

通知

- 1、考试时间：11月28号晚上
- 2、本次课的学习内容是必考内容。至少有10分左右的考题。
- 3、本周有第3次、第4次实验。
- 4、作业：
课本上例题4.6.11（P215）、例题6.7.1（P339）、例题6.7.2（P340）、例题6.7.3（P341），在作业本上独立写出对应的代码，然后与课本上的代码进行比较和更正。

4.6 用VerilogHDL描述组合逻辑电路

4.6.1 组合逻辑电路的行为级建模

4.6.2 分模块、分层次的电路设计

'timescale 定义时间单位和精度

在Verilog HDL 模型中，所有时延都用单位时间表述。使用 **'timescale** 编译器指令将时间单位与实际时间相关联。该指令用于定义时延的单位和时延精度。

'timescale编译器指令格式为：

'timescale time_unit / time_precision

time_unit和time_precision由值1、10、和100以及单位s、ms、us、ns、ps和fs组成。

例如： **'timescale 1ns/100ps**

表示**时延单位为1ns**，**时延精度为100ps**。

'timescale 编译器指令在模块说明外部出现，并且影响后面所有的时延值。例如： **'timescale 10ns/1ns**

.....

initial结构

- **initial**结构的主要功能就是进行初始化，是设计者进行信号和变量初始化的时候常用的形式。
- **initial**结构仅在仿真开始的时候被激活一次，然后该结构中的所有语句被执行一次，执行结束后就不再执行。

仅含一条语句时

```
initial a=1;
```

包含多条语句时

```
initial  
begin  
    a=1;  
    b=0;  
end
```

- 用initial生成信号

initial

begin

a=0;b=0;

#15 a=0;b=1;

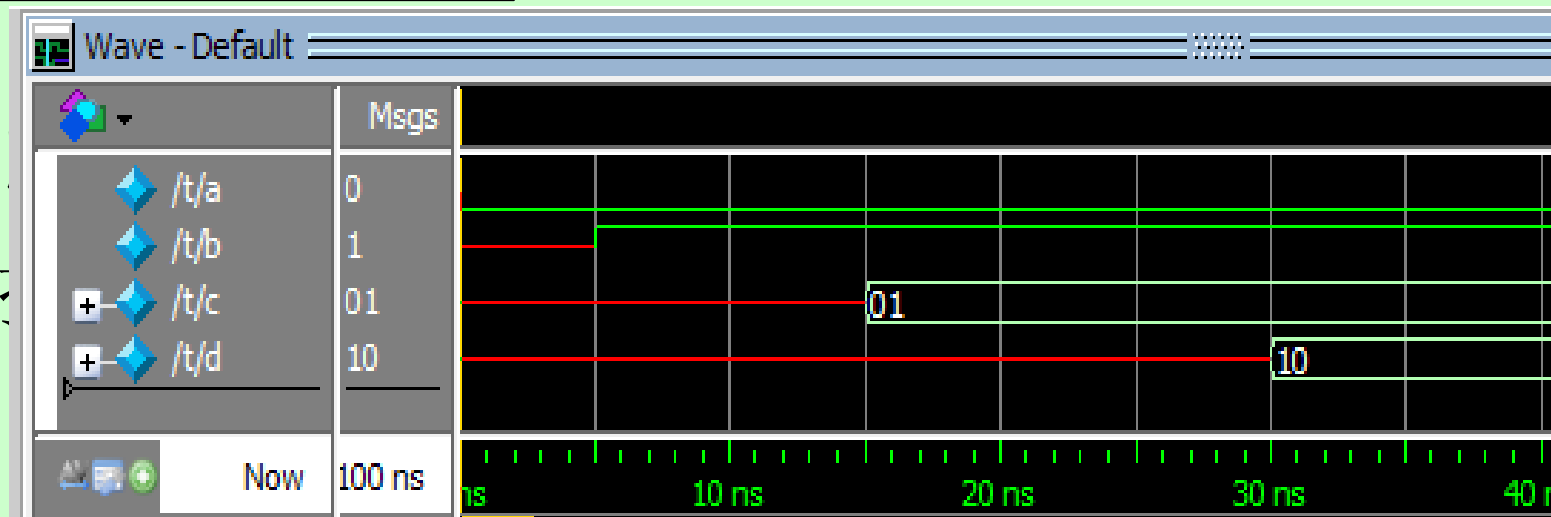
#15 a=1;b=0;

#15 a=1;b=1;

#15 a=0;b=0;

end

顺序块



```
initial
begin
    a=0;
    b=1;
    c={a,b};
    d={b,a};
end
```

```
initial
begin
    a=0;
    #5 b=1;
    #10 c={a,b};
    #15 d={b,a};
end
```

always结构

- always结构在仿真过程中是时刻活动的，它的语句结构如下：

always <时序控制方式> 执行语句

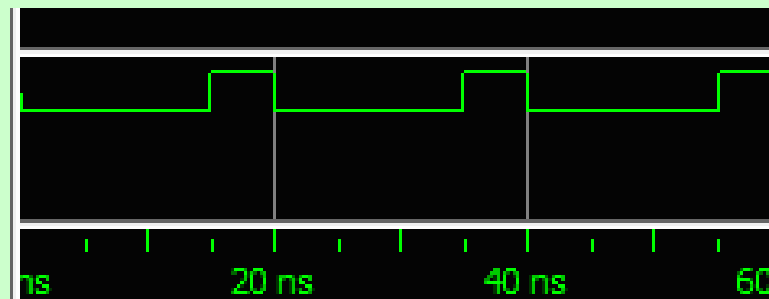
- 如果没有控制方式的参与，此结构中的语句可能会一直执行并发生死锁，或者变成类似数据流级的语句。
- initial clock=0;
always #10 clock=~clock;

时序控制方式

- 基于延迟的控制

```
always #5 a=~a;
```

```
initial clock=0;  
always  
begin  
#15 clock=1;  
#5 clock=0;  
end
```



基于事件的控制

- “@” 引导的事件列表

always @ (敏感事件列表)

- 可以使用or或者 “,”来隔开 多个事件

always @ (a or b)

sum =a+b;

- 只要事件发生就执行always中的语句

always @(A or B or C or D or S1 or S0 or En_)

- 事件过多可用*号

always @(*)

- 以某一个信号的名称作为敏感事件，表示的是对信号的电平值敏感，即信号只要发生了变化，就要执行**always**结构，所有的组合逻辑电路采用的都是这种控制方式
- 时序电路采用的敏感列表一般是边沿敏感的，信号的边沿用**posedge**（上升沿）和**negedge**（下降沿）来表示
always @ (posedge clock)
always @ (posedge clock or negedge reset)

