数 字 逻 辑

Digital Logic Circuit

丁 贤 庆

ahhfdxq@163.com

Home work（P350）

- 1、本周有实验。地点：电气实验楼505房间
- 2、期末考试里，第六章有30分左右的考题。
- 3、本周三（四）课前交作业，交完作业才正式上课。
- 4 、本周三（四）课上，会有10分钟的课堂小测。
- 5、本次的作业：　　无

第16周和第17周，新的实验时间表。（表中黑色字体带删除线的，是旧的实验时间。）

		周次:	第16周					
		星期一	星期二	星期三	星期四	星期五	星期六	星期日
上午	8:00~9:50	计科2班				物联网1班		
	10:10~12:00		物联网2班	计科5班				
下午	2:00~3:50					计科1班		
	4:00~5:50			~~计科4班~~				
晚上	7:00~8:50（默认）	计科1班	计科3班	~~计科5班~~	计科4班			~~物联网1班~~

计算机学院

数字逻辑电路

						周次:	第17周	
		星期一	星期二	星期三	星期四	星期五	星期六	星期日
上午	8:00~9:50	~~计科2班~~		计科4班				
	10:10~12:00			计科2班				
下午	2:00~3:50							
	4:00~5:50			~~计科4班~~				
晚上	7:00~8:50 (默认)	~~计科1班~~	计科3班	计科5班	物联网2班			

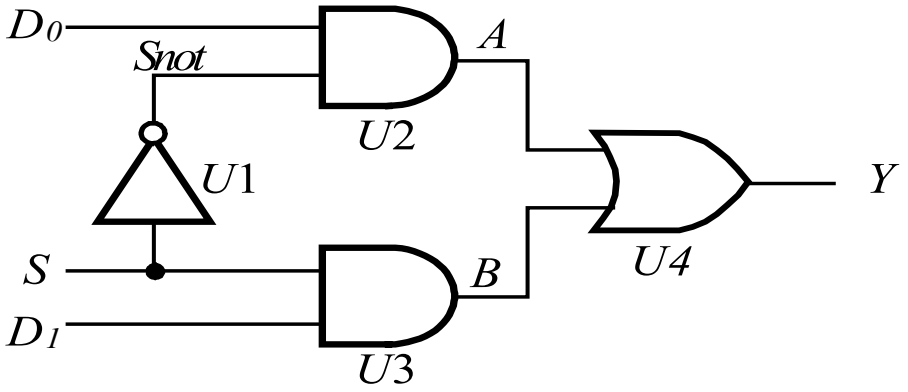
逻辑功能的描述方式有三种不同风格：

结构描述方式（门级描述方式）

数据流描述方式

行为描述方式。

例　用结构描述方式建立门电路Verilog模型



模块名

```
module mux2to1(D0, D1, S, Y );
 input D0, D1, S;  //定义输入信号
 output  Y;    //定义输出信号
 wire   Snot, A, B ; //定义内部节点信号数据类型
 //下面对电路的逻辑功能进行描述
   not   U1(Snot, S);
   and   U2(A, D0, Snot);
   and    U3(B, D1, S);
   or    U4(Y, A, B);
endmodule
```

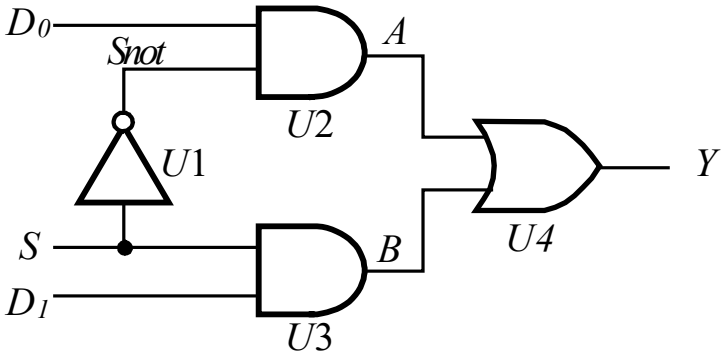
端口类型说明

数据类型说明

电路结构描述

结构描述方式就是通过主模块，子模块的方式描述电路的结构，其中子模块的实现，可以通过元件例化的方式重复调用。

例　用数据流描述方式建立模型



$$Y = D_0 \cdot \overline{S} + D_1 \cdot S$$

```
module mux2to1_dataflow(D0, D1, S, Y );
  input D0, D1, S;
  output Y;
  wire Y ;
//下面是逻辑功能描述
  assign Y = (~S & D0) | (S & D1); //表达式左边Y是wire型
endmodule
```

端口类型说明

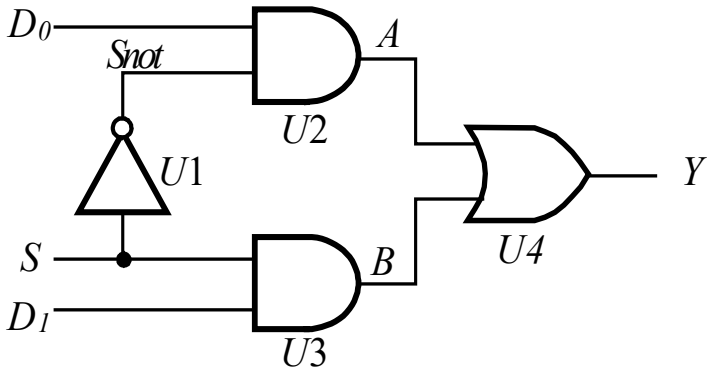
数据类型说明

电路输出量描述

注意，在assign语句中，左边变量的数据类型必须是wire型。

数据流描述方式就是用输出信号量与输入变量之间的函数表达式来描述电路的特性。

例　用行为描述方式建立模型



$$Y = D_0 \cdot \overline{S} + D_1 \cdot S$$

```
module mux2to1_bh(D0, D1, S, Y );
  input D0, D1, S;
  output Y;
  reg  Y ;
//逻辑功能描述
   always @(S or D0 or D1)
         if (S == 1)  Y = D1; //也可以写成 if (S)  Y = D1;
         else  Y = D0;   //注意表达式左边的Y是reg型
endmodule
```

