数字系统设计

华东理工大学电子与通信工程系
主讲:木昌洪
Email：changhongmu@ecust.edu.cn

第8章半导体存储器与可编程逻辑器件

了解半导体存储器（RAM和ROM）的电路结构、分类和特点。掌握半导体存储器的工作原理和扩展存储容量的方法。掌握基于Verilog HDL的存储器设计方法。掌握阵列型PLD和单元型PLD的基本结构和特点。掌握PLD的设计方法、设计流程。

- 8.1 概述
- 8.2 随机存储器
- 8.3 只读存储器
- 8.4 基于Verilog HDL的存储器设计
- 8.5 PLD的基本原理
- 8.6 PLD的设计技术

本 章 重 点

- 半导体存储器的工作原理；
- 扩展存储容量的方法;
- 基于Verilog HDL的存储器设计方法；
- 阵列型PLD和单元型PLD的基本结构和特点；
- PLD的设计方法和设计流程

内容概要

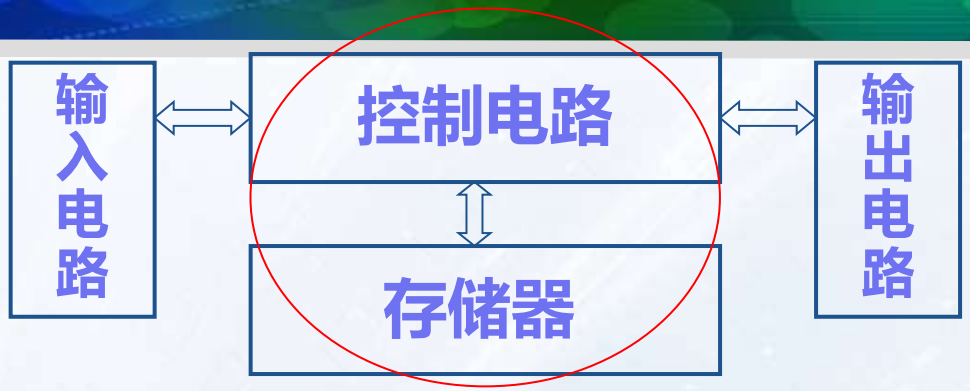
8.1.1 程序逻辑电路的结构及特点

8.1.2 半导体存储器的结构

8.1.3 半导体存储器的分类

8.1.1 程序逻辑电路的结构及特点

- 结构

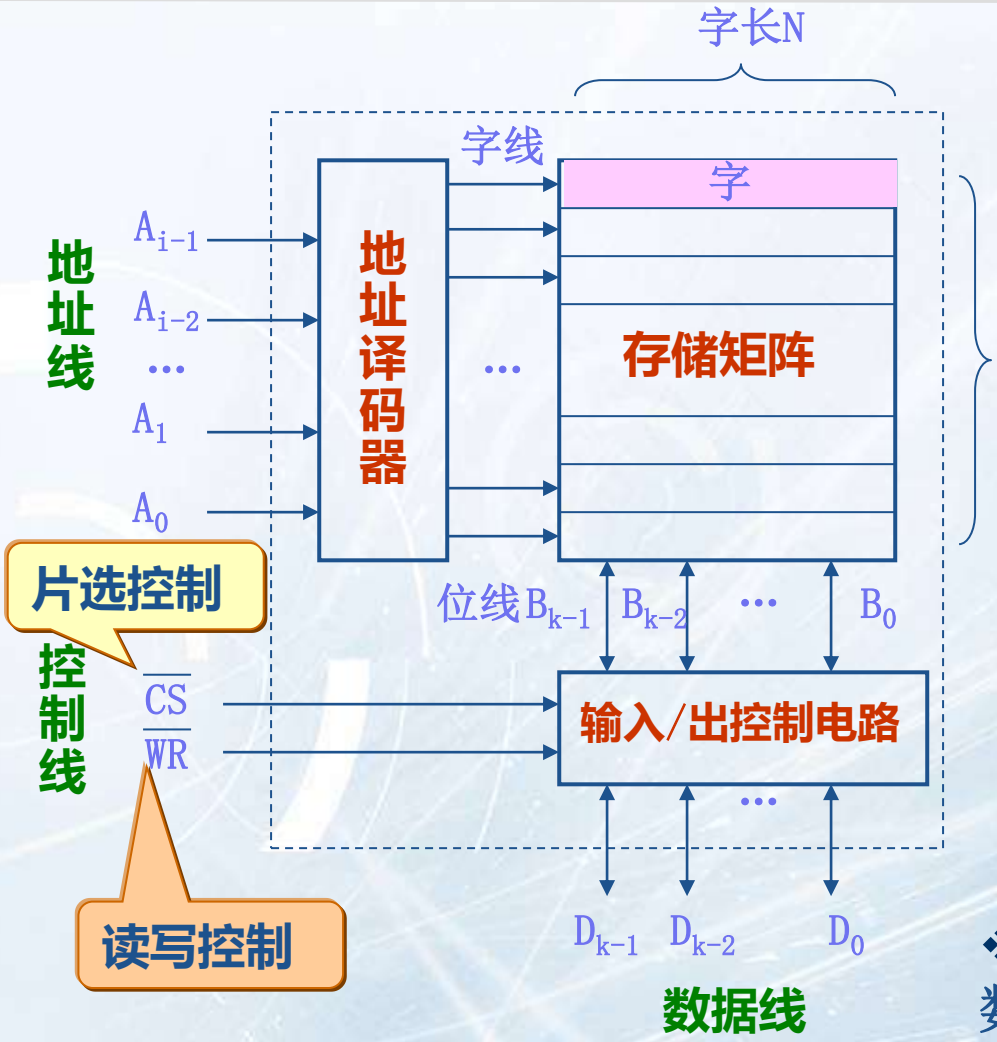


- 控制电路——包括计数器、寄存器等时序逻辑电路和译码器、运算器等组合逻辑电路。从存储器中取出程序或数据，译码后使电路完成相应的操作。
- 半导体存储器——能存储大量二值信息的半导体器件，可以存放程序和数据。
  - 输入电路——完成外部信息和指令、程序的输入。
  - 输出电路——完成处理结果信息及数据的输出。
- 特点：软硬结合，用一块相同的硬件电路，通过改变存储器中的程序或数据，完成多种功能的操作。

8.1.2 半导体存储器的结构

- 在计算机和数字系统中，都需要对大量的数据进行存储，半导体存储器是这些数字系统不可缺少的组成部分。
- 由于计算机处理的数据量越来越大，运算速度越来越快，这就要求存储器有更大的存储容量和更快的存取速度。因此存储容量和存取速度是衡量存储器性能的重要指标
  - 动态存储器容量已达10^9位/片，高速随机存储器的存取时间只有10ns左右
- 半导体存储器的存储单元数目庞大、器件引脚有限，不可能像寄存器那样把每个存储单元的输入和输出直接引出。
- 解决办法：给每个存储单元编一个地址，所有存储单元共用一组输入/输出引脚。只有被输入地址代码指定的存储单元才能与公共的输入/输出引脚接通，进行数据的写入或读出。

半导体存储器的结构图



- 存储矩阵——存放数据的主体，由许多存储单元排列而成
- 每个存储单元能存储1位二进制代码，若干个存储单元形成一个存储组——字，每个字包含的存储单元的个数称为字长。
- 地址译码器——产生到存储器“字”的地址码的器件，其输入称为地址线，输出称为字线。
- 输入/输出控制电路——控制存储器数据的流向和状态（读或写）

半导体存储器的输入和输出信号线

- 地址线用来寻址某一个存储单元。
  - 地址空间：地址线的条数决定了存储器的地址空间。有i条地址线的译码器，最多可有2^i条字线，能为2^i个字提供地址线，则存储器的字数为2^i个。
- 控制线包括片选控制信号/CS和读写控制信号/WR。
  - 当/CS=0时，存储器为正常工作状态；
  - 当/CS=1时所有输入、输出端为高阻态，不能对存储器进行读/写操作。
  - 当/CS=0、/WR=0时，执行写入操作，将数据线上的数据写入存储器中；
  - 当/CS=0、/WR=1时，执行读出操作，将数据从存储器中读出，送到数据线上。
- 数据线既是数据输入端又是数据输出端，由/CS和/WR来控制数据的流向。数据线的条数决定存储器的字长。