4.6.1 组合逻辑电路的行为级建模（真值表）

组合逻辑电路的行为级描述一般使用assign结构和过程赋值语句、条件语句（if-else）、多路分支语句（case-endcase）和for循环语句等。

在if语句、case语句和for语句的前面，一般都需要添加always语句。在always过程语句中，输出一般都是寄存器reg变量。

电路描述中，assign语句前不能添加always语句。

1、条件语句（if语句）

条件语句就是根据判断条件是否成立，确定下一步的运算。

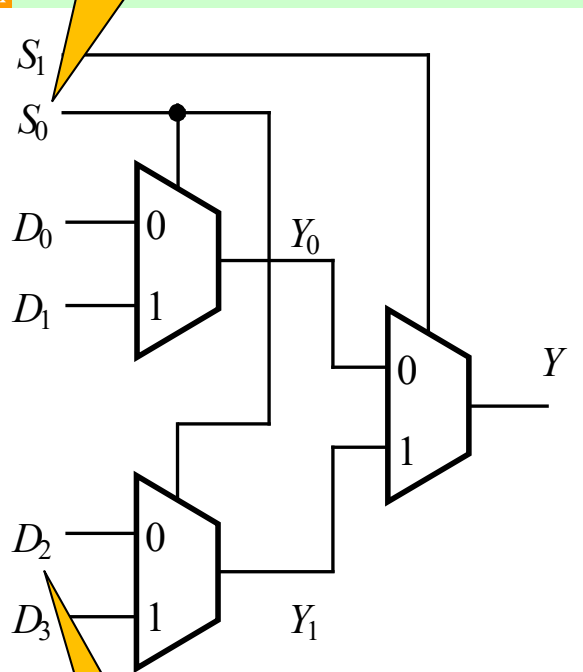
Verilog语言中有3种形式的if语句：

- (1) **if (condition_expr) true_statement; //if语句.....**
- (2) **if (condition_expr) true_statement;**
else false_statement; //if.....else语句
- (3) **if (condition_expr1) true_statement1;**
else if (condition_expr2) true_statement2;
else if (condition_expr3) true_statement3;
.....
else default_statement; //if.....else if..... else语句

if后面的条件表达式一般为逻辑表达式或关系表达式。执行if语句时，首先计算表达式的值，若结果为0、x或z，按“假”处理；若结果为1，按“真”处理，并执行相应的语句。

例：使用if-else语句对4选1数据选择器的行为进行描述

对应于总线S



```
module mux4to1_bh(D, S, Y);
    input [3:0] D; //输入端口
    input [1:0] S; //输入端
    output reg Y; //输出端
```

```
    always @(D, S) //电路功能描述
    if (S == 2'b00) Y = D[0];
    else if (S == 2'b01) Y = D[1];
    else if (S == 2'b10) Y = D[2];
    else Y = D[3];
```

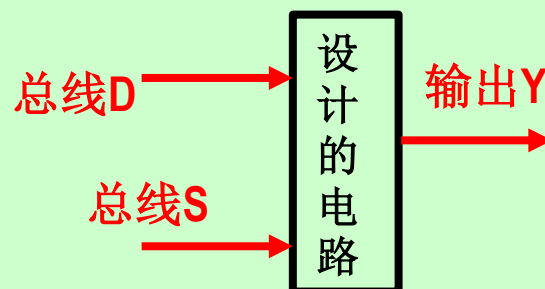
```
endmodule
```

对应于总线S

选择输入		输出
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

注意：过程赋值语句只能给寄存器型变量赋值，因此，输出变量Y的数据类型定义为reg。

对应于总线D



2、多路分支语句（case语句）

是一种多分支条件选择语句，一般形式如下

```
case (case_expr)
```

```
    item_expr1: statement1;
```

```
    item_expr2: statement2;
```

```
    .....
```

```
    default: default_statement; //default语句可以省略
```

```
endcase
```

注意：当分支项中的语句是**多条语句**，必须在最前面写上关键词**begin**，在最后写上关键词**end**，成为**顺序语句块**。

另外，用关键词**casex**和**casez**表示含有**无关项x**和**高阻z**的情况。

例：对具有使能端 E_n 的4选1数据选择器的行为进行Verilog描述。
 当 $E_n=0$ 时，数据选择器工作， $E_n=1$ 时，禁止工作，输出为0。

```
Module m4d1 (S, Y, En);
```

```
input [3:0] S;
```

```
input En;
```

对应于总线S

选择输入		输 出
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

//2005 syntax, 过程语

$E_n=1$ 时，输出为0
 选择器工作

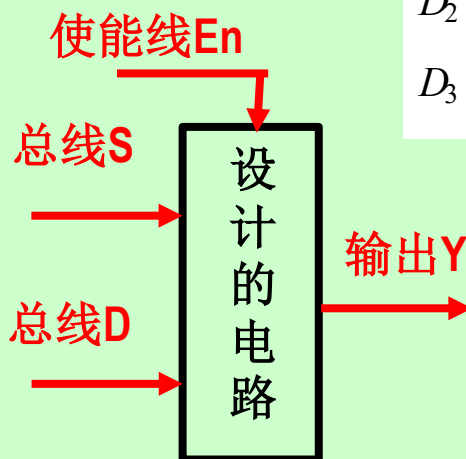
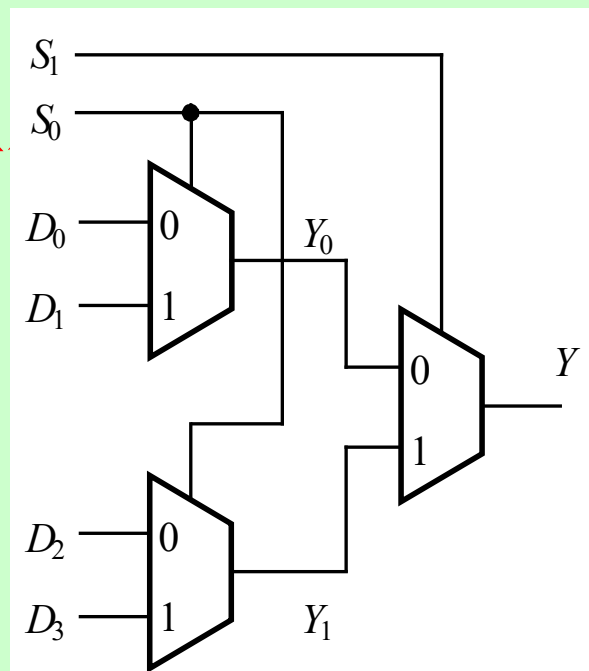
```
2'd2: Y = D[0];
```

```
2'd3: Y = D[1];
```