数据类型说明

敏感信号列表

电路真值表的描述

行为描述方式就是用真值表来描述电路的特性。

真值表

S	Y
0	D_0
1	D_1

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
```

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;

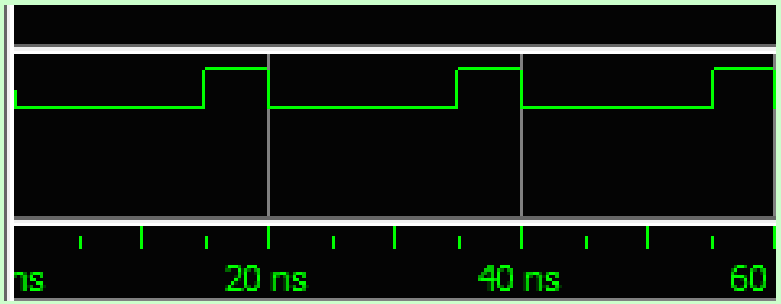
  always #10 clock=~clock;

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

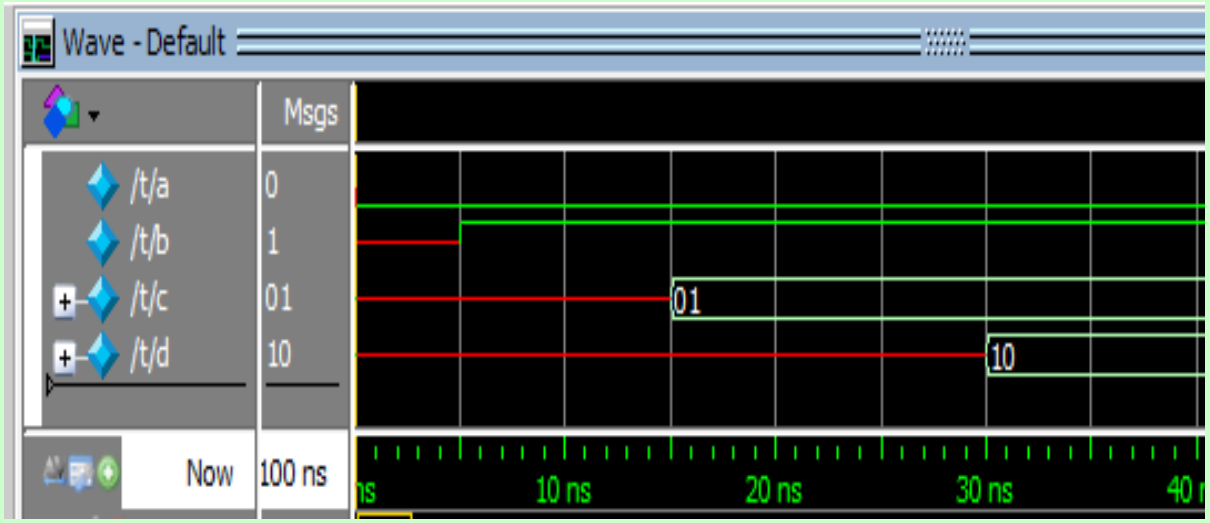
  always @(*)

- 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)
```

顺序块

- 以关键字begin...end将多条语句封装成块
- 按顺序执行




```
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
```

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。

行为级描述中，assign语句前不需要添加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……

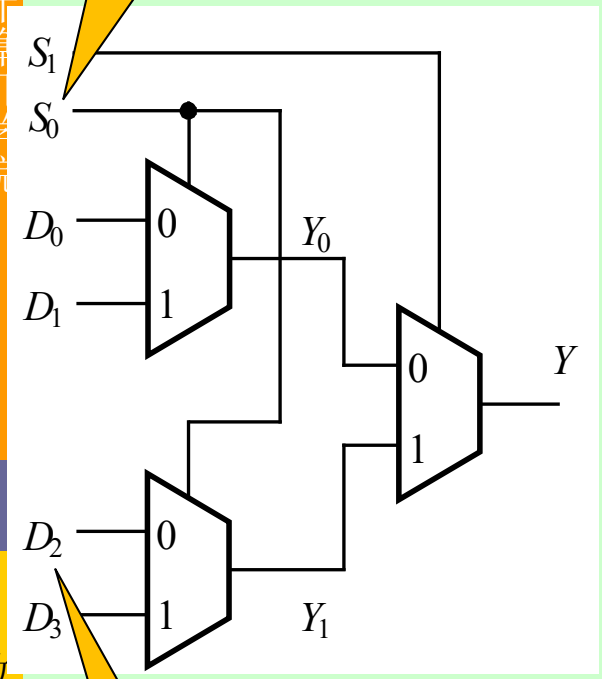
（2） if (condition_expr) true_statement;
else false_ statement; //if……else ……语句

（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



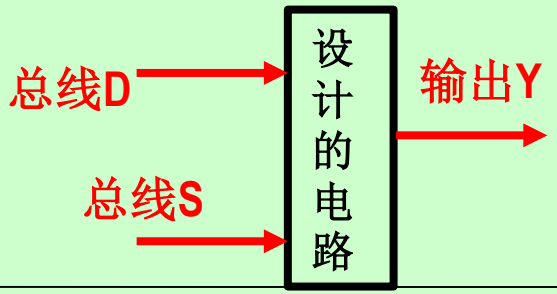

```
module mux4to1_bh(D, S, Y);
  input [3:0]  D; //输入端口
  input [1:0]  S; //输入端
  output  reg  Y; //输出端
  always @(D, S) //电路功能描
    if (S == 2'b00)     Y = D[0
    else if (S== 2'b01)  Y = D
    else if (S== 2'b10)  Y = D
    else                 Y = D[3];
endmodule
```

