

数 字 逻 辑

丁 贤 庆

ahhfdxq@163.com

第三次实验时间

地点：电气楼505房间

物联网2班：下周二上午10： 10---11:50

物联网1班：本周日晚上19： 00---20:40

计科1班：下周一晚上19： 00---20:40

计科2班：下周一上午8： 00---9:40

计科3班：下周二晚上19： 00---20:40

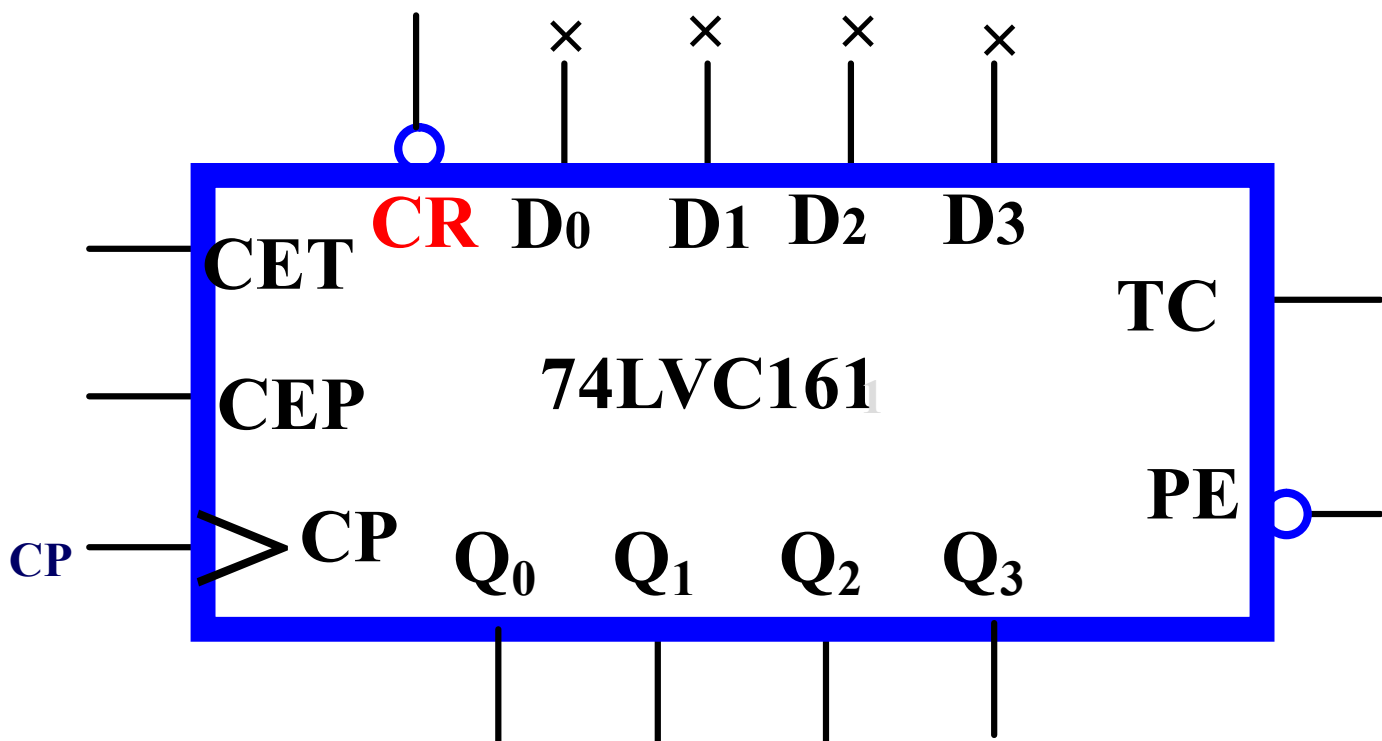
计科4班：下周三下午16： 00---17:40

计科5班：下周三晚上19： 00---20:40

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、下周三（四）课前交作业，交完作业才正式上课。
- ✎ 4、下周三（四）课上，会有**10**分钟的课堂小测。
- ✎ 5、本次的作业(不用抄题目)
 - ✎ 2.4.3(5)(6) (7)
 - ✎ 2.4.4(3)(4)
 - ✎ 2.5.5
 - ✎ 2.5.6（不要求仿真，只要求写出逻辑函数式）

74LVC161



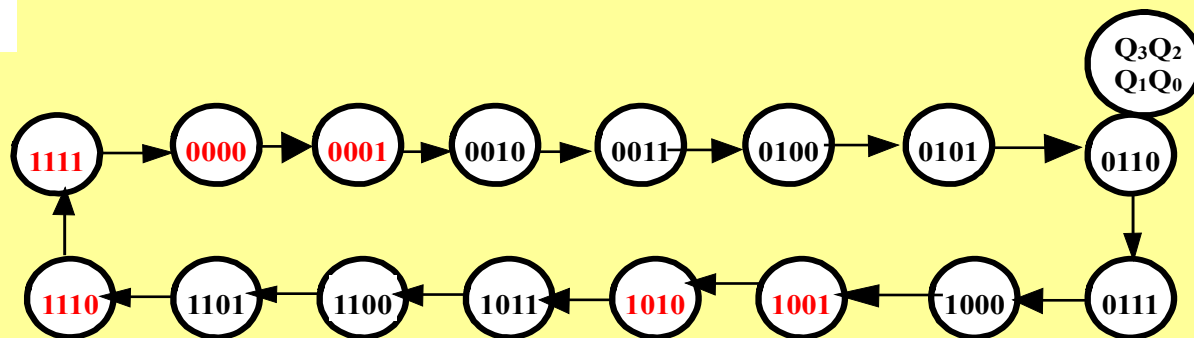
74LVC161的功能表。

输入					功能	
CP	\overline{CR}	\overline{PE}	CEP	CET	说明	解释
×	0	×	×	×	异步复位	$Q_3Q_2Q_1Q_0=0000$
↑	1	0	×	×	同步置数	$Q_3^{n+1}Q_2^{n+1}Q_1^{n+1}Q_0^{n+1}=D_3D_2D_1D_0$
×	1	1	0	1	保持	保持
×	1	1	×	0		保持
↑	1	1	1	1	计数	加1计数

如何才能得到N进制计数器？

1、反馈清零法：利用异步清零输入端**CR**，得到**N**进制计数器

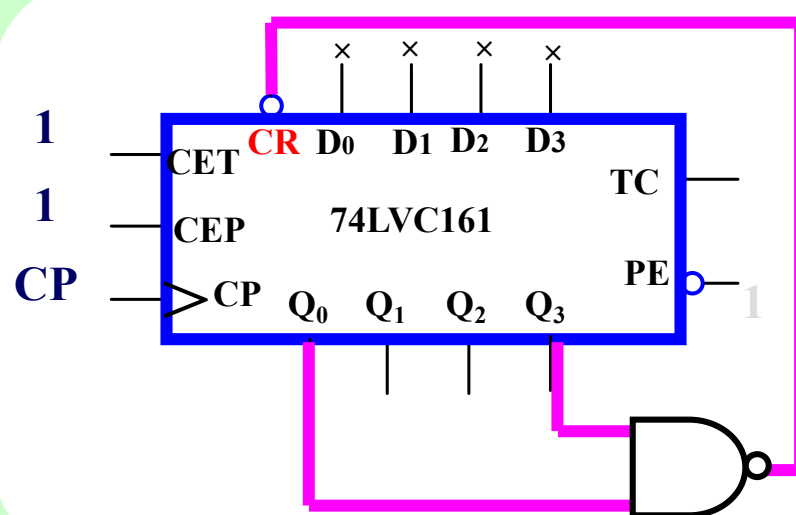
2、反馈置数法：利用同步置数端**PE**，在**M**进制计数器的计数过程中，跳过**M-N**个状态，得到**N**进制计数器



(4) 应用 例 用74LVC161构成九进制加计数器。

(a) 反馈清零法：利用异步清零输入端，在M进制计数器的计数过程中，跳过M-N-1个状态，得到N进制计数器的方法。

CP	Q ₃	Q ₂	Q ₁	Q ₀
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
...			
8	1	0	0	0
9	1	0	0	1
...	...			
15	1	1	1	1



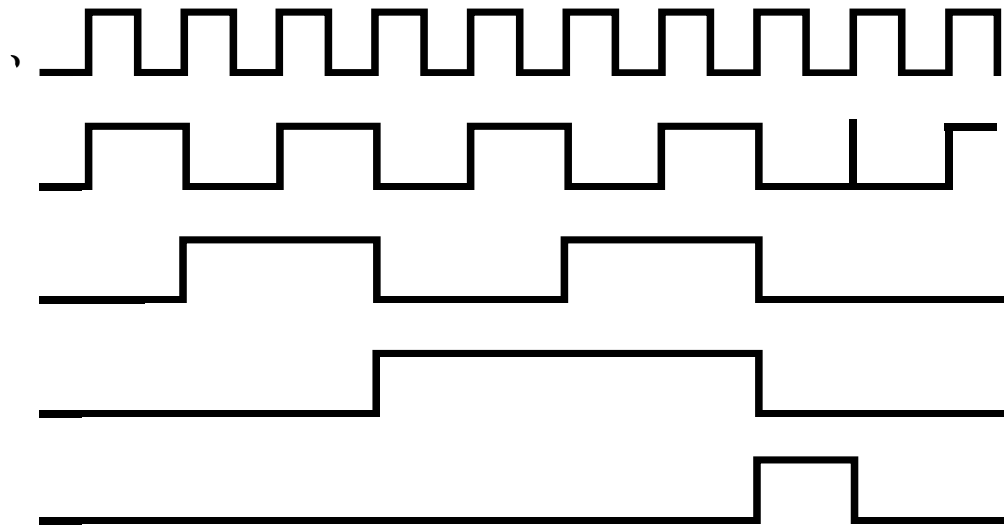
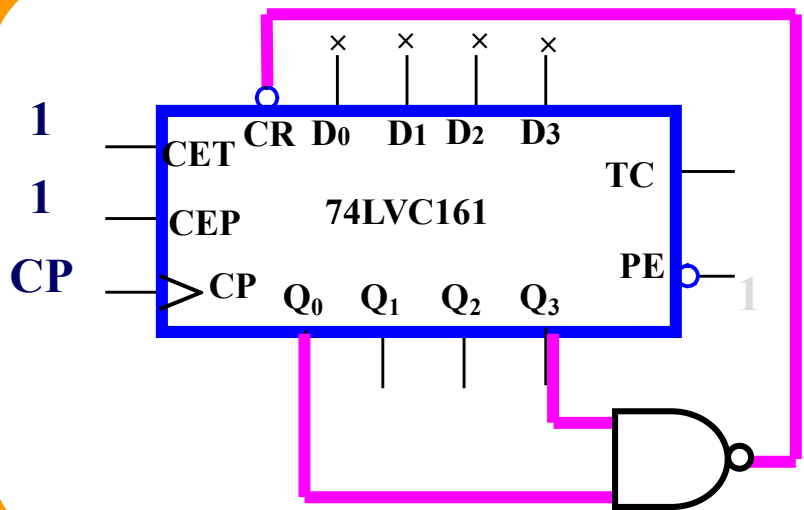
$$CR = \overline{Q_0 \cdot Q_3} = 0$$

