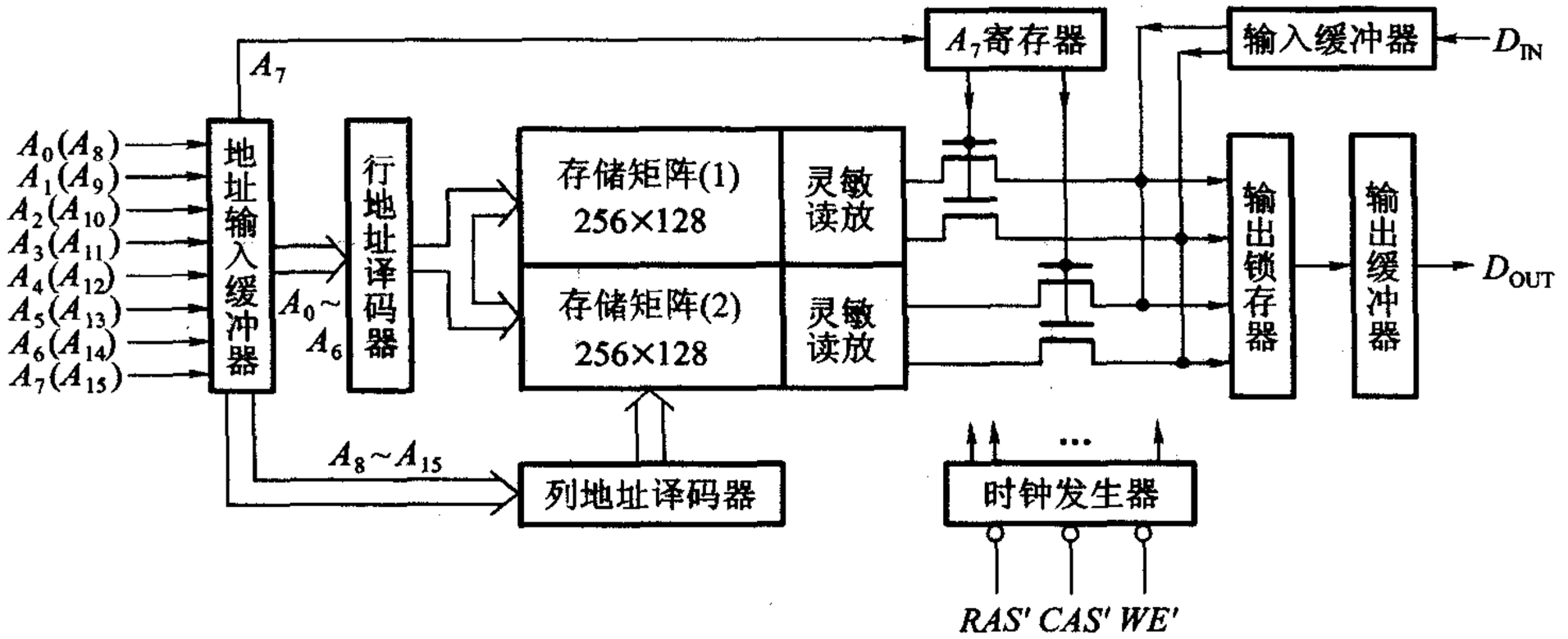


图 7.3.9　DRAM 的总体结构框图

出电路从两行中选出一行。A_8 ~ A_{15}被送往列地址译码器，列地址译码器的输出从 256 列中选中一列。

当 $WE'=1$ 时进行读操作，被输入地址代码选中单元中的数据经过输出锁存器、输出三态缓冲器到达数据输出端 D_{OUT}。当 $WE'=0$ 时进行写操作，加到数据输入端 D_{IN}的数据经过输入缓冲器写入由输入地址指定的单元中去。