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

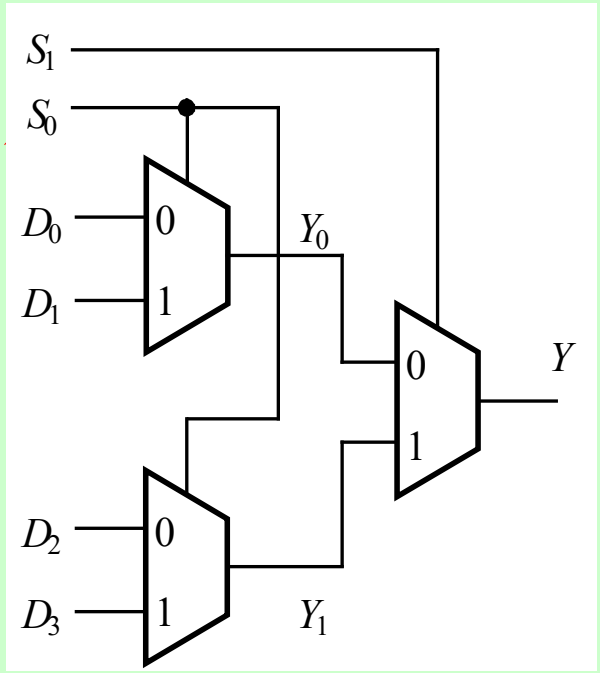
```
case (case_expr)
      item_expr1:  statement1;
      item_expr2:  statement2;
       ……
     default:   default_statement;   //default语句可以省略
endcase
```

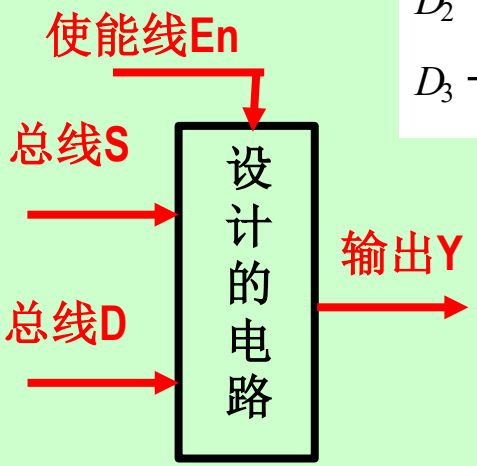
注意：当分支项中的语句是多条语句，必须在最前面写上关键词begin，在最后写上关键词end，成为顺序语句块。

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En 的4选1数据选择器的行为进行Verilog描述。
当En=0时，数据选择器工作，En=1时，禁止工作，输出为0。

```
Module mux4to1_bh (D, S, Y，  En);
  input [3:0]   D，  [1:0]    S;
   input En;
  output  reg  Y;
  always @(D, S, En)        //2005 syntax, 过程语
Begin
   if (En==1)   Y = 0;    //En=1时，输出为0
   else                  //En=0时，选择器工作
    case (S)
          2'd0: Y = D[0];
          2'd1: Y = D[1];
          2'd2: Y = D[2];
          2'd3: Y = D[3];
        endcase
end
endmodule
```





例：对基本的4线-2线优先编码器的行为

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	0	1	0	0

总线W

总线Y

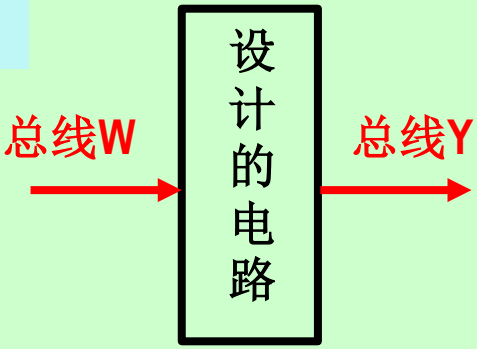
对应于总线W

对应于总线Y

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

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



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语言描述具有高电

A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	0	1	1	1	1	1	1	1
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

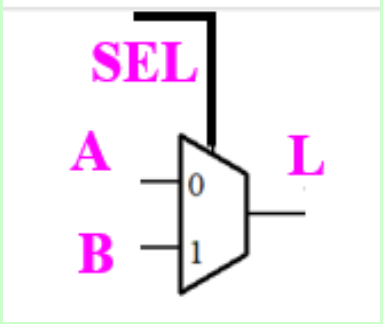
```
module ecoder3to8_bh(A,En,Y);
    input [2:0]   A;
     input En ;
    output reg  [7:0]   Y;
    integer k;          //声明一个整
    always @(A, En)    //声明过程语句
    begin
            Y = 8'b1111_1111;     //设译码器输出的默认值
          for(k = 0; k <= 7; k = k+1) //下面的if-else语句循环8次
            if ((En==1) && (A== k) )
                Y[k] = 0;  //当En=1时，根据A进行译码
            else
                Y[k] = 1;  //处理使能无效或输入无效的情况
        end

endmodule
```

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。

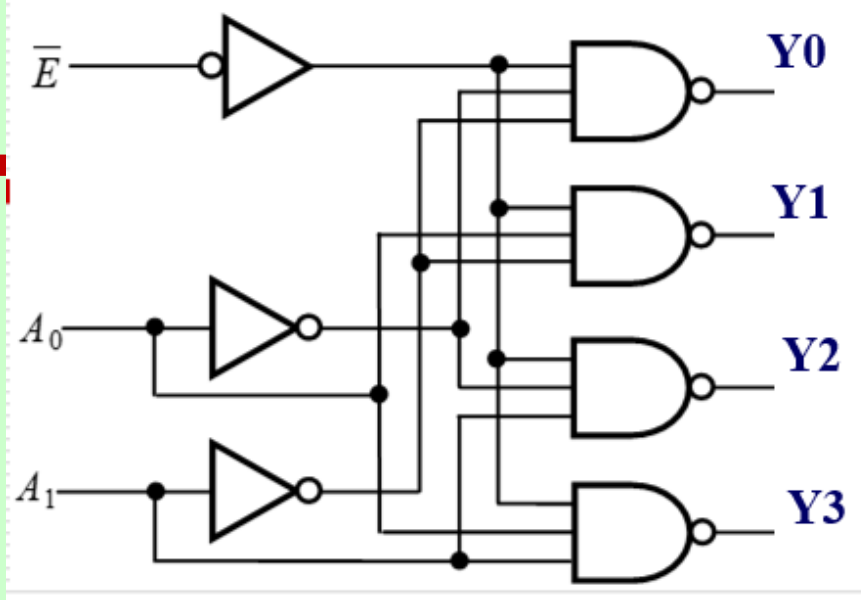
```
module mux2x1_df (A,B,SEL,L);
  input  A, B, SEL;
output  (wire)   L;
  assign L = SEL ?  B:A;
endmodule
```