对应于总线D

```
end
```

```
endmodule
```



例：对基本的4线-2线优先编码器的行

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

```
  always @(W)           //过程语句
  casex (W)             //有无无关项时用casex ( ) 语句
```

```
    4'b1xxx: Y = 3;
    4'b01xx: Y = 2;
    4'b001x: Y = 1;
    4'b0001: Y = 0;
```

```
  default: Y=1'bz;
  //W无效时, Y为高阻
  endcase
endmodule
```

输 入				输 出	
W_3	W_2	W_1	W_0	Y_1	Y_0
1	x	x	x	1	1
0	1	x	x	1	0
0	0	1	x	0	1
0		0	1	0	0

总线W

总线Y

对应于总线W

对应于总线Y

总线W

总线Y

设计的电路

3、for循环语句

一般形式如下

```
for (initial_assignment; condition; step_assignment)
```

```
begin
```

```
statement;
```

```
end
```

initial_assignment为循环变量的**初始值**。

Condition为**循环的条件**，若为真，执行过程赋值语句
statement，若不成立，循环结束，执行for后面的语句。

step_assignment为**循环变量的步长**，每次迭代后，循环
变量将增加或减少一个步长。

试用Verilog语言

A ₂ A ₁ A ₀	Y[0]	Y[1]	Y[A]	Y[5]	Y[6]	Y[7]
A	1	1	1	0	1	1	1	1

```
module ecoder3to8_bh(A,En,Y);
```

```
    input [2:0] A;
```

```
    input En ;
```

```
    output reg [7:0] Y;
```

```
    integer k; //声明一个整型变量
```

```
    always @(A, En) //声明过程的语句
```

```
    begin
```

```
        Y = 8'b1111_1111;
```

```
        for(k = 0; k <= 7; k++)
```

```
        { if ((En==1) &&
```

```
            Y[k] = 0; //当使能有效且
```

```
            else
```

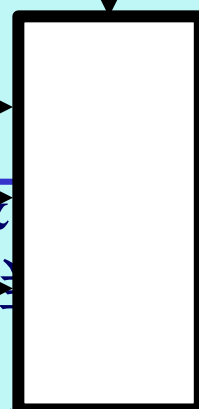
```
            Y[k] = 1; //处理使能无效或输入无效的情况
```

```
        end
```

```
    endmodule
```

0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

En



译码器输出的默认值

使能En

//下面的if-else语句



循环8次

总线Y

总线Y

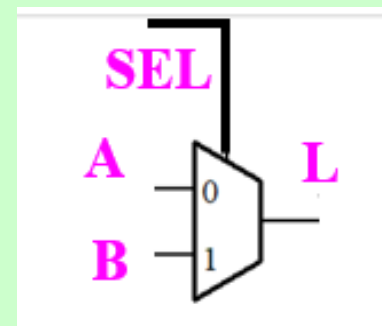
时, 根据A进行译码

总线A

循环8次

例：用条件运算符描述了一个2选1的数据选择器。

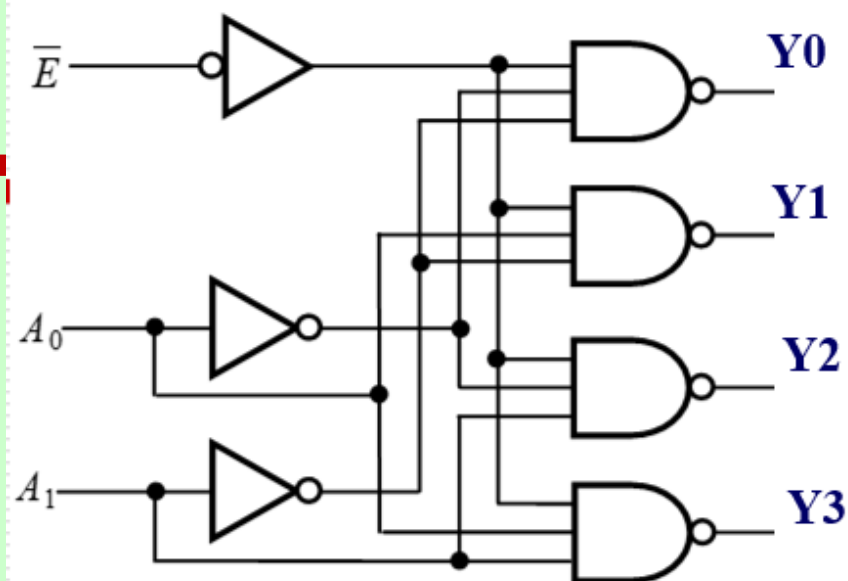
```
module mux2x1_df (A,B,SEL,L);  
    input A, B, SEL;  
    output wire L;  
    assign L = SEL ? B:A;  
endmodule
```



在连续赋值语句中，如果SEL=1，则输出L=B；否则L=A。

```
module mux2x1_df (A,B,SEL,L);  
    input A,B,SEL;  
    output reg L;  
    always @(B,A,SEL) //用always语句和条件运算符建模  
        L = SEL ? B : A;  
endmodule
```

例：用数据流建模方法对2线-4线译码器的行为进行描述。



```
module decoder_df (A1,A0,E,Y);
  input A1,A0,E;
  output [3:0] Y;
```

```
    assign Y[0] = ~(~A1 & ~A0 & ~E);
```

```
    assign Y[1] = ~(~A1 & A0 & ~E);
```

```
    assign Y[2] = ~(A1 & ~A0 & ~E);
```

```
    assign Y[3] = ~(A1 & A0 & ~E);
```

