```
module Vr7segdec(
    input [3:0] DIG,
    input EN,
    output wire SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG);
    reg [6:0] SEGS;
    always @ (*) begin
        if (EN)
            case (DIG) // Segment patterns  abcdefg
                0: SEGS = 7'b1111110; // 0
                1: SEGS = 7'b0110000; // 1
                2: SEGS = 7'b1101101; // 2
                3: SEGS = 7'b1111001; // 3
                4: SEGS = 7'b0110011; // 4
                5: SEGS = 7'b1011011; // 5
                6: SEGS = 7'b0011111; // 6 (no 'tail') // 6: SEGS = 7'b1011111; // 6 ('tail' included)
                7: SEGS = 7'b1110000; // 7
                8: SEGS = 7'b1111111; // 8
                9: SEGS = 7'b1110011; // 9 (no 'tail') // 9: SEGS = 7'b1111011; // 9 ('tail' included)
                // 10: SEGS = 7'b1110111; // A //11: SEGS = 7'b0011111; // b //12: SEGS = 7'b1001110; // C
                // 13: SEGS = 7'b0111101; // d //14: SEGS = 7'b1001111; // E //15: SEGS = 7'b1000111; // F
            default: SEGS = 7'b0000000;
            endcase
        else SEGS = 7'b0000000;
        end
        assign {SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG} = SEGS;
    endmodule
```

# 七段译码器的Verilog测试平台

```
`timescale 1ns / 100ps
module Vr7seg_tb ();
    reg EN;
    reg [3:0] DIG;
    wire SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
    integer i;
    Vr7segdec UUT (.DIG(DIG), .EN(EN), .SEGA(SEGA), .SEGB(SEGB), .SEGC(SEGC), .SEGD(SEGD), .SEGE(SEGE),
    .SEGF(SEGF), .SEGG(SEGG) ); //这里的名字要与被实例化的模块名字一致。
    initial
    begin
        EN = 1; // Enable all
        for (i=0; i<16; i=i+1)
            begin
                DIG = i;
                #5
                $write("Iteration %0d\n", i);
                if (SEGA) $write(" _ \n"); else $write("\n");
                if (SEGF) $write("|"); else $write(" ");
                if (SEGG) $write(" "); else $write(" ");
                if (SEGB) $write("| \n"); else $write("\n");
                if (SEGE) $write("|"); else $write(" ");
                if (SEGD) $write(" "); else $write(" ");
                if (SEGC) $write("| \n"); else $write("\n");
                #5 ;
            end
        $write("Done\n");
    end
endmodule
```



if (SEGA) \$write(" \_ \n"); else \$write("\n");  
if (SEGF) \$write("|"); else \$write(" ");  
if (SEGG) \$write(" "); else \$write(" ");  
if (SEGB) \$write("| \n"); else \$write("\n");  
if (SEGE) \$write("|"); else \$write(" ");  
if (SEGD) \$write(" "); else \$write(" ");  
if (SEGC) \$write("| \n"); else \$write("\n");