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。



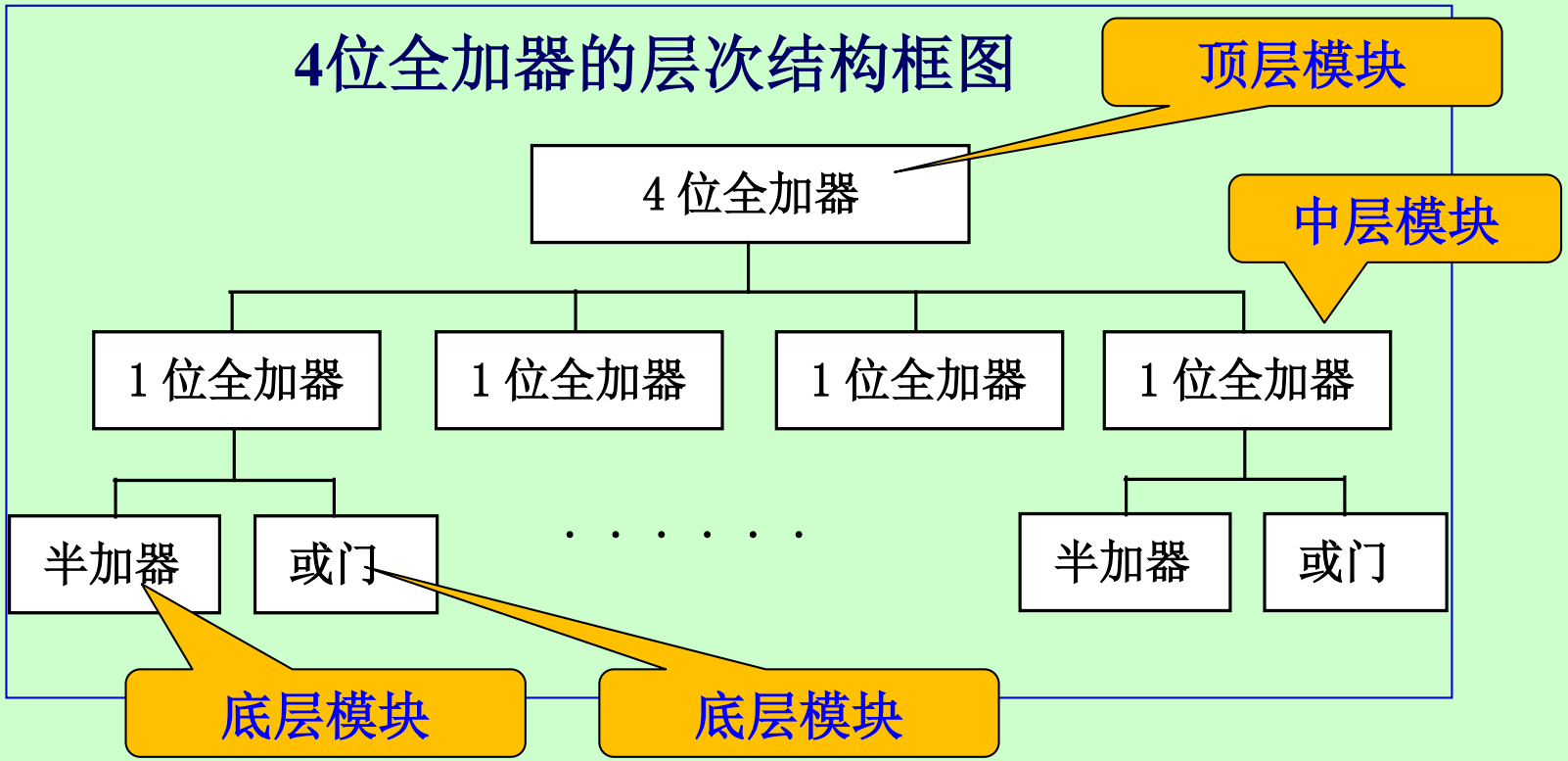

```
module decoder_df (A1,A0,E,Y);
  input  A1,A0,E;
  output [3:0]   Y;
     assign   Y[0] = ~(~A1 & ~A0 & ~E);
     assign  Y[1] = ~(~A1 & A0 & ~E);
     assign  Y[2] = ~(A1 & ~A0 & ~E);
     assign  Y[3] = ~(A1 & A0 & ~E);
endmodule
```