复习思考题

R7.3.1　静态随机存储器和动态随机存储器的根本区别是什么？它们各有何优、缺点？各适用于什么场合？

R7.3.2　RAM 和 ROM 的主要区别是什么？它们各适用于哪些场合？

7.4　存储器容量的扩展

当使用一片 ROM 或 RAM 器件不能满足对存储容量的要求时，就需要将若干片 ROM 或 RAM 组合起来，形成一个容量更大的存储器。

7.4.1　位扩展方式

如果每一片 ROM 或 RAM 中的字数已经够用而每个字的位数不够用，则应采用位扩展的连接方式，将多片 ROM 或 RAM 组合成位数更多的存储器。

RAM 的位扩展连接方法如图 7.4.1 所示。在这个例子中,用 8 片 1024 × 1 位的 RAM 接成了一个 1024 × 8 位的 RAM。

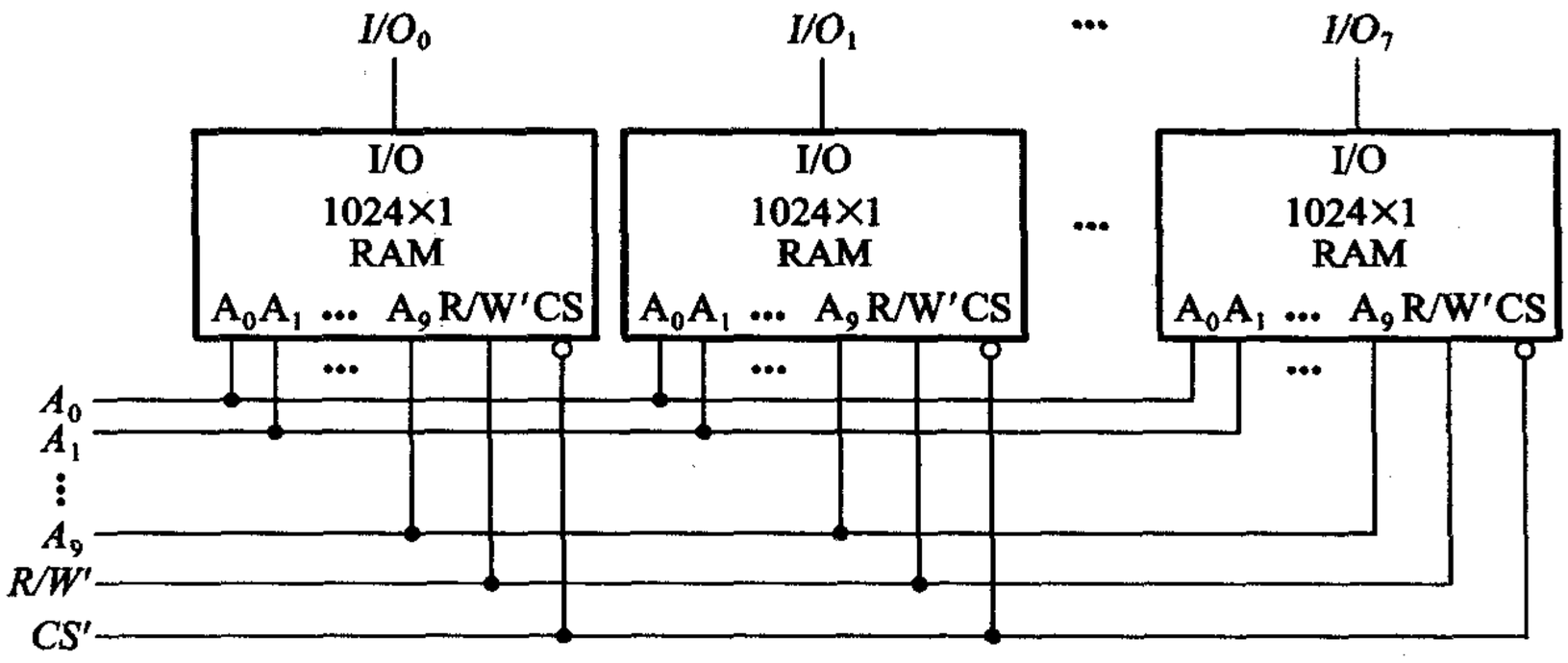


图 7.4.1 RAM 的位扩展接法

连接的方法十分简单,只需将 8 片的所有地址线、R/W'、CS'分别并联起来就行了。每一片的 I/O 端作为整个 RAM 输入/输出数据端的一位。总的存储容量为每一片存储容量的 8 倍。

ROM 芯片上没有读/写控制端 R/W',在进行位扩展时其余引出端的连接方法和 RAM 完全相同。

7.4.2 字扩展方式

如果每一片存储器的数据位数够用而字数不够用,则需要采用字扩展方式,将多片存储器(RAM 或 ROM)芯片接成一个字数更多的存储器。

图 7.4.2 是用字扩展方式将 4 片 256 × 8 位的 RAM 接成一个 1024 × 8 位 RAM 的例子。因为 4 片中共有 1024 个字,所以必须给它们编成 1024 个不同的地址。然而每片集成电路上的地址输入端只有 8 位($A_0 \sim A_7$),给出的地址范围全都是 0 ~ 255,无法区分 4 片中同样的地址单元。

