数字系统设计习题
参考答案

3.6 请用开关代数定理化简下面的逻辑函数：

(a) $F = W \cdot X \cdot Y \cdot Z \cdot (W \cdot X \cdot Y \cdot Z' + W \cdot X' \cdot Y \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y' \cdot Z)$

$$
\begin{aligned}
F &= W \cdot X \cdot Y \cdot Z \cdot (W \cdot X \cdot Y \cdot Z' + W \cdot X' \cdot Y \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y' \cdot Z) \\
&= W \cdot X \cdot Y \cdot Z \cdot W \cdot X \cdot Y \cdot Z' + W \cdot X \cdot Y \cdot Z \cdot W \cdot X' \cdot Y \cdot Z \\
&\quad + W \cdot X \cdot Y \cdot Z \cdot W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z \cdot W \cdot X \cdot Y' \cdot Z && \text{(分配律)} \\
&= W \cdot W \cdot X \cdot X \cdot Y \cdot Y \cdot Z \cdot Z' + W \cdot W \cdot X \cdot X' \cdot Y \cdot Y \cdot Z \cdot Z \\
&\quad + W \cdot W' \cdot X \cdot X \cdot Y \cdot Y \cdot Z \cdot Z + W \cdot W \cdot X \cdot X \cdot Y \cdot Y' \cdot Z \cdot Z && \text{(交换律)} \\
&= W \cdot X \cdot Y \cdot 0 + W \cdot 0 \cdot Y \cdot Z + 0 \cdot X \cdot Y \cdot Z + W \cdot X \cdot 0 \cdot Z && \text{(同一律、互补律)} \\
&= 0 + 0 + 0 + 0 && \text{(空元素)} \\
&= 0 && (A4D)
\end{aligned}
$$

作业一

3.10 请写出下面各个逻辑函数的标准和及标准积：

(a) $F = \Sigma_{X,Y,Z}(0,1,3)$ (b) $F = \prod_{A,B,C}(0,2,4)$

课本 73~74 页

在真值表和最小项的对应关系基础上，很容易从真值表生成逻辑函数的代数表达式。一个逻辑函数的标准和（canonical sum）是使函数输出为 1 的真值表行（输入组合）所对应的最小项之和。例如，表 3-5 逻辑函数的标准和为：

逻辑函数的标准积（canonical product）是使函数输出为 0 的输入组合所对应的最大项之积。例如，表 3-5 中逻辑函数的标准积为：

最小项列表和最大项列表之间的转换是很容易的。对 n 变量函数，可能的最小项和最大项编号都是在集合 $\{0, 1, \cdots, 2^n-1\}$ 之中，最小项和最大项列表就是包括这些编号的一个子集。在列表类型之间转换，只需对集合求反，例如：

$$\Sigma_{A,B,C}(0,1,2,3) = \prod_{A,B,C}(4,5,6,7)$$

作业一

在真值表和最小项的对应关系基础上，很容易从真值表生成逻辑函数的代数表达式。一个逻辑函数的标准和（canonical sum）是使函数输出为 1 的真值表行（输入组合）所对应的最小项之和。例如，表 3-5 逻辑函数的标准和为：

完整的思路（以标准和为例）：

① 写出真值表

② 找出函数输出为 1 的真值表行

③ 写出这些行对应的最小项表达

④ 将对应最小项相加

作业一

行	X	Y	Z	F	最小项	最大项
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

(a) $F = X' \cdot Y' \cdot Z' + X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z$

$F = (X + Y' + Z) \cdot (X' + Y + Z) \cdot (X' + Y + Z') \cdot (X' + Y' + Z) \cdot (X' + Y' + Z')$

(b) $F = A' \cdot B' \cdot C + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' + A \cdot B \cdot C$

$F = (A + B + C) \cdot (A + B' + C) \cdot (A' + B + C)$

作业一

3.16 对下面每个逻辑表达式，用卡诺图找出相应的两级“与－或”或者“或－与”电路的所有静态冒险，并设计实现同样逻辑函数的无冒险电路。

(c) $F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$

对如何**画卡诺图**不了解的同学，请参考PPT课件DSD-02 第3节组合电路综合

对如何**利用卡诺图发现静态冒险**不了解的同学，请参考PPT课件DSD-03 第3节时序冒险

思路：

① 参考卡诺图步骤，画出完整卡诺图

② 找出全部的“相切现象”

③ 添加一致项

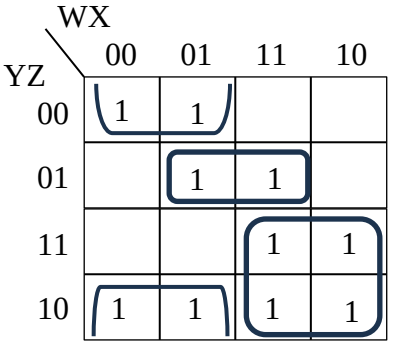
④ 写出最终的逻辑函数，如有需要画出对应的电路图

3.16 对下面每个逻辑表达式，用卡诺图找出相应的两级“与－或”或者“或－与”电路的所有静态冒险，并设计实现同样逻辑函数的无冒险电路。

(c) $F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$

画出卡诺图（多种画法）

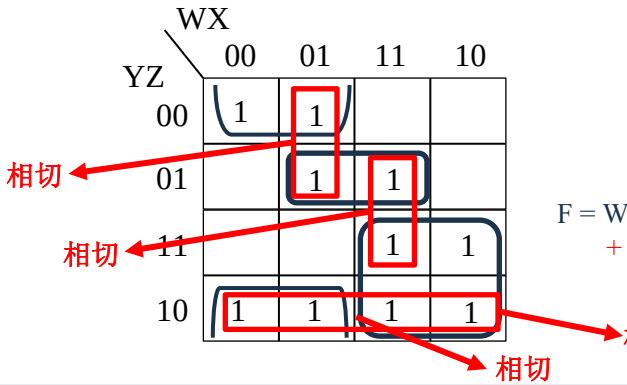
W	X	Y	Z	F
0	0	0	0	1
0	0	0	1	
0	0	1	0	1
0	0	1	1	
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	1
1	0	1	1	1
1	1	0	0	
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