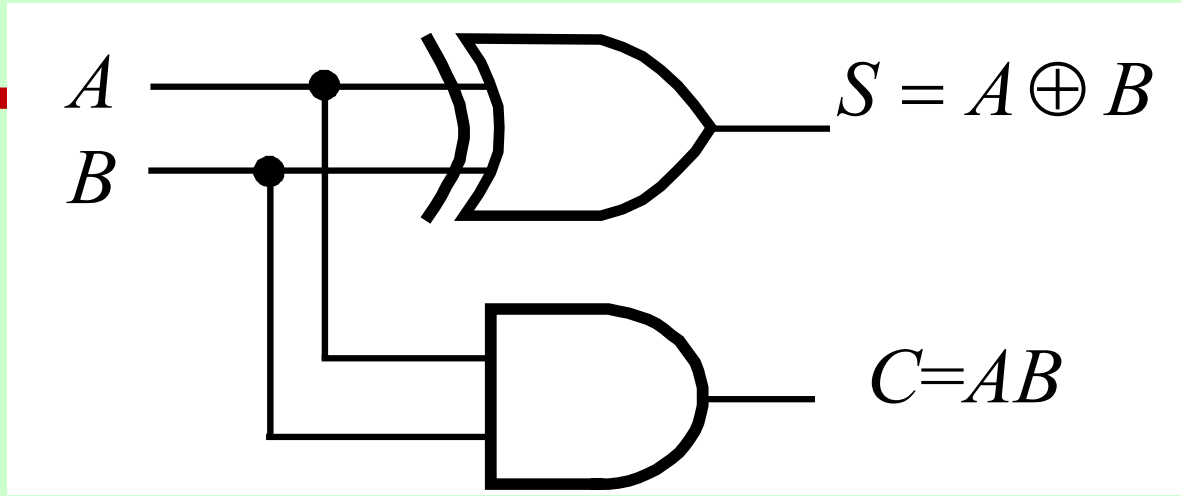
行为级描述中，assign语句前不需要添加always语句。

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

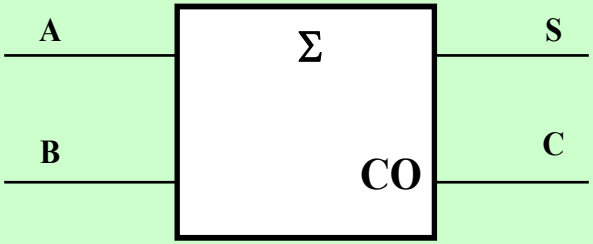




底层模块

```
module halfadder (S,C,A,B);
    input A,B;
    output S,C;
    xor (S,A,B);
    and (C,A,B);
endmodule
```

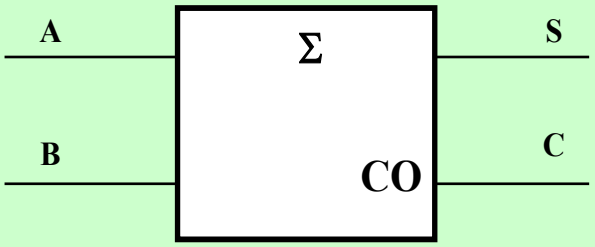
半加器的门级描述



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
```

底层模块

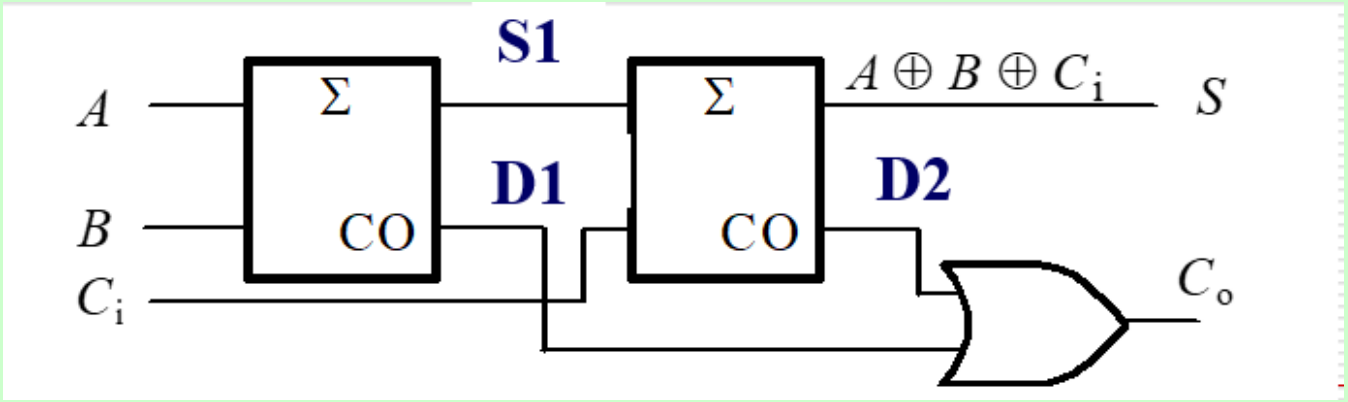


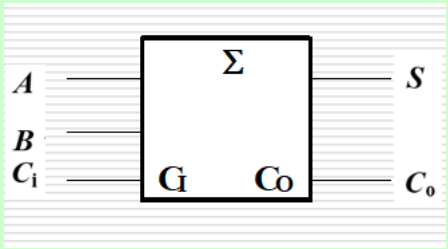
halfadder (S,C,A,B);

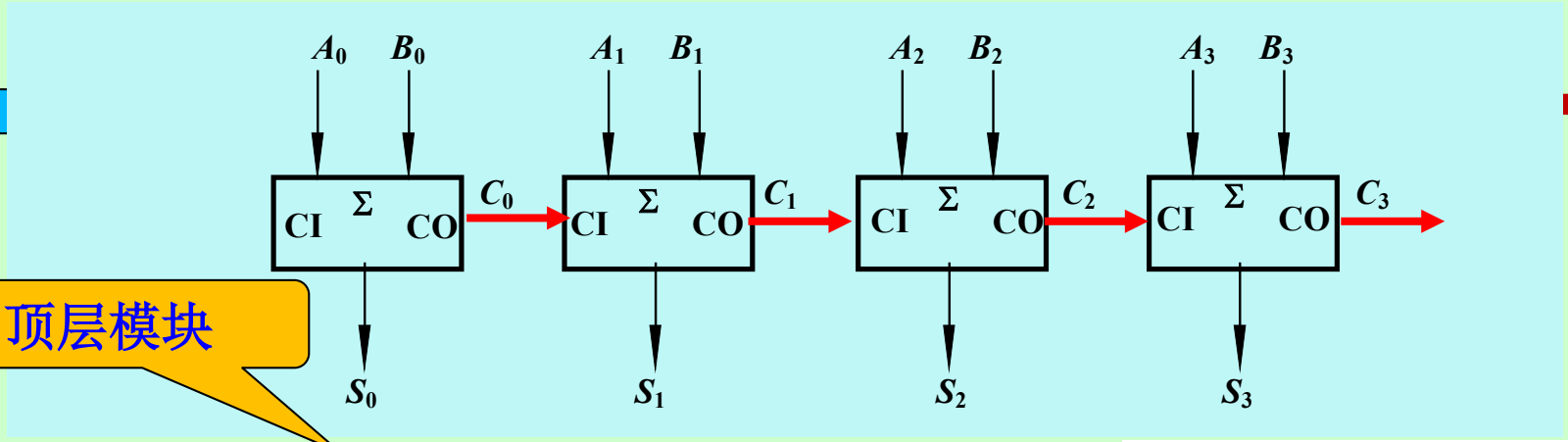
全加器的描述-调用半加器

halfadder HA1 (. S(S1), . C(D1), . A(A), .B(B));

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



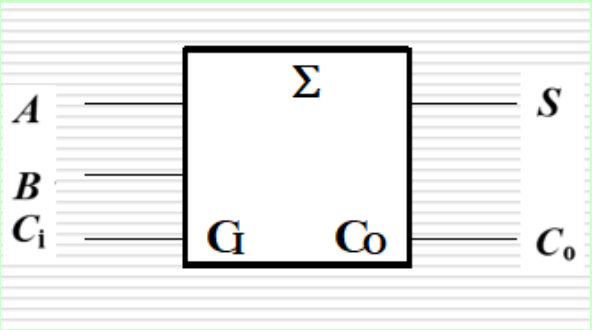




顶层模块