Iteration 5

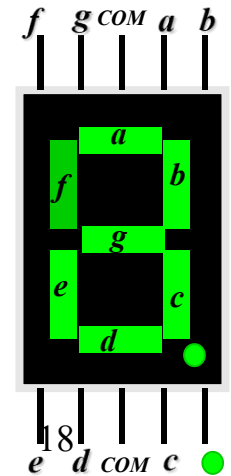
```
 _
|
 _
|
```

Iteration 6

```
 |
|
|
```

Iteration 7

```
 _
|
```



# 用Verilog实现七段译码器——修改版

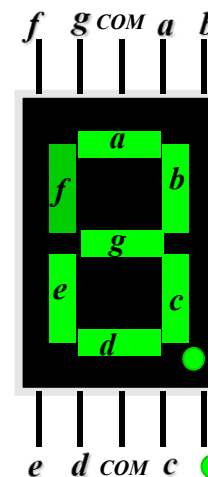
```
module Vr7segdec(
    input [3:0] DIG,
    input EN,
    output reg [6:0] SEGS;
    always @(*) begin
        if (EN)
            case (DIG) // Segment patterns  abcdefg
                0: SEGS = 7'b1111110; // 0
                1: SEGS = 7'b0110000; // 1
                2: SEGS = 7'b1101101; // 2
                3: SEGS = 7'b1111001; // 3
                4: SEGS = 7'b0110011; // 4
                5: SEGS = 7'b1011011; // 5
                6: SEGS = 7'b0011111; // 6 (no 'tail') // 6: SEGS = 7'b1011111; // 6 ('tail' included)
                7: SEGS = 7'b1110000; // 7
                8: SEGS = 7'b1111111; // 8
                9: SEGS = 7'b1110011; // 9 (no 'tail') // 9: SEGS = 7'b1111011; // 9 ('tail' included)
                // 10: SEGS = 7'b1110111; // A // 11: SEGS = 7'b0011111; // b //12: SEGS = 7'b1001110; // C
                // 13: SEGS = 7'b0111101; // d // 14: SEGS = 7'b1001111; // E //15: SEGS = 7'b1000111; // F
            default: SEGS = 7'b0000000;
            endcase
        else SEGS = 7'b0000000;
    end
endmodule
```

# 七段译码器的Verilog测试平台——修改版

```
`timescale 1ns / 100ps
module Vr7seg_tb ();
    reg EN;
    reg [3:0] DIG;
    wire [6:0] SEGS;
    integer i;
    Vr7segE UUT (.DIG(DIG), .EN(EN), .SEGS(SEGS));
    initial
    begin
        EN = 1; // Enable all
        for (i=0; i<16; i=i+1)
            begin
                DIG = i;
                #5
                $write("Iteration %0d\n", i);
                if (SEGS[6]) $write("__\n"); else $write("\n");
                if (SEGS[1]) $write("|"); else $write(" ");
                if (SEGS[0]) $write("__"); else $write(" ");
                if (SEGS[5]) $write("\n"); else $write("\n");
                if (SEGS[2]) $write("|"); else $write(" ");
                if (SEGS[3]) $write("__"); else $write(" ");
                if (SEGS[4]) $write("\n"); else $write("\n");
                #5 ;
            end
        $write("Done\n");
    end
endmodule
```



```
if (SEGS[6]) $write("__\n"); else $write("\n");
if (SEGS[1]) $write("|"); else $write(" ");
if (SEGS[0]) $write("__"); else $write(" ");
if (SEGS[5]) $write("\n"); else $write("\n");
if (SEGS[2]) $write("|"); else $write(" ");
if (SEGS[3]) $write("__"); else $write(" ");
if (SEGS[4]) $write("\n"); else $write("\n");
```



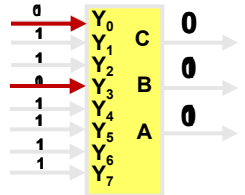
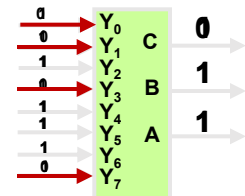
# 组合逻辑元件的Verilog实现

---

- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

# 编码器

- 特点：多输入、多输出的组合逻辑电路
- 功能：将二进制码按照一定规律编排，使其具有特定含义，与译码器互逆。

常用编码器	特点	编码演示
普通编码器 (二进制编码器)	$N$ 位，任何时刻 $N$ 根输入线中只能有一个输入有效， $N$ ( $N=2^n$ ) 中取一。 $n$ 位二进制码	 <p>(8 线-3 线编码器)</p>
优先编码器	<ul style="list-style-type: none"><li>• 允许同时输入两个以上有效输入信号</li><li>• 能按照预先设定的优先级别，只对其优先级最高的输入进行编码。</li></ul>	 <p>(8 线-3 线优先编码器)</p>

