组合逻辑电路设计与Verilog基础

北京科技大学 计算机系

齐悦

目录

Verilog概述

- **Verilog HDL**与**C**语言在很多地方非常相似。
- **C**语言的各函数之间是串行的，而**Verilog**的各个模块间是并行的
- 体会如何用Verilog描述电路

C语言	Verilog	功能	C语言	Verilog	功能
+	+	加	>=	>=	大于等于
-	-	减	<=	<=	小于等于
*	*	乘	==	==	等于
/	/	除	!=	!=	不等于
%	%	取模	~	~	取反
!	!	逻辑非	&	&	按位与
&&	&&	逻辑与	|	|	按位或
||	||	逻辑或	^	^	按位异或
>	>	大于	<<	<<	左移
<	<	小于	>>	>>	右移

Verilog概述 - 初探

常用信号类型：
wire – 线网类型
reg – 寄存器类型

标 识 符

Verilog中有三种端口类型：
input - 输入端口
output - 输出端口
inout - 双向端口

Module name　　*Module ports*

```
module Add_half ( sum, c_out, a, b );
    input       a, b;
    output      sum, c_out;
    wire        c_out_bar;

    xor (sum, a, b);
    nand (c_out_bar, a, b);
    not (c_out, c_out_bar);
endmodule
```

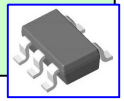
Declaration of port modes

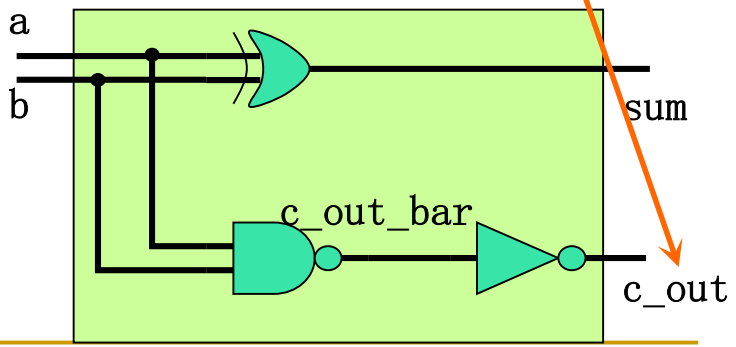
Declaration of internal signal

Instantiation of primitive gates

Verilog 关键字

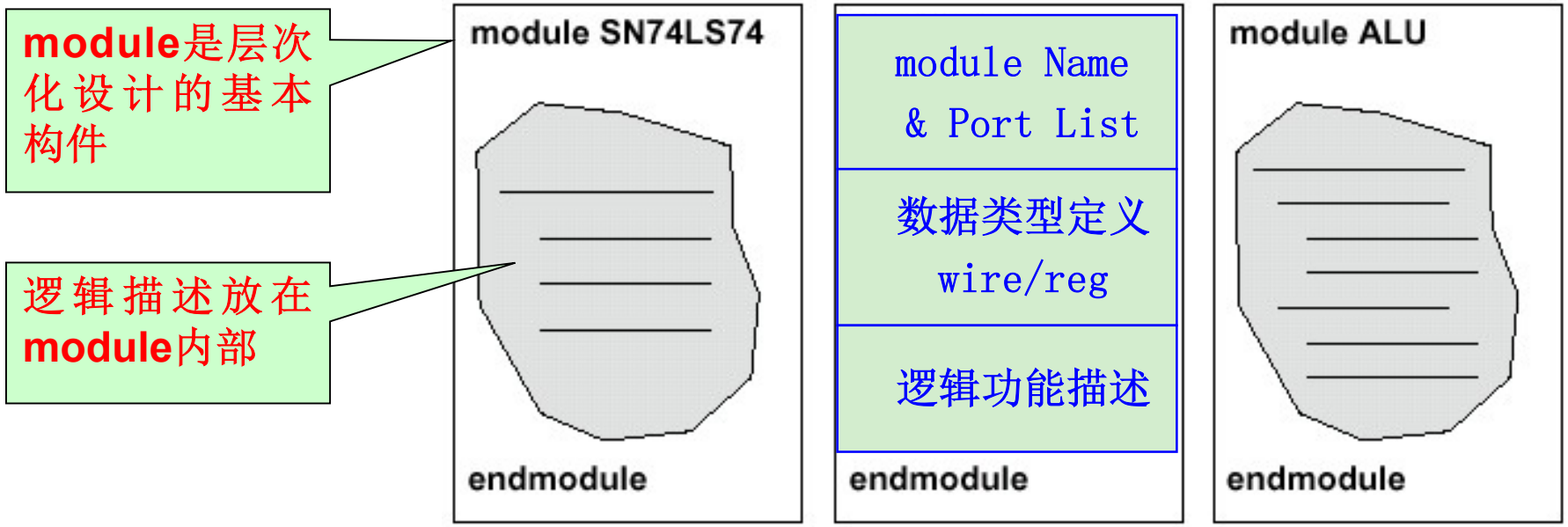
端口等价于硬件的引脚(pin)





Verilog概述 - 初探

- **module模块**
  - 每一个模块的描述从关键词*module*开始，有一个名称（如SN74LS74，DFF，ALU等等），由关键词*endmodule*结束
  - 物理块，如IC或ASIC单元
  - 逻辑块，如一个CPU设计的ALU部分
  - 整个系统



Verilog概述 - 初探

标识符与关键字

- 标识符就是用户为程序中的**Verilog** 对象所起的名字
- **a-z, A-Z, 0-9, _, $**
- 标识符必须以字母或者下横线开头
- 标识符最长可以达到**1023**个字符
- **Verilog**语言是大小写敏感的
- 所有的**Verilog**关键字都是小写

```
module adder(a,b,cin,s,cout);
  input a,b,cin;
  output s,cout;
  …
endmodule
```