半导体存储器的存储容量

- 字：若干个存储单元构成的一个存储组。字是一个整体，有共同的地址，共同用来代表某种信息，并共同写入存储器或从存储器中读出。
- 字长：构成存储器字中二进制数的位数。字长有1、4、8、16、32位等，一般把8位字长称为1字节（Byte）；16位字长称为1字（Word）。
- D_0~D_{k-1}是存储器的数据线。数据线的条数决定存储器的字长。
- 字数：存储矩阵中所包含的存储组的个数。地址线的条数决定存储器的字数。若有i条地址线，则存储器的字数为2^i个
  - 存储容量：存储矩阵能存放的二进制代码的总位数。

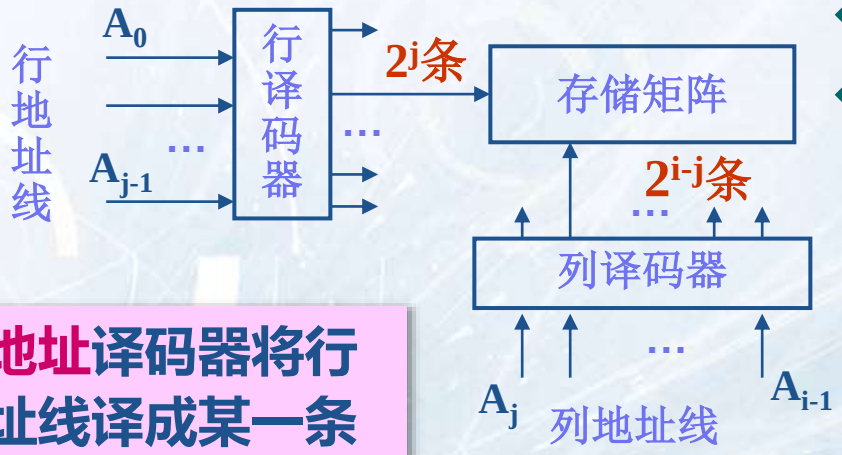
    存储容量 = 字数M ×字长N（位bit）(1B=8b)

    若地址线i=10，则M=2^i=2^{10}=1024

    1024=1K；1024K=1M；1024M=1G；1024G=1T

半导体存储器的译码

- 地址译码器——产生到存储器“字”的地址
- 译码方式：
  - 线译码：若地址线为i条，需i线-2^i线译码器，译码线数＝ 2^i
  - 矩阵译码：将地址线分为两组，分别为行地址译码器和列地址译码器的输入。若行地址线和列地址线各i/2条，译码线数＝$2 \times 2^{i/2}$



- 线译码输出线数＝1024（i＝10）
- 矩阵译码输出线数＝64（i＝10）

- **行地址译码器将行地址线译成某一条字线的输出高、低电平信号，以选中一行存储单元（即一个字）**

- **列地址译码器将列地址线译成某一条列译码线的输出高、低电平信号，从字线选中的一行存储单元中选择1位（或几位）**

- **矩阵译码的优点：译码输出线条数大为减少**
- **在集成电路中，译码线也占用芯片的面积，因此减少译码线的条数可以扩大芯片的集成度。**

半导体存储器的输入/输出控制电路

❖输入/输出控制电路——控制存储器数据的流向和片选使能

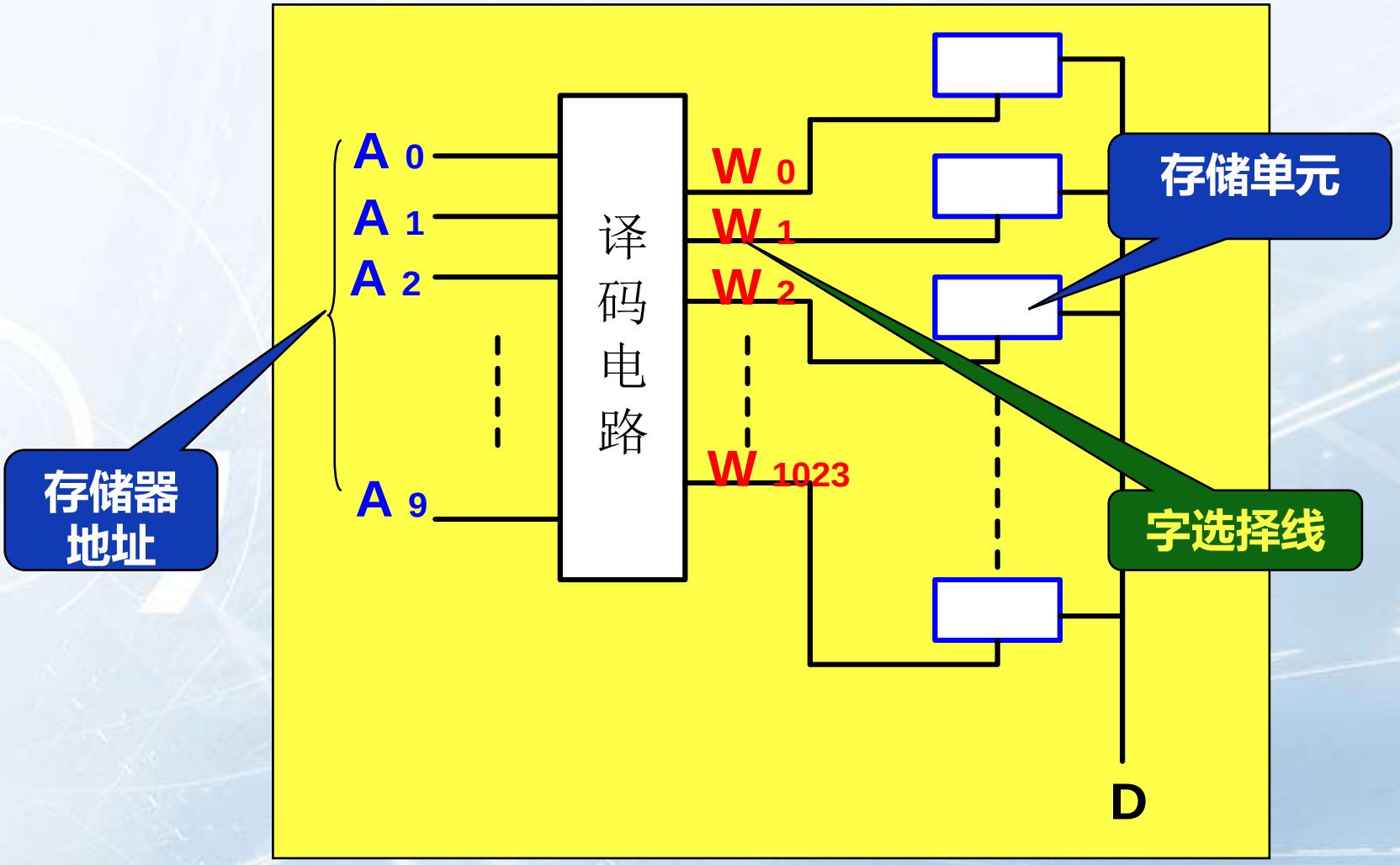
- 写操作：通过数据线将外部数据送入存储器的某些存储单元中保存
- 读操作：把存在存储器的某些存储单元中的数据取出送到数据线上，供其它器件或设备使用。