# 用Verilog实现优先编码器——用多重if

---

```
module Vr8inprior (  
    input wire [7:0] I,  
    output reg [2:0] A,  
    output reg IDLE);    //IDLE 为0表示优先编码器处于工作状态。  
    always @ (*) begin  
        IDLE=1'b0;  
        if (I[7]) A = 3'h7; //第一  
  
        else if (I[6]) A = 3'h6; //第二优先  
        else if (I[5]) A = 3'h5; //第三优先  
        else if (I[4]) A = 3'h4; //第四优先  
        else if (I[3]) A = 3'h3; //第五优先  
        else if (I[2]) A = 3'h2; //第六优先  
        else if (I[1]) A = 3'h1; //第七优先  
        else if (I[0]) A = 3'h0; //第八优先  
        else begin A = 3'h0; IDLE=1'b1; end; //IDLE 为1表示优先编码器处于空闲状态，A值任意  
    end  
endmodule
```

# 采用case语句的8输入优先编码器

```
module Vr8inprior (  
    input  wire [7:0] I,  
    output reg [2:0] A,  
    output reg IDLE);  
  
    always @ (*) begin  
        if (!I) IDLE = 1'b0;  
        else IDLE = 1'b1;  
    end
```

归约与 &, 归约或 |, 归约异或 ^

**a = 4'b0100**

**&a = 0      (0&1&0&0)**

**| a = 1      (0 | 1 | 0 | 0)**


**^a = 1      (0 ^ 1 ^ 0 ^ 0)**

```
        always @ (*) begin  
            case (1)  
                I[7] : A = 3'h7;    //第一优先  
                I[6] : A = 3'h6;    //第二优先  
                I[5] : A = 3'h5;  
                I[4] : A = 3'h4;  
                I[3] : A = 3'h3;  
                I[2] : A = 3'h2;  
                I[1] : A = 3'h1;  
                I[0] : A = 3'h0;    //第八优先  
                default: A = 3'h0;  
            endcase  
        end  
    endmodule
```



# 8输入优先编码器模块的测试平台

```
`timescale 1ns / 100ps
module Vr8inprior_tb(); //无输入输出变量定义
    reg [7:0] I;
    wire [2:0] A;
    wire IDLE;
    integer ii, errors;
    Vr8inprior UUT ( .I(I), .A(A), .IDLE(IDLE) );
    initial begin
        errors = 0;
        for (ii=0; ii<256; ii=ii+1) begin
            I = ii;
            #10 ;
            if (
                // Identify all error cases
                ( (I==8'b0) && (IDLE!=1'b1) ) // Should be idle
                || ( (I>8'b0) && (IDLE==1'b1) ) // Should not be idle
                || ( (I>8'b0) && (I<2**A) ) // I should be at least 2**A
                || ( (I>8'b0) && (I>=2**(A+1)) ) // but less than 2**(A+1)
            )
                begin
                    errors = errors+1;
                    $display("Error: I=%b, A=%b, IDLE=%b",I,A,IDLE);
                end
            end
            $display("Test done, %d errors\n",errors);
            $stop(1);
        end
    endmodule
```