因此,必须增加两位地址代码 A_8、A_9,使地址代码增加到 10 位,才能得到 $2^{10} = 1024$ 个地址。如果取第一片的 $A_9A_8 = \mathbf{00}$,第二片的 $A_9A_8 = \mathbf{01}$,第三片的 $A_9A_8 = \mathbf{10}$,第四片的 $A_9A_8 = \mathbf{11}$,那么 4 片的地址分配将如表 7.4.1 中所示。

由表 7.4.1 可见,4 片 RAM 的低 8 位地址是相同的,所以接线时将它们分别并联起来就行了。由于每片 RAM 上只有 8 个地址输入端,所以 A_8、A_9 的输入端只好借用 CS'端。图中使用 2 线 - 4 线译码器将 A_9A_8 的 4 种编码 **00**、**01**、**10**、**11** 分别译成 Y'_0、Y'_1、Y'_2、Y'_3 4 个低电平输出信号,然后用它们分别去控制 4 片 RAM 的 CS'端。

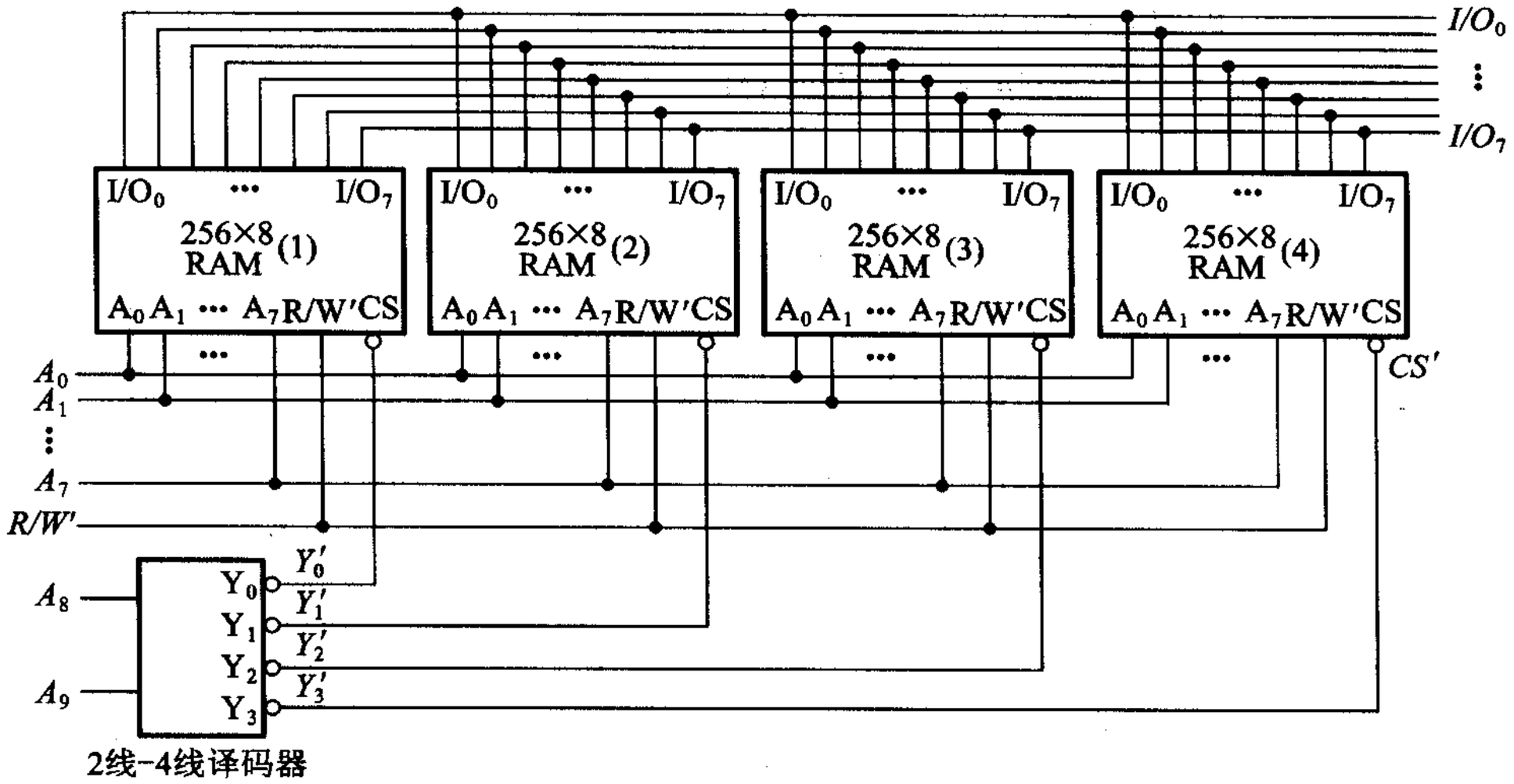


图 7.4.2 RAM 的字扩展接法

表 7.4.1 图 7.4.2 中各片 RAM 电路的地址分配

器件编号	A_9A_8	Y_0' Y_1' Y_2' Y_3'	地址范围 A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0 （等效十进制数）
RAM (1)	0 0	0 1 1 1	00 00000000 ~ 00 11111111 (0) (255)
RAM (2)	0 1	1 0 1 1	01 00000000 ~ 01 11111111 (256) (511)
RAM (3)	1 0	1 1 0 1	10 00000000 ~ 10 11111111 (512) (767)
RAM (4)	1 1	1 1 1 0	11 00000000 ~ 11 11111111 (768) (1023)