F = W’Z’ + XY’Z + WY

3.16 对下面每个逻辑表达式，用卡诺图找出相应的两级“与－或”或者“或－与”电路的所有静态冒险，并设计实现同样逻辑函数的无冒险电路。

(c) $F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$



F = W’Z’ + XY’Z + WY
+ W’XY’ + WXZ + YZ’

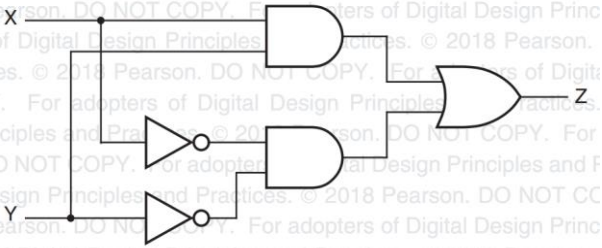
3.31 异或非（NOR 或 XNOR）门是 2 输入门，当且仅当两个输入相等时输出为 1。请写出异或非函数的真值表、“积之和”表达式和相应的“与－或”电路。

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

真值表

$Z = X \cdot Y + X' \cdot Y'$

标准和

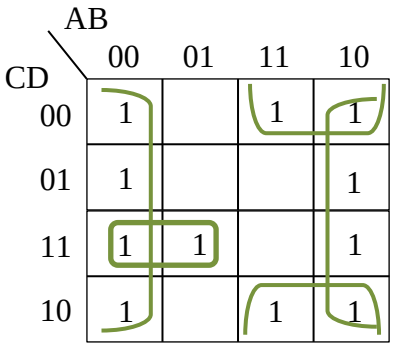


两级电路

3.48 求取下列各式的最简与或表达式（提示：利用卡诺图化简）。

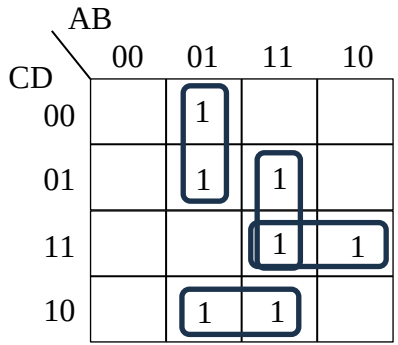
卡诺图化简

(e) $F = \prod_{A,B,C,D}(4,5,6,13,15)$



F = B’ + AD’ + A’CD

(f) $F = \sum_{A,B,C,D}(4,5,6,11,13,14,15)$



F = A’BC’ + ABD + ACD + BCD’

下以 $F = X \cdot Y + Z$ 为例，展示求解最简或与式的两种常见方式

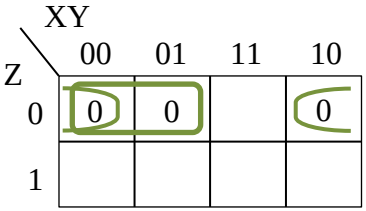
- 方式一：
  - 求得目标函数的最简与或式（利用卡诺图化简）
  - 求得最简与或式的反函数，并利用德摩根律化简
  - 展开化简后的式子，能获得反函数的最简与或式
  - 对反函数的最简与或式进行求反，再利用德摩根律化简即可得最简或与式

$$F = X \cdot Y + Z$$

$$\begin{aligned}\overline{F} &= \overline{X \cdot Y + Z} \\ &= \overline{X \cdot Y} \cdot \overline{Z} \\ &= (\overline{X} + \overline{Y}) \cdot \overline{Z} \\ &= \overline{X} \cdot \overline{Z} + \overline{Y} \cdot \overline{Z}\end{aligned}$$

$$\begin{aligned}\overline{\overline{F}} &= \overline{\overline{X} \cdot \overline{Z} + \overline{Y} \cdot \overline{Z}} \\ &= \overline{\overline{X} \cdot \overline{Z}} \cdot \overline{\overline{Y} \cdot \overline{Z}} \\ &= (X + Z) \cdot (Y + Z)\end{aligned}$$

- 方式二：
  - 在利用卡诺图化简的时候，将“画圈”的对象从“1”变成“0”（这样做能得到**反函数的最简与或式**）
  - 对反函数的最简与或式进行求反，再利用德摩根律化简即可得**最简或与式**



$$\overline{F} = \overline{X}\cdot\overline{Z} + \overline{Y}\cdot\overline{Z}$$

$$\overline{\overline{F}} = \overline{\overline{X}\cdot\overline{Z} + \overline{Y}\cdot\overline{Z}}$$

$$= \overline{\overline{X}\cdot\overline{Z}}\cdot\overline{\overline{Y}\cdot\overline{Z}}$$

$$= (X + Z)\cdot(Y + Z)$$

作业一

4.8 重新设计图 3-16 的报警电路，用反相门代替非反相门，并根据需要增加或者删除反相器。采用圈到圈的逻辑设计的思想画出电路的逻辑图，并对所有信号命名。

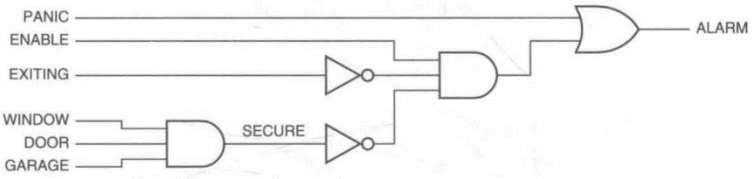
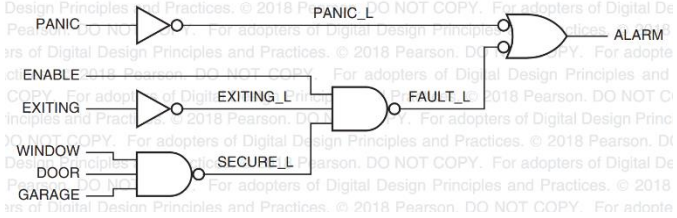