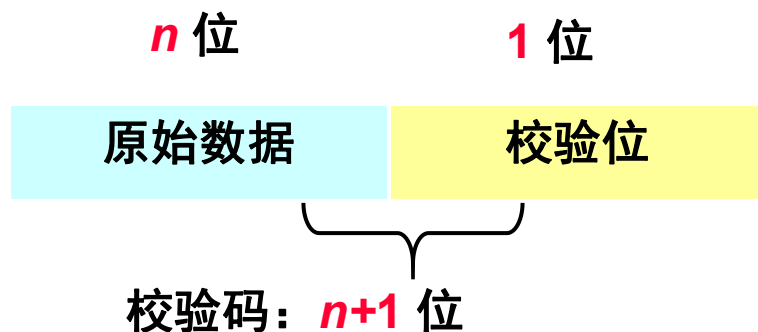
```
for (ii=0; ii<256; ii=ii+1) begin
    I = ii;
    #10 ;
    if (
        //识别所有的错误情况
        ( (I==8'b0) && (IDLE!=1'b1) ) //输入全0却非空闲
        || ( (I>8'b0) && (IDLE==1'b1) ) //输入不全0却空闲
        || ( (I>8'b0) && (I<2**A) ) //输出的编号不符合规则
        || ( (I>8'b0) && (I>=2**(A+1)) ) ) begin
        errors = errors+1;
        $display("Error: I=%b, A=%b,IDLE=%b",I,A,IDLE);
    end
end
```

# 组合逻辑元件的Verilog实现

---

- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

# 校验位计算方法



偶校验位逻辑值的表达式:

$$P_E = A_3 \oplus A_2 \oplus A_1 \oplus A_0$$

奇校验位逻辑值的表达式:

$$P_O = A_3 \oplus A_2 \oplus A_1 \oplus A_0$$

异或门真值表

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

奇偶校验器一般  
由异或门构成

异或门特性

- ◆ 两个输入中有奇数个“1”，输出为1；  
有偶数个“1”，输出为0。
- ◆ 扩展：n个1位二进制数中有奇数个“1”，输出为1；  
有偶数个“1”，输出为0。

4位二进制数校验码真值表

$A_3 A_2 A_1 A_0$	$P_E$	$P_O$
0 0 0 0	0	1
0 0 0 1	1	0
0 0 1 0	1	0
0 0 1 1	0	1
0 1 0 0	1	0
0 1 0 1	0	1
0 1 1 0	0	1
0 1 1 1	1	0
1 0 0 0	1	0
1 0 0 1	0	1
1 0 1 0	0	1
1 0 1 1	1	0
1 1 0 0	0	1
1 1 0 1	1	0
1 1 1 0	1	0
1 1 1 1	0	1

# 用Verilog实现9输入奇偶校验电路

---

```
module Vrparity9s(  
    input wire [8:0] I,  
    output wire EVEN, ODD);  
    wire Y1, Y2, Y3, Y3N;  
    Vrxor3 U1 (I[0], I[1], I[2], Y1);  
    Vrxor3 U2 (I[3], I[4], I[5], Y2);  
    Vrxor3 U3 (I[6], I[7], I[8], Y3);  
    assign Y3N = ~Y3;  
    Vrxor3 U4 (Y1, Y2, Y3, ODD);  
    Vrxor3 U5 (Y1, Y2, Y3N, EVEN);  
endmodule
```

```
module Vrxor3(  
    input wire A, B, C,  
    output wire Y);  
  
    assign Y = A ^ B ^ C;  
endmodule
```

3输入异或器件的Verilog描述

# 组合逻辑元件的Verilog实现

---

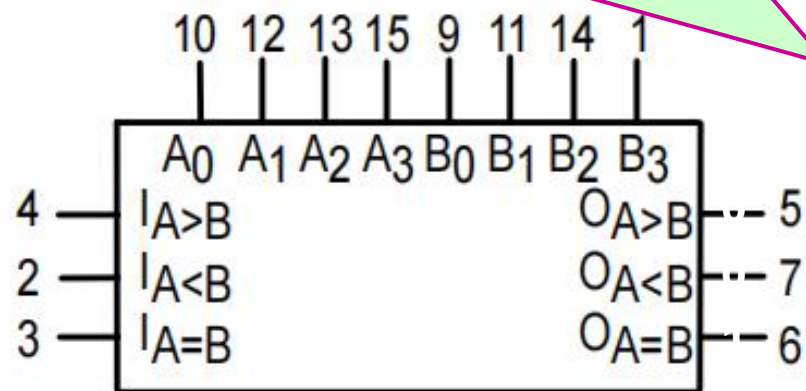
- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