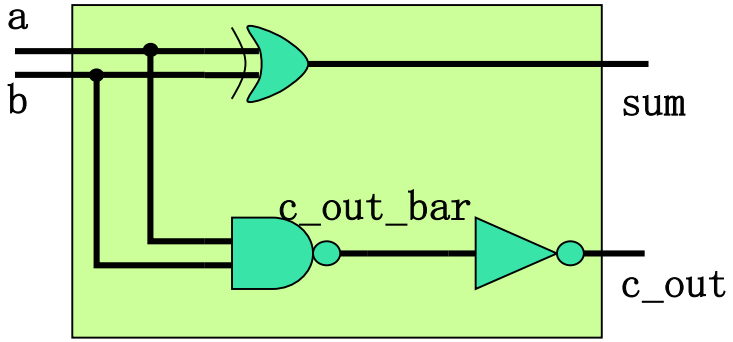
标识符

always	and	assign	begin	buf
bufif0	bufif1	case	casex	casez
cmos	deassign	default	defparam	disable
edge	else	end	endcase	endfunction
endmodule	endprimitive	endspecify	endtable	endtask
event	for	force	forever	fork
function	highz0	highz1	if	initial
inout	input	integer	join	large
macromodule	medium	module	nand	negedge
nmos	nor	not	notif0	notif1
pull0	pull1	pulldown	pullup	rcmos
reg	release	repeat	rnmos	rpmos
rtran	rtranif0	rtranif1	scalared	small
specify	specparam	strong0	strong1	supply0
supply1	table	task	time	tran
tranif0	tranif1	tri	tri0	tri1
triand	trior	vectored	wait	wand
weak0	weak1	while	wire	wor
xnor	xor			

Verilog概述 - 建模方式

- Verilog 对电路建模方式有三类：
  - 结构化描述方式
  - 数据流描述方式
  - 行为描述方式

结构化描述方式

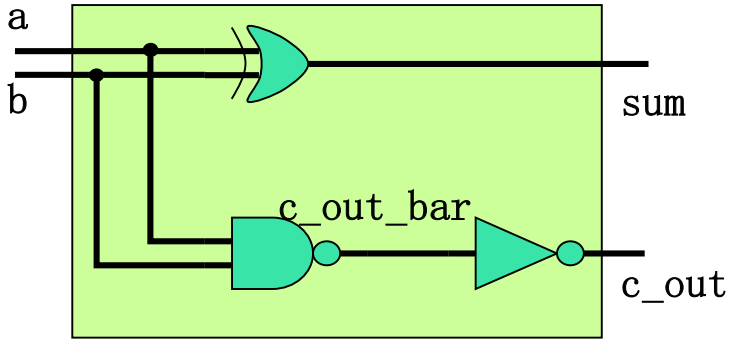



```
module Add_half ( sum, c_out, a, b );
    input       a, b;
    output      sum, c_out;
    wire        c_out_bar;

    xor (sum, a, b);
    nand (c_out_bar, a, b);
    not (c_out, c_out_bar);
endmodule
```

- **wire（线网net）**
  - 表示元件之间的物理连接
  - wire a,b;
  - wire信号的缺省值是z

- **基本门级元件**
  - Verilog中已有一些建立好的逻辑门和开关模型
  - 逻辑门：and、nand、or…
  - 缓冲器与非门：buf、not
  - 三态门
  - MOS开关
  - …

数据流描述方式

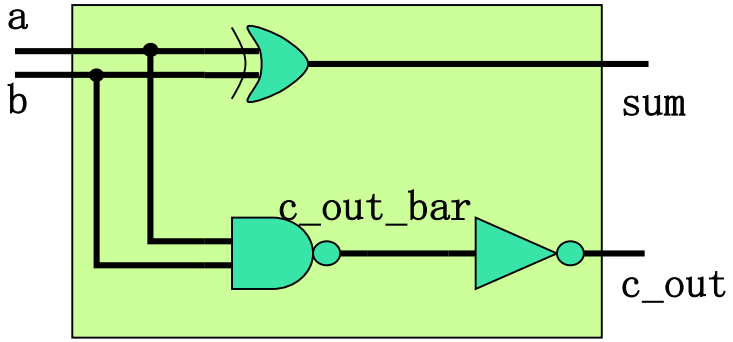



```
module Add_half ( sum, c_out, a, b );
    input       a, b;
    output      sum, c_out;

    assign      sum = a ^ b;
    assign      c_out = a & b;

endmodule
```

- **assign**
  - 持续赋值语句
  - 用来描述组合逻辑电路
- **位运算符**
  - ~、&、|、^、^~
  - 按位操作
  - 两个操作数位数不同时，位数少的高位零扩展

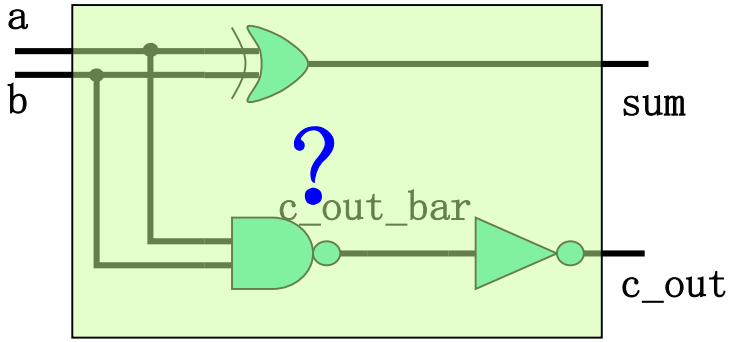
行为描述方式

```
module Add_half ( sum, c_out, a, b );
    input       a, b;
    output      sum, c_out;
    reg         sum, c_out;
    always @ ( a or b ) begin
        sum = a ^ b;
        c_out = a & b;
    end
endmodule
```