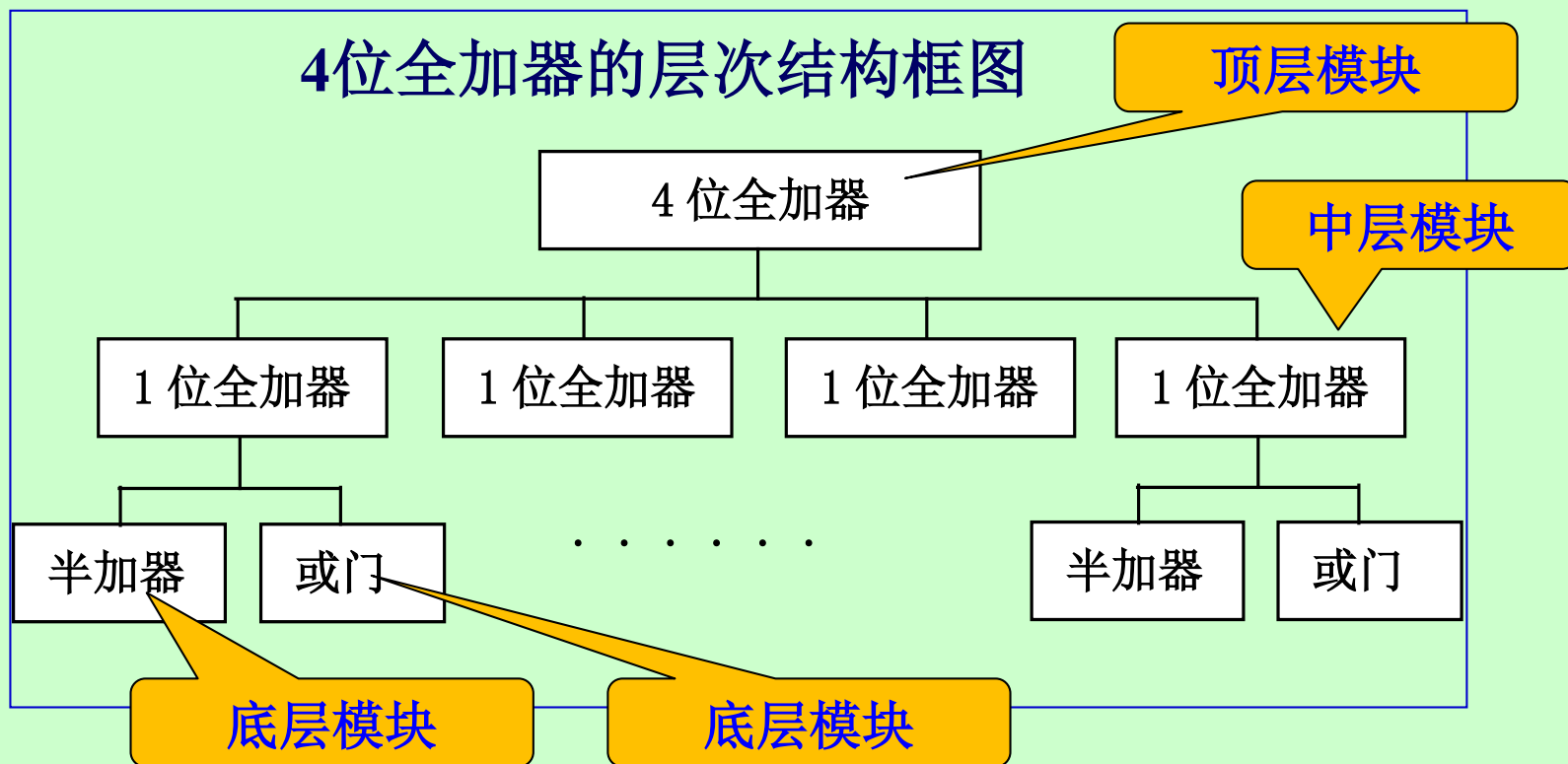
```
endmodule
```

行为级描述中，assign语句前不需要添加always语句。

4.6.2 分模块、分层次的电路设计

分层次的电路设计:在电路设计中,将两个或多个模块组合起来描述电路逻辑功能的设计方法。

设计方法: 自顶向下和自底向上两种常用的设计方法



底层模块

```
module halfadder (S,C,A,B);
```

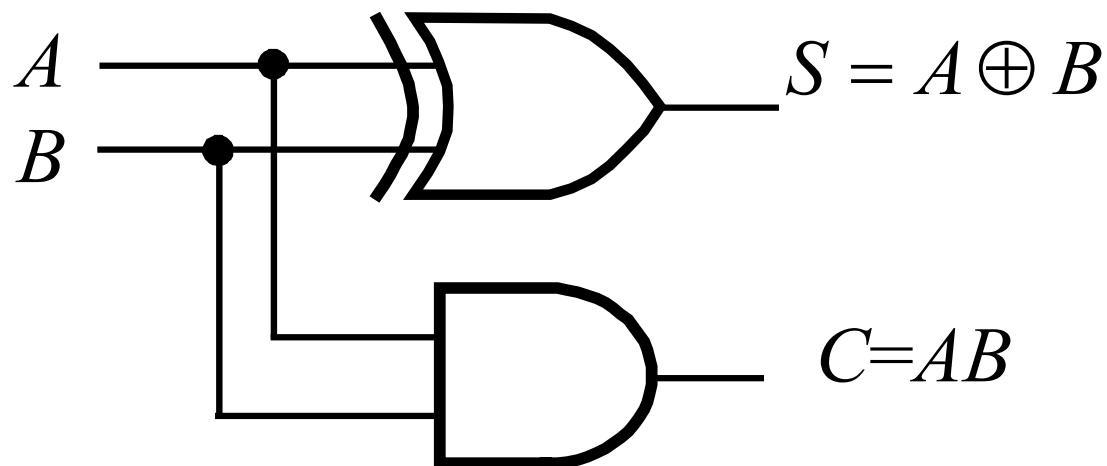
```
  input A,B;
```

```
  output S,C;
```

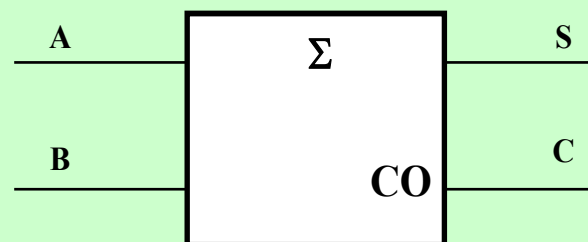
```
  xor (S,A,B);
```

```
  and (C,A,B);
```

```
endmodule
```



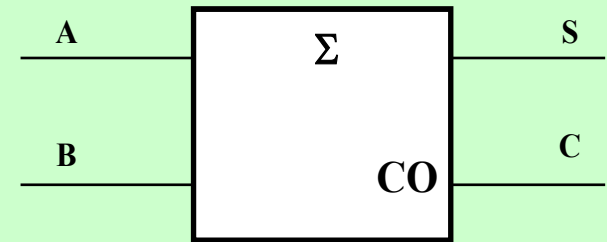
} 半加器的门级描述



中层模块

```

module fulladder (S,CO,A,B,CI);
    input A,B,CI;
    output S,CO;
    wire S1,D1,D2; //内部节点信号
    halfadder HA1 (S1,D1,A,B);
    halfadder HA2 (S,D2,S1,CI);
    or g1(CO,D2,D1);
endmodule
    
```