采用异步清零法时，由于异步清零最后一个状态保持时间很短，通常忽略不计。

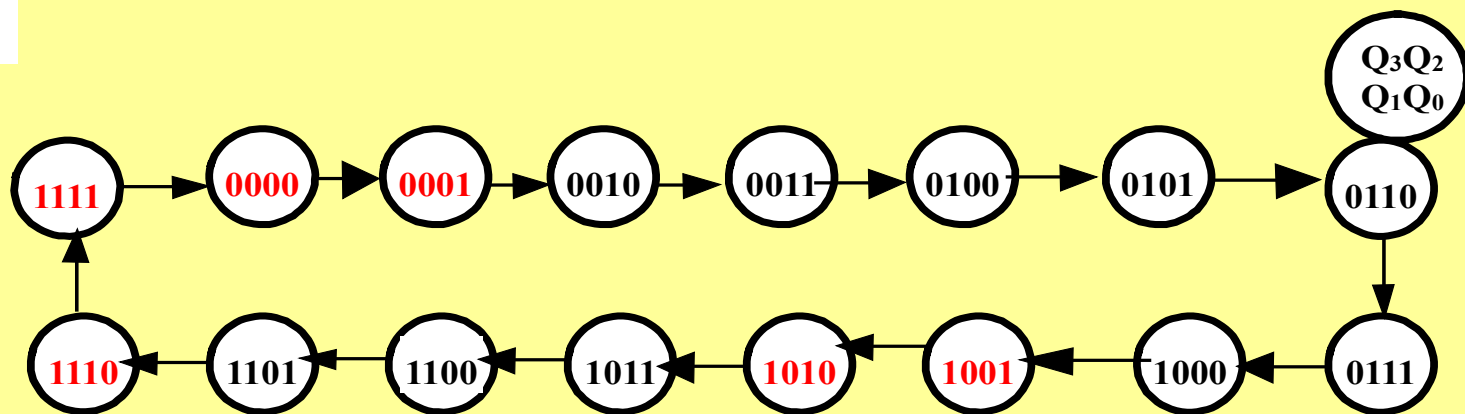
设法跳过16-10=6个状态

计数器的模：计数器状态图中闭合圈包含的稳定的有效状态的数目。本题中最后得到的是模9计数器

工作波形



状态图

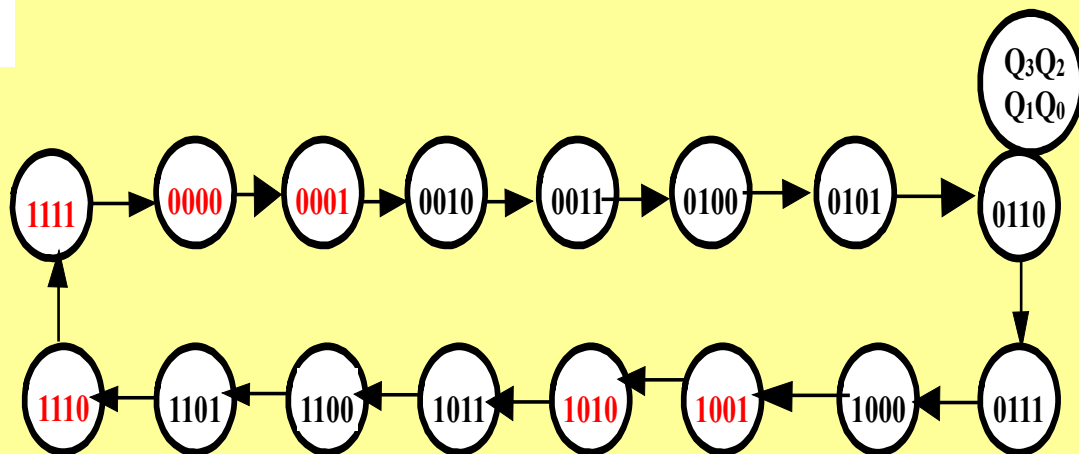
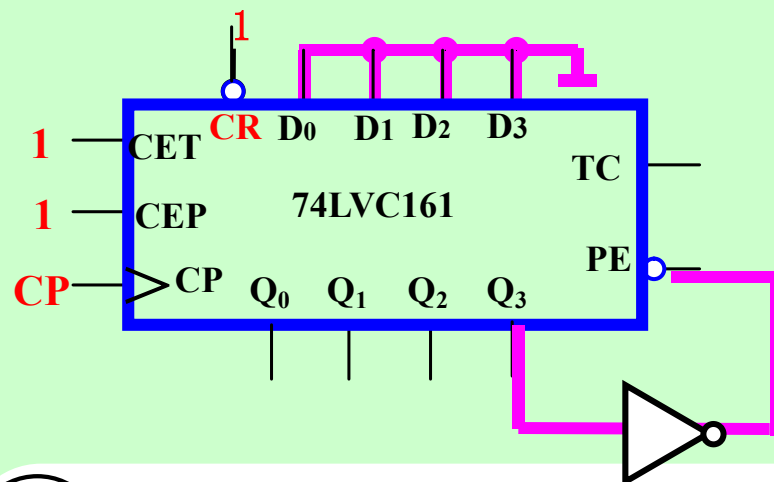


利用同步置数端构成九进制计数器

(b) 反馈置数法:利用同步置数端, 在M进制计数器的计数过程中, 跳过M-N个状态, 得到N进制计数器的方法。

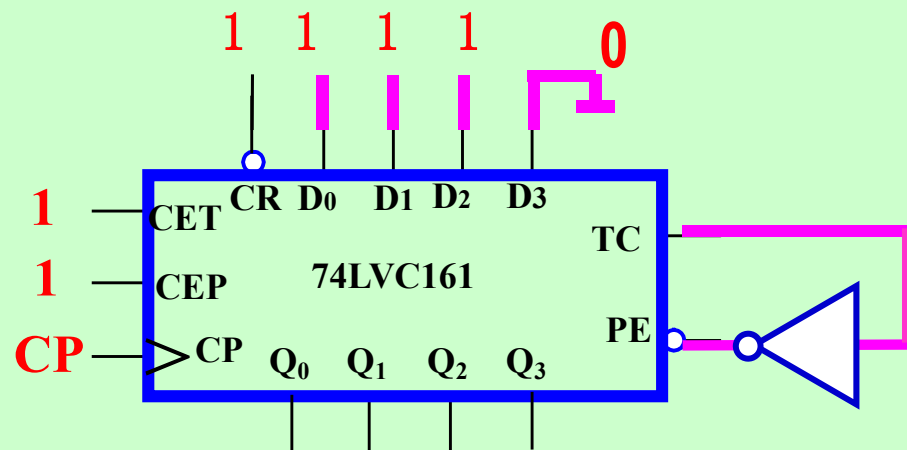
CP	Q ₃	Q ₂	Q ₁	Q ₀
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
...			
8	1	0	0	0

$$PE = \overline{Q_3} = 0$$

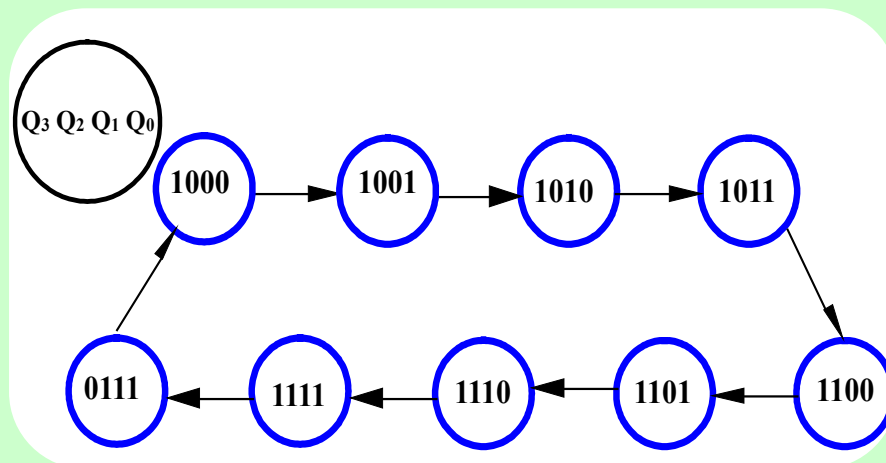


采用后九种状态作为有效状态，用**反馈置数法** 构成九进制加计数器。

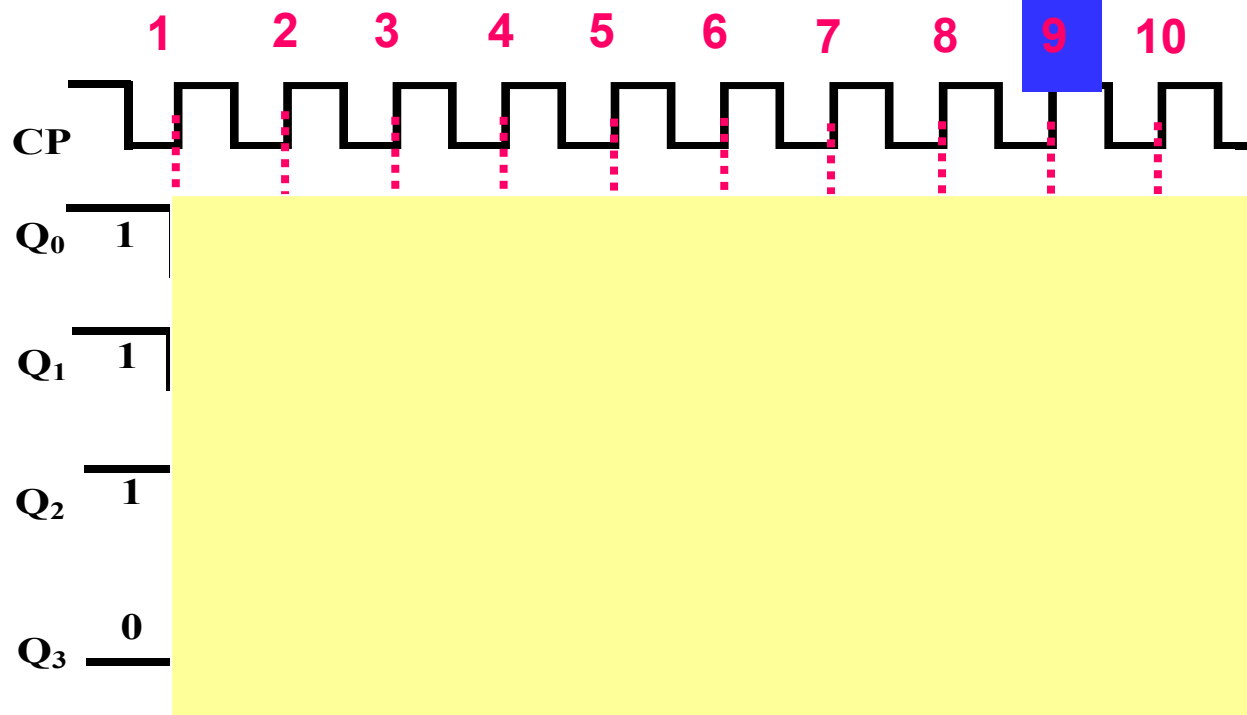
Q_3	Q_2	Q_1	Q_0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1



$$TC = CET \cdot Q_3 \cdot Q_2 \cdot Q_1 \cdot Q_0 = 1$$



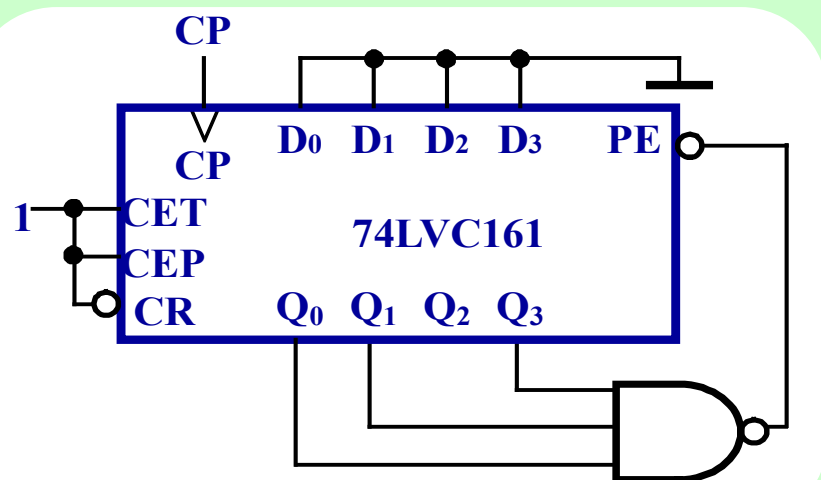
波形图:



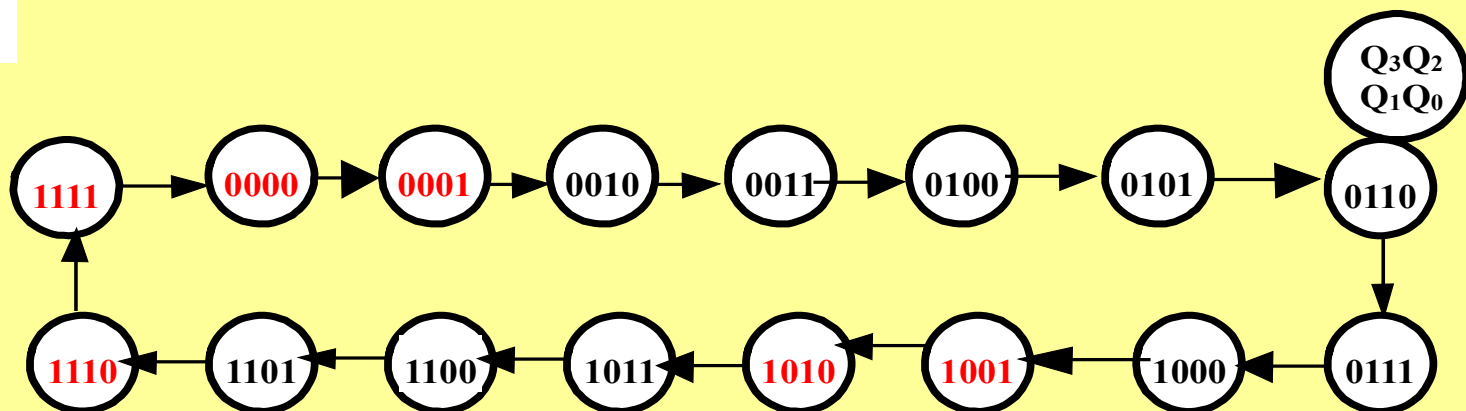
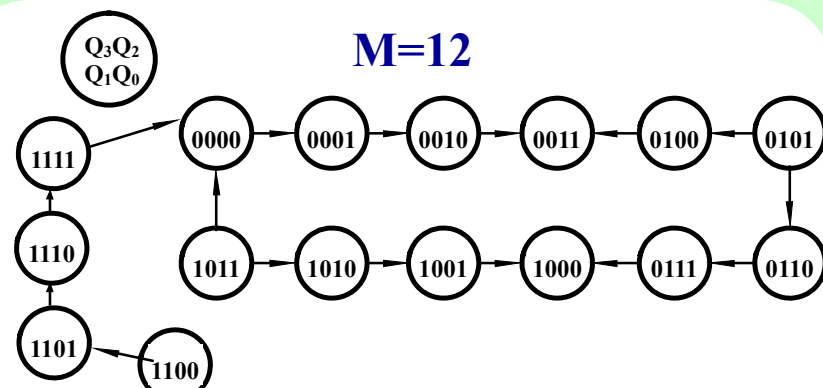
该计数器的模为9。

计数器的模：计数器状态图中闭合圈包含的稳定的有效状态的数目。本题中最后得到的是模9计数器

分析下图所示的时序逻辑电路，试画出其状态图和在CP脉冲作用下 Q_3 、 Q_2 、 Q_1 、 Q_0 的波形，并指出计数器的模是多少？



$$PE = \overline{Q_3 \cdot Q_1 \cdot Q_0} = 0$$

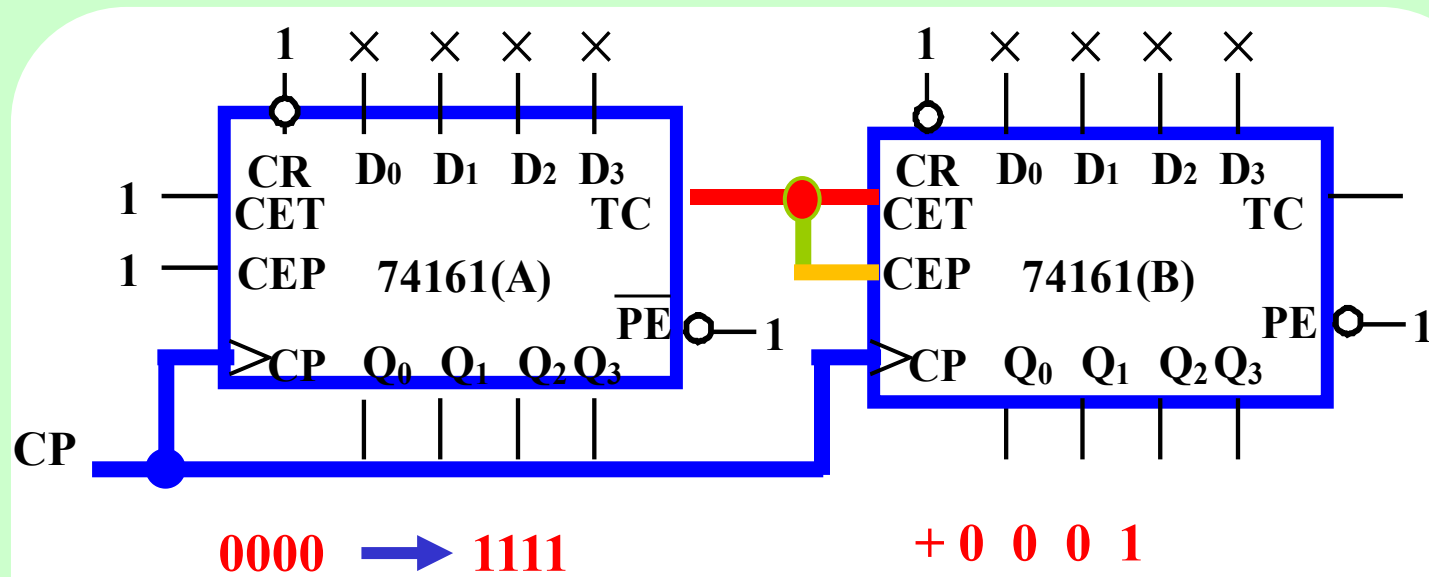


例 用74VC161组成256进制计数器。

解：设计思想

- 1片74161是16进制计数器
- $256 = 16 \times 16$
- 所以256进制计数器需用两片74161构成
- 片与片之间的连接通常有两种方式：
 - 并行进位（低位片的进位信号作为高位片的使能信号）
 - 串行进位（低位片的进位信号作为高位片的时钟脉冲，即异步计数方式）

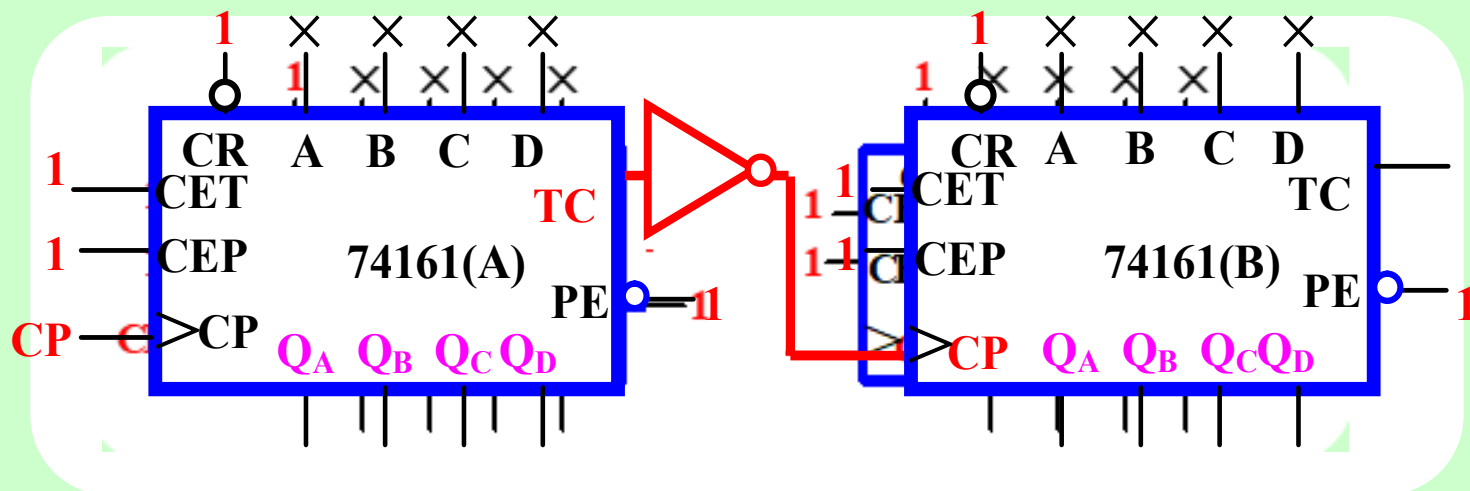
并行进位：低位片的进位作为高位片的使能(采用同步时钟)