片选控制$\overline{CS}$ = { 0 芯片工作；1 芯片禁止，输出为高阻状态

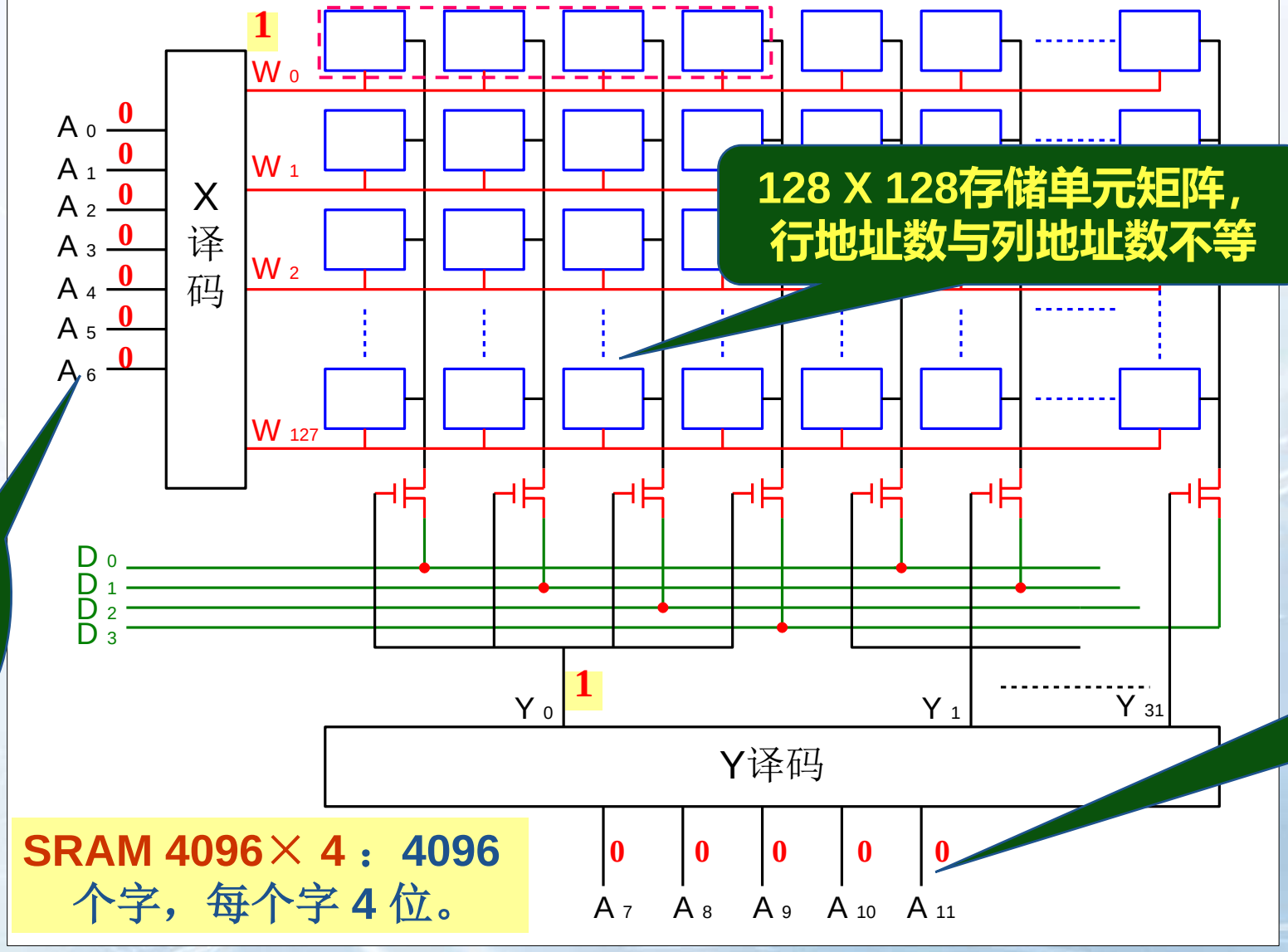
读写控制$\overline{WR}$= { 0 写操作（输入数据）；1 读操作（输出数据）

存储芯片结构（一维地址结构）

- 1024×1位SRAM：1024 个字，每个字有 1 个二进制位

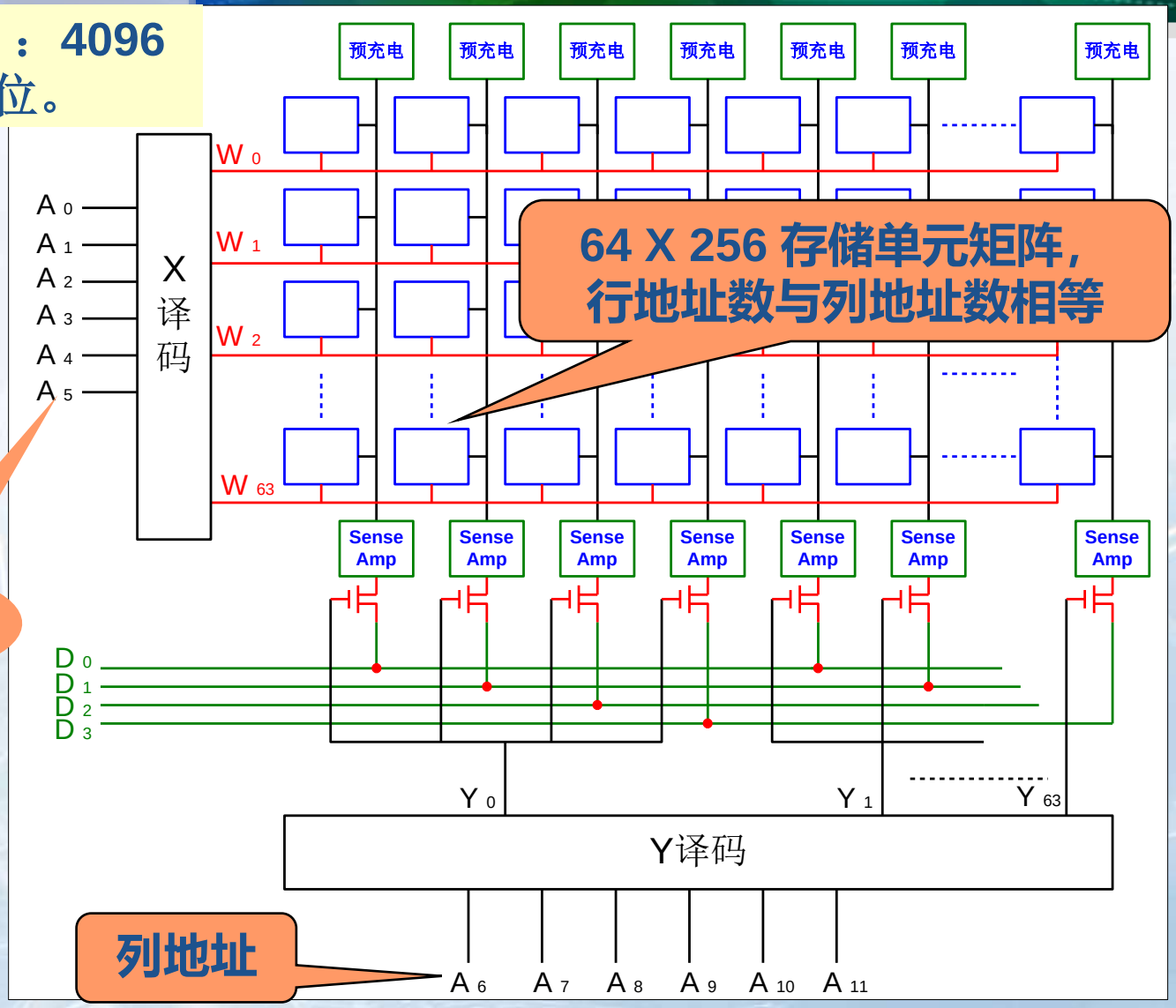


存储芯片结构（SRAM二维地址结构）



存储芯片结构（DRAM二维地址结构）

DRAM 4096× 4：4096个字，每个字4位。



8.1.3 半导体存储器的分类

❖分类(按存、取功能)