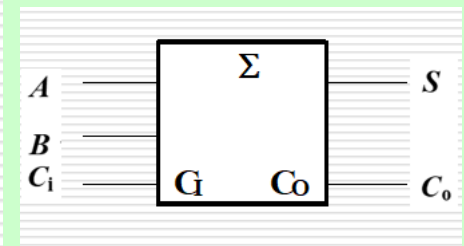
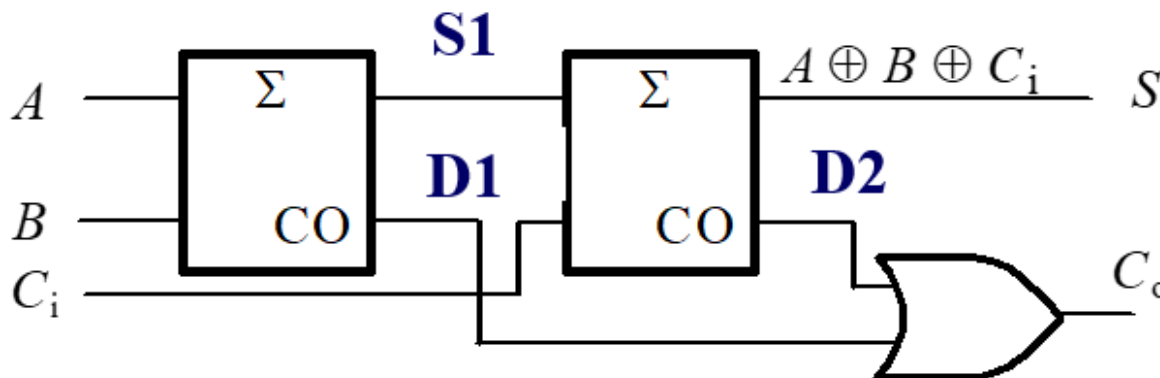
halfadder (S,C,A,B);

全加器的描述-调用半加器

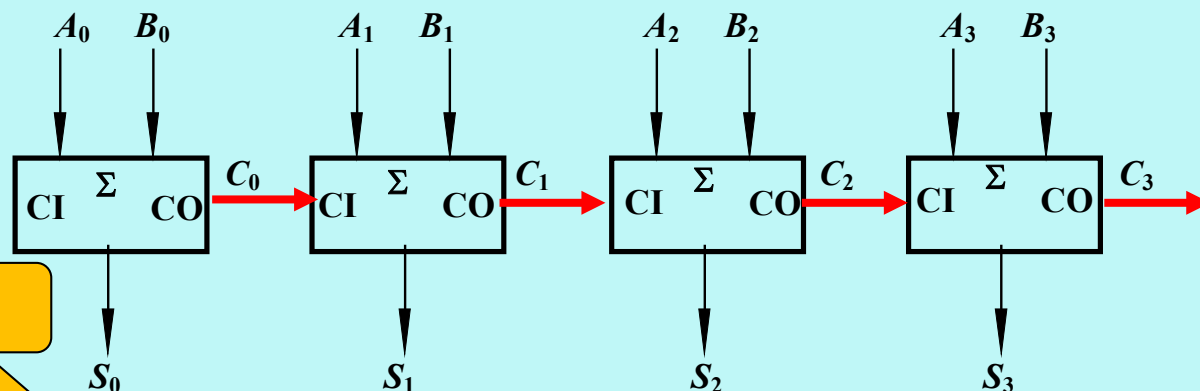
底层模块

```

halfadder HA1 (. S(S1), . C(D1), . A(A), .B(B));
halfadder HA2 (. S(S), . C(D2), . A(S1), .B(CI));
    
```



顶层模块



```
module _4bit_adder (S,C3,A,B,C_1);
```

```
  input [3:0] A,B;
```

```
  input C_1;
```

```
  output [3:0] S;
```

```
  output C3;
```

```
  wire C0,C1,C2; //内部进位信号
```

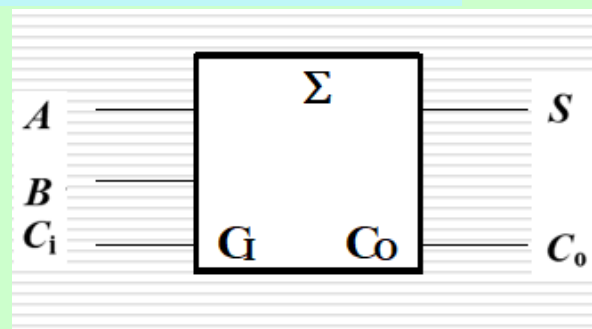
```
  fulladder FA0 (S[0],C0,A[0],B[0],C_1),
```

```
                FA1 (S[1],C1,A[1],B[1],C0),
```

```
                FA2 (S[2],C2,A[2],B[2],C1),
```

```
                FA3 (S[3],C3,A[3],B[3],C2);
```

```
endmodule
```



```
fulladder (S,CO,A,B,CI);
```

4位全加器的描述
--调用1位全加器

```
fulladder FA0 (. S(S[0]),.CO(C0),.A(A[0]),.B(B[0]),.CI(C_1));
```

```
fulladder FA1 (. S(S[1]),.CO(C1),.A(A[1]),.B(B[1]),.CI(C0));
```

```
fulladder FA2 (. S(S[2]),.CO(C2),.A(A[2]),.B(B[2]),.CI(C1));
```

```
fulladder FA3 (. S(S[3]),.CO(C3),.A(A[3]),.B(B[3]),.CI(C2));
```


5.6 用Verilog HDL描述锁存器和触发器

敏感事件分为电平敏感事件和边沿触发事件：

电平敏感事件（如锁存器）：

always@(sel or a or b)

sel、a、b中任意一个电平发生变化，后面的过程赋值语句将执行一次。

边沿敏感事件（如触发器）：

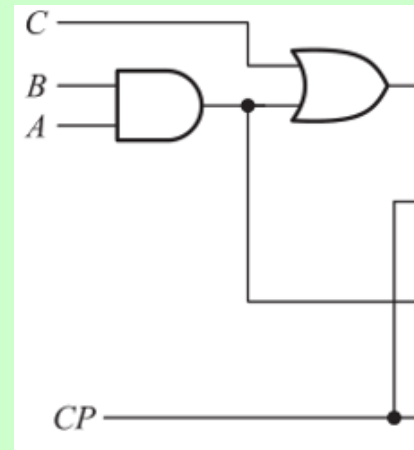
always@(posedge CP or negedge CR)