- **reg（寄存器register）**
  - 过程块输出信号
  - reg信号的缺省值是x
- **begin end顺序块**
  - 将多条语句组合在一起
  - 块内语句顺序执行，上一条语句执行结束后下一条开始执行
- **wire与reg**
  - assign语句中被赋值的信号定义成wire
  - 过程块中被赋值的信号定义成reg
  - 除了应该定义成reg的都定义成wire
- **always过程块**
  - always块语句重复执行
  - 敏感信号表
  - 多个always块之间是并行执行的

行为描述方式




```
module Add_half ( sum, c_out, a, b );
    input       a, b;
    output      sum, c_out;
    reg         sum, c_out;

    always @ ( a or b ) begin
        {c_out, sum} <= a+b;
    end
endmodule
```

- **{ } 位拼接运算符**
  - 用于位的重组和矢量构造
  - 拼接时不限定操作数的数目。在操作符符号{ }中，用逗号将操作数分开
  - {a,b,c,d}
  - 可以在赋值号左边，也可以右边

1 Verilog 建模方式

2 组合逻辑电路简介

3 常用组合逻辑模块的Verilog描述

4 组合逻辑电路可综合描述的常见问题

5 以加法器为例讲述关键路径的时延问题

6 测试平台搭建

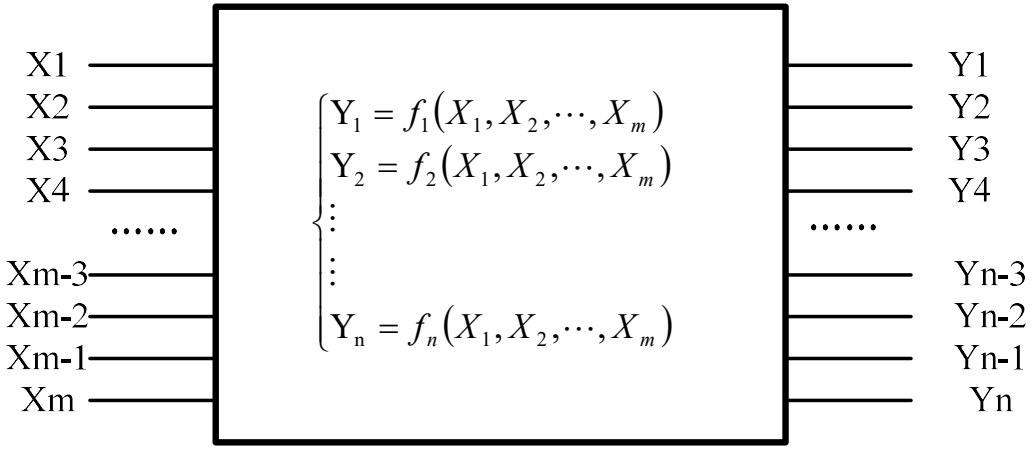
组合逻辑电路概述

- 数字电路按其完成逻辑功能的不同特点，划分为组合逻辑电路和时序逻辑电路两大类
- 复杂数字逻辑由组合逻辑电路和时序逻辑电路构成
- 常用的组合逻辑器件包括编码器、译码器、多路选择器、比较器、加法器等

组合逻辑电路和时序逻辑电路
主要区别是什么?

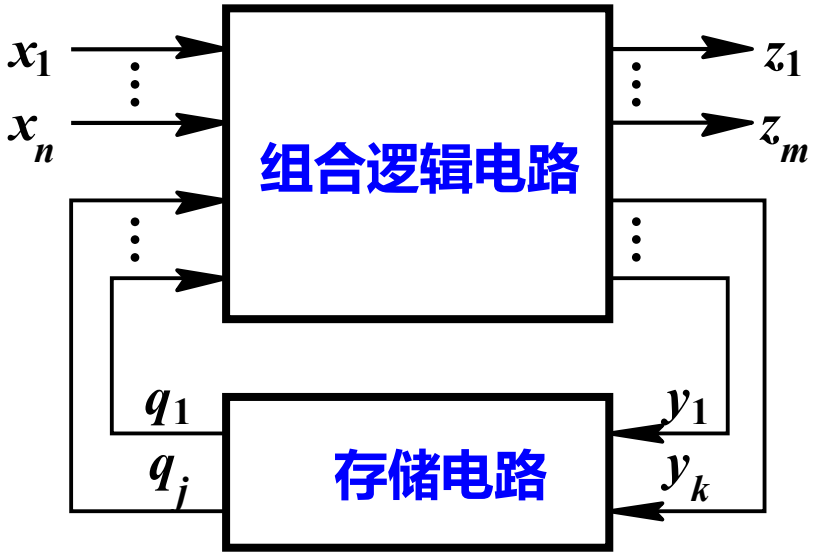
组合逻辑电路

- 从功能上，组合逻辑电路在任一时刻的输出仅和电路当前的输入有关
- 从内部结构上，组合电路都是单纯由逻辑单元组成（**不含存储单元，电路无记忆功能**）



时序电路

- 还**和电路**的内部**状态有关**
- 还**有**寄存器等**存储单元**，电路内有反馈



逻辑关系：

$$\begin{cases} z_m = f_m(x_1, x_2, \cdots, x_n, q_1^n, q_2^n, \cdots, q_j^n) & \text{输出方程} \\ y_k = g_k(x_1, x_2, \cdots, x_n, q_1^n, q_2^n, \cdots, q_j^n) & \text{驱动方程} \\ q_j^{n+1} = h_j(y_1, y_2, \cdots, y_k, q_1^n, q_2^n, \cdots, q_j^n) & \text{状态方程} \end{cases}$$