图 3-16 从逻辑表达式直接导出报警电路

圈到圈设计参考课本 104 页 4.1.6 节



5.2 编写图 6-15 中组合电路的一个结构化 Verilog 模块。

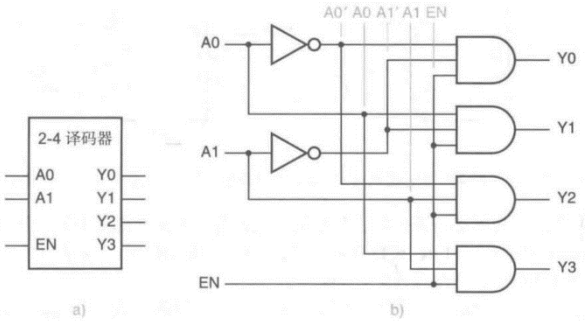


图 6-15 2-4 译码器：a）输入和输出；b）逻辑图

结构化描述方式： 是使用实例化低层次模块的方法，即调用其他已经定义过的低层次模块对整个电路的功能进行描述，或者直接调用Verilog内部预先定义的基本门级元件描述电路的结构。

数据流描述方式： 是使用连续赋值语句(assign)对电路的逻辑功能进行描述，该方式特别便于对组合逻辑电路建模。

行为级描述方式： 是使用过程块语句结构(always)和比较抽象的高级程序语句对电路的逻辑功能进行描述。

作业一

5.2 编写图 6-15 中组合电路的一个结构化 Verilog 模块。

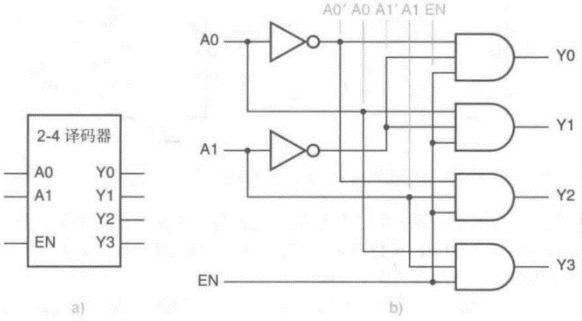


图 6-15 2-4 译码器：a）输入和输出；b）逻辑图

```
module Vr24decCkt (
  input A0, A1, EN,
  output Y0, Y1, Y2, Y3
);
  wire A0_L, A1_L;

  not U1 (A0_L, A0);
  not U2 (A1_L, A1);
  and U3 (Y0, A0_L, A1_L, EN);
  and U4 (Y1, A0,   A1_L, EN);
  and U5 (Y2, A0_L, A1,   EN);
  and U6 (Y3, A0,   A1,   EN);
endmodule
```

作业一

5.3 编写图 3-16 中报警电路的一个数据流风格 Verilog 模块。

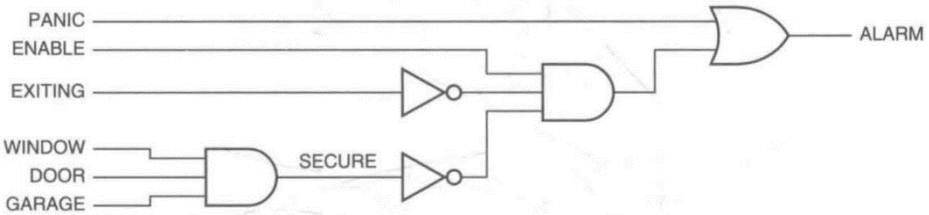


图 3-16 从逻辑表达式直接导出报警电路

```
module VrAlarmCktd (
  input panic, enable, exiting, window, door, garage,
  output alarm
);
  wire secure;

  assign secure = window & door & garage;
  assign alarm = panic | ( enable & ~exiting & ~secure );
endmodule
```

5.18 BUT 门的一个可能的定义是“如果 A1 和 B1 是 1，但 A2 或 B2 有一个是 0，则 Y1 是 1，Y2 的定义与 Y1 对称”。为这样的 BUT 门编写一个行为化风格的 Verilog 模块。

```
module VrButGate
  (A1, B1, A2, B2, Y1, Y2);
input A1, B1, A2, B2;
output reg Y1, Y2;

always @ (A1, B1, A2, B2)
  begin
    if ( (A1&B1)==1'b1 && (A2&B2)==1'b0 ) Y1 = 1;
    else Y1 = 0;
    if ( (A2&B2)==1'b1 && (A1&B1)==1'b0) Y2 = 1;
    else Y2 = 0;
  end
endmodule
```

作业一

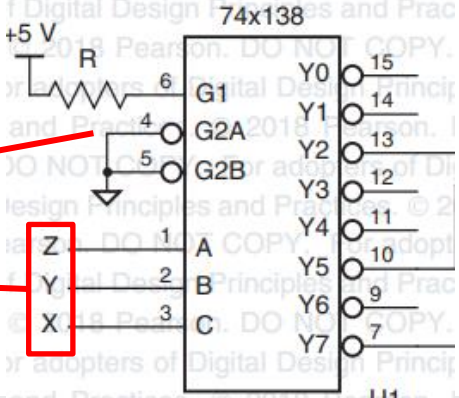
6.7 指出用一块或多块 74x138 二进制译码器以及与非门，如何构建下面每个单输出或多输出的逻辑功能（提示：每个实现应该等效于一个最小项之和）。