- 只读存储器**ROM**（**Read Only Memory**）
  - 使用中（指正常工作状态下）只读不写
  - 优点：电路结构简单；掉电后信息不会丢失。
  - 缺点：不能修改或重新写入数据，只适用于存储固定数据。
- 随机存储器**RAM**（**Random Access Memory**）
  - 也叫随机读/写存储器。使用中可读可写
  - 优点：正常工作状态下可以快速地随时向存储器写入数据或从中读出数据
  - 缺点：掉电后信息会丢失。

? ❖ **问题：计算机的内存是ROM还是RAM？DDR、DDR2、DDR3是什么?**

随机存取存储器RAM的分类（1/2）

- 静态随机存取存储器（SRAM）
  - 使用中可读可写，不需要刷新。
  - 用触发器作为存储单元存放1 和0，只要不掉电即可持续保持内容不变。
  - 优点：存取速度比DRAM快
  - 缺点：集成度不如DRAM高
  - 按制造工艺的不同，分为双极型和MOS型SRAM。
  - 双极型SRAM制造工艺复杂，集成度较低，功耗大；但存取速度快，主要用在一些高速数字系统中。
  - MOS型SRAM功耗低、集成度高，包括NMOS型和CMOS型。CMOS型微功耗，大容量SRAM几乎都采用CMOS工艺。

随机存取存储器RAM的分类（2/2）

- 动态随机存取存储器（DRAM）
  - 使用中可读可写，但需要定时刷新。
  - 基本存储电路为带驱动晶体管的电容。电容上有无电荷状态被视为逻辑1 和0。随着时间的推移，电容上的电荷会逐渐减少，为保持其内容必须周期性地对其进行刷新（对电容充电）。
  - 优点：结构非常简单，集成度远高于SRAM
  - 缺点：存取速度不如SRAM快
  - 为保持数据必须设置刷新电路，硬件系统复杂。

只读存储器ROM的分类

- 固定ROM（掩膜MROM）：数据在生产时写入，出厂后数据不能更改。使用中只读不写。
- 可编程ROM（PROM，Programmable Read Only Memory）：数据可由用户一次性编程写入，写入后的数据不能再更改。使用中只读不写。
- 光可擦可编程ROM（EPROM，Erasable Programmable Read Only Memory）：数据可用紫外光擦去并由用户多次编程写入，使用中只读不写。写入需要使用专门的编程器完成；数据擦除需要使用专门的擦除器完成，擦除速度很慢。
- 电可擦可编程ROM（EEPROM，Electricity Erasable Programmable Read Only Memory）：数据可用电擦去并由用户多次编程写入，使用中只读不写。擦除速度较快。
- 快闪存储器（Flash Memory）：数据可用电擦去并由用户多次编程写入，使用中只读不写。擦除速度很快。

8.2 随机存储器

内容概要

8.2.1 静态随机存储器SRAM

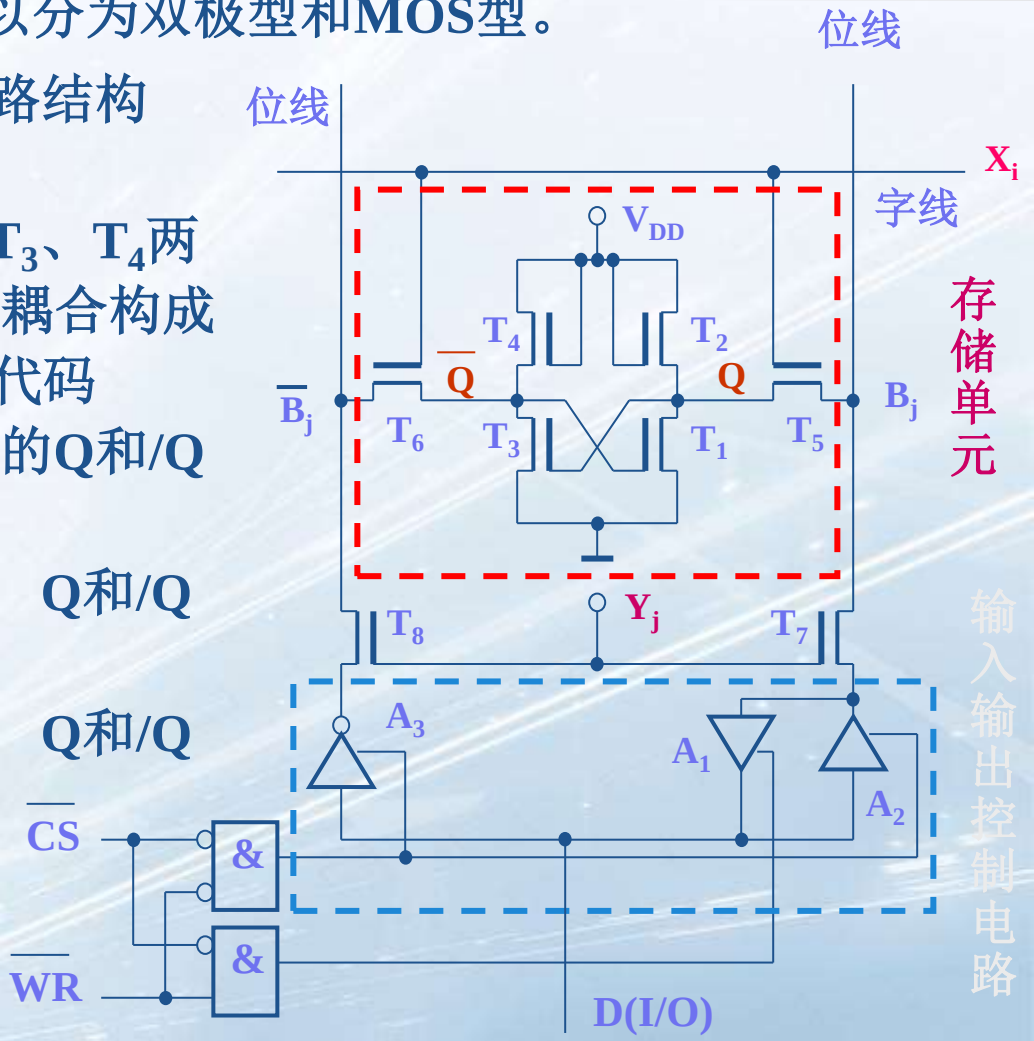
8.2.2 动态随机存储器DRAM

8.2.3 RAM典型芯片

8.2.4 RAM芯片扩展

8.2.1 静态随机存储器SRAM

- 根据制造工艺的不同，SRAM可以分为双极型和MOS型。
- 右图为6管NMOS静态存储单元电路结构
  - T_1、T_3：工作管
  - T_2、T_4：负载管，T_1、T_2及T_3、T_4两个增强型NMOS管反相器交叉耦合构成基本RS触发器，记忆1位二值代码
  - T_5、T_6：门控管，控制RS FF的Q和/Q与位线B_j和/B_j之间的连接
    - 字线X_i=1时，T_5、T_6导通，Q和/Q与位线连接；
    - 字线X_i=0时，T_5、T_6截止，Q和/Q与位线不连接。
- T_7、T_8：每列存储单元公用的门控管，用于与输入输出控制电路的连接，开关状态由列地址译码器的输出Y_j控制