Verilog如何描述组合逻辑

- module中描述组合逻辑的语句
  - 使用**assign**描述
  - 使用**always**描述

简单逻辑函数

复杂组合逻辑

```
assign q = (al==1?) d : 0 ;

always @(al or d)
   begin
      if (al==1)  q = d ;
      else  q = 0;
   end
```

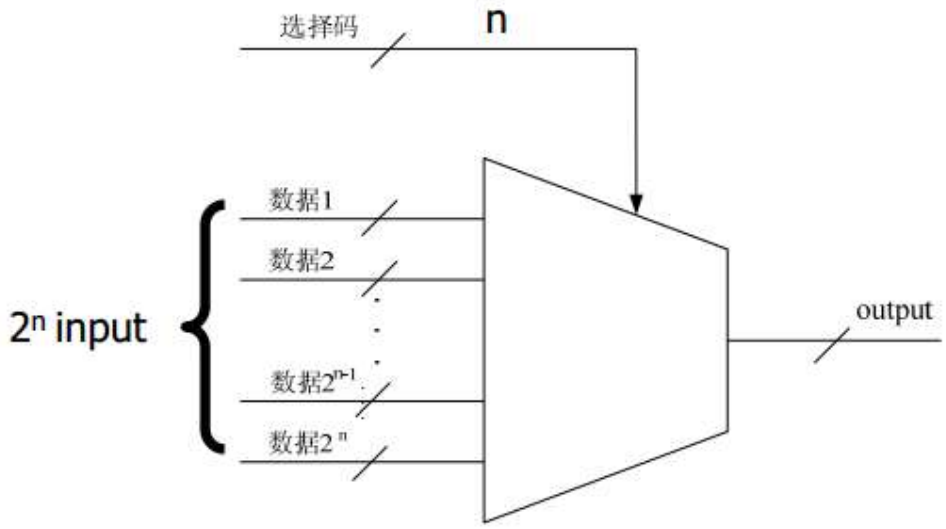
目录

多路选择器（数据选择器）

- 多路选择器是一个多输入单输出的组合逻辑电路
- 根据选择码，从多个输入数据流选取一个输出

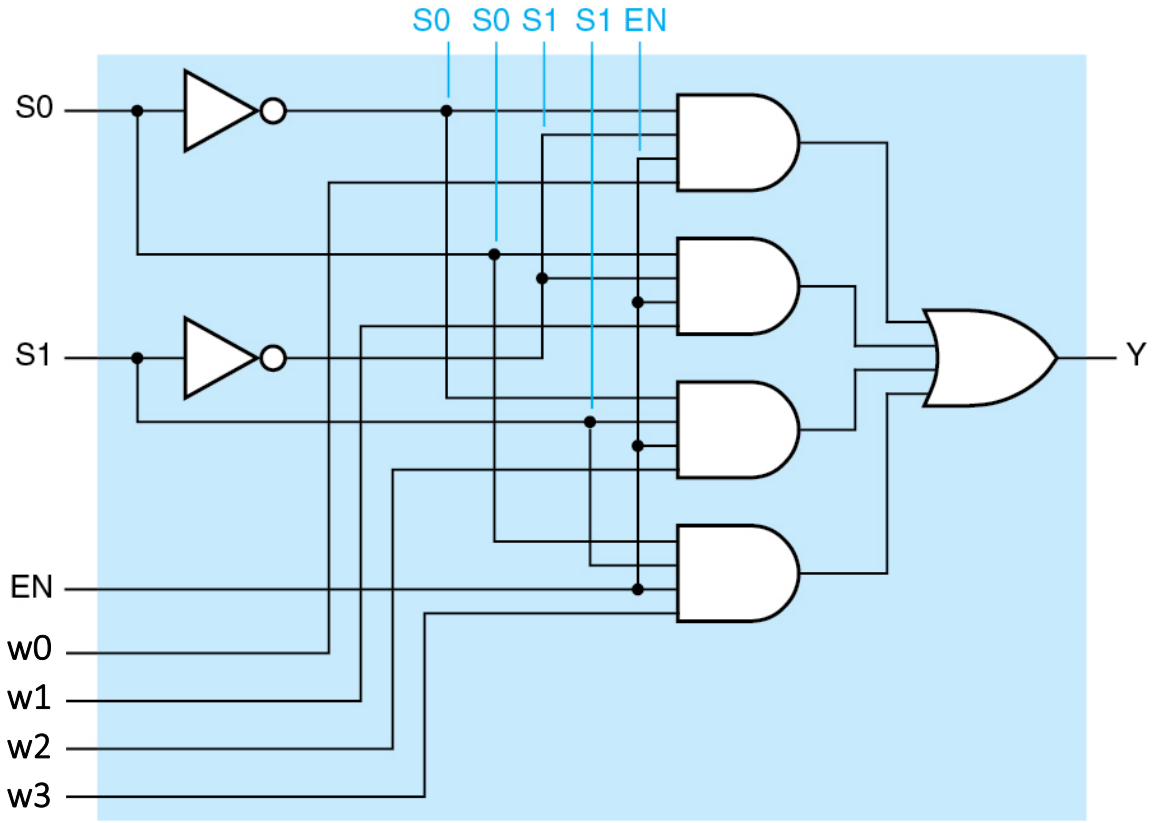
典型Verilog描述语句有以下三种

- assign语句 + 条件操作符
- if...else 及其嵌套语句
- case多分支语句



多路选择器—4选1

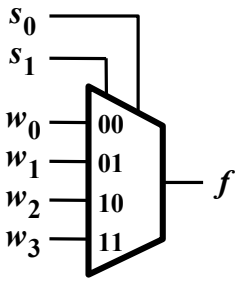
- **电路结构**



多路选择器—4选1

- 数据流描述 - assign

```
module mux4to1 (w0, w1, w2, w3, S, f);
    input w0, w1, w2, w3;
    input [1:0] S;
    output f;
    assign f = S[1] ? (S[0] ? w3 : w2) : (S[0] ? w1 : w0);
endmodule
```