CP的上升沿或CR的下降沿来到，后面的过程语句就会执行。

过程赋值语句有阻塞型和非阻塞型：

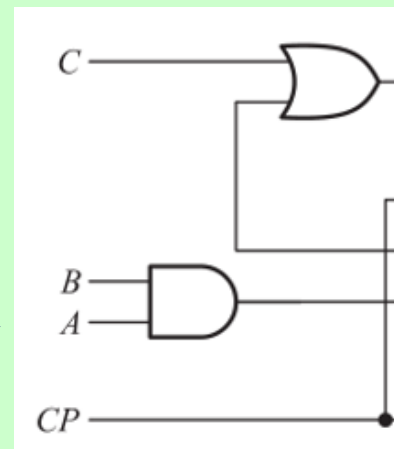
阻塞型用“=”表示，多条语句顺序执行。

```
always@(posedge CP)
begin
    f = A & B;
    g = f | C;
end
```



非阻塞型用“<=”表示，语句块内部的语句并行执行。

```
always@(posedge CP)
begin
    f <= A & B;
    g <= f | C;    // g用CP上升沿前的f值
end
```



5.6.2 锁存器和触发器的Verilog建模实例

```
module D_latch (Q, D, E); //D锁存器的描述
    output Q;
    input D, E;
    reg Q;
    always @(E or D)
        if (E) Q <= D; //Same as: if (E== 1)
endmodule
```

```
module DFF (
    output reg Q,
    input D, CP
); //D触发器的描述
    always @(posedge CP)
        Q <= D;
endmodule
```

例：分析下面Verilog模块, 说明其逻辑功能。

```
module async_set_rst_DFF (  
    output reg Q, QN,  
    input D, CP, Sd, Rd  
);
```

```
always @(posedge CP, negedge Sd, negedge Rd)
```

```
begin
```

```
    if (~Sd || ~Rd)
```

```
        if (~Sd ) begin Q <= 1'b1; QN <= 1'b0; end//异步置数
```

```
        else      begin Q <= 1'b0; QN <= 1'b1; end//异步清零
```

```
    else
```

```
        begin Q <= D; QN <= ~D; end
```

```
end
```

```
endmodule
```