# 多位数值比较器

接低位芯片的比较结果，用于芯片扩展。

## 四位数值比较器74LS85

比较2个4位二进制数的大小时，3个输入端 $I_{A>B}$ 、 $I_{A<B}$ 、 $I_{A=B}$ 应接001；当 $A_3A_2A_1A_0=B_3B_2B_1B_0$ 时，比较器的输出 $Y_{A>B}Y_{A<B}Y_{A=B}=001$



当 $A_3A_2A_1A_0=B_3B_2B_1B_0$ 时，比较器的输出复现3个输入端 $I_{A>B}$ 、 $I_{A<B}$ 、 $I_{A=B}$ 的状态。

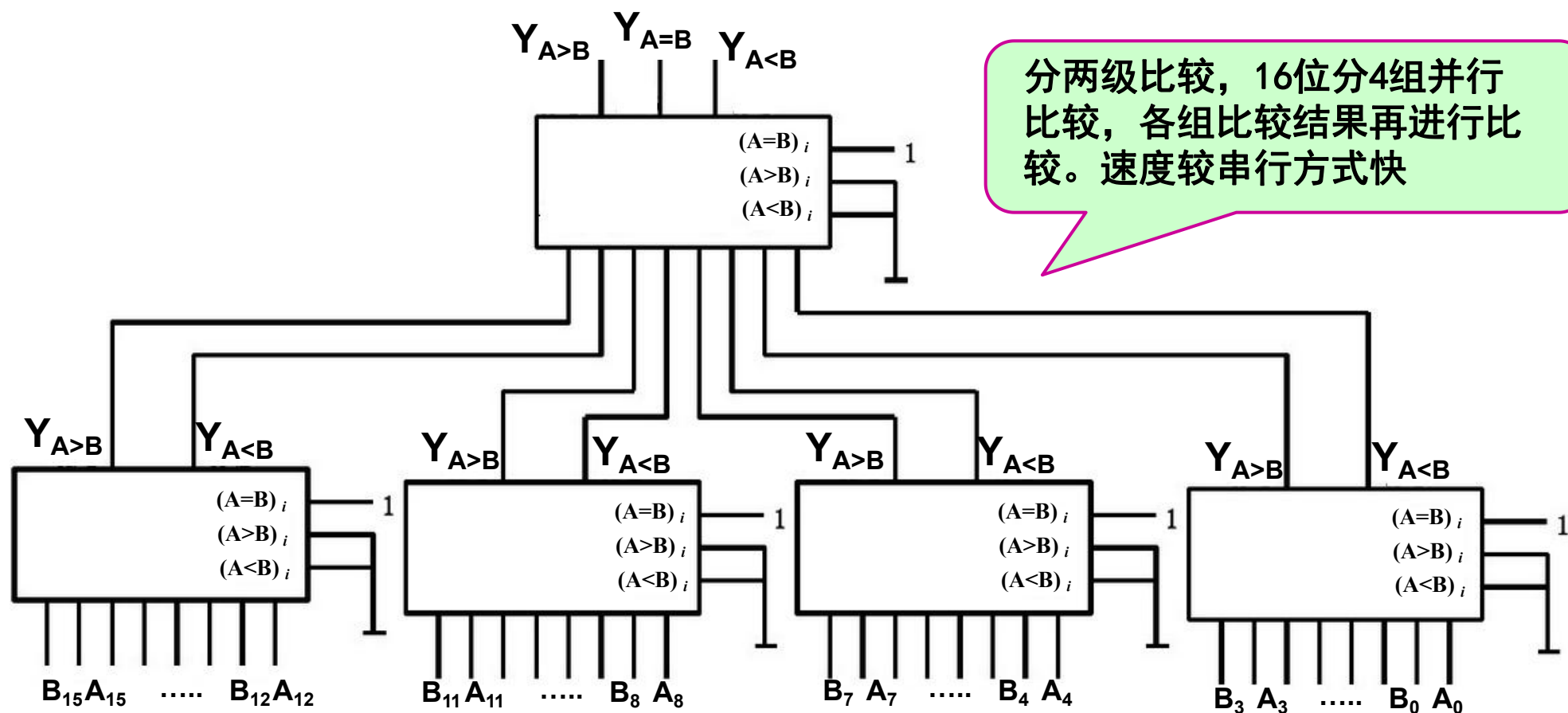
比较输入				级联输入I			输出O						
A <sub>3</sub>	B <sub>3</sub>	A <sub>2</sub>	B <sub>2</sub>	A <sub>1</sub>	B <sub>1</sub>	A <sub>0</sub>	B <sub>0</sub>	(A>B)	(A<B)	(A=B)	O <sub>A&gt;B</sub>	O <sub>A&lt;B</sub>	O <sub>A=B</sub>
A <sub>3</sub> > B <sub>3</sub>		X		X		X		X	X	X	1	0	0
A <sub>3</sub> < B <sub>3</sub>		X		X		X		X	X	X	0	1	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> > B <sub>2</sub>		X		X		X	X	X	1	0	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> < B <sub>2</sub>		X		X		X	X	X	0	1	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> > B <sub>1</sub>		X		X	X	X	1	0	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> < B <sub>1</sub>		X		X	X	X	0	1	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> > B <sub>0</sub>		X	X	X	1	0	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> < B <sub>0</sub>		X	X	X	0	1	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> = B <sub>0</sub>		X	X	1	0	0	1
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> = B <sub>0</sub>		0	1	0	0	1	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> = B <sub>0</sub>		1	0	0	1	0	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> = B <sub>0</sub>		1	1	0	0	0	0
A <sub>3</sub> = B <sub>3</sub>		A <sub>2</sub> = B <sub>2</sub>		A <sub>1</sub> = B <sub>1</sub>		A <sub>0</sub> = B <sub>0</sub>		0	0	0	1	1	0