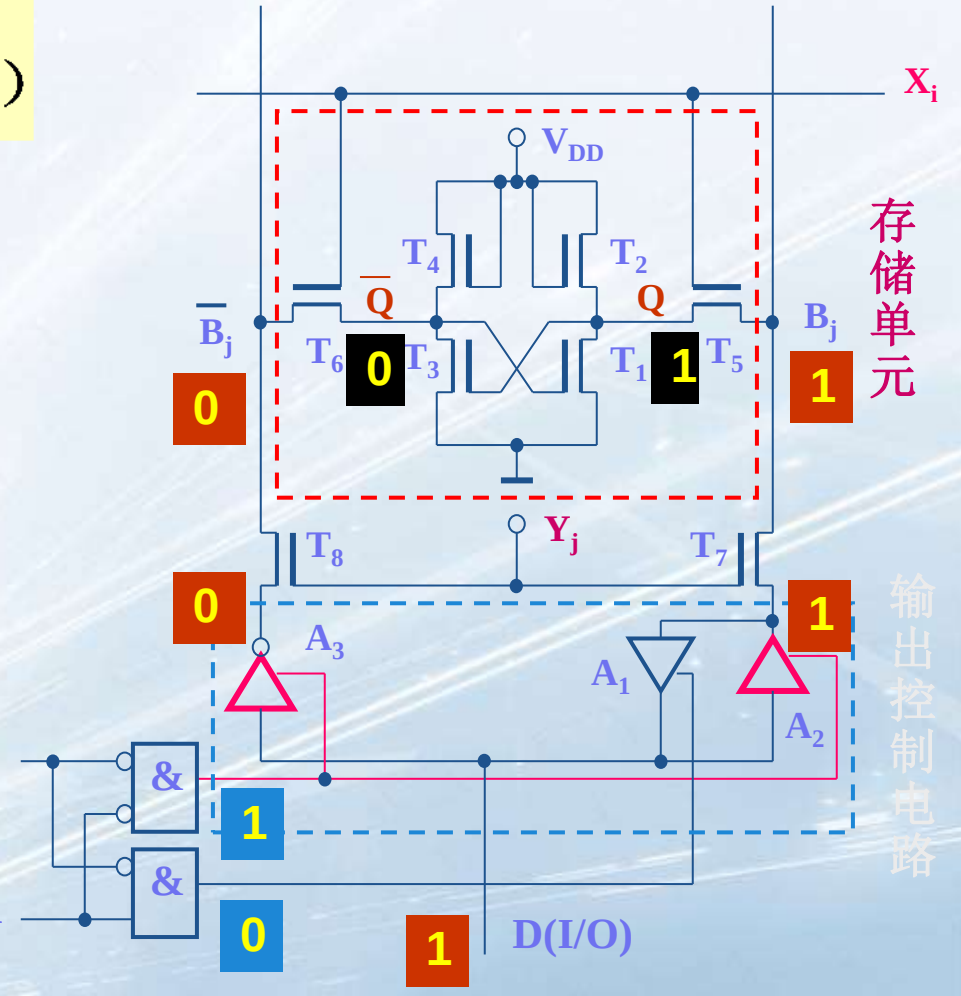
静态随机存储器SRAM工作原理（写操作）

写操作

条件：$\overline{CS}=0, \overline{WR}=0,$

$X_i=1,\ Y_j=1$（地址选中）

- T_5、T_6、T_7、T_8均导通， Q和/Q与位线B_j和/B_j接通
- A1 截止，A2 和A3 导通，加在D端的数据被写入存储单元
  - 若D=1，则A_2 输出为1， A_3 输出为0， 即B_j为1， /B_j为0，使T_3导通， T_1截止，写入Q= 1。
  - 若D=0，则A_2 输出为0， A_3 输出为1， 即B_j为0， /B_j为1，使T_1导通，T_3截止，写入Q= 0。