6.7 用Verilog HDL描述时序逻辑电路

6.7.1 移位寄存器的Verilog建模

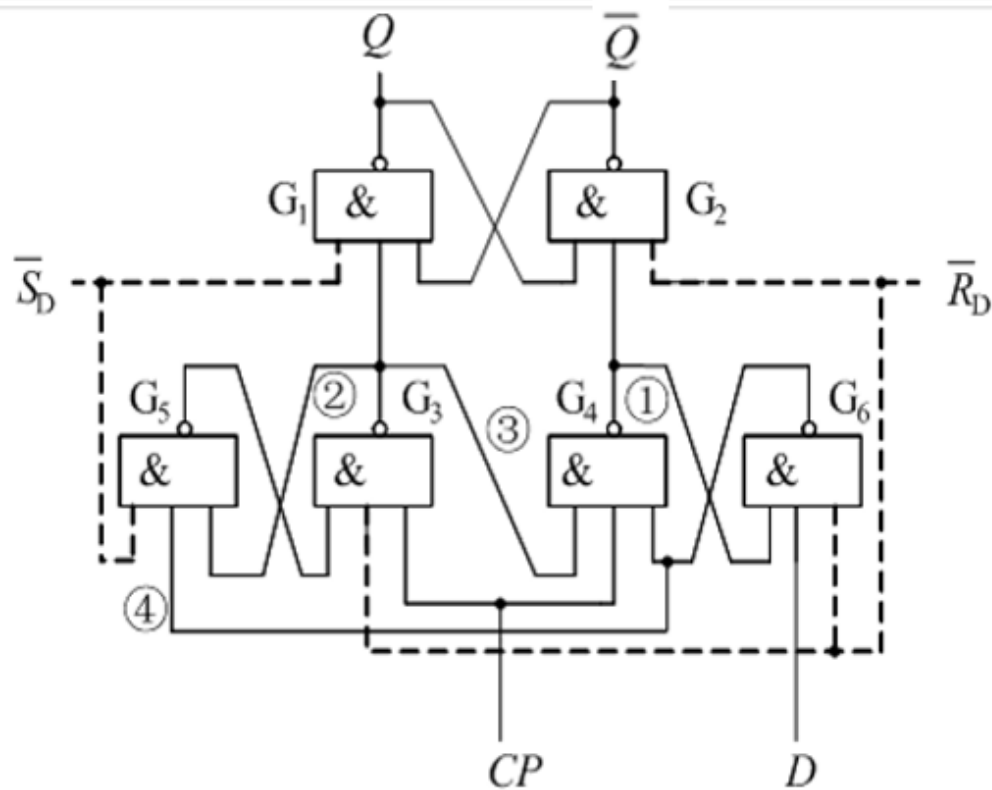
6.7.2 计数器的Verilog建模

6.7.3 状态图的Verilog建模

6.7.4 数字钟的Verilog建模

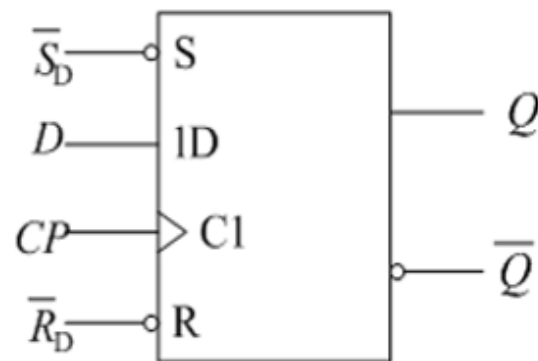
清零端和置1端引脚

维持-阻塞式D触发器



(a) 电路结构

图

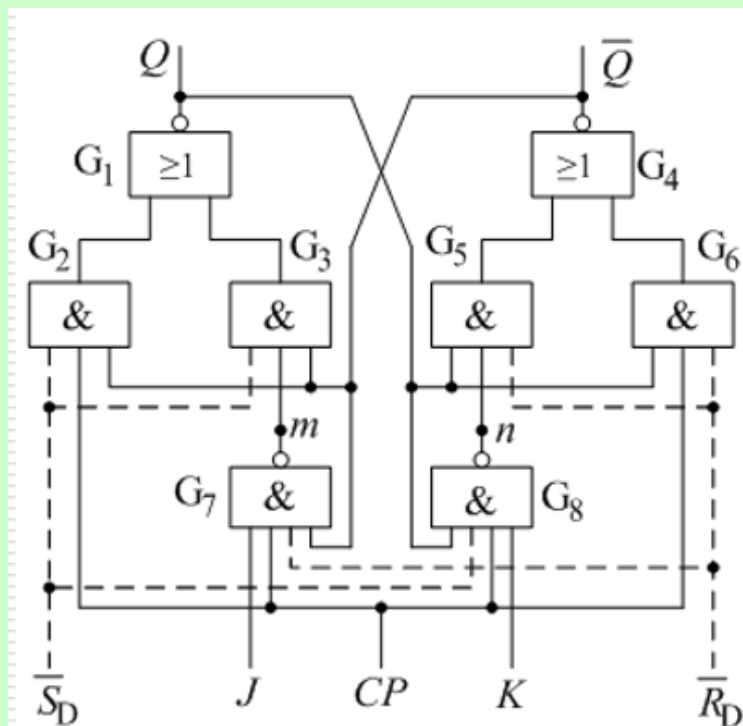


(b) 图形符号

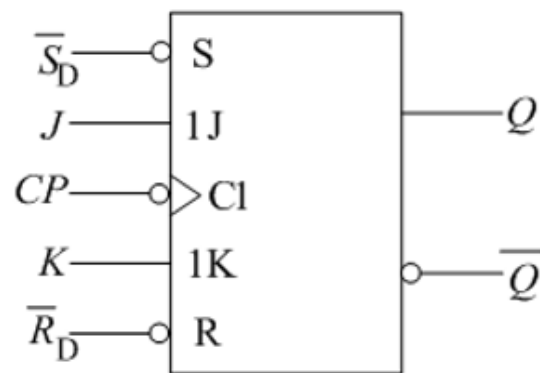
维持-阻塞式 D 触发器

清零端和置1端引脚

下降沿触发JK触发器



(a) 电路结构



(b) 图形符号

图 边沿 JK 触发器

6.7.1 移位寄存器的Verilog建模

用行为级描述always描述一个4位双向移位寄存器，有异步清零、同步置数、左移、右移和保持。功能同74HC194。

```
module shift74x194 (S1, S0, D, Dsl, Dsr, Q, CP, CR);  
    input S1, S0;           //控制输入  
    input Dsl, Dsr;         //串行输入  
    input CP, CR;           //时钟及清零  
    input [3:0] D;          //并行输入  
    output [3:0] Q;         //寄存器输出  
    reg [3:0] Q;
```

6.7.1 移位寄存器的Verilog建模

always @ (posedge CP or negedge CR)

if (~CR) Q <= 4'b0000;

else

case ({S1,S0})

2'b00: Q <= Q;

//保持

2'b01: Q <= {Q[2:0],Dsr};

//右移

2'b10: Q <= {Dsl,Q[3:1]};

//左移

2'b11: Q <= D;