此外，由于每一片 RAM 的数据端 $I/O_1 \sim I/O_8$ 都设置了由 CS' 控制的三态输出缓冲器，而现在它们的 CS' 任何时候只有一个处于低电平，故可将它们的数据端并联起来，作为整个 RAM 的 8 位数据输入/输出端。

上述字扩展接法也同样适用于 ROM 电路。

如果一片 RAM 或 ROM 的位数和字数都不够用，就需要同时采用位扩展和字扩展方法，用多片器件组成一个大的存储器系统，以满足对存储容量的要求。

复习思考题

R7.4.1 在图 7.4.2 的例子中，可否改用 A_1A_0 译出 $Y_0' \sim Y_3'$ 信号，而将 $A_2 \sim A_9$ 接到各片

RAM 的 $A_0 \sim A_7$ 输入端?

7.5 用存储器实现组合逻辑函数

表 7.5.1 是一个 ROM 的数据表。如果将输入地址 A_1 和 A_0 视为两个输入逻辑变量,同时将输出数据 D_0、D_1、D_2 和 D_3 视为一组输出逻辑变量,则 D_0、D_1、D_2 和 D_3 就是一组 A_0、A_1 的组合逻辑函数,表 7.5.1 也就是这一组多输出组合逻辑函数的真值表。

表 7.5.1 一个 ROM 的数据表

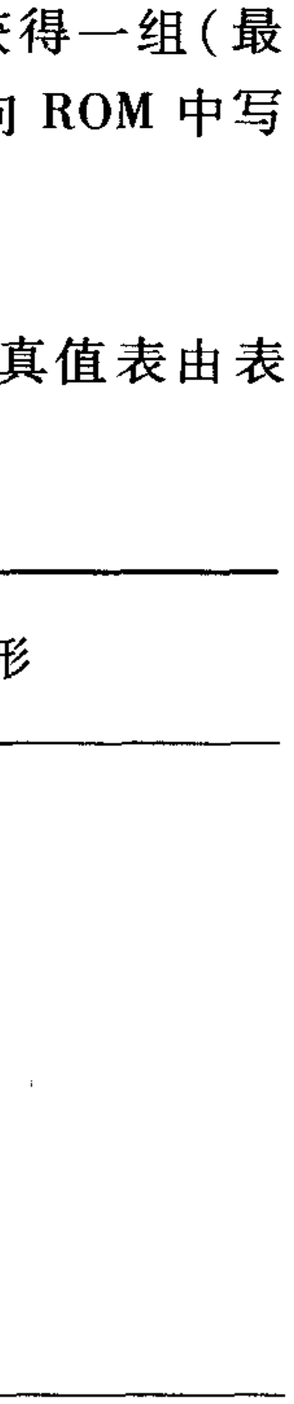
A_1	A_0	D_0	D_1	D_2	D_3	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	1	0	1	1	0	0	1	1	0
0	1	1	0	1	1	1	1	1	1	0	0

另外,由图 7.2.2 所示 ROM 的电路结构图上也可以看到,其中译码器的输出包含了输入变量全部的最小项,而每一位数据输出又都是若干个最小项之和,因而任何形式的组合逻辑函数均能通过向 ROM 中写入相应的数据来实现。

不难推想,用具有 n 位输入地址、m 位数据输出的 ROM 可以获得一组(最多为 m 个)任何形式的 n 变量组合逻辑函数,只要根据函数的形式向 ROM 中写入相应的数据即可。这个原理也适用于 RAM。