```
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
```

4位全加器的描述
--调用1位全加器



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

```
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）：

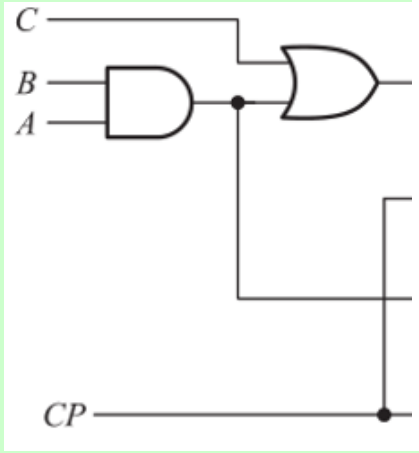
always@(posedge CP or negedge CR)

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：

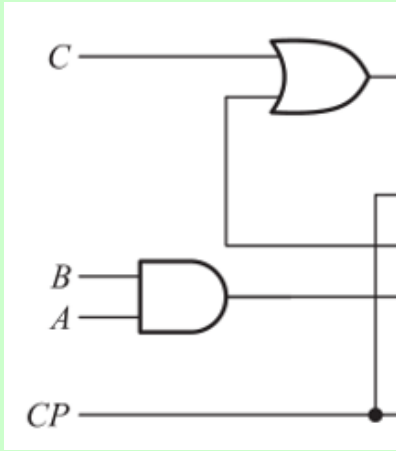
阻塞型用“＝”表示，多条语句顺序执行。

```
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