//并行输入

endcase

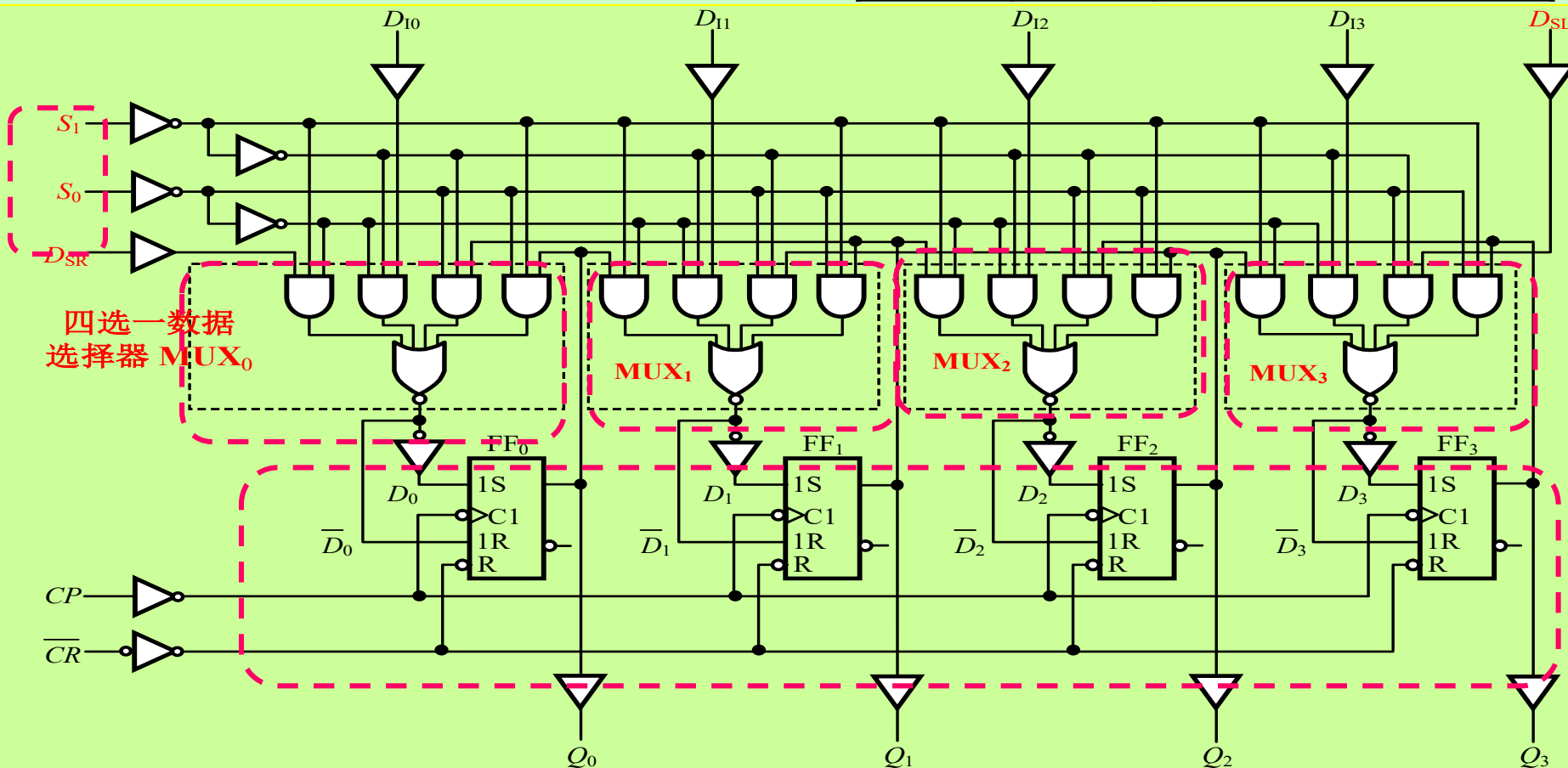
endmodule

(b) 典型集成电路

CMOS 4位双向移位寄存器74HC/HCT194

74HCT194 的功能表

S1	S0	功能
0	0	保持
0	1	低位往高位移动
1	0	高位往低位移动
1	1	并行置入



6.7.2 计数器的Verilog建模实例

用Verilog描述具有使能端、异步置零、同步置数、计数、保持的16进制计数器,类似74HC161芯片。

```
module counter74x161_beh ( //Verilog 2001, 2005 syntax
    input CEP, CET, PE, CP, CR, //输入端口声明
    input [3:0] D,                //并行数据输入
    output TC,                    //进位输出
    output reg [3:0] Q            //数据输出端口及变量的数据类型声明
);
    wire CE;                      //中间变量声明
```

6.7.2 计数器的Verilog建模实例

```
assign CE=CEP&CET; //CE=1时，计数器计数
```

```
assign TC=CET&PE&(Q == 4'b1111); //产生进位输出信号
```

```
always @(posedge CP, negedge CR) //Verilog 2001, 2005 syntax
```

```
if (~CR) Q<=4'b0000; //实现异步清零功能
```

```
else if (~PE) Q<=D; //PE=0, 同步装入输入数据
```

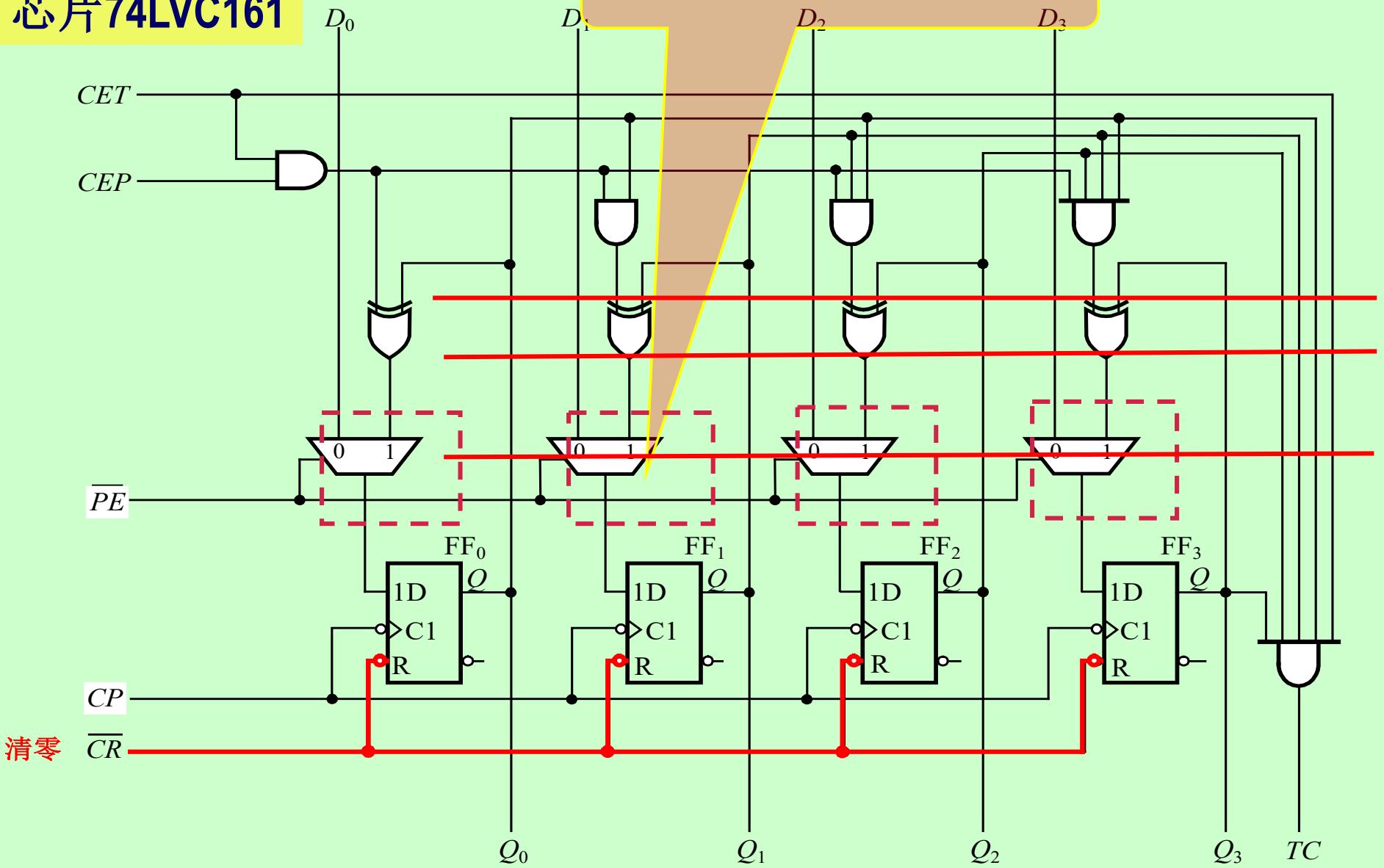
```
else if (CE) Q<=Q+1'b1; //加1计数
```

```
else Q<=Q; //输出保持不变
```

```
endmodule
```

芯片74LVC161

2选1数据选择器



完

了解

//带参数的4位加法器子模块

```
module adder #(parameter n=4 )
```

```
(
```

```
    input [n-1:0] A,B,
```

```
    input Ci,
```

```
    output [n-1:0] S,
```

```
    output Co
```

```
);
```

```
assign {Co,S} = A + B + Ci;
```

```
endmodule
```



4位全加器的描述

了解

//按名称关联端口及参数

```
module adder_hier (  
    input [7:0] A1,B1,  
    input Cin1,  
    output [7:0] Sum1,  
    output Cout1  
);  
adder #( .n(8)) U1  
(  
    .A(A1),  
    .B(B1),  
    .Ci(Cin1),  
    .S( Sum1),  
    .Co( Cout1)  
);
```

8位全加器