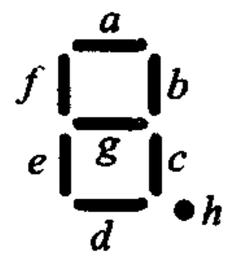
【例 7.5.1】 试用 ROM 设计一个八段字符显示的译码器,其真值表由表 7.5.2 给出。

表 7.5.2 例 7.5.1 的真值表

输入				输出								字形
D	C	B	A	a	b	c	d	e	f	g	h	
0	0	0	0	1	1	1	1	1	1	0	1	
0	0	0	1	0	1	1	0	0	0	0	1	
0	0	1	0	1	1	0	1	1	0	1	1	
0	0	1	1	1	1	1	1	0	0	1	1	

续表

输入				输出								字形
D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	
0	1	0	0	0	1	1	0	0	1	1	1	4
0	1	0	1	1	0	1	1	0	1	1	1	5
0	1	1	0	1	0	1	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	1	7
1	0	0	0	1	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	1	9
1	0	1	0	1	1	1	1	1	0	1	0	
1	0	1	1	0	0	1	1	1	1	1	0	b
1	1	0	0	0	0	0	1	1	0	1	0	c
1	1	0	1	0	1	1	1	1	0	1	0	d
1	1	1	0	1	1	0	1	1	1	1	0	E
1	1	1	1	1	0	0	0	1	1	1	0	F



解： 由给定的真值表可见，应取输入地址为4位、输出数据为8位的(16×8位)ROM来实现这个译码电路。以地址输入端 A_3、A_2、A_1、A_0 作为BCD代码的 D、C、B、A 4位的输入端，以数据输出端 $D_0 \sim D_7$ 作为 $a \sim h$ 的输出端，如图7.5.1所示，就得到了所要求的译码器。

如果制成掩模ROM，则可依照表7.5.2画出存储矩阵的连接电路，如图7.5.1中所示。图中以结点上接入二极管表示存入**0**，未接入二极管表示存入**1**。由表7.5.2可以看出，由于数据中**0**的数目比**1**的数目少得多，所以用接入二极管表示存入**0**比用接入二极管表示存入**1**要节省器件。