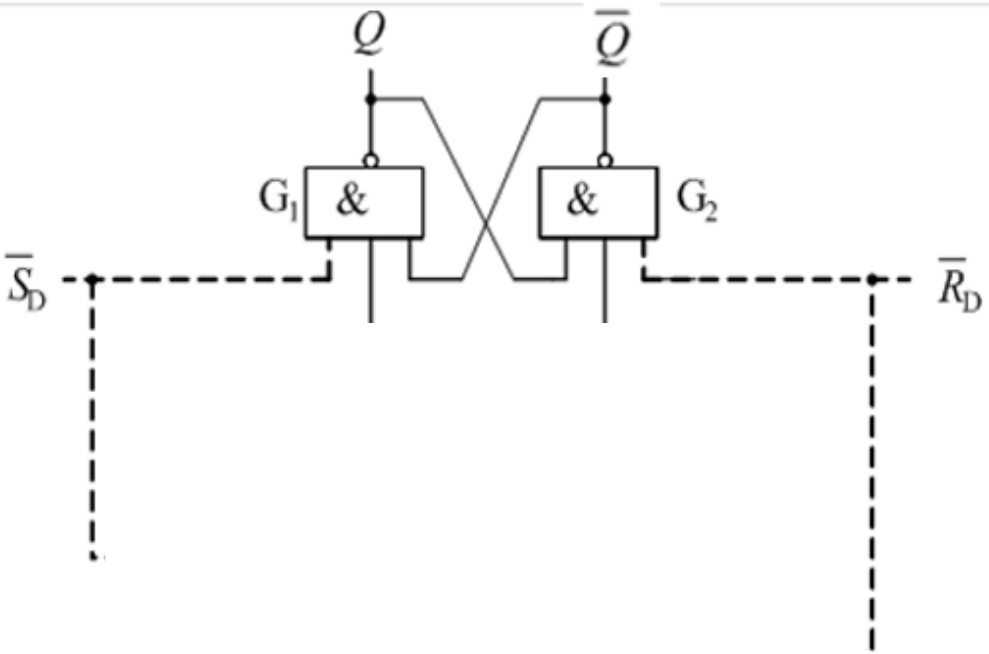
6.7.4　数字钟的Verilog建模

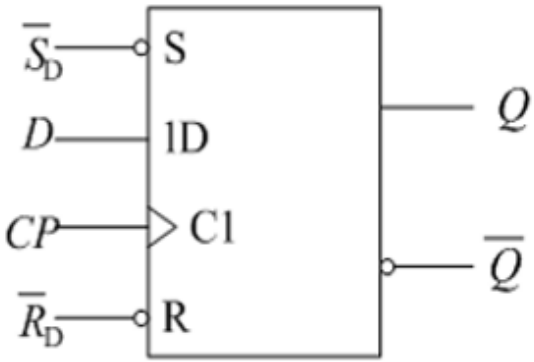
补充

S=Set　　R=Reset

清零端和置1端引脚

维持-阻塞式D触发器





（a）电路结构　　（b） 图形符号

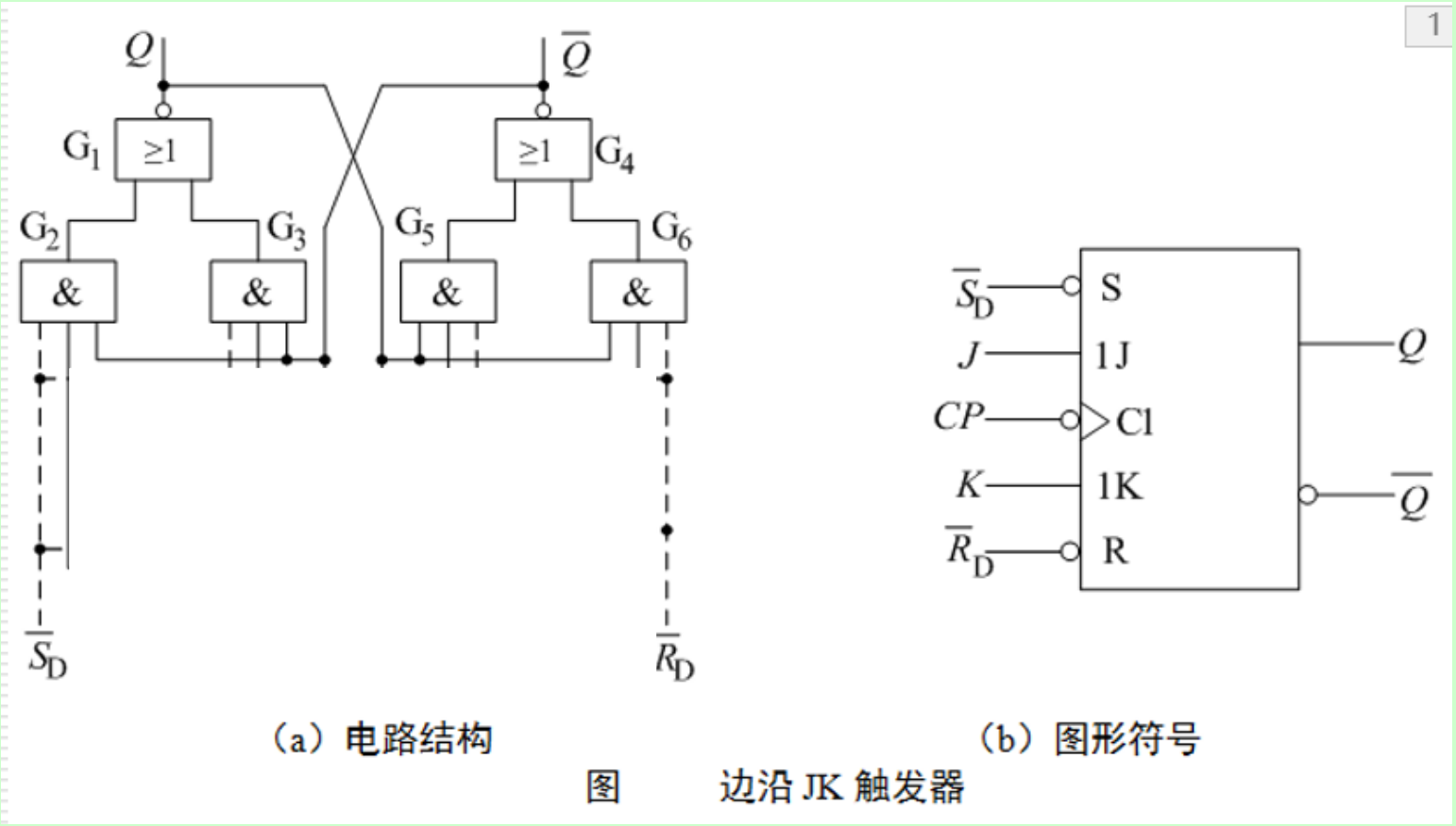
图　　维持-阻塞式 D 触发器

补充

S=Set　　R=Reset

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

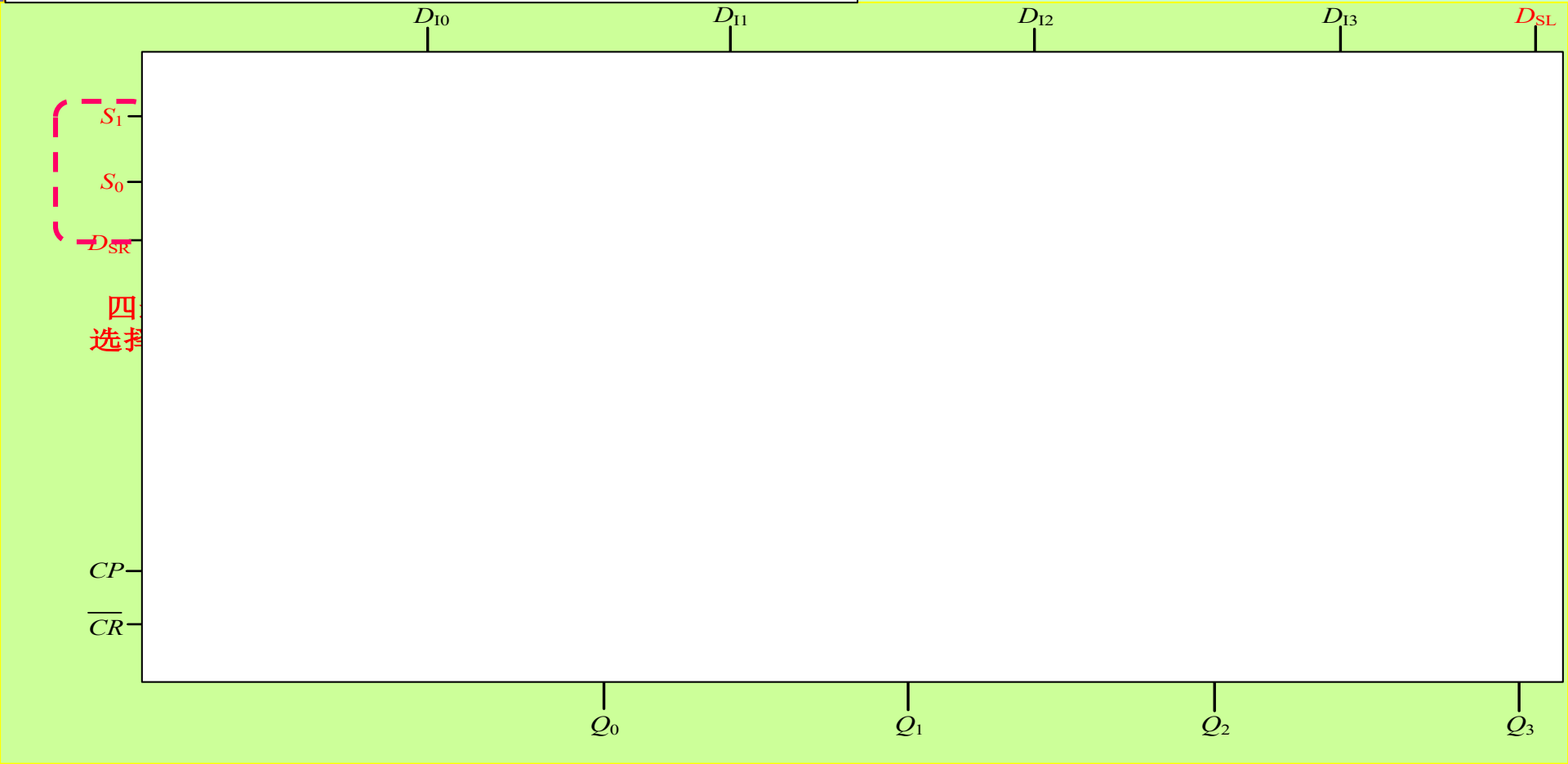
用行为级描述always描述一个 4 位双向移位寄存器，有异步清零、同步置数、左移、右移和保持。功能同74HC194。