(b) $F = \prod_{A,B,C}(2,4,5,6,7)$　　(c) $F = \sum_{A,B,C,D}(0,6,10,14)$

① 不要改器件端口！！！

② 使能端口也要指明

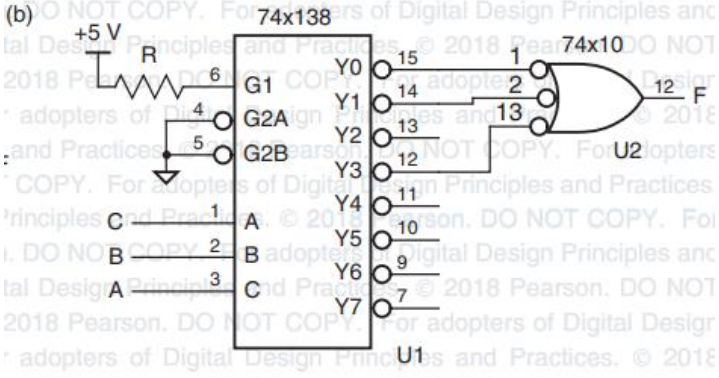
③ 注意输入数据的顺序



作业一

6.7 指出用一块或多块 74x138 二进制译码器以及与非门，如何构建下面每个单输出或多输出的逻辑功能（提示：每个实现应该等效于一个最小项之和）。

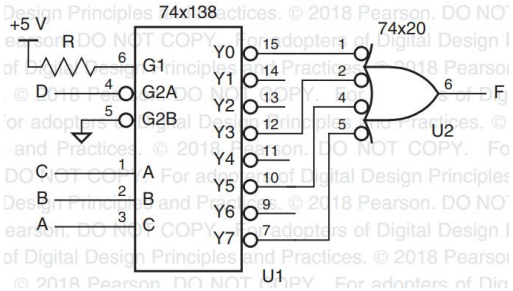
(b) $F = \prod_{A,B,C}(2,4,5,6,7)$ (c) $F = \sum_{A,B,C,D}(0,6,10,14)$

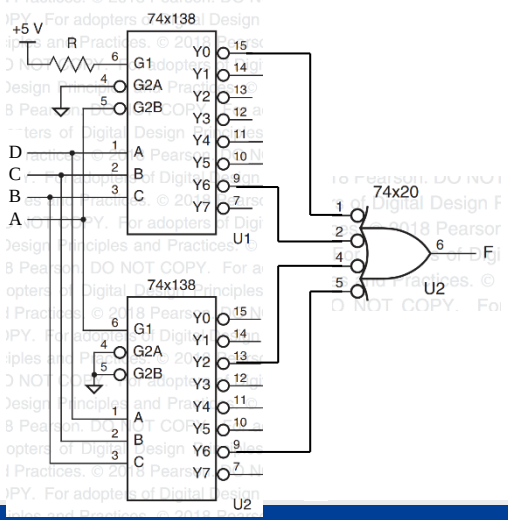


6.7 指出用一块或多块 74x138 二进制译码器以及与非门，如何构建下面每个单输出或多输出的逻辑功能（提示：每个实现应该等效于一个最小项之和）。

(c) $F = \sum_{A,B,C,D}(0,6,10,14)$

有多种答案，想求稳的话，可以像右边一样，变量数>3直接用多块译码器

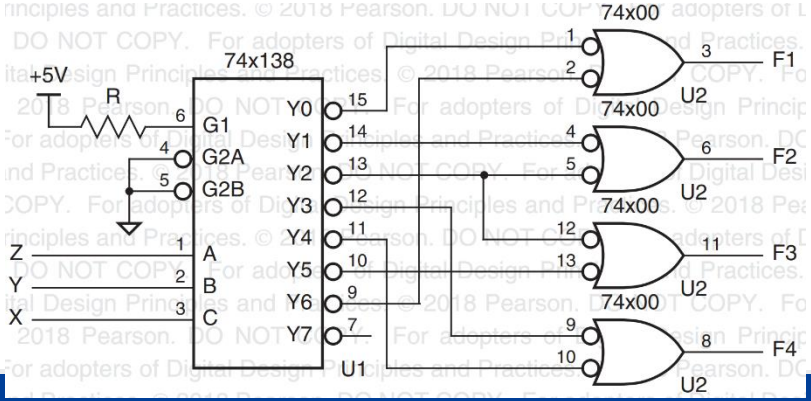




6.32 请表示出如何用一个输出低电平有效的 3-8 译码器和四个 2 输入与非门构建下面所有的 4 个函数：

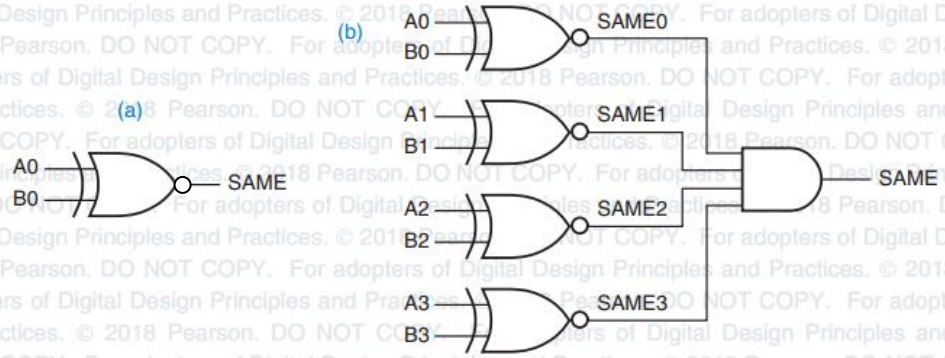
$F1 = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z'$　　$F2 = X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z'$

$F3 = X' \cdot Y \cdot Z' + X \cdot Y' \cdot Z$　　$F4 = X \cdot Y' \cdot Z' + X' \cdot Y \cdot Z$



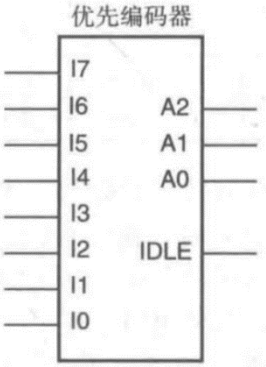
7.9 按照图 7-21b 的风格，用同或门（XNOR）和与门画出一个 4 位比较器的逻辑图，确保使用与其有效电平相对应的逻辑符号和信号名。