# 用Verilog实现8位数值比较器

---

```
module Vr8bitcmp(  
    input [7:0] P,Q,  
    output reg PGTQ, PEQQ, PLTQ);  
  
    always @ (*)  
        if (P == Q)  
            begin PGTQ = 1'b0; PEQQ = 1'b1; PLTQ = 1'b0; end  
        else if (P > Q)  
            begin PGTQ = 1'b1; PEQQ = 1'b0; PLTQ = 1'b0; end  
        else  
            begin PGTQ = 1'b0; PEQQ = 1'b0; PLTQ = 1'b1; end  
endmodule
```

# 数值比较器的级联—— ②并行方式





# 64位数值比较器

---

采用9个8位数值比较器构成的64位比较器的Verilog描述

```
module Vr64bitcmp_sh(  
    input [63:0] P,Q,  
    output PGTQ, PEQQ, PLTQ);  
    wire [7:0] GT, EQ, LT;  
    Vr8bitcmp U1(P[7:0], Q[7:0], GT[0], EQ[0], LT[0]);  
    Vr8bitcmp U2(P[15:8], Q[15:8], GT[1], EQ[1], LT[1]);  
    Vr8bitcmp U3(P[23:16], Q[23:16], GT[2], EQ[2], LT[2]);  
    Vr8bitcmp U4(P[31:24], Q[31:24], GT[3], EQ[3], LT[3]);  
    Vr8bitcmp U5(P[39:32], Q[39:32], GT[4], EQ[4], LT[4]);  
    Vr8bitcmp U6(P[47:40], Q[47:40], GT[5], EQ[5], LT[5]);  
    Vr8bitcmp U7(P[55:48], Q[55:48], GT[6], EQ[6], LT[6]);  
    Vr8bitcmp U8(P[63:56], Q[63:56], GT[7], EQ[7], LT[7]);  
    Vr8bitcmp U9(GT, LT, PGTQ, PEQQ, PLTQ);  
endmodule
```