(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

- 多位宽端口
  - 以 [MSB：LSB] 格式
  - 如果 input [0:2] S，则S[0]为高位
- 条件运算符? ：
  - 三目运算符
  - result = condition ? true_value:false_value；

行为级描述 - if else 分支语句

```
module mux4to1 (W, S, f);
    input [0:3] W;
    input [1:0] S;
    output reg f;

    always @(*)
        if (S == 0)
            f = W[0];
        else if (S == 1)
            f = W[1];
        else if (S == 2)
            f = W[2];
        else if (S == 3)
            f = W[3];
endmodule
```

```
module mux4to1 (w0, w1, w2, w3, S, f);
    input w0, w1, w2, w3;
    input [1:0] S;
    output reg f;

    always @(W, S)    //always @(W or S)
        if (S == 2'b00)
            f = w0;
        else if (S == 2'b01)
            f = w1;
        else if (S == 2'b10)
            f = w2;
        else if (S == 2'b11)
            f = w3;
endmodule
```

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

- if 语句
- 常量
- (*)
- 注释
  - //
  - /* … */

if语句

```
if ( 表达式 )
  语句
else
  语句
```

```
if ( 表达式 )
  语句
else if ( 表达式 )
  语句
else
  语句
```

```
if ( a > b )
  res = 1;
else if ( a < c )
  begin
    res = 2;
    out = 4;
  end
else
  res = 3;
```

- **关系运算符**
  - >、<、>=、<=
  - 其结果是1’b1、1’b0或1’bx

- 为确保正确关联，使用begin…end块语句指定其作用域
- 可以多层嵌套。在嵌套if序列中，else和前面最近的if相配对

数的表示方式

```
if (S == 2'b01)
```

X可以用来定义十六进制数的4位二进制状态，八进制数的3位，二进制数的1位。Z的表示方法同X类似。

- <size>‘<base><value>
  - size: 以bit为单位
  - base: b(二进制),o(八进制),d(十进制),h(16进制)
  - value:和进制相应的数值，x, z, ? (x,z不区分大小写)
- 例
  - 16 //默认32位十进制数, 32’d16
  - 8'd16
  - 8'h10
  - 8'b0001_0010 //下划线“_”是为了增加可读性
  - 32‘bx0x1 //32位，高位补x
  - 2'b? //?、z、Z都表示高阻
  - 10'b10 //左边添0补齐, 0000000010
  - 3'b1001_0011 //与3'b011相等
  - 5'h0FFF //与5’h1F相等

行为级描述 - case 分支语句

```
module mux4to1 (W, S, f);
    input [0:3] W;
    input [1:0] S;
    output reg f;

    always @(W, S)
        case (S)
            0: f = W[0];
            1: f = W[1];
            2: f = W[2];
            3: f = W[3];
            default: f = 1'b0;
        endcase
endmodule
```

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

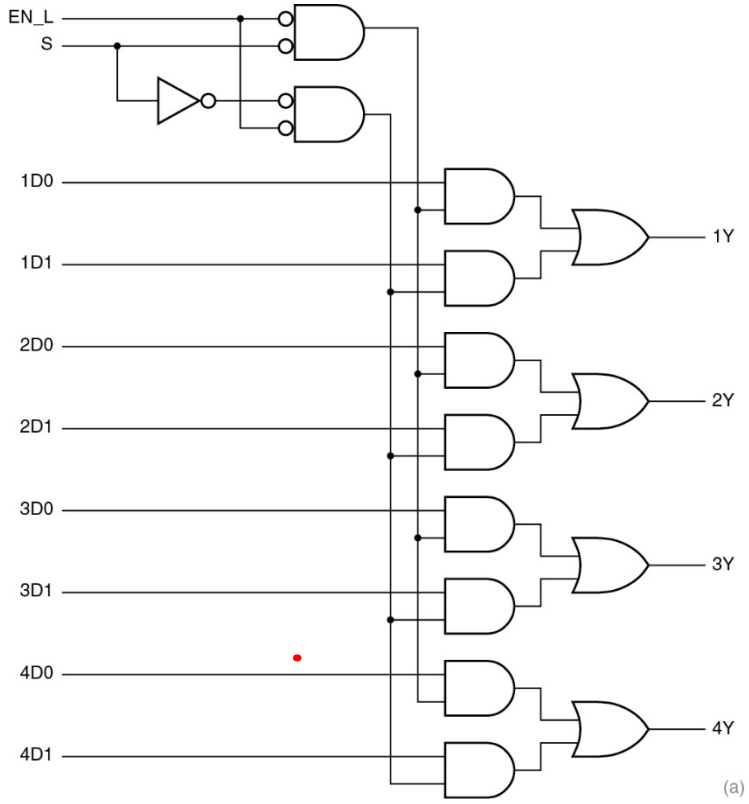
(b) Truth table

case语句

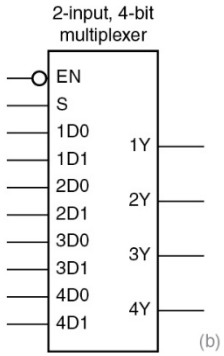
- default语句可选，在没有任何条件成立时执行
- 如果未说明default，Verilog不执行任何动作
- 多个default语句非法