计数状态：0000 0000 ~ 1111 1111

$$N = 16 \times 16 = 256$$

串行进位：低位片的进位作为高位片的时钟



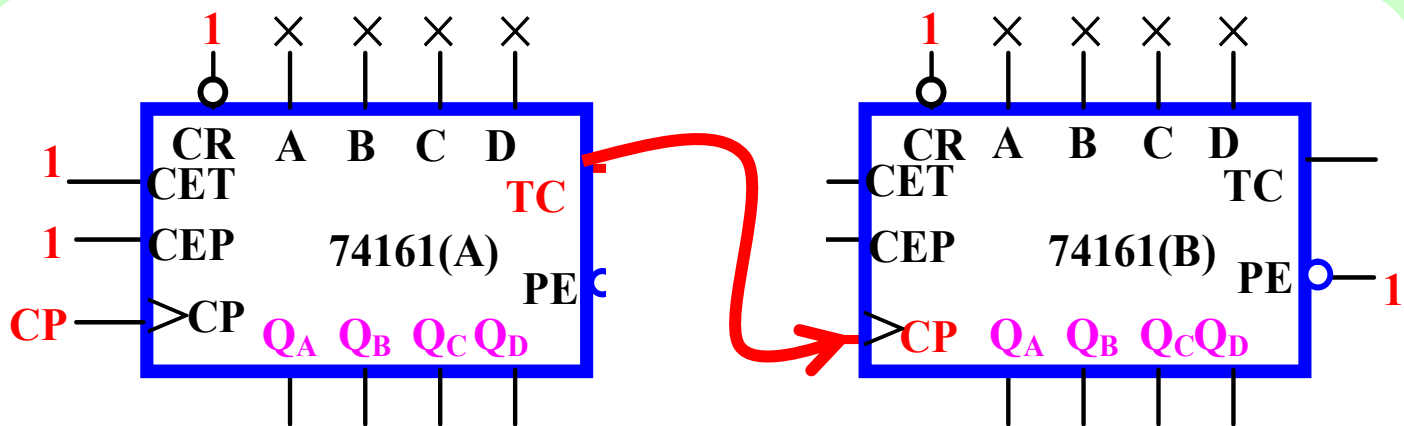
0000 → 1111 + 0 0 0 1

计数状态：0000 0000 ~ 1111 1111

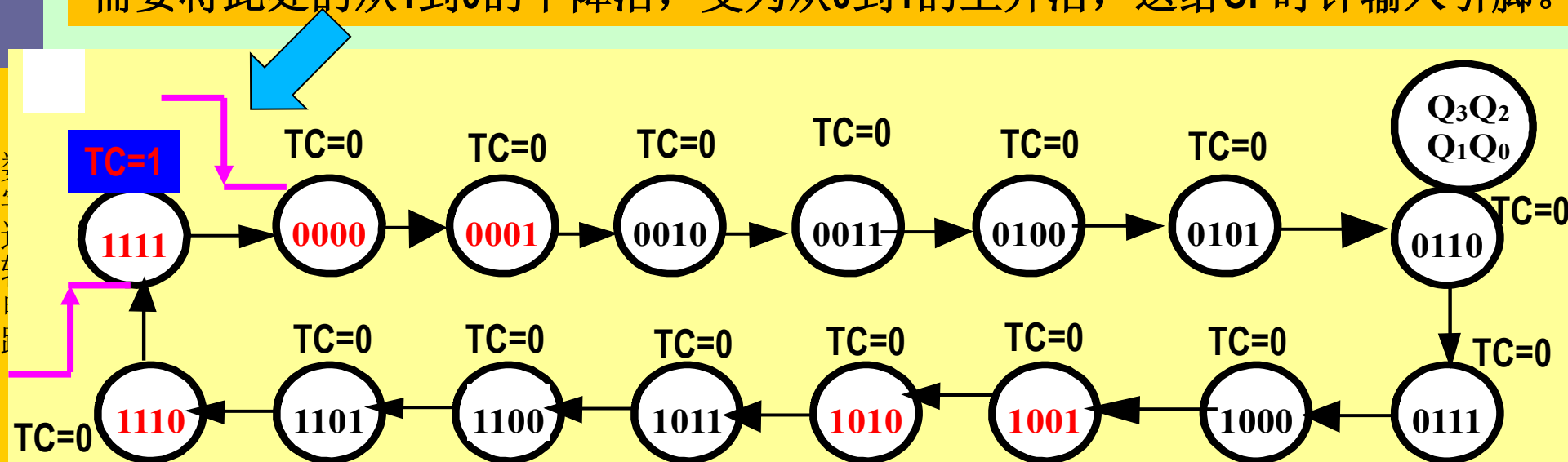
采用串行进位时，为什么低TC要经反相器后作为高位的CP？

加个反相器，是将TC由1变0的下降沿，变成161芯片所需要的由0变1的上升沿。

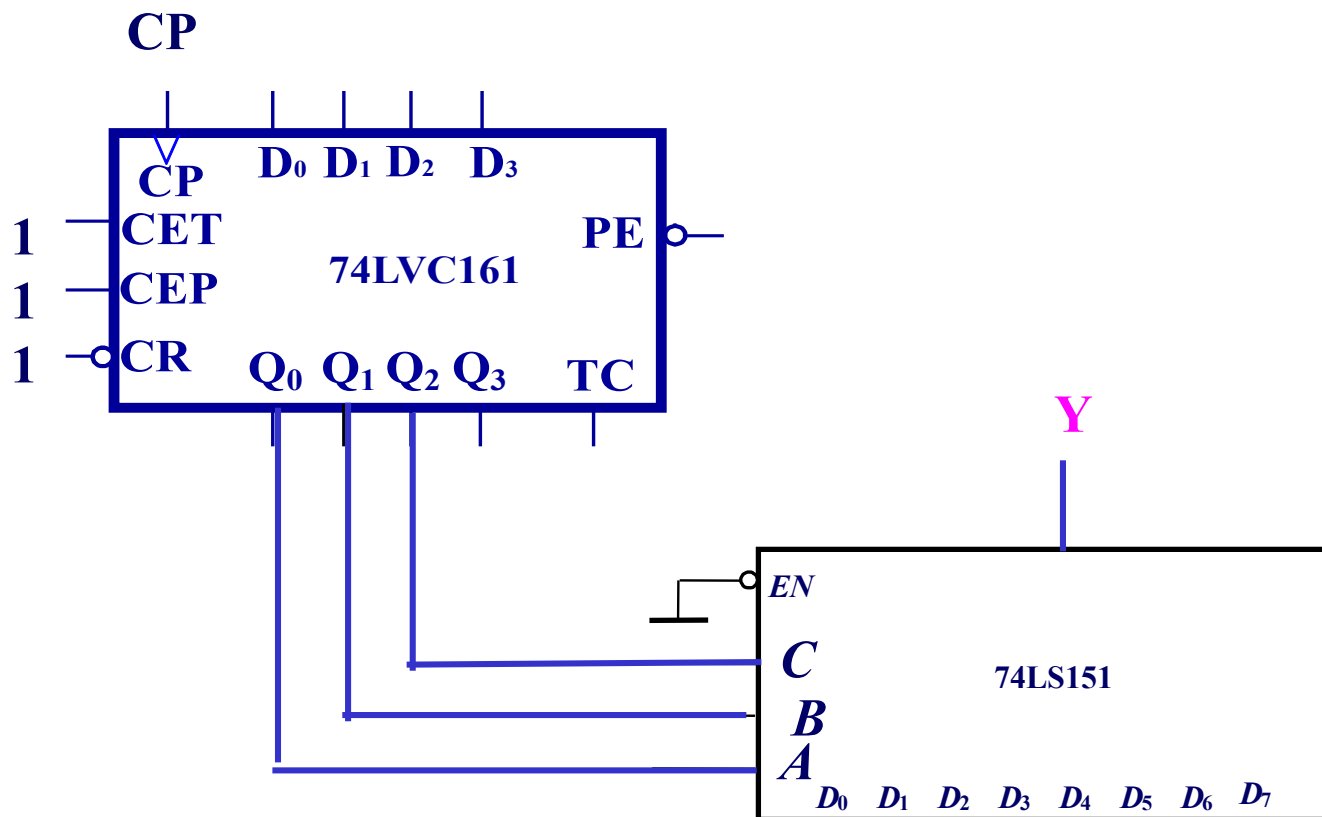
串行进位：低位片的进位作为高位片的时钟



需要将此处的从1到0的下降沿，变为从0到1的上升沿，送给CP时钟输入引脚。



在CP的作用下，Y端产生00010111循环序列信号



如要求Y端产生10110010循环序列信号，如何改变电路的连接？

3. 环形计数器

(1) 工作原理

① 基本环形计数器

Q3连线到D0，就构成基本环形计数器

置初态 $Q_3Q_2Q_1Q_0=0001$,

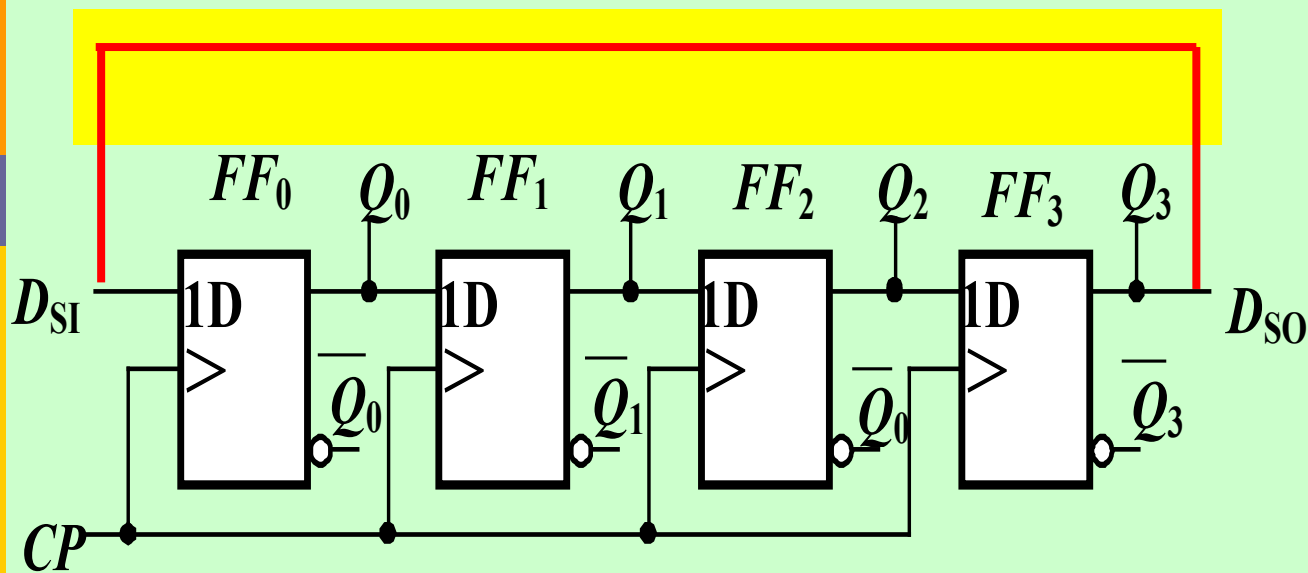
第一个CP: $Q_3Q_2Q_1Q_0=0010$,

第二个CP: $Q_3Q_2Q_1Q_0=0100$,

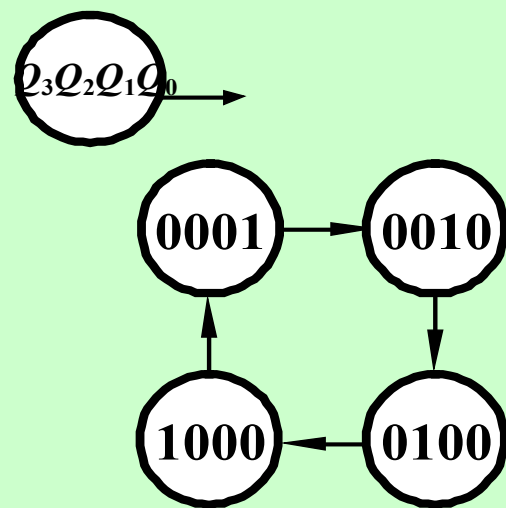
第三个CP: $Q_3Q_2Q_1Q_0=1000$,

第四个CP: $Q_3Q_2Q_1Q_0=0001$,

第五个CP: $Q_3Q_2Q_1Q_0=0010$,



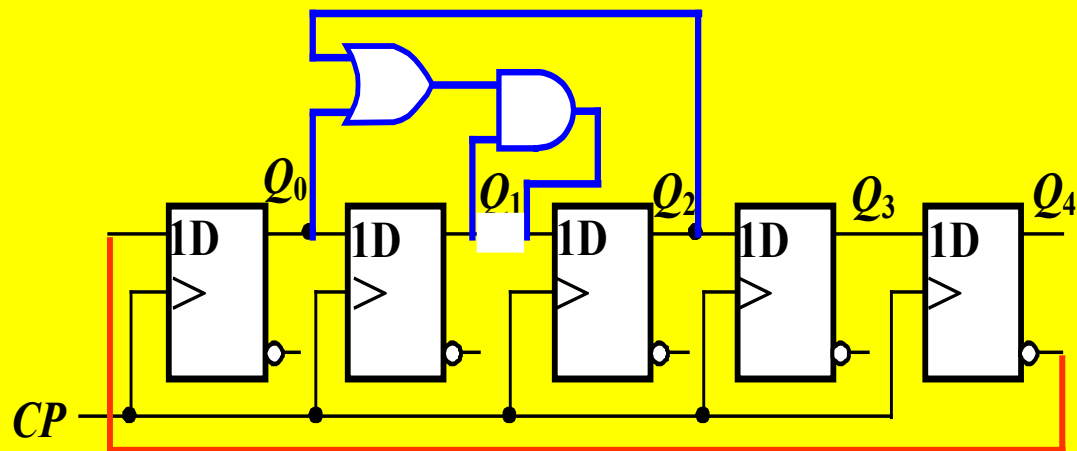
状态图



② 扭环形计数器

a、电路

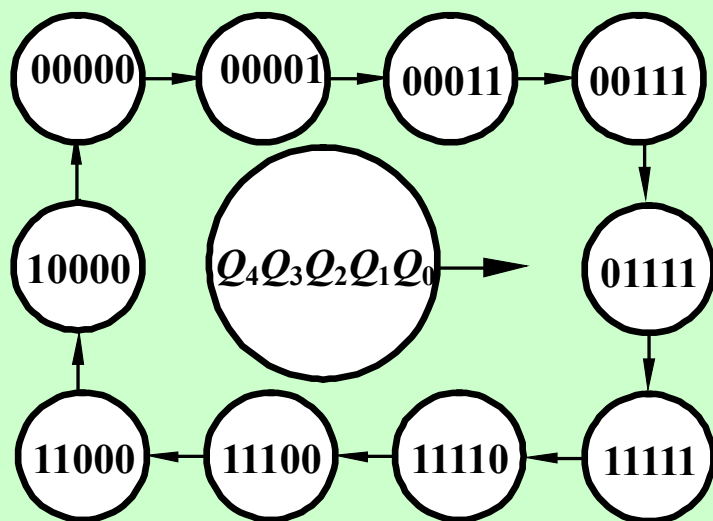