# N位比较器的测试平台

---

```
`timescale 1 ns / 100 ps
module VrNbitcmp_tb();
    parameter N = 64;    // Input width of comparator UUT
    parameter SEED = 1; // Set a different pseudorandom seed here if desired
    reg [N-1:0] P, Q;
    wire PGTQ, PEQQ, PLTQ;
    integer ii, errors;
    Vr64bitcmp_sh UUT ( .P(P), .Q(Q), .PGTQ(PGTQ), .PEQQ(PEQQ), .PLTQ(PLTQ) );
    initial begin
        errors = 0;
        P = $random(SEED); // Set pattern based on seed parameter
        for (ii=0; ii<10000; ii=ii+1) begin
            P = $random; Q = $random;
            #10 ;
            if ( (PGTQ) !== (P>Q) || (PLTQ) !== (P<Q) || (PEQQ) !== (P==Q) ) begin
                errors = errors + 1;
                $display("P=%b(%0d), Q=%b(%0d), PGTQ=%b, PEQQ=%b, PLTQ=%b", P, P, Q, Q, PGTQ, PEQQ, PLTQ);
            end;
        end
        $display("Test done, %0d errors", errors);
    end
endmodule
```

# 组合逻辑元件的Verilog实现

---

- 多路复用器(Multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 加法器

# 全加器

- 除了A、B操作数，还有来自低位的进位C<sub>i</sub>N

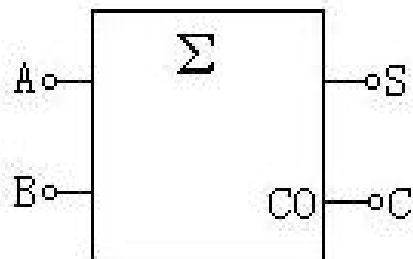
$$\begin{array}{r} 1\ 0\ 1\ 1\ \dots\dots\dots A \\ 1\ 1\ 1\ 0\ \dots\dots\dots B \\ + \qquad\qquad 0\ \dots\dots\dots C_i \\ \hline \dots\dots\dots S_i \end{array}$$

$$A = a_3 a_2 a_1 a_0 = 1011$$

$$B = b_3 b_2 b_1 b_0 = 1110$$

# 异或门的应用——加法器

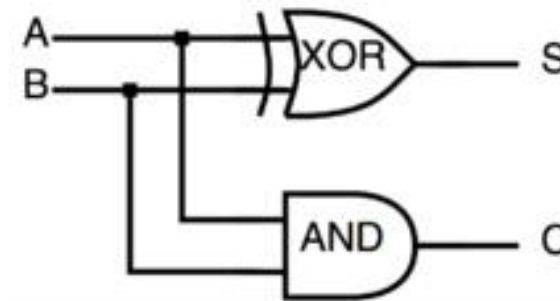
## • 半加器 (Half-adder)



半加器逻辑符号

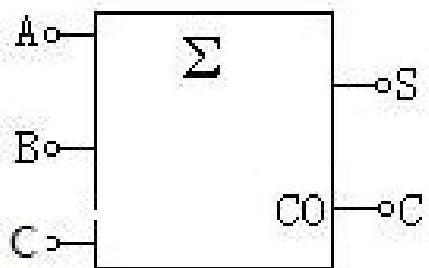
输入		输出	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