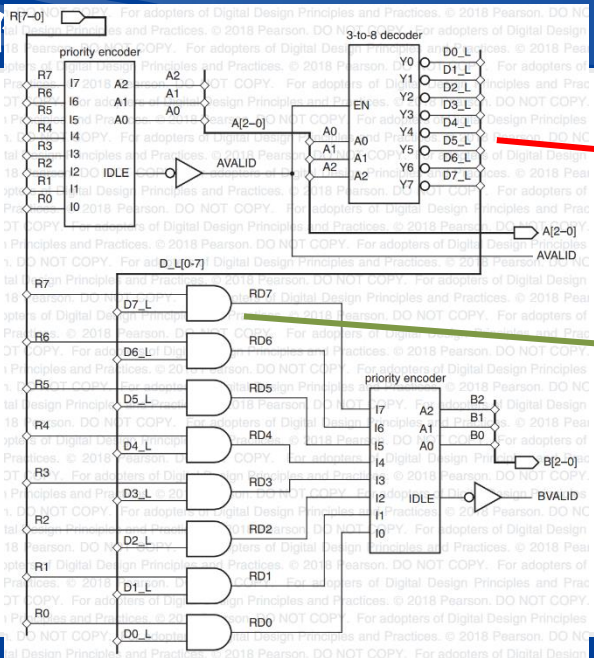
7.9



作业二

7.26 设计一个组合逻辑电路，它有 8 个高电平有效的请求输入 R0 ~ R7 和 8 个输出 A2 ~ A0、AVALID、B2 ~ B0 及 BVALID，其中输入 R7 的优先级最高，输出“A”识别出最高优先级的有效输入，输出 B 识别出优先级第二高的有效输入。你的设计可以使用分立门电路、译码器以及图 7-11 中的 8 输入优先编码器。





优先编码器+译码器能过滤出最高优先级的输入

之后再通过与门，将最高优先级的输入屏蔽。从而使得第二个优先编码器输出第二高优先级的输入

8.16 假设图 8-8 的 4 位加法器没有输出 C4。(有些带有组间先行进位输出的 MSI 加法器就是这种情况。)写出整个加法的进位输出("C16")关于图中现有信号的逻辑方程。

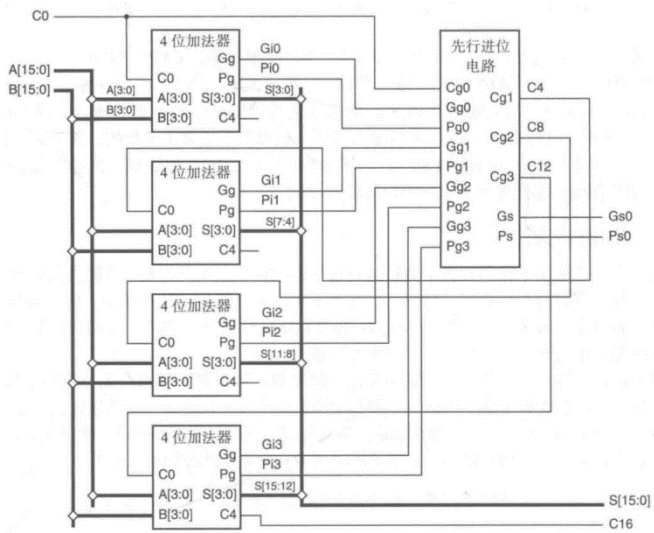


图 8-8 4 位一组的 16 位组间先行进位加法器

$$C16 = Gs0 + C0 \cdot Ps0$$

参考**课本 281 页往后** 或 **PPT课件DSD-08 第一节加法和减法**

9.9 分析图 9-33 中的状态机。写出激励方程、激励/转移表以及状态/输出表（状态 Q1Q2Q3 = 000 ~ 111 使用状态名 A ~ H）。

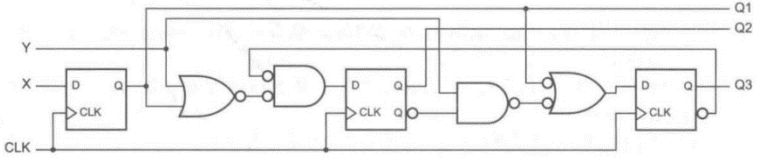


图 9-33

在图 9-8 中有两个 D 触发器，其输出端的信号分别记为 Q0 和 Q1。这两个输出就是状态变量，它们的值就是状态机当前的状态值。与之相对应，将两个 D 触发器的 D 输入信号分别记为 D0 和 D1。这些输入信号在每一个时钟触发沿向 D 触发器提供激励（excitation）。

激励信号为现态和输入的函数，创建这些函数的电路通常被称为激励逻辑（excitation logic）。可以从逻辑图中导出激励方程（excitation equation）：

$$D0 = Q0 \cdot EN' + Q0' \cdot EN$$
$$D1 = Q1 \cdot EN' + Q1' \cdot Q0 \cdot EN + Q1 \cdot Q0' \cdot EN$$

激励方程定义
课本 336 页

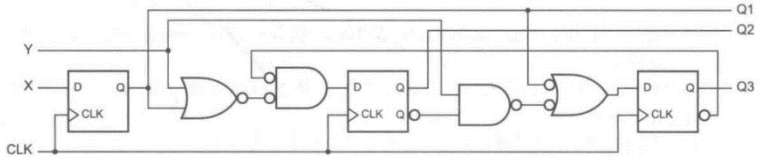


图 9-33

9.9 激励方程

$$D1 = X$$

$$D2 = (Q1 + Y) \cdot Q3$$

$$D3 = Q2' \cdot Y + Q1'$$

作业二

9.9 分析图 9-33 中的状态机。写出激励方程、激励 / 转移表以及状态 / 输出表（状态 Q1Q2Q3 = 000 ~ 111 使用状态名 A ~ H）。

这些方程式把状态变量的下一个值表示成现态和输入的函数，称之为转移方程（transition equation）。

表 9-1a 展示了对应可能的状态 / 输入组合，利用转移方程进行计算而得到的转移表（transition table）。按照惯例，转移表在左边列出状态变量的取值组合，在表的上边列出输入组合，具体形式如上例所示。