■ 例 - 4输入8位多路选择器

module mux8b_4to1 (W, S, f);



2输入4位



s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

endmodule

例 - 4输入8位多路选择器

```
module mux4to1 (w0, w1, w2, w3, S, f);
    input [7:0] w0, w1, w2, w3;
    input [1:0] S;
    output reg[7:0] f;

    always @(*)
        case (S)
            0: f = w0;
            1: f = w1;
            2: f = w2;
            3: f = w3;
            default: f = 8'b0;
        endcase
endmodule
```

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

- **多位宽变量**
  - 以 [MSB：LSB] 格式
  - reg [3: 0] v;　　// 4位寄存器
  - reg [7: 0] m, n;　// 两个8位寄存器
  - wire [31: 0] databus, addrbus;
  - wire [35: 4] abus, bbus;
  - assign　abus = databus;
  - assign　out[5:2] = v;
  - assign　out[7] = m[1];

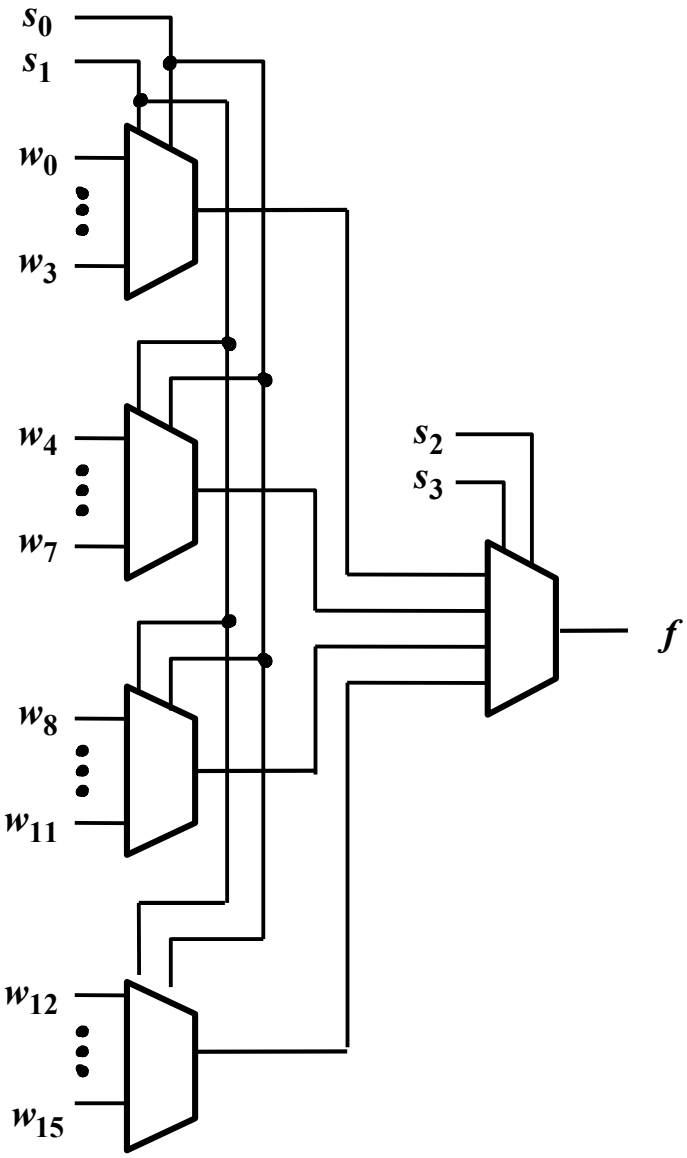
- 课堂小练习：用Verilog描述4输入8位选择器
  - 要求：不使用always语句

```
module mux4to1 (w0, w1, w2, w3, S, f);
      input [7:0] w0, w1, w2, w3;
      input [1:0] S;
      output reg[7:0] f;

                  ?

endmodule
```

- **结构化描述例：用4选1搭建16选1选通器**

```
module mux16to1 (W, S, f);
    input [0:15] W;
    input [3:0] S;
    output f;
    wire [0:3] M;

    mux4to1 Mux1 (W[0:3], S[1:0], M[0]);
    mux4to1 Mux2 (W[4:7], S[1:0], M[1]);
    mux4to1 Mux3 (W[8:11], S[1:0], M[2]);
    mux4to1 Mux4 (W[12:15], S[1:0], M[3]);
    mux4to1 Mux5 (M[0:3], S[3:2], f);
endmodule
```

模块实例化



元件例化

- **格式：**
  - **模块名 <实例名> (<端口列表>);**
- **端口列表有两种表示方式**
  - **端口名字关联：**
    （.端口名（信号名1），.端口名(信号名2),……）
  - **位置关联：隐式给出端口与信号之间的关系**
    （信号名1，信号名2,……）

  例化的端口列表中信号的顺序要与该模块定义的端口列表中端口顺序严格一致
- 模块实例化与调用程序不同。每个实例都是模块的一个完全的拷贝，相互独立、并行

```
AND  u1(a,b,and_out);
AND  u1(.a(a), .b(b), .o(and_out));
```