\bar{Q}_4 连线到 D_0 ，就构成扭环计数器



b、状态表

状态编号	Q_4	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	1
3	0	0	1	1	1
4	0	1	1	1	1
5	1	1	1	1	1
6	1	1	1	1	0
7	1	1	1	0	0
8	1	1	0	0	0
9	1	0	0	0	0

c、状态图



2.5 硬件描述语言Verilog HDL基础

2.5.1 Verilog语言的基本语法规则

2.5.2 变量的数据类型

2.5.3 运算符及其优先级

2.5.4 Verilog内部的基本门级元件

2.5.5 Verilog程序的基本结构

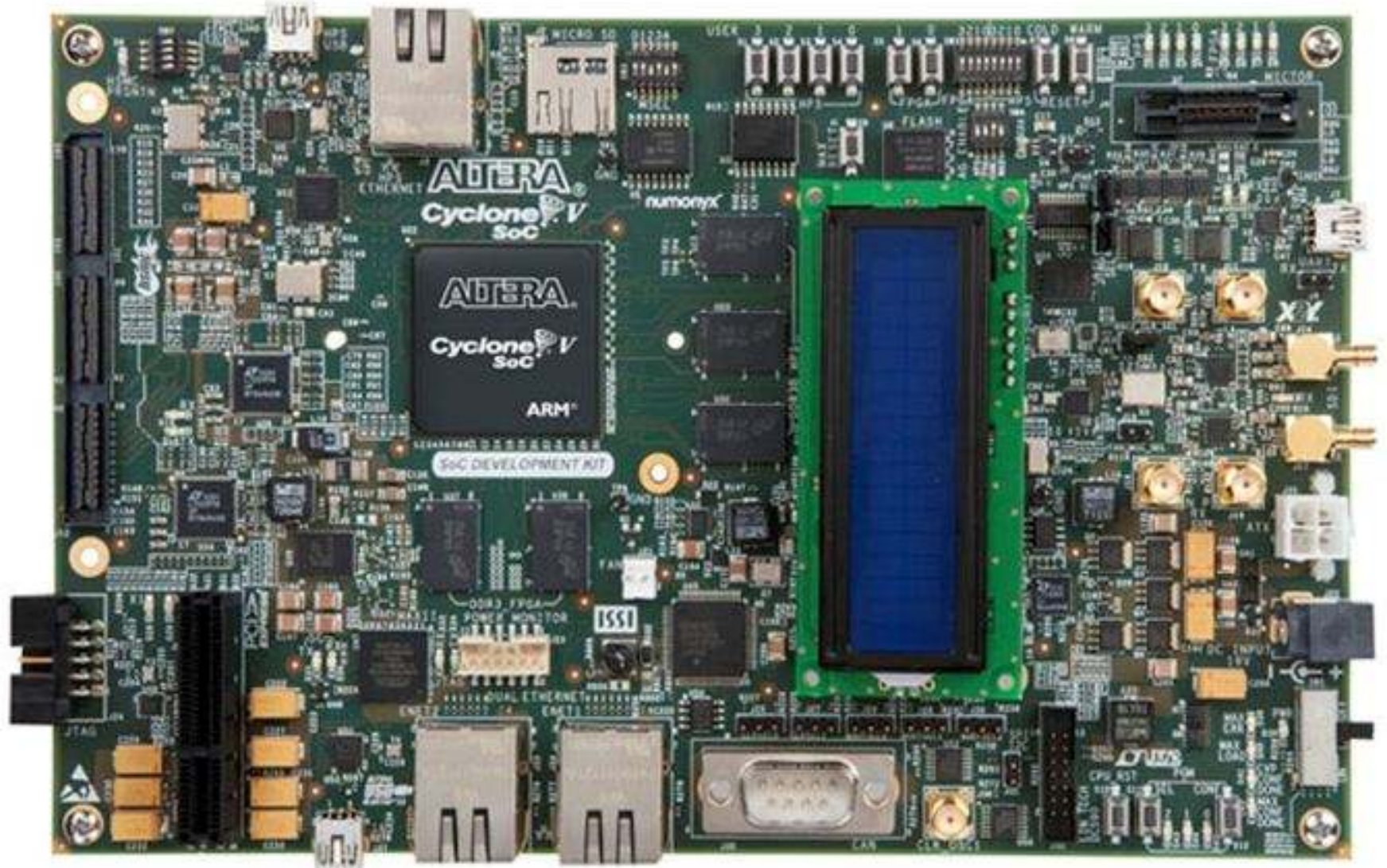
2.5.6 逻辑功能的仿真与测试

一、FPGA简介（补充知识）

- **FPGA** (Field Programmable Gate Array)即**现场可编程门阵列**，它是在PAL、GAL、EPLD等可编程器件的基础上进一步发展的产物。它是作为专用集成电路(ASIC)领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。**FPGA**的使用非常灵活，**同一片FPGA通过不同的编程数据可以产生不同的电路功能**。**FPGA**在通信、数据处理、网络、仪器、工业控制、军事和航空航天等众多领域得到了广泛应用。随着功耗和成本的进一步降低，**FPGA**还将进入更多的应用领域。目前市场上最大的**两个FPGA厂商分别为 Xilinx 和 Altera**。

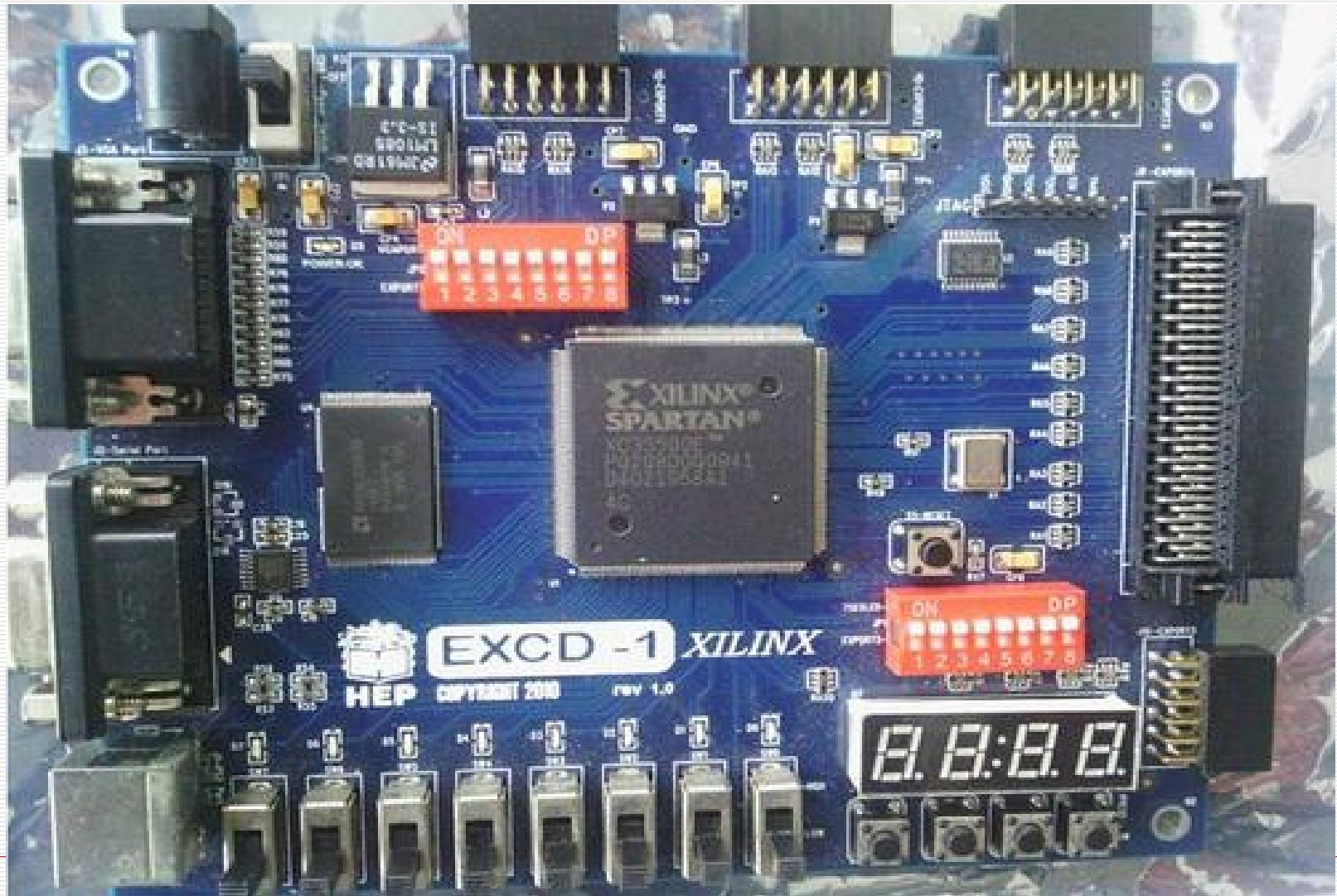


ALTERA FPGA开发板 (QUARTUS开发环境)



XILINX FPGA开发板

(ISE 或者 VIVADO 开发环境)