转移方程、转移表
课本 336 页

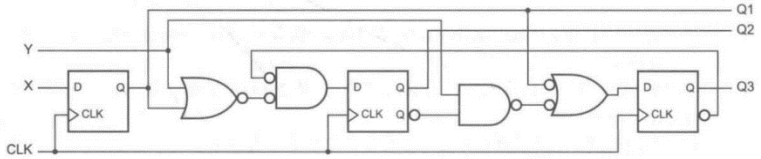


图 9-33

9.9 转移表

Q1 Q2 Q3	XY 00	01	10	11
000	001	001	101	101
001	001	011	101	111
010	001	001	101	101
011	001	011	101	111
100	**000**	001	100	101
101	**010**	011	110	111
110	**000**	000	100	100
111	**010**	010	110	110
	Q1* Q2* Q3*			

9.9 分析图 9-33 中的状态机。写出激励方程、激励 / 转移表以及状态 / 输出表（状态 Q1Q2Q3 = 000 ~ 111 使用状态名 A ~ H）。

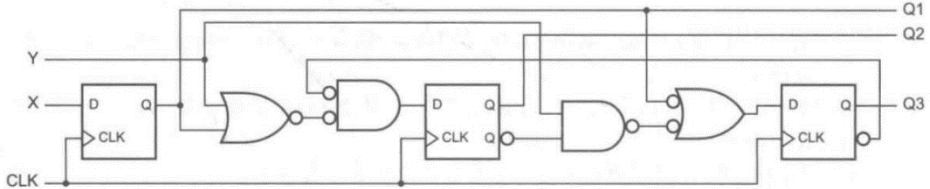


图 9-33

2. 用 F 和 G 构造一个状态 / 输出表（state/output table）。对于现态和输入的每一个可能组合，这个表都会完全地指定电路的次态和输出。

状态/输出表
课本 335 页

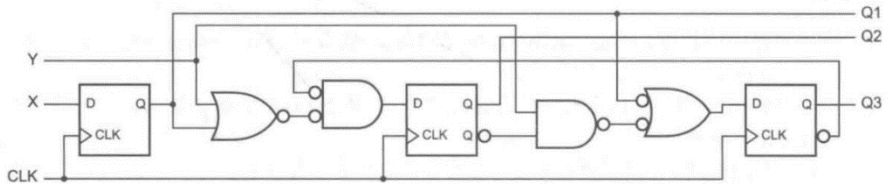


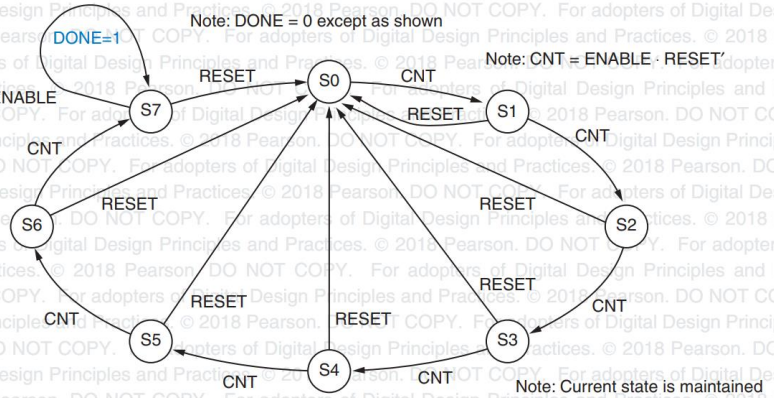
图 9-33

9.9 状态/输出表

S	XY = 00	XY = 01	XY = 10	XY = 11
A	B	B	F	F
B	B	D	F	H
C	B	B	F	F
D	B	D	F	H
E	C	B	E	F
F	A	D	G	H
G	C	A	E	E
H	B	C	G	G
	S*			

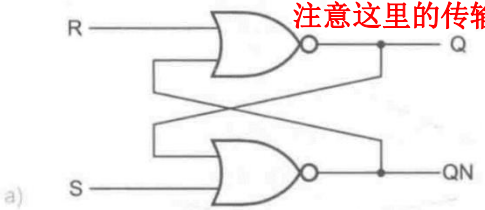
这类题目的流程固定，
一定要注意表格的格式

9.30 给一个“粘性计数器”状态机创建一个无歧义的状态图或 ASM 图，该状态机有 8 个状态 S0 ~ S7。除了 CLOCK，这个状态机还应该有两个输入 RESET 和 ENABLE，以及一个输出 DONE。每当 RESET 有效时，状态机就进入状态 S0。当 RESET 无效且 ENABLE 有效时，状态机进入下一计数状态。然而，一旦到达状态 S7，就会停留在这个状态直到 RESET 再次有效。当且仅当状态机的状态为 S7 且 ENABLE 有效时，输出 DONE 为 1。



10.9 画出图 10-4 中 S-R 锁存器的输出波形，其输入波形如图 10-47 所示。假设输入和输出信号的上升和下降时间为 0，或非门的传输延迟是 10ns（图中每个时间分段是 10ns）。

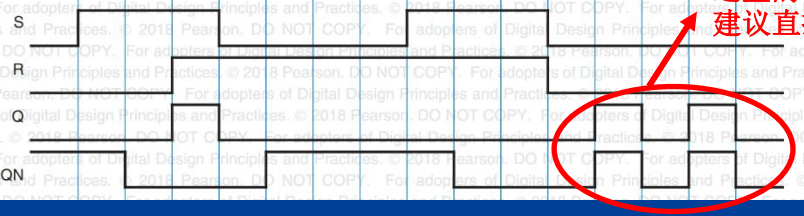
注意这里的传输延迟！！！



S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

图 10-4　S-R 锁存器：a）使用或非门设计的电路；b）功能表

10.9 The latch oscillates if S and R are negated simultaneously. Many simulator programs will exhibit this same behavior when confronted with such input waveforms.