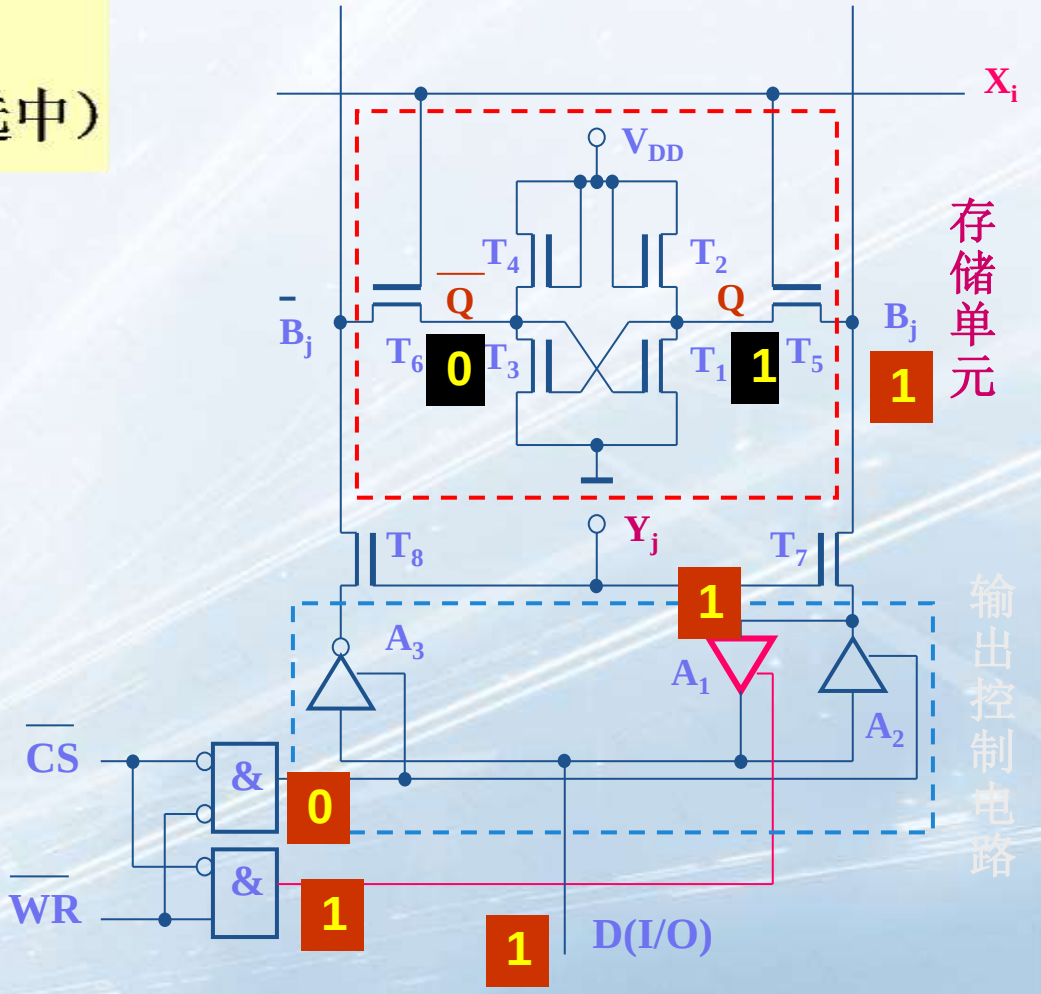
静态随机存储器SRAM工作原理（读操作）

- 读操作

条件：$\overline{CS}=0, \overline{WR}=1$，

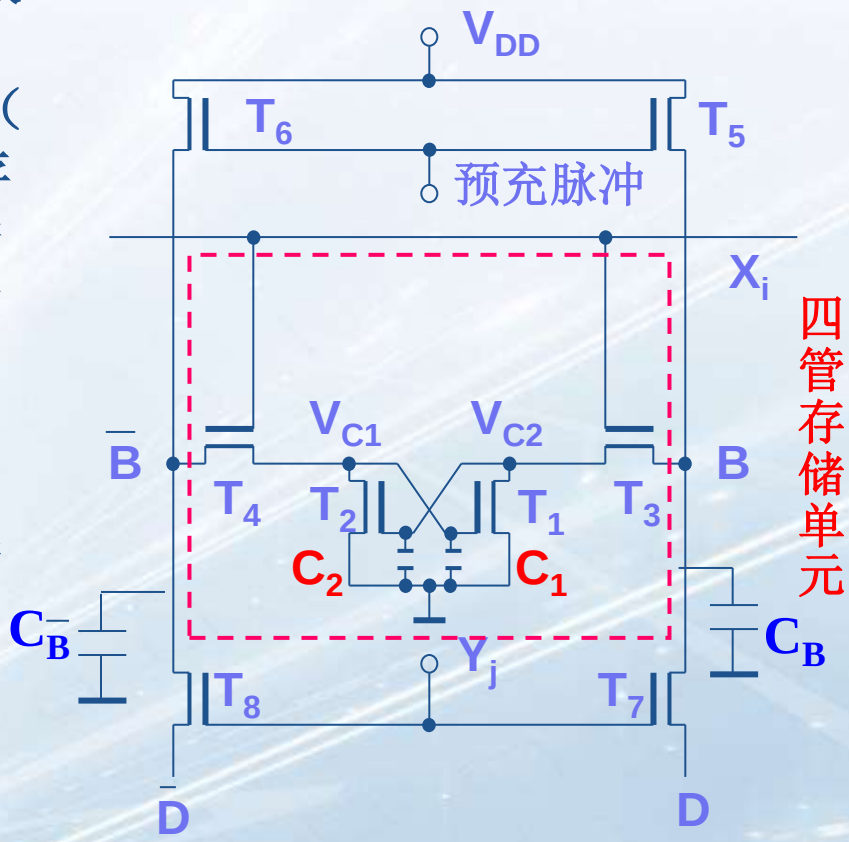
$X_i=1$，$Y_j=1$（地址选中）

- T_5、T_6、T_7、T_8均导通， Q和/Q与位线B_j和/B_j接通
- A_1导通，A_2 和A_3截止，Q端的状态经A_1送到D端，数据被读出
  - 若Q=1， /Q=0，则B_j为1，A_1 输入为1， 从D端输出D= 1。
  - 若Q=0， /Q=1，则B_j为0，A_1 输入为0， 从D端输出D= 0。



8.2.2 动态随机存储器DRAM

- DRAM的存储单元是利用MOS管的栅极电容可以存储电荷的原理制成的。
- 由于栅极电容的容量（C2、C1）很小（几pF），且存在漏电流，所以电荷保存的时间有限。为了及时补充漏掉的电荷，必须给栅极电容补充电荷，这种操作称为刷新（或再生）。
- DRAM必须辅以必要的刷新控制电路，操作也比较复杂。
- 早期的动态存储单元为四管电路或三管电路
  - T_1、T_2：增强型NMOS管，栅极和漏极交叉相连，数据以电荷的形式存储在其栅极电容C_1、C_2上，C_1、C_2上的电压控制T_1、T_2的截止或导通，产生位线B和/B上的高、低电平
  - T_5、T_6：对位线的预充电电路



四管MOS动态存储 单元结构

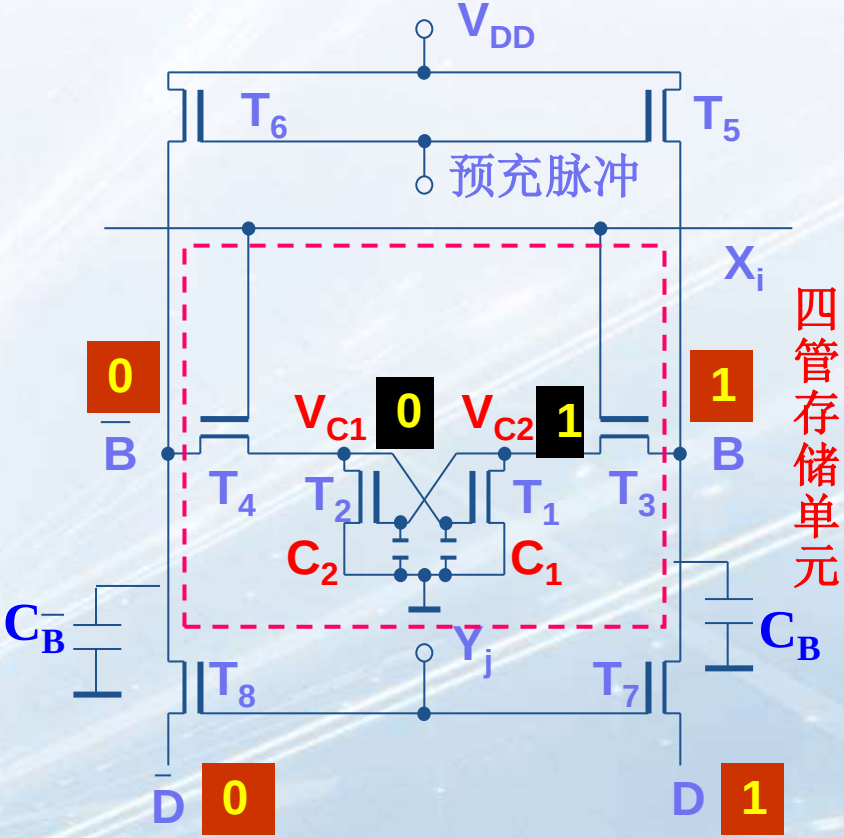
四管MOS动态存储单元工作原理（写操作）

- 写操作

条件：$\overline{CS}=0, \overline{WR}=0,$

$X_i=1,\ Y_j=1$（地址选中）

- 存储单元被选中。T_3、T_4、T_7、T_8均导通， 数据加到D和/D上，通过T_7、T_8传到位线B和/B，经过T_3、T_4写入C_1或C_2
  - 若D=1，/D=0，则C_2被充电，C_1没有被充电， 使T_1截止，T_2导通（接地），V_{C1}= 0， V_{C2}= 1，写入数据“1”。
  - 若D=0， /D=1，则C_1被充电，C_2没有被充电， 使T_1导通（接地）， T_2截止，V_{C1}= 1， V_{C2}= 0，写入数据“0”。



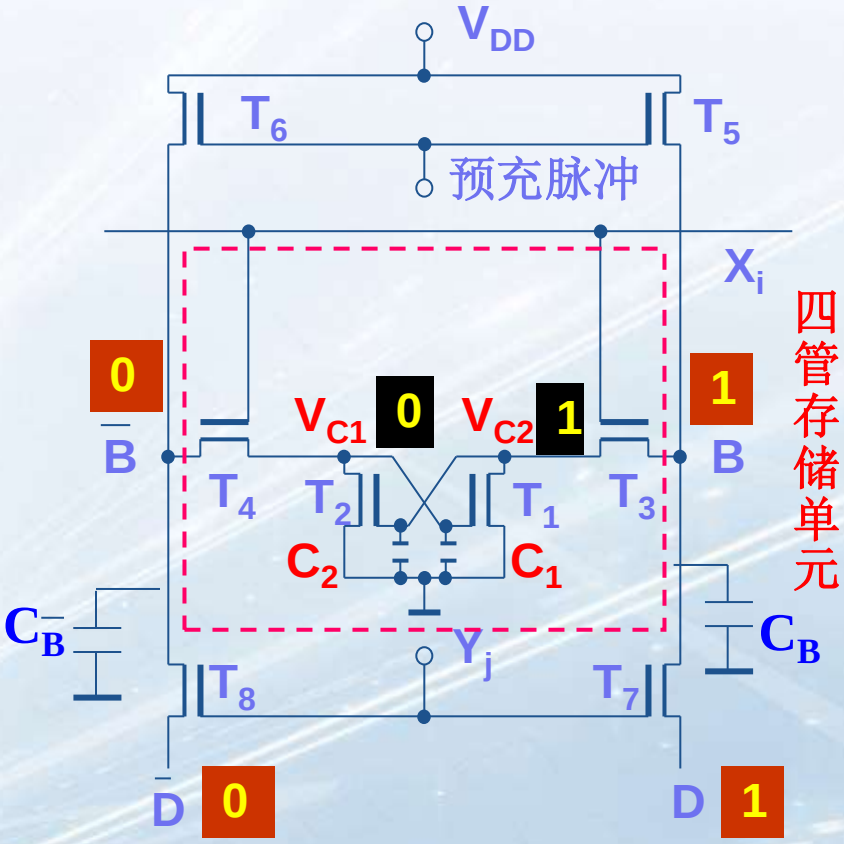
四管动态存储单元结构

四管MOS动态存储单元工作原理（读操作）

- 读操作

$$\text{条件}:\overline{CS}=0,\overline{WR}=1,$$
$$X_i=1,\ Y_j=1(\text{地址选中})$$

- 存储单元被选中。T_3、T_4、T_7、T_8均导通，C_1或C_2中存储的电荷以电压的形式经过T_3、T_4传到位线B和/B，再通过T_7、T_8出现在数据线D和/D上，数据被读出
  - 若$V_{C2}=1$，$V_{C1}=0$，则B=1，/B=0，使D=1，/D=0
  - 若$V_{C2}=0$，$V_{C1}=1$，则B=0，/B=1，使D=0，/D=1