逻辑表达式:  $S=A \oplus B$ ;  $C=A \cdot B$



半加器的逻辑实现

## • 全加器 (Full-adder)

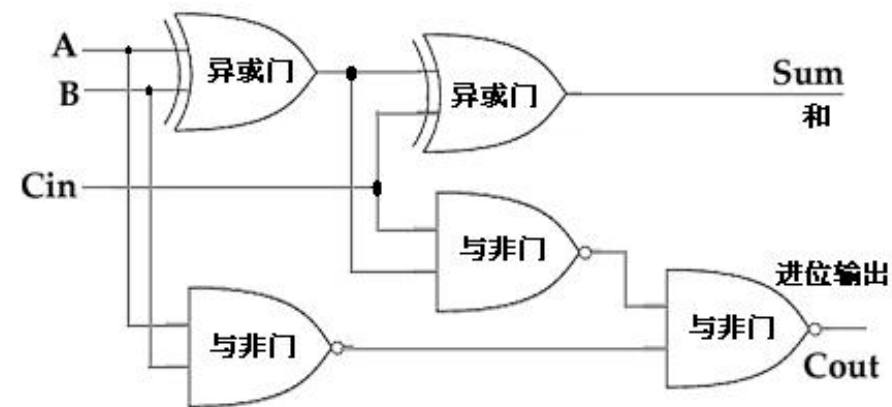


全加器逻辑符号

输入			输出	
$C_{in}$	A	B	$C_{out}$	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$Sum=A \oplus B \oplus C_{in}$$

$$C_{out}=A \cdot B+C_{in} \cdot (A \oplus B)$$



# 用Verilog实现全加器

---

```
module VrNbitadder    //两个N位二进制数A和B相加，进位为COUT，和为S
    #(parameter N=64)  // 模块外定义参数，加数与和的宽度
    (
        input [N-1:0] A, B,          //
        input CIN,                  // 低位来的进位
        output [N-1:0] S,           //和
        output COUT);              //当前位向高位的进位

        assign {COUT, S} = A + B + CIN;
endmodule
```

参数N指定加数与和的宽度。

N位无符号数加法可以产生N+1位的和，所以赋值语句左侧将进位输出COUT与N位输出和S级联，接受N+1位的和。

# 可以同时处理有符号数和无符号数的全加器

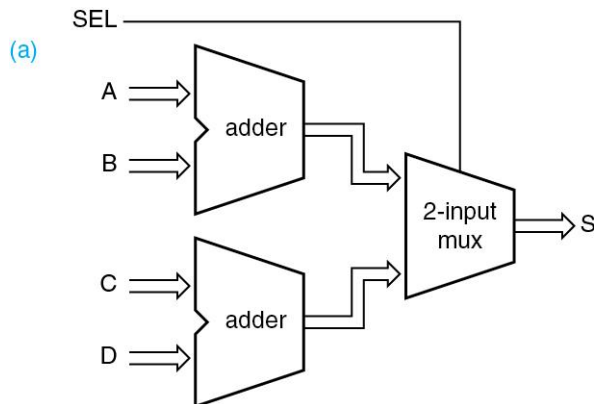
---

```
module Vradders(  
    input [7:0] A, B, C, D,  
    output [7:0] S, T,  
    output OVFL, COUT);  
    //有符号数全加器: S=A+B, OVFL用于判断是否溢出  
    assign S = A + B;  
    assign OVFL = (A[7] == B[7]) && (S[7] != A[7]);    // OVFL判断是否溢  
        //A, B符号相同, 和的最高位(符号位)与A或B的最高位不同, 溢出  
  
    // 无符号数全加器 T=C+D, COUT为进位  
    assign {COUT, T} = C + D;  
endmodule
```

# 加法器复用模块

加法和减法电路的价格昂贵，需要尽可能的对其复用

```
module Vraddersh(  
    input SEL,  
    input [7:0] A, B, C, D,  
    output reg [7:0] S);  
  
    always @ (SEL, A, B, C, D)  
        if (SEL) S = A + B;  
        else S = C + D;  
endmodule
```



```
module Vraddersc(  
    input SEL,  
    input [7:0] A, B, C, D,  
    output [7:0] S);  
  
    assign S = (SEL) ? A + B : C + D;  
endmodule
```