这里的不确定状态，建议直接画成阴影

作业三

10.16 一个上升沿触发的 S-R 触发器具有两个控制输入 S 和 R，它们的作用和 S-R 锁存器中的相同，只不过这里只在 CLK 输入的上升沿对控制输入进行采样并改变输出的状态。请说明怎样使用一个 D 触发器和组合逻辑来建立一个置位优先的 S-R 触发器。

非常完整的套路：

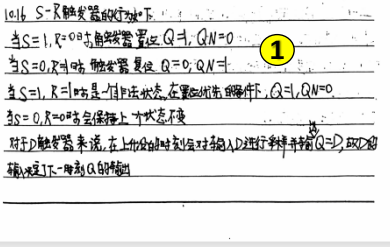
① 找到输入和输出，用遍历的方式描述输入和输出之间的变化

② 根据上一步的描述，绘制出真值表

③ 根据真值表绘制出对应的卡诺图

④ 根据卡诺图获得最简与或/或与表达式

⑤ 根据表达式绘制对应的逻辑电路



非常完整的套路：

1. 找到输入和输出，用遍历的方式描述输入和输出之间的变化
2. 根据上一步的描述，绘制出真值表
3. 根据真值表绘制出对应的卡诺图
4. 根据卡诺图获得最简与或/或与表达式
5. 根据表达式绘制对应的逻辑电路

10.30 通过分析图 10-9 中 D 锁存器内部的反馈回路，来解释如果 D 输入信号在 G 信号的建立或保持时间内发生变化，亚稳定性是如何出现的。

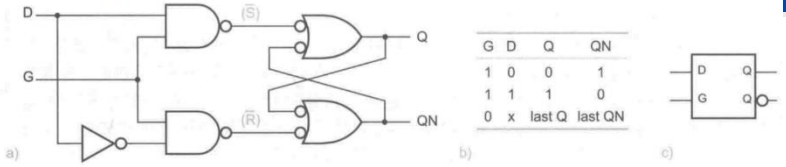
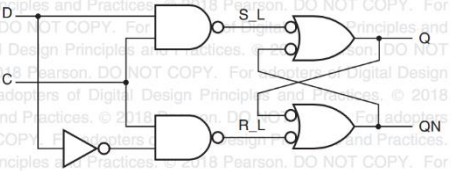


图 10-9 D 锁存器：a）使用与非门设计的电路；b）功能表；c）逻辑符号

10.30 Refer to the figure to the right. The details of metastability triggering depend on the details of the internal circuit timing.



For example, suppose that the inverter has nonzero delay, and that C and D are 1. Then when D changes from 1 to 0, both S_L and R_L will be asserted for a short time. If C changes from 1 to 0 during this time, then S_L and R_L will be negated simultaneously; this is analogous to the situation in the previous exercise.

On the other hand, let us assume for the sake of argument that the inverter has zero delay, and that C and D are 1. The feedback loop is in one of the stable states depicted in Figure 7–3. If D is changed from 1 to 0, and assuming that the delays through the NAND gates to S_L and R_L are equal, the "operating point" of the feedback loop starts moving toward the other stable state. However, if C is changed to 0 while this is happening, the circuit may stop halfway at the metastable operating point.

作业四

11.38 CNTR4UD 是和二进制计数器 CNTR4U 具有相同输入输出的升序/降序计数器，增加一个 UP/DN 输入，用于控制计数器是升序（UP/DN=1）还是降序计数。RCO 输出的功能也取决于 UP/DN；当升序计数时它在 1111 状态有效，当是降序计数时它在 0000 状态有效。采用一个 CNTR4UD 和最多一个分立逻辑门，设计一个模 16 计数器。计数序列如下：7，6，5，4，3，2，1，0，8，9，10，11，12，13，14，15，7，…

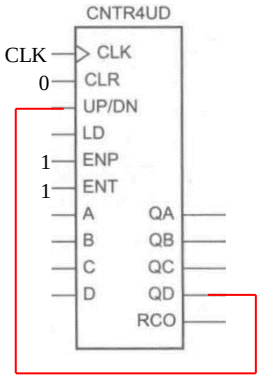
对CNTR4U不了解的同学从课本 11.1.3 节，418 页开始看

CNTR4U 相关题目我一般只关注两点：1）**什么时候复位**，复位的时机可以由输出信号 Q 相关的组合逻辑电路决定（或者用RCO信号），复位信号一般选用 LD （如果是归零的话就用 CLR）；2）**复位到什么值**，一般由输入信号 D 决定，一些题也能用输出信号的组合（或者其他特殊信号的组合）实现

作业四

这道题增加了一个新的输入信号：升序/降序

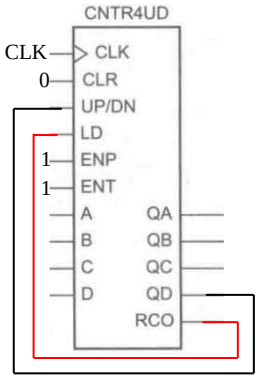
- 观察计数序列可以发现从 **7~0 是降序**的，而从 **8~15 是升序**的。它们的区别在于：7~0 的最高位是 0，而 8~15 的最高位是 1。因此我们可以使用**输出信号的最高位来控制升序/降序**



作业四

剩下要解决的点在于，如何处理0→8和15→7的跳变，这其实就可以理解为两次复位

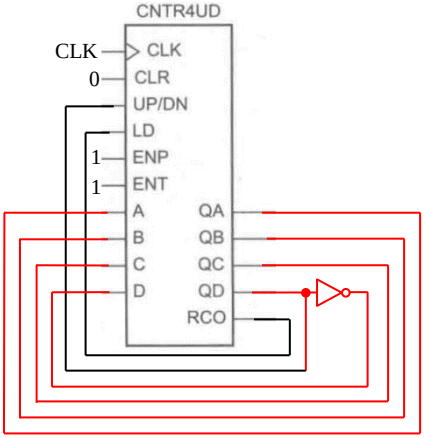
- 什么时候复位：输出信号为 0000 （0）时，根据前一步结果可知 UP/DN 输入信号为 0，再根据题目描述可知 RCO 输出信号此时为 1；输出信号为 1111 （15）时，根据前一步结果可知 UP/DN 输入信号为 1，再根据题目描述可知 RCO 输出信号此时为 1。其余时间 RCO 信号均为0，因此只要 RCO 信号为 1 就触发复位，即将 RCO 输出与 LD 输入相连