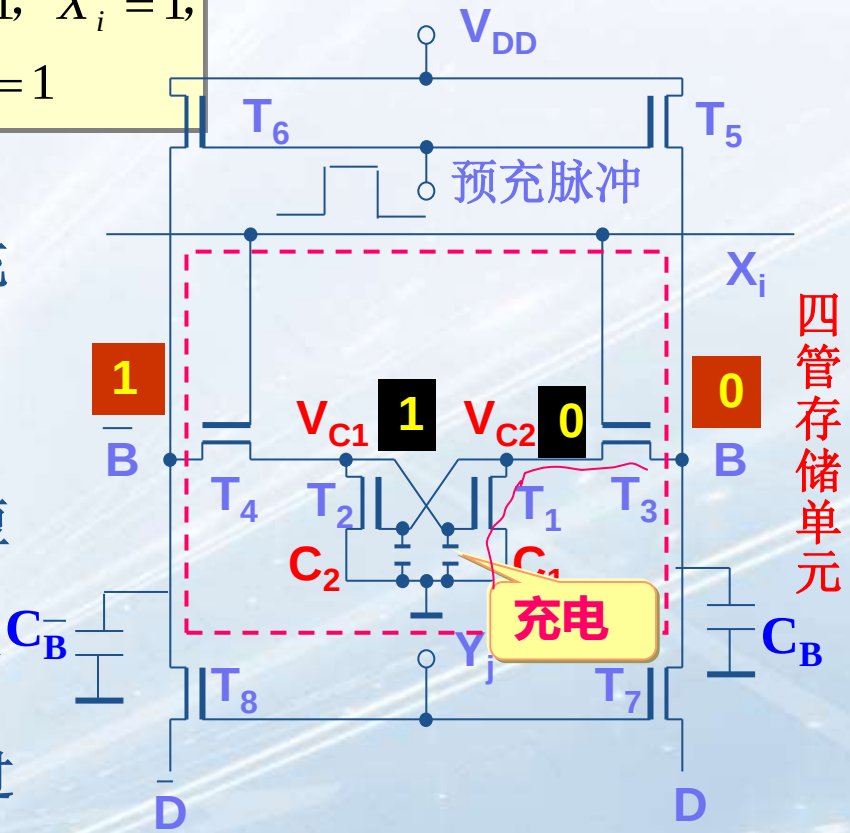
四管动态存储单元结构

四管MOS动态存储单元工作原理（刷新操作）

刷新操作(读操作)

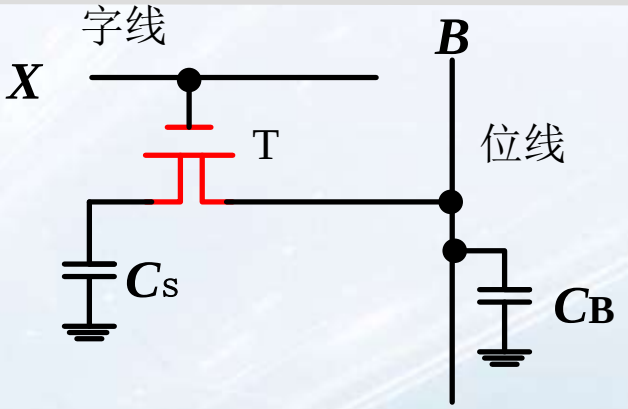
条件 $\overline{CS}=0, \overline{WR}=1$，$X_i=1$，$Y_j=1$，预冲脉冲$=1$

- T_5、T_6：对位线的预充电电路
- 刷新开始时，先在T_5、T_6的栅极加预充电脉冲，使T_5、T_6导通，位线B和/B与V_{DD}接通，将位线上的分布电容C_B和$C_{/B}$充至高电平。
- 预充电脉冲消失后，位线上的高电平短时间内由C_B和$C_{/B}$维持
- 当X_i、Y_j为高电平时，假定存储单元存储的数据为0，则T_1导通（接地），T_2截止，$V_{C2}=0$，$V_{C1}=1$，这时C_B将通过T_3和T_1放电，使位线B变为低电平，C_2将不能充电，使V_{C2}保持0；
  - 因T_2截止，/B上保持的高电平可以对C1充电，使C_1上的电荷不仅不会丢失，反而得到补充，即刷新，使V_{C1}保持1



单管MOS动态存储单元

- 四管存储单元的优点是外围电路比较简单，刷新时不需要另加外部逻辑； 读出信号也较大
- 缺点：管子多，占用的芯片面积大
- 单管MOS动态存储单元由一只N沟道增强型MOS管T和一个电容C_S构成。其电路结构最简单，是所有大容量DRAM首选的存储单元
  - 优点：元件数量少，集成度高
  - 缺点：需要有高鉴别能力的读出放大器配合工作，外围电路比较复杂。



单管MOS动态存储单元

- C_S电容 <<C_B电容
- Cs上有电荷表示 “1”
- Cs上无电荷表示 “0”
- 保持状态：字线低电平，T截止，理论上内部保持稳定状态。

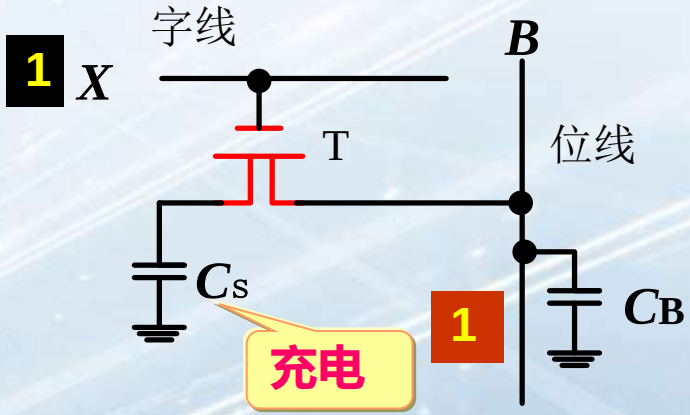
单管MOS动态存储单元工作原理（写入）

❖写入：

- D （数据） 线加高电平（1）或低电平（0），所写数据加到C_B上，若字线X=1，T导通→对C_S充电或放电→位线上的数据经过T被存入C_S中；
  - 写1时，D线高电平，对C_S充电；
  - 写0时，D线低电平，C_S放电

❖保持

- 若字线X=0，T截止→无放电回路→信息存储在C_S中(会缓慢泄漏)



❖ 在保存二进制信息“1”的状态下，Cs有电荷，但Cs存在漏电流，Cs上的电荷会逐渐消失，状态不能长久保持。在电荷泄漏到威胁所保存的数据性质之前，需要补充所泄漏的电荷，以保持数据性质不变。这种电荷的补充称之为刷新（或再生）。