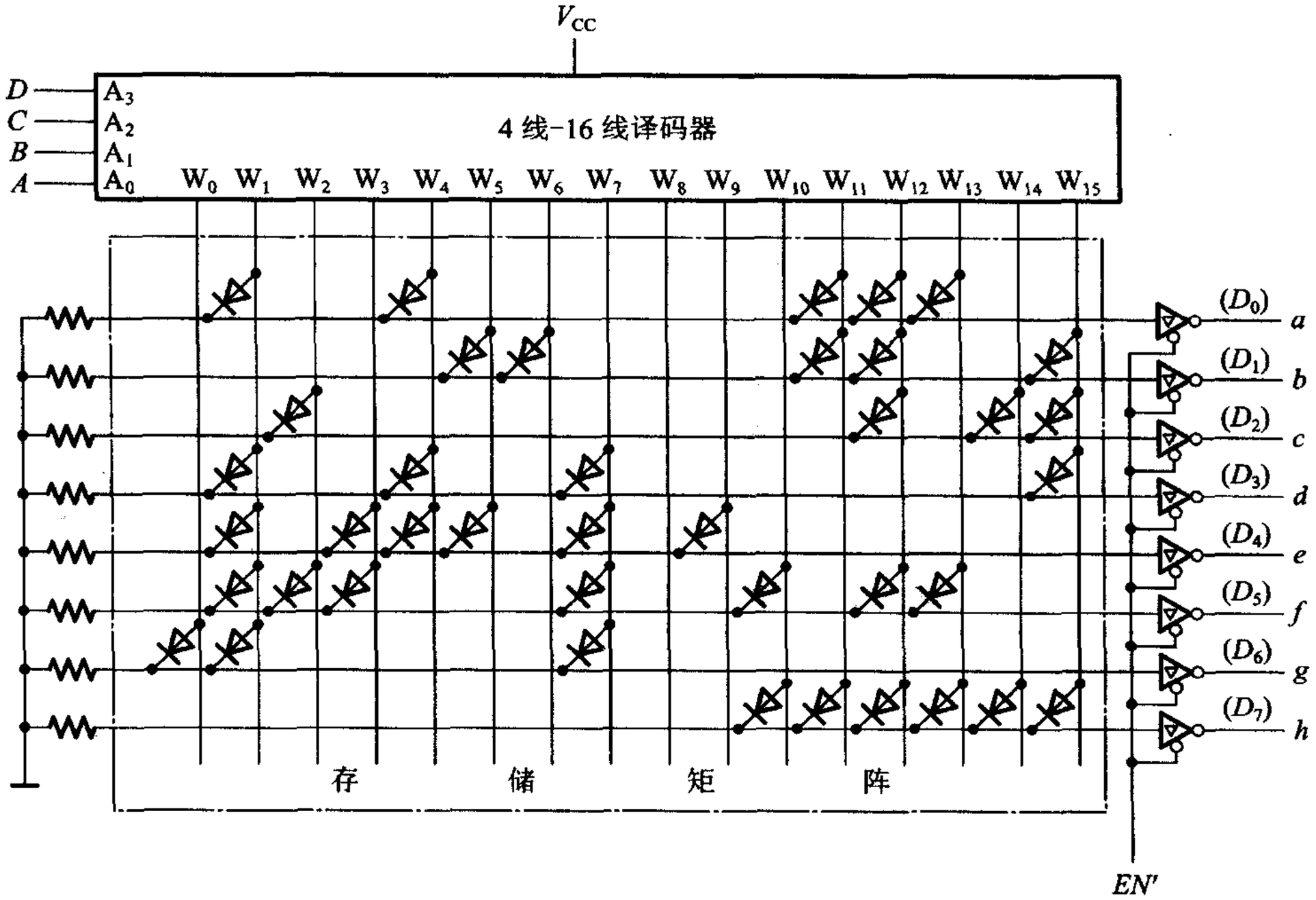


图 7.5.1 例 7.5.1 的电路

如果使用 EPROM 实现这个译码器,则只要将表 7.5.2 中左边的 *DCBA* 当作输入地址代码、右边的 *abcdefg* 当作数据,依次对应地写入 EPROM 就行了。

【例 7.5.2】 试用 ROM 产生如下的一组多输出逻辑函数。

$$\begin{cases} Y_1 = A'BC + A'B'C \\ Y_2 = AB'CD' + BCD' + A'BCD \\ Y_3 = ABCD' + A'BC'D' \\ Y_4 = A'B'CD' + ABCD \end{cases} \tag{7.5.1}$$

解: 将式(7.5.1)化为最小项之和的形式得到

$$\begin{cases} Y_1 = A'BCD' + A'BCD + A'B'CD' + A'B'CD \\ Y_2 = AB'CD' + A'BCD' + ABCD' + A'BCD \\ Y_3 = ABCD' + A'BC'D' \\ Y_4 = A'B'CD' + ABCD \end{cases} \tag{7.5.2}$$

或写成

$$\begin{cases} Y_1 = m_2 + m_3 + m_6 + m_7 \\ Y_2 = m_6 + m_7 + m_{10} + m_{14} \\ Y_3 = m_4 + m_{14} \\ Y_4 = m_2 + m_{15} \end{cases} \tag{7.5.3}$$

取有 4 位地址输入、4 位数据输出的 16 × 4 位 ROM，将 A、B、C、D 4 个输入变量分别接至地址输入端 A_3、A_2、A_1、A_0，按照逻辑函数的要求存入相应的数据，即可在数据输出端 D_3、D_2、D_1、D_0 得到 Y_1、Y_2、Y_3、Y_4。

因为每个输入地址对应一个 A、B、C、D 的最小项，并使地址译码器的一条输出线（字线）为 **1**，而每一位数据输出都是若干字线输出的逻辑**或**，故可按照式（7.5.3）列出 ROM 存储矩阵内应存入的数据表，如表 7.5.3 所示。

表 7.5.3　例 7.5.2 中 ROM 的数据表

函数 / 最小项	Y_1	Y_2	Y_3	Y_4	
$A'B'C'D'$ m_0	0	0	0	0	W_0 0000
$A'B'C'D$ m_1	0	0	0	0	W_1 0001
$A'B'CD'$ m_2	1	0	0	1	W_2 0010
$A'B'CD$ m_3	1	0	0	0	W_3 0011
$A'BC'D'$ m_4	0	0	1	0	W_4 0100
$A'BC'D$ m_5	0	0	0	0	W_5 0101
$A'BCD'$ m_6	1	1	0	0	W_6 0110
$A'BCD$ m_7	1	1	0	0	W_7 0111
$AB'C'D'$ m_8	0	0	0	0	W_8 1000
$AB'C'D$ m_9	0	0	0	0	W_9 1001
$AB'CD'$ m_{10}	0	1	0	0	W_{10} 1010
$AB'CD$ m_{11}	0	0	0	0	W_{11} 1011