```
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
```

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

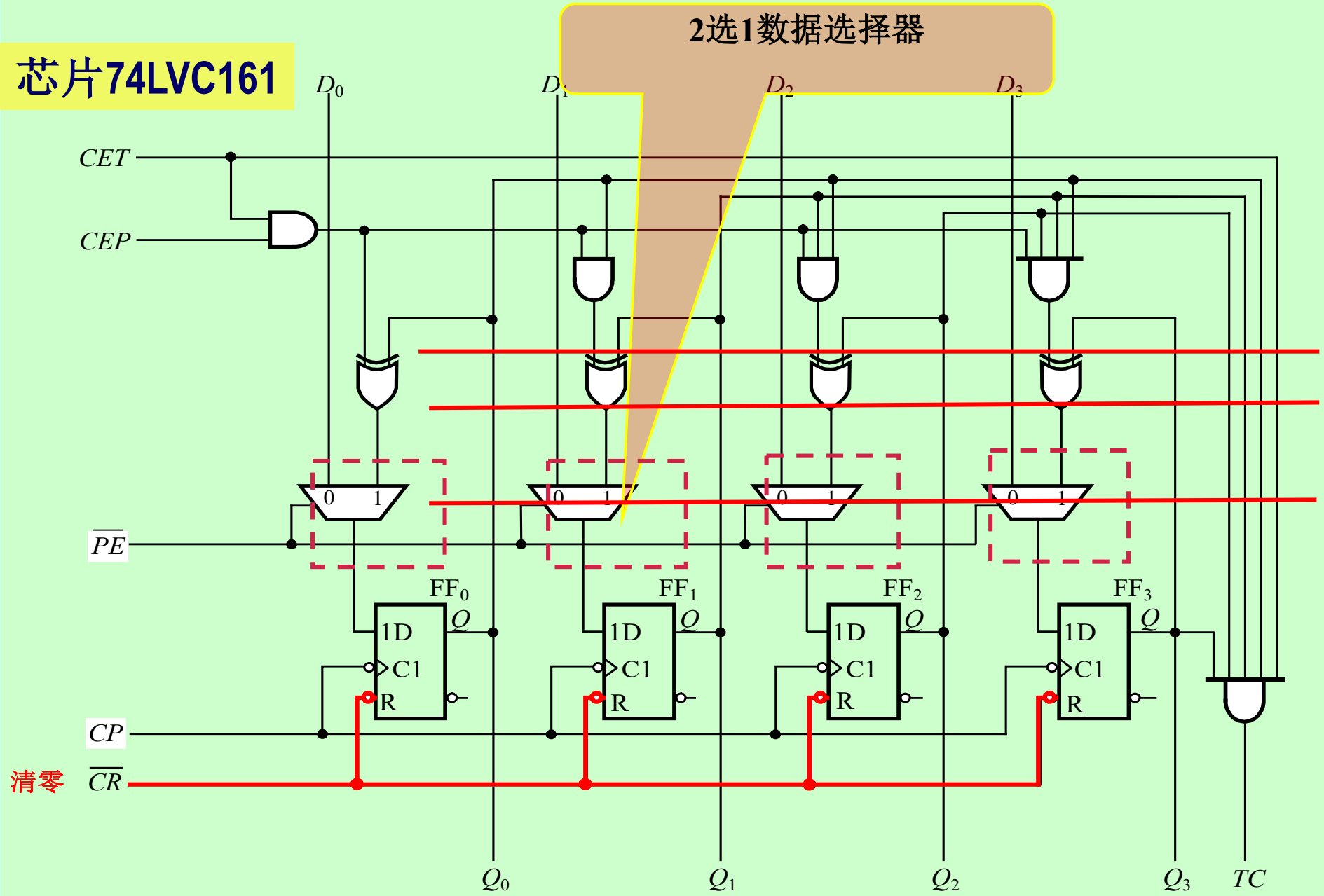
用Verilog描述具有使能端、异步置零、同步置数、计数、保持的16进制计数器,类似74HC161芯片。

```
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
D_0
D_1
D_2
D_3
CET
CEP
$\overline{PE}$
0
1
FF_0
FF_1
FF_2
FF_3
1D
C1
R
Q
CP
清零
$\overline{CR}$
Q_0
Q_1
Q_2
Q_3
TC

完