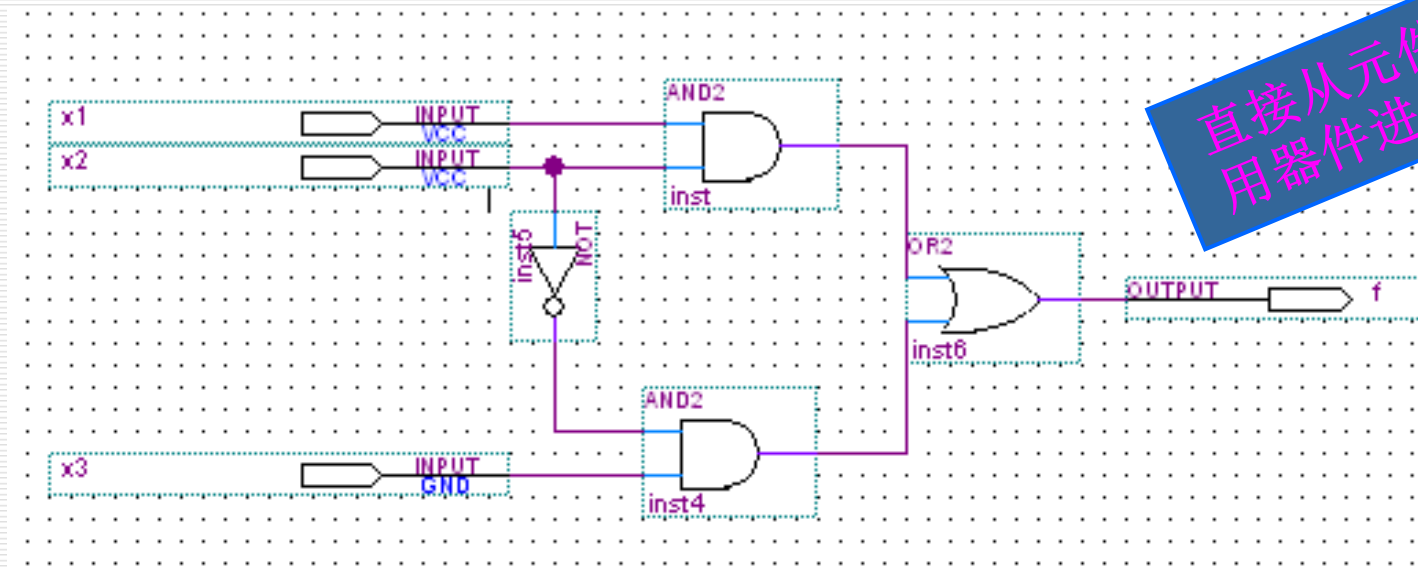
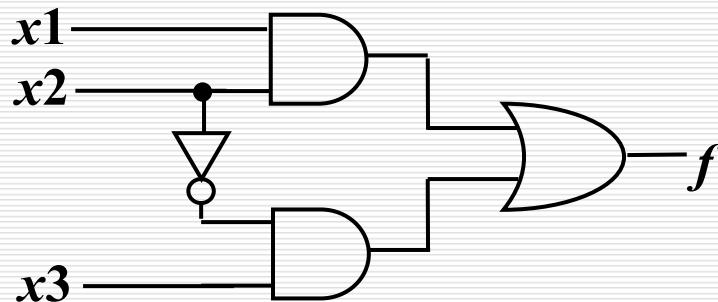


1、原理图设计

可编程器件的逻辑设计

元件库

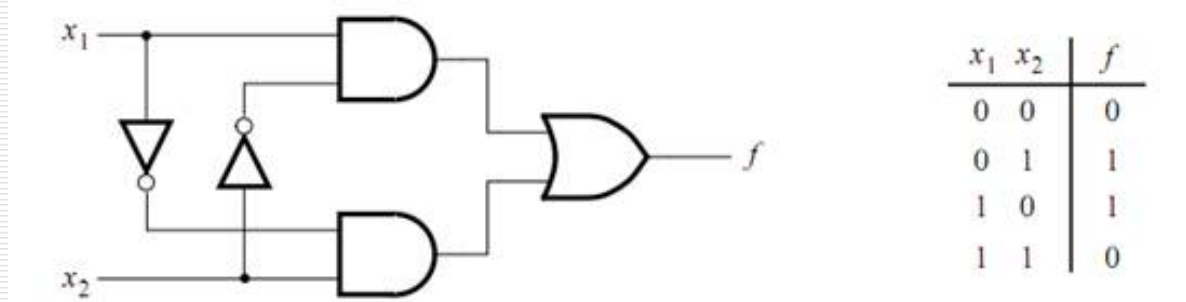
CAD工具提供一系列表示不同输入端数的各种类型门的图形符号。



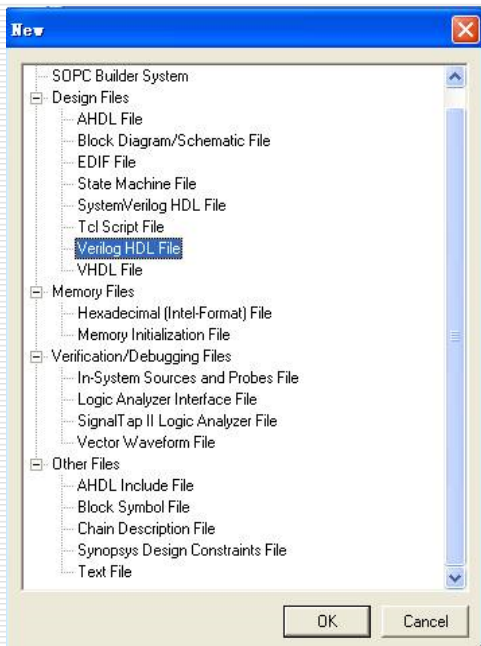
直接从元件库中调用器件进行设计。

2、用Verilog代码设计(与原理图类似)

- 实现一个2路输入控制灯开关的电路，如下图， x_1 ， x_2 为2个开关， f 为电路输出。



- 使用Quartus II的编程环境，在它的的文本编辑器输入设计。
点击菜单栏File > New出现下图，选择Verilog HDL File，ok确认。然后在编辑器里编辑代码如下。最后后点击菜单栏File > Save as, 保存文件名为light.v



→

```
1 module light(x1,x2,f);  
2   input x1,x2;  
3   output f;  
4   assign f=(x1 & ~x2) | (~x1 & x2);  
5 endmodule
```


两种常用的硬件描述语言VHDL和Verilog HDL

VHDL（了解）


VHDL主要用于描述数字系统的结构、行为、功能和接口。

Verilog HDL(主要介绍)

Verilog HDL是在C语言的基础上发展而来的硬件描述语言，具有简洁、高效、易用的特点。

二. Verilog HDL介绍

1、什么是Verilog HDL

- Verilog HDL是一种用于数字逻辑电路设计的硬件描述语言 (Hardware Description Language)，可以用来进行数字电路的仿真验证、时序分析、逻辑综合。
 - 用Verilog HDL描述的电路设计就是该电路的Verilog HDL模型。
 - Verilog HDL 既是一种行为描述语言也是一种结构描述语言。
 - 既可以用电路的功能描述，也可以用元器件及其之间的连接来建立Verilog HDL模型。
-

2、Verilog HDL的发展历史

- 1983年，由GDA（GateWay Design Automation）公司的Phil Moorby首创；
 - 1989年，Cadence公司收购了GDA公司；
 - 1990年，Cadence公司公开发表Verilog HDL；
 - 1995年，IEEE制定并公开发表Verilog HDL1364-1995标准；
 - 1999年，模拟和数字电路都适用的Verilog标准公开发表
-

2.5 硬件描述语言Verilog HDL基础

硬件描述语言HDL(Hardware Description Language)类似于高级程序设计语言. 它是一种以文本形式来描述数字系统硬件的结构和行为的语言, 用它来表示逻辑电路图、逻辑表达式, 复杂数字逻辑系统完成的逻辑功能。HDL是高层次自动化设计的起点和基础.

计算机对HDL的处理：

逻辑仿真

逻辑仿真 是指用**计算机仿真软件**对数字逻辑电路的结构和行为进行**预测**. 仿真器对HDL描述进行解释, **以文本形式或时序波形图形式**给出**电路的输出**。在仿真期间如发现设计中存在错误, 就要对HDL描述进行及时的修改。

逻辑综合

逻辑综合 是指从HDL描述的**数字逻辑电路模型**中**导出电路基本元件列表**以及**元件之间的连接关系**（常称为**门级网表**）的过程。类似对高级程序语言设计进行编译产生目标代码的过程. 产生门级元件及其连接关系的数据库, 根据这个数据库可以制作出**集成电路或印刷电路板PCB**。

2.5.1 Verilog语言的基本语法规则

为对数字电路进行描述（常称为建模），**Verilog语言**规定了一套完整的语法结构。

1. **间隔符**: Verilog 的间隔符主要起分隔文本的作用，可以使文本错落有致，便于阅读与修改。

间隔符包括空格符（\b）、**TAB 键**（\t）、换行符（\n）及换页符。

2. **注释符**: **注释**只是为了改善程序的可读性, 在编译时不起作用。

多行注释符 (用于写多行注释): **/* --- */**;

单行注释符 : 以**//**开始到行尾结束为注释文字。

3. 标识符和关键词

标识符:给对象（如模块名、电路的输入与输出端口、变量等）

取名所用的字符串。以英文字母或下划线开始

如，clk、counter8、_net、bus_A。

关键词:是Verilog语言本身规定的特殊字符串，用来定义语言的结构。例如，module、endmodule、input、output、wire、reg、and等都是关键词。关键词都是小写，关键词不能作为标识符使用。

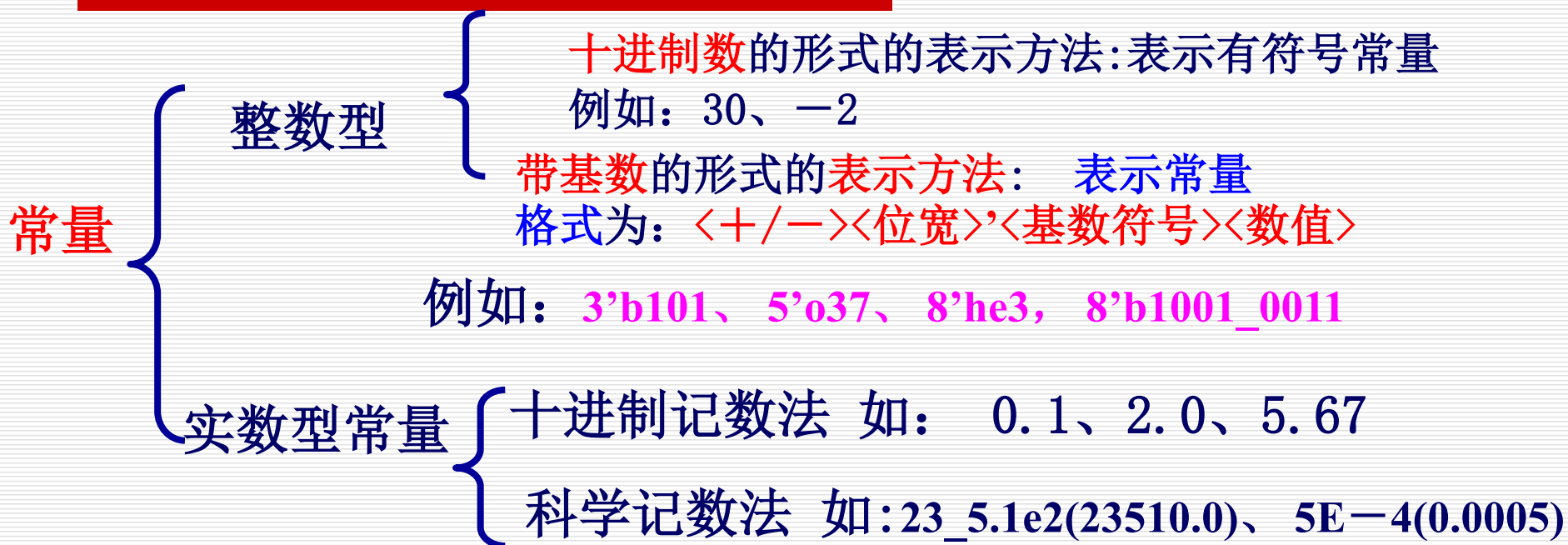
4. 逻辑值集合

为了表示数字逻辑电路的逻辑状态，Verilog语言规定了

4种基本的逻辑值。

0	逻辑0、逻辑假
1	逻辑1、逻辑真
x或X	不确定的值（未知状态）
z或Z	高阻态

5. 常量及其表示



用一个标识符来代表一个常量, 称为**符号常量**。定义的格式为:

parameter 参数名1=常量表达式1, 参数名2=常量表达式2, ...;
如 **parameter** BIT=1, BYTE=8, PI=3.14;

6. 字符串: 字符串是**双撇号**内的字符序列。

2.5.2 变量的数据类型

1、**线网类型**:是指输出始终根据输入的变化而更新其值的变量,它一般指的是硬件电路中的各种物理连接.