续表

函数 / 最小项	Y_1	Y_2	Y_3	Y_4	
$ABC'D'$ m_{12}	**0**	**0**	**0**	**0**	W_{12} **1100**
$ABC'D$ m_{13}	**0**	**0**	**0**	**0**	W_{13} **1101**
$ABCD'$ m_{14}	**0**	**1**	**1**	**0**	W_{14} **1110**
$ABCD$ m_{15}	**0**	**0**	**0**	**1**	W_{15} **1111**
	D_3	D_2	D_1	D_0	地址 / 数据

如果使用 EPROM 实现上述一组逻辑函数，则只要按表 7.5.3 将所有的数据写入对应的地址单元即可。

在使用 PROM 或掩模 ROM 时，还可以根据表 7.5.3 画出存储矩阵的结点连接图，如图 7.5.2 所示。为了简化作图，在接入存储器件的矩阵交叉点上画一个圆点，以代替存储器件。图中以接入存储器件表示存 **1**，以不接存储器件表示存 **0**。

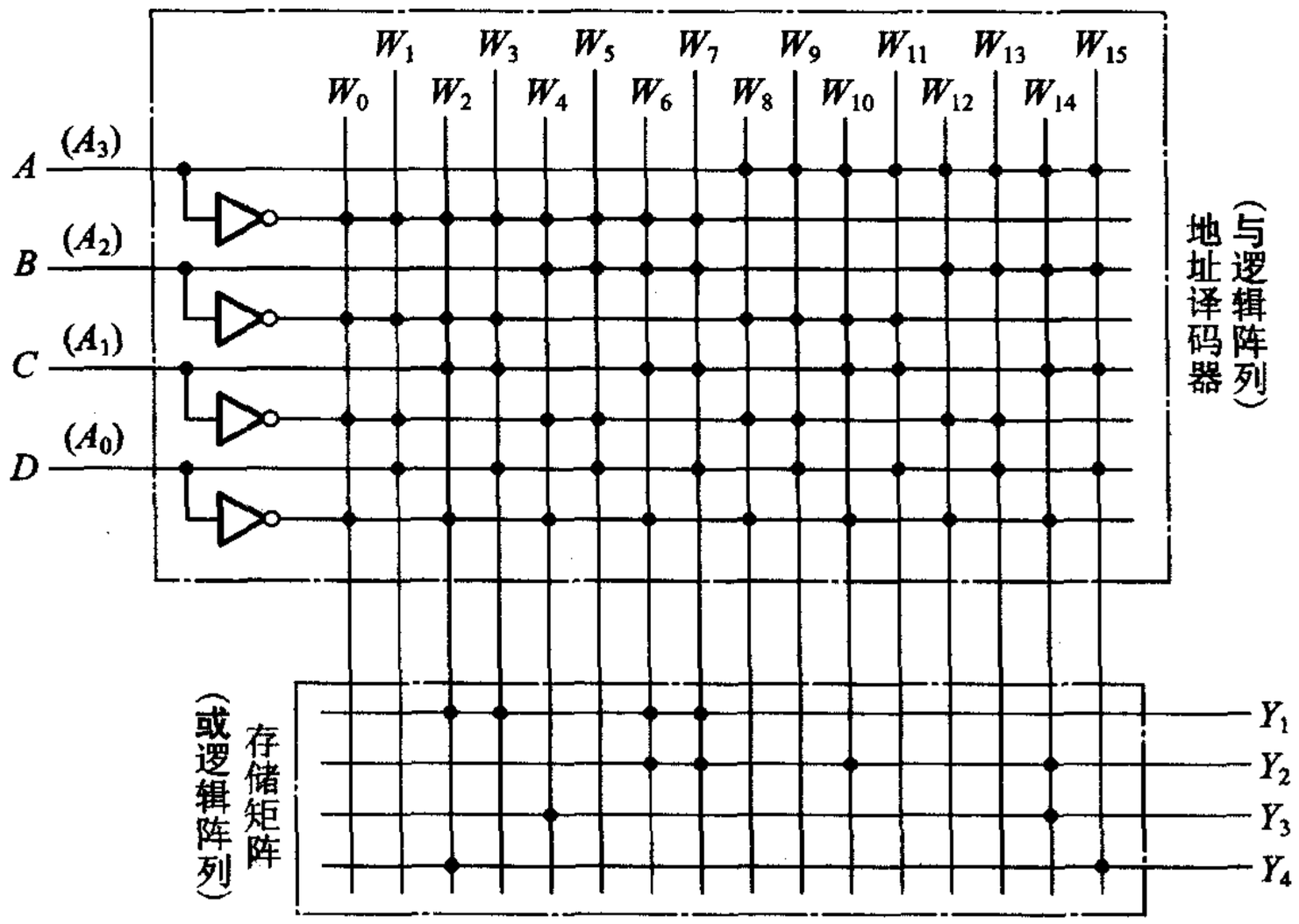


图 7.5.2 例 7.5.2 的 ROM 点阵图

复习思考题

R7.5.1　在用 ROM 产生一组多输出组合逻辑函数时，如果 ROM 的地址输入端数目大于输入逻辑变量数，ROM 的数据输出端数目大于函数输出端数，则应如何处理这些多余的地址输入端和数据输出端？

R7.5.2　在用 ROM 产生一组多输出逻辑函数时，如果输入逻辑变量的数目大于 ROM 地址输入端的数目，或者函数输出端的数目大于 ROM 数据输出端的数目，应当怎么办？

本章小结

半导体存储器是一种能存储大量数据或信息的半导体器件。由于要求存储的数据量往往很大，而器件的引脚数目不可能无限制地增加，因而不可能将每个存储单元电路的输入和输出端都固定地各接到一个引脚上。因此，存储器的电路结构形式与第六章里讲的寄存器不同。

在半导体存储器中采用了按地址存放数据的方法，只有那些被输入地址代码指定的存储单元才能与输入/输出端接通，可以对这些被指定的单元进行读/写操作。而输入/输出端是公用的。为此，存储器的电路结构中必须包含地址译码器、存储矩阵和输入/输出电路（或读/写控制电路）这三个组成部分。

半导体存储器有许多不同的类型。首先从读、写的功能上分成只读存储器（ROM）和随机存储器（RAM）两大类。其次，根据存储单元电路结构和工作原理的不同，又将 ROM 分为掩模 ROM、PROM、EPROM、E^2PROM、快闪存储器等几种类型；将 RAM 分为静态 RAM 和动态 RAM 两类。掌握各种类型半导体存储器在电路结构和性能上的不同特点，将为我们合理选用这些器件提供理论依据。