多路选择器的Verilog编码风格

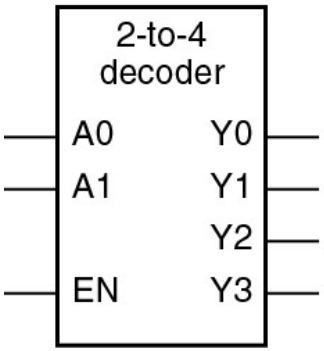
- 二选一建议使用**if else**
- 四选一以上建议使用**case**
- 多选一超过**8**输入时，建议拆分成多个小规模选通器

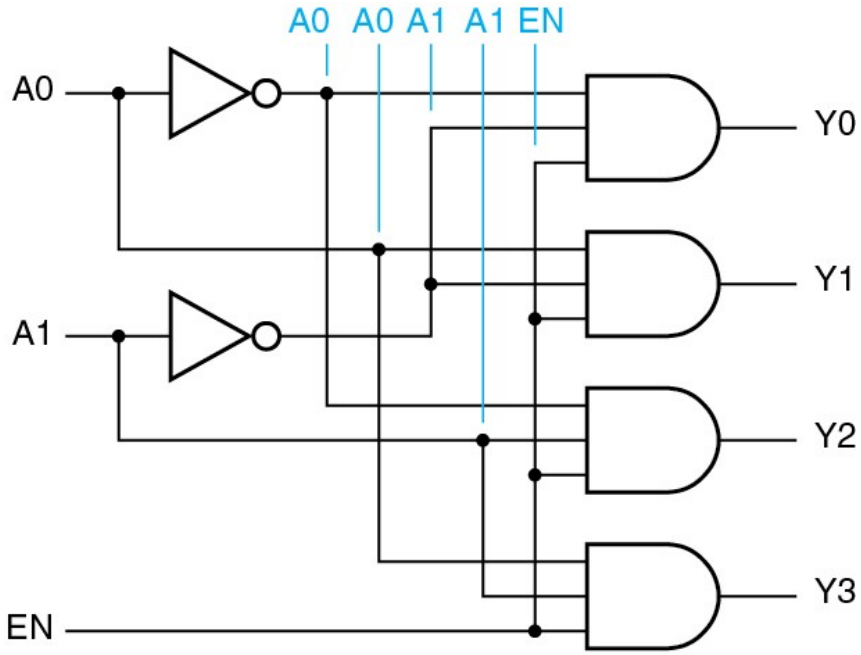
译码器/编码器

- 将一组形式的二进制数据转化为另一种形式的二进制数据
- 编码器将多位输入数据流编码成更短的码流，使得编码器的输出端口减少，具有2^n（或小于）输入位的编码器可提供n位编码输出线
- 译码器将先前编码过的数据解码，是编码器的逆过程，n位的输入可代表2^n种不同的信息

译码器/编码器

- **电路结构**





描述译码器/编码器的语句

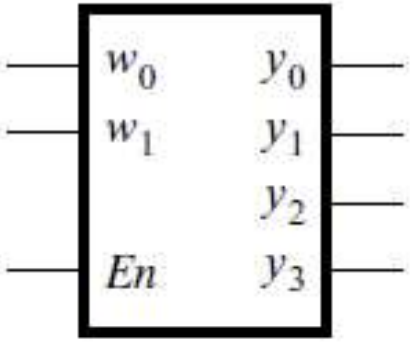
- 通常，采用下列语句描述：
- **If…else**及其嵌套结构
- **case/casex/casez**结构

2-4译码器

```
module dec2to4 (W, En, Y);
    input [1:0] W;
    input En;
    output reg [0:3] Y;
    always @(W, En)
        case ({En, W})
            3'b100: Y = 4'b1000;
            3'b101: Y = 4'b0100;
            3'b110: Y = 4'b0010;
            3'b111: Y = 4'b0001;
            default: Y = 4'b0000;
        endcase
endmodule
```

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



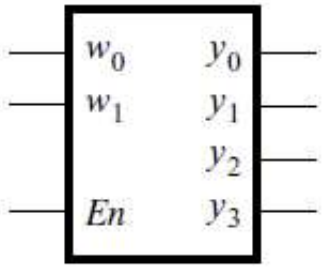
(b) Graphical symbol

2-4译码器的另一种描述方式

```
module dec2to4 (W, En, Y);
    input [1:0] W;
    input En;
    output reg [0:3] Y;

    always @(W, En)
    begin
      if (En == 0)
        Y = 4'b0000;
      else
        case (W)
          0: Y = 4'b1000;
          1: Y = 4'b0100;
          2: Y = 4'b0010;
          default: Y = 4'b0001;
        endcase
    end
endmodule
```