常用的**网络类型**由**关键词wire**定义

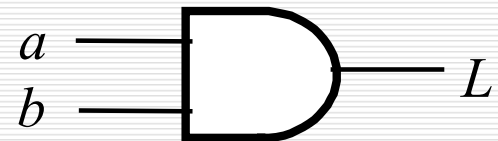
单线格式

wire 数据名1, 数据名2,, 数据名n;

例如: **wire** x,y,z; //定义3条信号线, 名字分别为: x,y,z

例:**wire** L; //将上述电路的输出信号L声明为网络型变量

例:网络型变量L的值由与门的驱动信号a和b所决定, 即 $L = a \& b$ 。a、b的值发生变化, 线网L的值会立即跟着变化。



□ wire型变量

- 常用来表示以assign语句赋值的组合逻辑信号。
- 模块中的输入/输出信号类型缺省为wire型。
- 可用做任何方程式的输入，或“assign”语句和实例元件的输出。

总线格式

wire[n-1:0] 数据名1, 数据名2,, 数据名m;
或 wire[n:l] 数据名1, 数据名2,, 数据名m;

变量宽度

例如: wire [16:1] datac; //声明一个16-bit宽的总线变量datac

例如: wire[7:0] m; //定义1条总线，名字为: m

总线位宽为8

该总线m中包含
m[7]~m[0]共8条信号线。

2、寄存器类型

4种寄存器类型的变量

寄存器型变量对应的是具有状态保持作用的电路元件等, 如触发器/寄存器。寄存器型变量只能在initial或always内部被赋值。

4种寄存器类型的变量

寄存器类型	功能说明
reg	常用的寄存器型变量
integer	32位带符号的整数型变量
real	64位带符号的实数型变量,
time	64位无符号的时间变量

抽象描述,
不对应具
体硬件

□ reg型变量

■ **定义**——在过程块中被赋值的信号，往往代表触发器，但不一定就是触发器（也可以是组合逻辑信号）！

单线格式

reg 数据名1, 数据名2,, 数据名n;

例如: **reg** x1,y1,z1; //定义3条寄存器型的信号线,
名字分别为: x1,y1,z1

总线格式

reg[n-1:0] 数据名1, 数据名2,, 数据名m;
或 **reg[n:1]** 数据名1, 数据名2,, 数据名m;

每个向量
位宽为n

共有m个reg
型向量

例: **reg** clock; //定义一个1位寄存器变量

reg [3:0] counter; //定义一个4位寄存器变量

➤ **[例]** **reg[4:1]** regc; //regc 为4位宽的reg型向量

❖ register型变量与wire型变量的根本区别是：

❖ register型变量需要被明确地赋值，并且在被重新赋值前一直保持原值。

❖ register型变量必须通过过程赋值语句赋值！不能通过assign语句赋值！

❖ 在过程块内被赋值的每个信号必须定义成register型！

reg与wire变量的区别举例

Verilog中reg与wire的区别

reg型变量既可生成触发器，也可生成组合逻辑； wire型变量只能生成组合逻辑。

➤ 用reg型变量生成组合逻辑举例：

```
module rw1( a, b, out1, out2 ) ;
```

```
    input a, b;
```

```
    output out1, out2;
```

```
    reg out1;
```

```
    wire out2;
```

连续赋值语句

```
    assign out2 = a ;
```

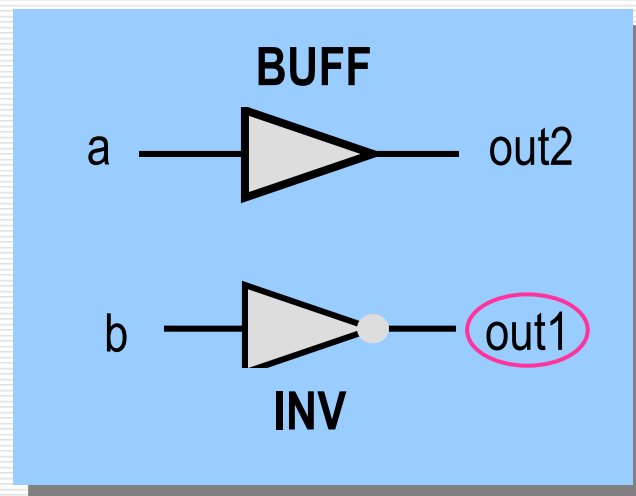
```
    always @(b)
```

电平触发

```
        out1 <= ~b;
```

```
endmodule
```

过程赋值语句



2.5.3 运算符及其优先级

1. 运算符

运算符分为算术运算符、逻辑运算符、关系运算符、移位运算符等

类型	符号	功能说明	类型	符号	功能说明
算术运算符	<div>+ - - * /</div>	二进制加 二进制减 2的补码 二进制乘 二进制除	关系运算符 (双目运算符)	<div>> < >= <= == !=</div>	大于 小于 大于或等于 小于或等于 相等 不相等
位运算符 (双目运算符)	<div>~ & ^ ^~ 或 ~^</div>	按位取反 按位与 按位或 按位异或 按位同或	缩位运算符 (单目运算符)	<div>& ~& ~ ^ ^~ 或 ~^</div>	缩位与 缩位与非 缩位或 缩位或非 缩位异或 缩位同或
逻辑运算符 (双目运算符)	<div>! && </div>	逻辑非 逻辑与 逻辑或	移位运算符 (双目运算符)	<div>>> <<</div>	右移 左移
位拼接运算符	<div>{} {}</div>	将多个操作数 拼接成为一个 操作数	条件运算符 (三目运算符)	<div>?:</div>	根据条件表达式是否成立，选择表达式

位拼接运算符{ }

作用是将两个或多个信号的某些位拼接起来成为一个新的操作数，进行运算操作。

设 $A=1'b1$ ， $B=2'b10$ ， $C=2'b00$

则 $\{B,C\}=4'b1000$

$\{A,B[1],C[0]\}=3'b110$

$\{A,B,C,3'b101\}=8'b11000101$ 。

对同一个操作数的重复拼接还可以双重大括号构成的运算符 $\{\{\}\}$

例如 $\{4\{A\}\}=4'b1111$ ， $\{2\{A\},2\{B\},C\}=8'b11101000$ 。

4个A进行拼接

2个A

2个B

1个C

所谓**缩位运算**，就是将这个数**自身的各位**进行相应的逻辑运算。

所谓**位运算**就是将两个数的对应位进行逻辑运算。

位运算符与缩位运算的比较

已知，**A: 4'b1010**、
B: 4'b1111,

位运算	$\sim A = 0101$ $\sim B = 0000$	$A \& B = 1010$	$A B = 1111$	$A \wedge B = 0101$	$A \sim \wedge B = 1010$
缩位运算	$\& A = 1 \& 0 \& 1 \& 0 = 0$	$\sim \& A = 1$ $\& B = 1$	$ A = 1$ $\sim B = 0$	$\wedge A = 0$ $\wedge B = 0$	$\sim \wedge A = 1$ $\sim \wedge B = 1$

2. 运算符的优先级

优先级的顺序从下向上依次增加。

类型	符号	优先级别
取反	! ~ -(求2的补码)	最高优先级
算术	* / + -	
移位	>> <<	
关系	< <= > >=	
等于	== !=	
缩位	& ~& ^ ^~ ~	
逻辑	&& 	
条件	?:	最低优先级

条件运算符 ? :

? : 是三目运算符，运算时根据条件表达式的值选择表达式。

一般用法：

condition_expr?expr1:expr2;

首先计算第一个操作数**condition_expr**的值，如果结果为逻辑1，则选择第二个操作数**expr1**的值作为结果返回，结果为逻辑0，选择第三个操作数**expr2**的值作为结果返回。

例如： **Z=X>Y? m : n;**

功能：当**X>Y**时，**Z=m**，否则**Z=n**;

2.5.4 Verilog内部的基本门级元件

门级建模:将逻辑电路图用HDL规定的文本语言表示出来。

基本门级元件模

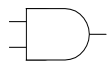
三态门

多输出门

多输入门

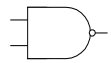
元件符号	功能说明	元件符号	功能说明
and	多输入端的与门	and	多输入端的与非门
or	多输入端的或门	nor	多输入端的或非门
xor	多输入端的异或门	xnor	多输入端的异或非门
buf	多输出端的缓冲器	not	多输出端的反相器
bufif1	控制信号高电平有效的三态缓冲器	notif1	控制信号高电平有效的三态反相器
bufif0	控制信号低电平有效的三态缓冲器	notif0	控制信号低电平有效的三态反相器

Verilog 基本门级元件



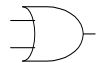
and

n-input AND gate



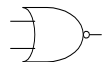
nand

n-input NAND gate



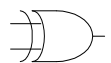
or

n-input OR gate



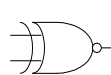
nor

n-input NOR gate



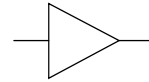
xor

**n-input exclusive
OR gate**



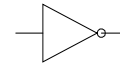
xnor

**n-input exclusive
NOR gate**



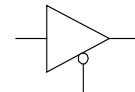
buf

n-output buffer



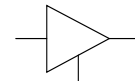
not

n-output inverter



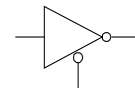
bufif0

**tri-state buffer;
Io enable**



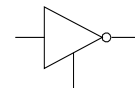
bufif1

**tri-state buffer;
hi enable**



notif0

**tri-state inverter;
Io enable**



notif1

**tri-state inverter;
hi enable**

1、多输入门 有and、or、xor等

只允许有一个输出，但可以有多多个输入。

and A1 (out, in1, in2, in3) ;

实例名

输出

库中元件名



and真值表

and		输入 1			
		0	1	X	Z
输入 2	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	Z	0	x	x	x