在一片存储器芯片的存储量不够用时，可以将多片存储器芯片组合起来，构成一个更大容量的存储器。当每片存储器的字数够用而每个字的位数不够用时，应采用位扩展的连接方式；当每片的字数不够用而每个字的位数够用时，应采用字扩展的连接方式；当每片的字数和位数都不够用时，则需同时采用位扩展和字扩展的连接方式。

存储器的应用领域极为广阔，凡是需要记录数据或各种信息的场合都离不开它。尤其在电子计算机中，存储器是必不可少的一个重要组成部分。此外，还可以用存储器来设计组合逻辑电路。只要将地址输入作为输入逻辑变量，将数据输出端作为函数输出端，并根据要产生的逻辑函数写入相应的数据，就能得到

所需要的组合逻辑电路了。

除了一般常见的 ROM、RAM 以外，有些场合也要用到一些特殊的存储器，例如根据移位寄存器工作原理构成的串行存储器，还有其他一些特殊类型的存储器，本书就不一一介绍了。

习　题

［题 7.1］　若存储器的容量为 512 K×8 位，则地址代码应取几位？

［题 7.2］　某台计算机的内存储器设置有 32 位的地址线，16 位并行数据输入/输出端，试计算它的最大存储量是多少？

［题 7.3］　试用 2 片 1024×8 位的 ROM 组成 1024×16 位的存储器。

［题 7.4］　试用 4 片 4 K×8 位的 RAM 接成 16 K×8 位的存储器。

［题 7.5］　试用 4 片 2114（1024×4 位的 RAM）和 3 线－8 线译码器 74HC138（见图 4.3.8）组成 4096×4 位的 RAM。

［题 7.6］　试用 16 片 2114（1024×4 位的 RAM）和 3 线－8 线译码器 74HC138（见图 4.3.8）接成一个 8 K×8 位的 RAM。

［题 7.7］　已知 ROM 的数据表如表 P7.7 所示，若将地址输入 A_3、A_2、A_1、A_0 作为 4 个输入逻辑变量，将数据输出 D_3、D_2、D_1、D_0 作为函数输出，试写出输出与输入间的逻辑函数式，并化为最简与或形式。

表 P7.7

地址输入				数据输出			
A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0
1	0	1	1	1	0	0	0
1	1	0	0	0	1	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	0	0	0	1