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

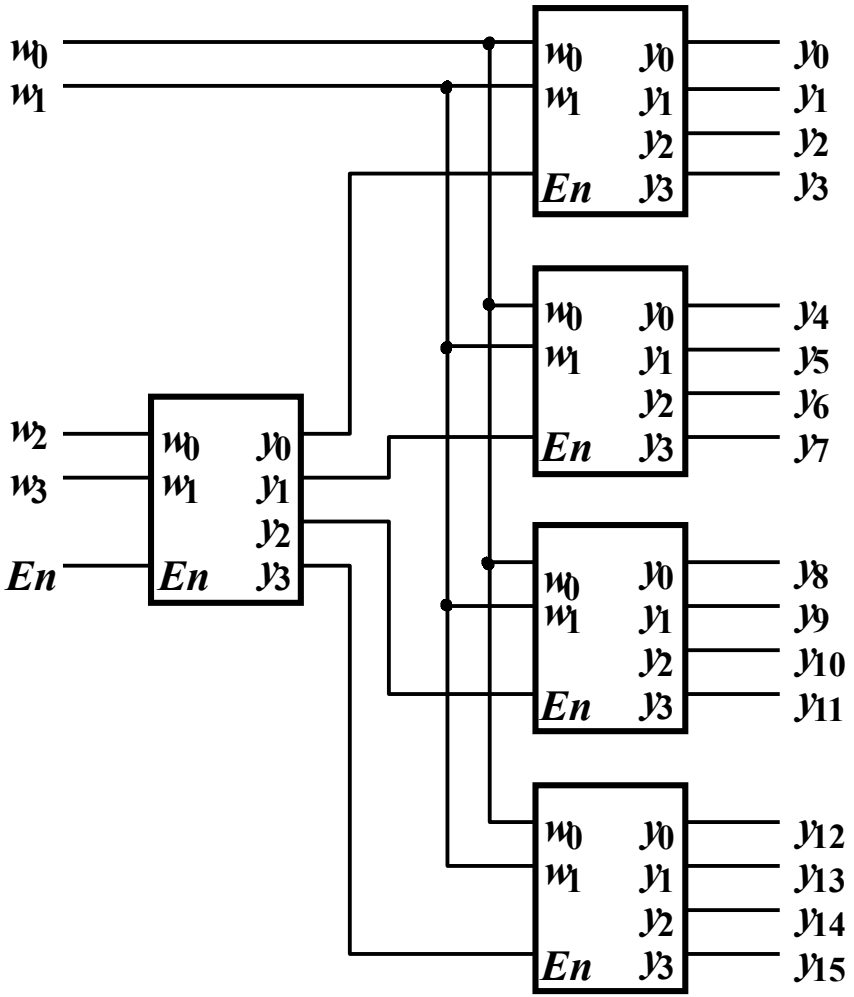
(a) Truth table



(b) Graphical symbol

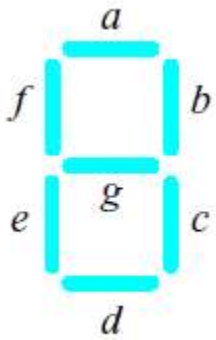
- **相等运算符**
  - 逻辑等：==、!=
  - case等：===、!==

结构化描述实现4-16译码器




```
module dec4to16 (W, En, Y);
      input [3:0] W;
      input En;
      output [0:15] Y;
      wire [0:3] M;

      dec2to4 Dec1 (W[3:2], En, M[0:3]);
      dec2to4 Dec2 (W[1:0], M[0],Y[0:3] );
      dec2to4 Dec3 (W[1:0], M[1],Y[4:7] );
      dec2to4 Dec4 (W[1:0], M[2], Y[8:11]);
      dec2to4 Dec5 (W[1:0], M[3],Y[12:15] );
endmodule
```

代码转化器（显示译码器）



w_3	w_2	w_1	w_0	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
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

```
module seg7 (hex, leds);
   input [3:0] hex;
   output reg [1:7] leds;

   always @(hex)
      case (hex)   //abcdefg
         0: leds = 7'b1111110;
         1: leds = 7'b0110000;
         2: leds = 7'b1101101;
         3: leds = 7'b1111001;
         4: leds = 7'b0110011;
         5: leds = 7'b1011011;
         6: leds = 7'b1011111;
         7: leds = 7'b1110000;
         8: leds = 7'b1111111;
         9: leds = 7'b1111011;
        10: leds = 7'b1110111;
        11: leds = 7'b0011111;
        12: leds = 7'b1001110;
        13: leds = 7'b0111101;
        14: leds = 7'b1001111;
        15: leds = 7'b1000111;
      endcase

endmodule
```

编码器

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

- **逻辑运算符**
  - !、&&、||
  - 逻辑操作符的结果为一位1，0或x

```
always @(W)
    case (W)
        4'b0001: Y = 2'b00;
        4'b0010: Y = 2'b01;
        4'b0100: Y = 2'b10;
        4'b1000: Y = 2'b11;
        default: Y = 2'b00;
    endcase
```

```
always @ (W)
begin
    if (W[3]) Y=2'b11;
    else if (W[2]) Y=2'b10;
    else if (W[1]) Y=2'b01;
    else Y=2'b00;
end
assign z=W[3]||W[2]||W[1]||W[0]
```

编码/译码电路的Verilog编码风格

- 通常使用**case**语句实现编码 / 译码模块
- 优先级编码器可以采用**if**语句实现
- 有时也可以用**for**循环结构实现，初学者不建议！！！

设计举例：简单ALU

Operation	Inputs s_2 s_1 s_0	Outputs F
Clear	0 0 0	0 0 0 0
B−A	0 0 1	$B - A$
A−B	0 1 0	$A - B$
ADD	0 1 1	$A + B$
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1

确定输入信号和输出信号

指令译码、操作数A、B；运算结果

确定输入和输出的逻辑状态关系

用Verilog正确描述电路功能

```
module alu(S, A, B, F); // 74381 ALU
    input [2:0] S;
    input [3:0] A, B;
    output reg [3:0]

    always @(S, A, B
        case (S)
            0: F = 4'b0000
            1: F = B - A;
            2: F = A - B;
            3: F = A + B;
            4: F = A ^ B;
            5: F = A | B;
            6: F = A & B;
            default: F = 4'b1111;
        endcase

endmodule
```

每个操作可能表示一个子电路模块

设计举例：裁判表决器

- 设计一个举重裁判表决器，举重比赛有3个裁判，1个主裁判和2个副裁判，只有当两个或两个以上裁判判明成功，并且其中有主裁判时，表决器判决举重成功

确定输入信号和输出信号

input a,b,c; output y;

确定输入和输出的逻辑状态关系

用Verilog正确描述电路功能

输入			输出
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

```
assign y=a&b+a&c;
```