X- 不确定状态

Z- 高阻态

1、多输入门 有and、or、xor等

or真值表

or		输入1			
		0	1	X	Z
输入 2	0	0	1	X	X
	1	1	1	1	1
	X	X	1	X	X
	Z	X	1	X	X

xor真值表

xor		输入1			
		0	1	X	Z
输入 2	0	0	1	X	X
	1	1	0	X	X
	X	X	X	X	X
	Z	X	X	X	X

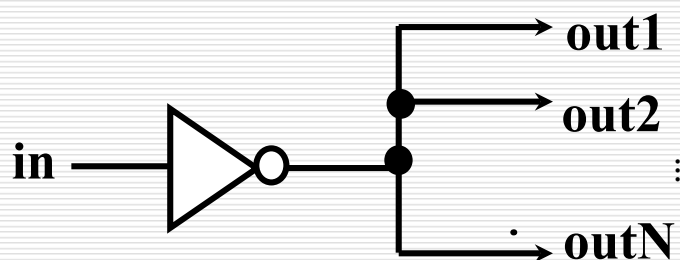
X- 不确定状态

Z- 高阻态

2、多输出门 有not、buf等

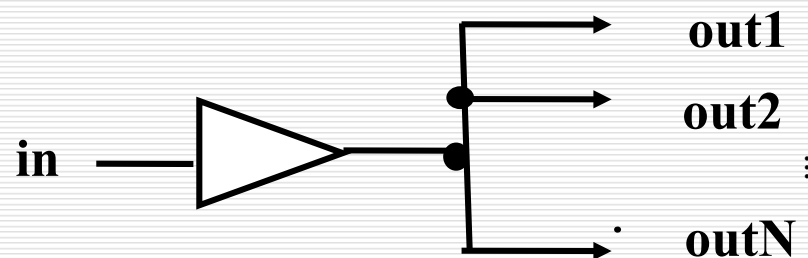
允许有多个输出，但只有一个输入。

not N1 (out1, out2, ..., in) ; buf B1 (out1, out2, ..., in) ;



not真值表

not	输入			
	0	1	x	z
输出	1	0	x	x



buf真值表

buf	输入			
	0	1	x	z
输出	0	1	x	x

X- 不确定状态

Z- 高阻态

3、三态门

有**bufif1**、**notif1**等

有一个输出、一个数据输入和一个输入控制。

如果输入控制信号无效，则三态门的输出为高阻态**z**。

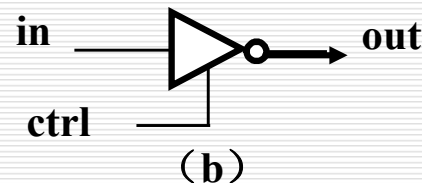
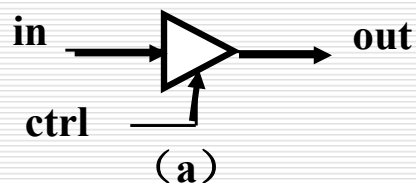


图 三态门元件模型
(a) bufif1 (b) notif1

bufif1真值表

bufif1		控制输入			
		0	1	x	z
数据输入	0	z	0	0/z	0/z
	1	z	1	1/z	1/z
	x	z	x	x	x
	z	z	x	x	x

notif1真值表

notif1		控制输入			
		0	1	x	z
数据输入	0	z	1	1/z	1/z
	1	z	0	0/z	0/z
	x	z	x	x	x
	z	z	x	x	x

2.5.5 Verilog程序的基本结构

模块是Verilog描述电路的基本单元。对数字电路建模时，用一个或多个模块。不同模块之间通过端口进行连接。

- 1、每个模块以关键词`module`开始，以`endmodule`结束。
- 2、每个模块先要进行端口的定义，并说明输入 (`input`)和输出 (`output`),然后对模块功能进行描述。
- 3、除了`endmodule`语句、`begin_end`语句和`fork_join`语句外，每个语句后必须有分号。
- 4、可以用`/* --- */`和`//.....`对程序的任何部分做注释。
- 5、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。

-
- 6、Verilog HDL程序是由**模块**构成的。模块是可以进行层次嵌套的。每个Verilog HDL**源文件中**只准有一个**顶层模块**，其他为**子模块**。
 - 7、序书写格式自由，一行可以写几个语句，一个语句也可以分多行写。
-

模块定义的一般语法结构如下：

module 模块名 (端口名1, 端口名2, 端口名3, ...);

端口类型说明(input, outout, inout);

参数定义(可选);

数据类型定义(wire, reg等);

} 说明部分

实例化低层模块和基本门级元件;

连续赋值语句(assign);

过程块结构(initial和always)

行为描述语句;

} 逻辑功能描述部分, 其顺序是任意的

endmodule

-
- Verilog的基本设计单元是“**模块** (block) ”。
 - Verilog 模块的结构由在**module**和**endmodule**关键词之间的**4**个主要部分组成：

- 1 端口定义
- 2 I/O说明
- 3 信号类型声明
- 4 功能描述

```
module block1(a, b, c, d );  
    input a, b, c;  
    output d;  
    wire x;  
    assign d = a | x;  
    assign x = ( b & ~c );  
endmodule
```

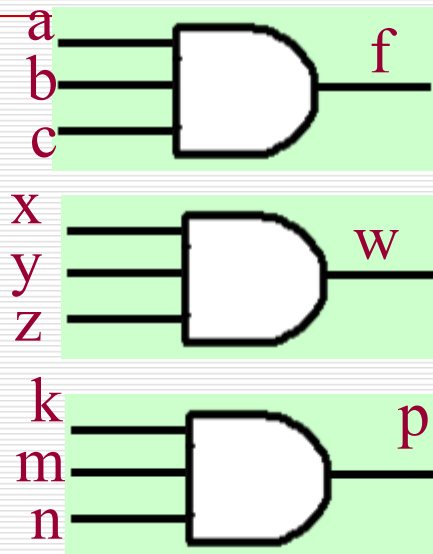
□ 在Verilog 模块中有2种常用方法可以描述电路的逻辑功能：

(1) 用assign 语句

连续赋值语句

```
assign x = ( b & ~c );
```

常用于描述
组合逻辑



(2) 元件例化 (instantiate)

门元件例化

```
and myand1( f,a,b,c);
```

门元件关键字

例化元件名

```
and myand2( w,x,y,z);
```

```
and myand3( p,k,m,n);
```

- ❖ 注1：元件例化即是调用Verilog HDL提供的元件；
- ❖ 注2：元件例化包括门元件例化和模块元件例化；
- ❖ 注3：每个实例元件的名字必须唯一！以避免与其它调用元件的实例相混淆。
- ❖ 注4：例化元件名也可以省略！

模块元件例化

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
```

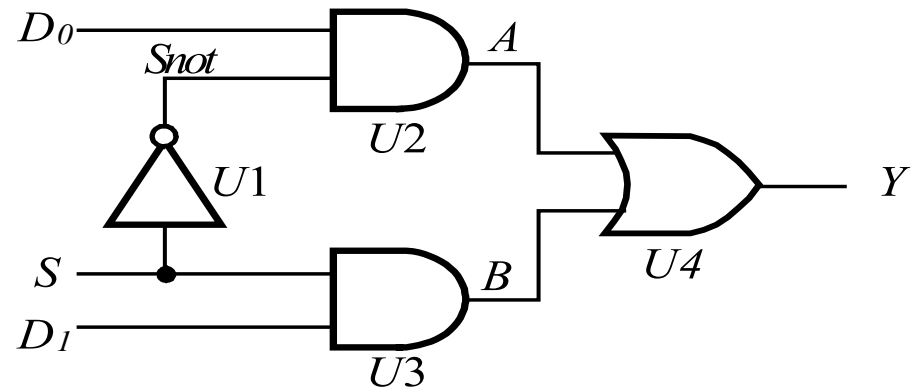
//下面对电路的逻辑功能进行描述

```
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

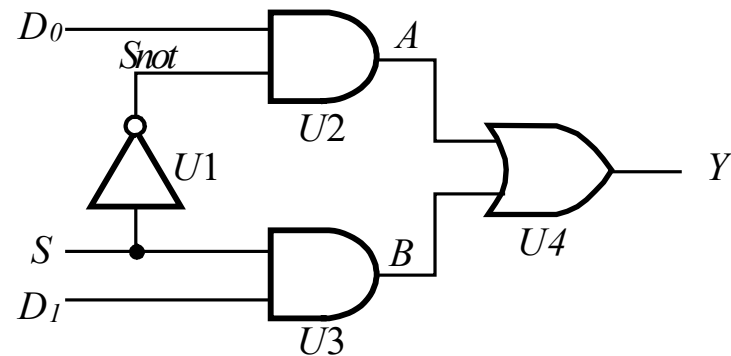
电路结构描述



— 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

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



```
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

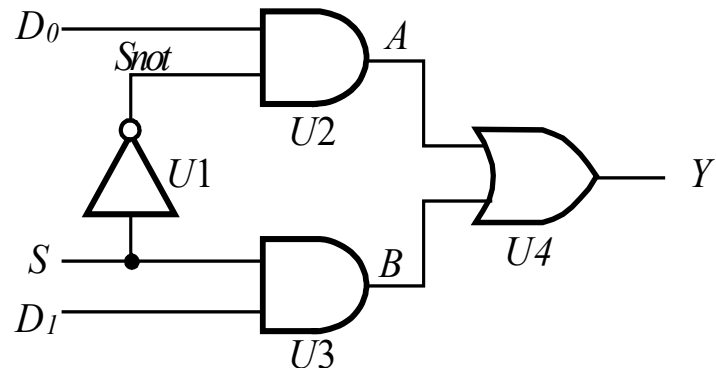
电路输出量描述

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

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



```
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
```

敏感信号列表

数据类型
说明

电路真值表的描述

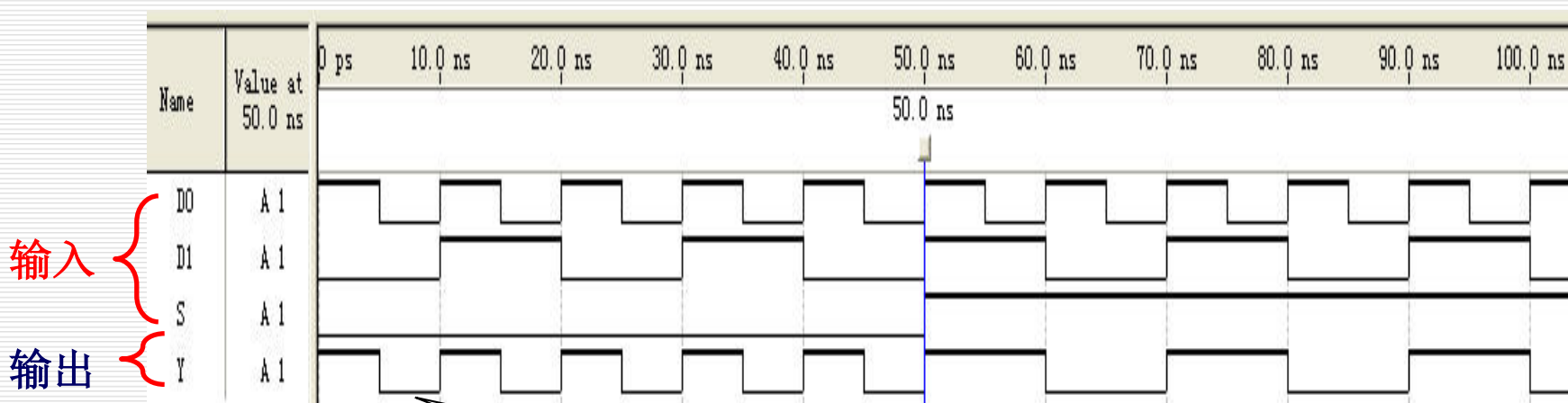
真值表

行为描述方式就是用真值表来描述电路的特性。

S	Y
0	D ₀
1	D ₁

2.5.6 逻辑功能的仿真与测试

逻辑电路的设计块完成后，就要测试这个设计块描述的逻辑功能是否正确。为此必须在输入端口加入测试信号，而从其输出端口检测其结果是否正确，这一过程常称为搭建测试平台。根据仿真软件的不同，搭建测试平台的方法也不同。



电路输入与输出量波形描述