作业四

处理0→8和15→7的跳变

- 复位到何值：输出信号为 0000 （0）时，需要复位到 1000（8）；输出信号为 1111（15）时，需要复位到 0111（7）。总结一下规律可以发现，复位值就是将输出信号的最高位取反。因此我们只需要把输出最高位取反后连接到输入最高位，再把输出的剩余位直接连到输入剩余位即可。



作业四

11.59 只用 4 个 D 触发器和 8 个门电路，设计一个 4 位 Johnson 计数器，并对 8 个计数状态进行译码。计数器不需要自校正功能。

*11.2.4 Johnson 计数器

把 n 位移位寄存器的串行输出取反，反馈到串行输入端，就构成了一个具有 2^n 种状态的计数器，称为扭环形（twisted-ring，或者 Moebius，或者 Johnson）计数器。图 11-22 就是这种计数器的基本电路，图 11-23 是其对应的时序图。这个计数器的正常状态列在表 11-3

课本 432 页

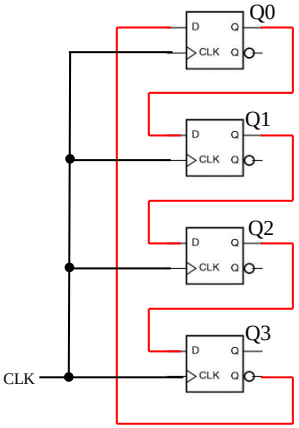
移位寄存器在课本 429 页

作业四

11.59 只用 4 个 D 触发器和 8 个门电路，设计一个 4 位 Johnson 计数器，并对 8 个计数状态进行译码。计数器不需要自校正功能。

Johnson计数器其实就是不停左移，然后把最高位翻转放到最低位。四个触发器就把前一个输出连到后一个输入即可，只不过最低位输入需要接收最高位输出的取反

状态	Q3	Q2	Q1	Q0
S0	0	0	0	0
S1	0	0	0	1
S2	0	0	1	1
S3	0	1	1	1
S4	1	1	1	1
S5	1	1	1	0
S6	1	1	0	0
S7	1	0	0	0

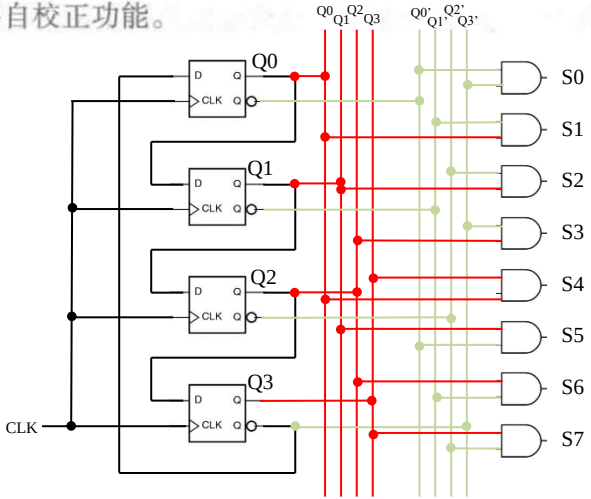


作业四

11.59　只用 4 个 D 触发器和 8 个门电路，设计一个 4 位 Johnson 计数器，并对 8 个计数状态进行译码。计数器不需要自校正功能。

之后译码，暴力点的方法就是给每个状态都配一个四输入的与门，因为同时使用4个D触发器的输出肯定能唯一界定每个状态。但我们仔细观察下，其实每个状态都可只用2个D触发器的输出界定，如下表的红框所示

状态	Q3	Q2	Q1	Q0
S0	0	0	0	0
S1	0	0	0	1
S2	0	0	1	1
S3	0	1	1	1
S4	1	1	1	1
S5	1	1	1	0
S6	1	1	0	0
S7	1	0	0	0



12.21 给有两个输入（X 和 INIT）和两个 Moore 型输出（EDGE 和 MISS）的状态机编写一个 Verilog 模块 Vredgemiss。状态机要可靠地检测输入 X 上的变化。状态机在每个时钟沿检测输入 X，如果在这个时钟沿的 X 值与前面一个时钟沿的值不同，状态机就使 EDGE 有效。EDGE 一旦有效，就一直保持有效直到 INIT 有效且至少维持一个时钟周期。如果在 EDGE 有效后 INIT 有效前，错过了一个或多个时钟沿，那么输出 MISS 有效。并且 MISS 在 INIT 有效前一直维持有效。注意“边界”情况。特别是，INIT 正好在一个时钟沿上有效，则仍然是 EDGE 有效，而 MISS 无效。

```
module Vredgemiss( CLOCK, INIT, X, EDGE, MISS );
  input CLOCK, INIT, X;
  output reg EDGE, MISS;
  reg LASTX;                          // Keep track of previous value of X

  // Note, no reset input.
  // Circuit is deterministically reset to known state in 2 ticks
  //   by INIT=1, X=0, for example.

  always @ (posedge CLOCK) begin    // Create the state memory and behavior together
    if (INIT) MISS <= 0;              // On INIT clear MISS if any
    else MISS <= MISS |               // Else set MISS if previous MISS or
                 (EDGE & (X != LASTX)); //   unacknowledged edge plus new edge
    EDGE <= (EDGE & ~INIT) | (X != LASTX);  // Set unacknowledged edge or new edge
    LASTX <= X;
  end
endmodule
```