单管MOS动态存储单元工作原理（读出）

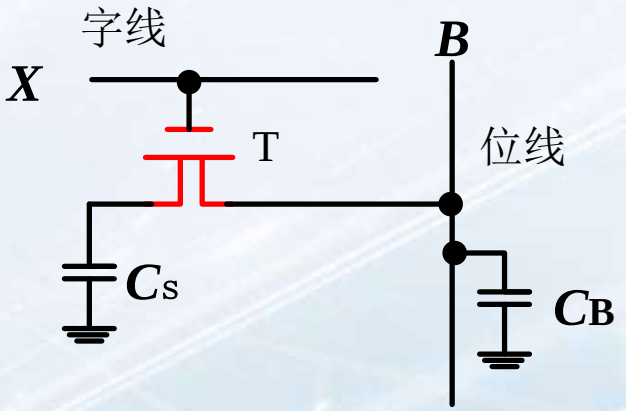
❖读出：

◆先在C_B上加正脉冲Vpre=2.5V →对C_B预充电

◆然后X=1，T导通→ C_S经T向C_B提供电荷，使位线获得读出的信号电平

■若电路保存信息1，Vcs=3.5V，电流方向从单元电路内部指向外，Cs在放电

■若电路保存信息 0，Vcs=0.0V，电流方向从外指向单元电路内部， Cs在充电

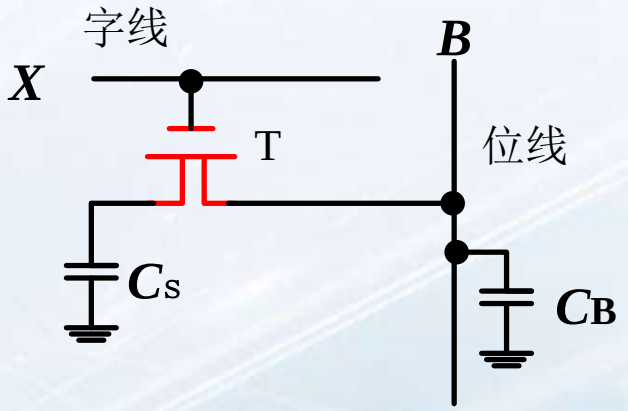


- 根据数据线上电流的方向可判断单元电路保存的是1还是 0。
- 读出过程实际上是C_S与C_B上的电荷重新分配的过程，也是Cs与C_B上的电压重新调整的过程。C_B上的电压，即是D（数据）线上的电压。

单管MOS动态存储单元的工作特征

- 设C_S上原来有正电荷，电压υ_{cs}为高电平，而位线电位$\upsilon_B=0$，则执行读操作后位线电平将上升为

$$v_B = \frac{C_S}{C_S + C_B} v_{Cs}$$

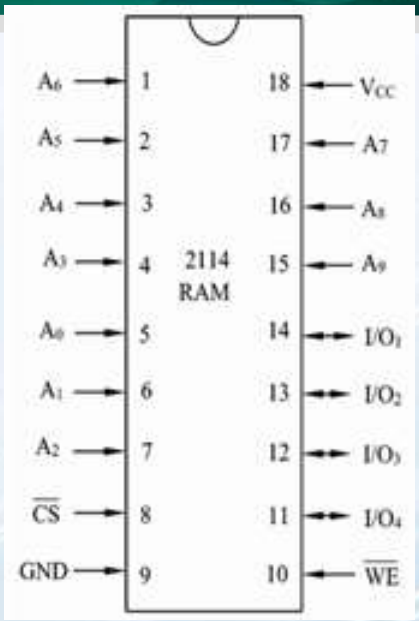


- 由于实际的存储电路位线上总是同时接有很多存储单元，使$C_B >> C_S$，所以位线上读出的电压信号很小
- 例如读操作之前$\upsilon_{cs}=5V$， $C_S/C_B=1/50$，则读操作之后位线上电平仅有0.1V；而且读出以后C_S上电压也只剩下0.1V ——破坏性读出
- 需要在DRAM中设置灵敏的读出放大器，将读出信号放大，并将存储单元中原来存储的信号恢复

- 刷新：由灵敏恢复/读出放大器在读出过程中同时完成。在D线上增加了灵敏恢复/读出放大器后读出过程实际上就是一次刷新过程

8.2.3 RAM典型芯片

- Intel 2114 （1K×4位）SRAM
  - 地址线10条A_0～A_9
  - 数据线4条I/O_0 ～ I/O_3
  - 片选$\overline{CS}$
  - 写控制$\overline{WE}$，为0时写操作；
    为1时读操作
  - 存储容量为$2^{10}\times4=1024\times4$位

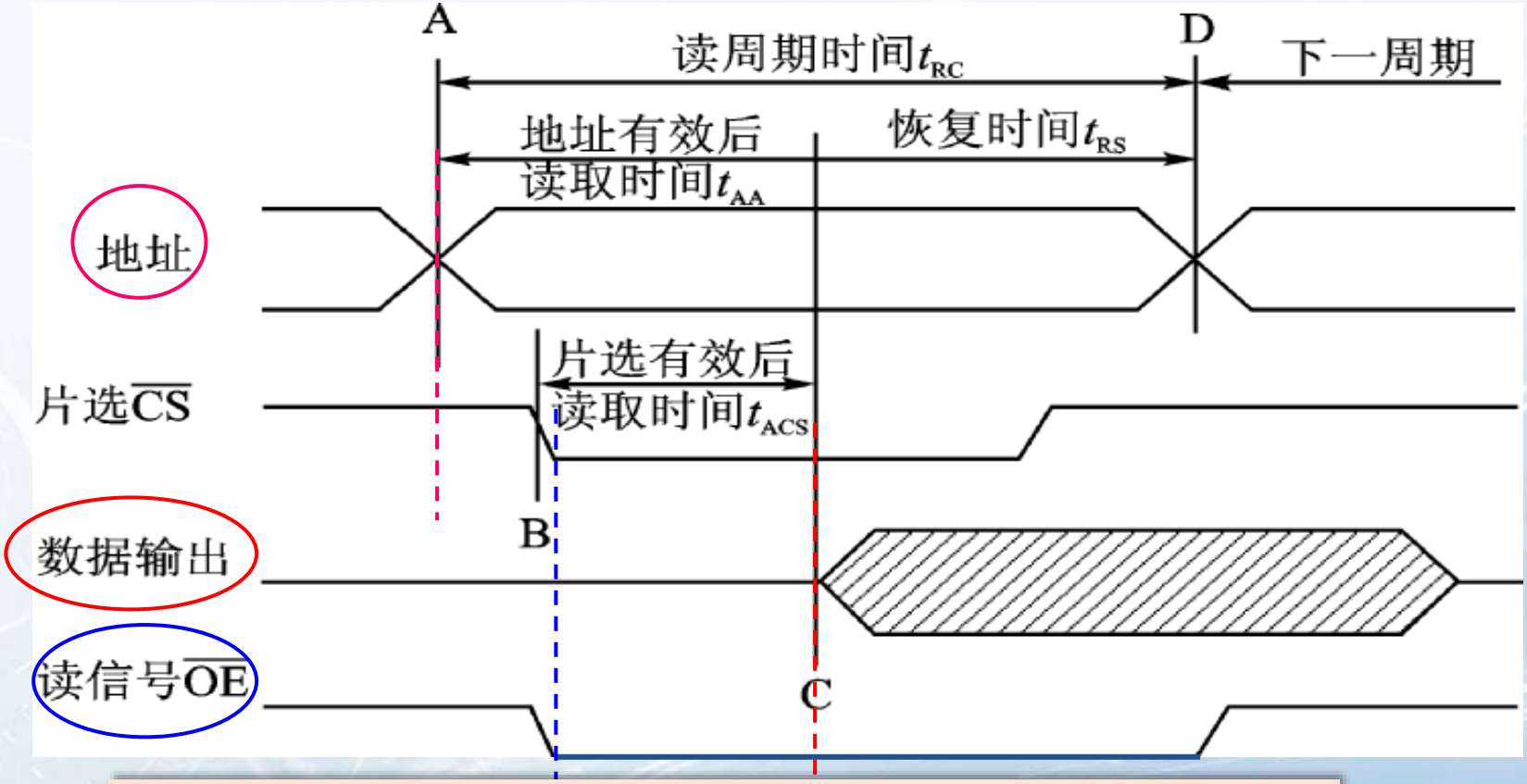


引脚图

- 存储芯片容量的基本描述（字数×每个字的位数）
  - 1K × 4：1024 个字，每个字 4 位（二进制位）。
  - 意味着任一时刻可以（也只能）访问1024个独立字中的任意一个字，每次读写的数据位数是一个字的容量（4位）。

SRAM读周期时序图

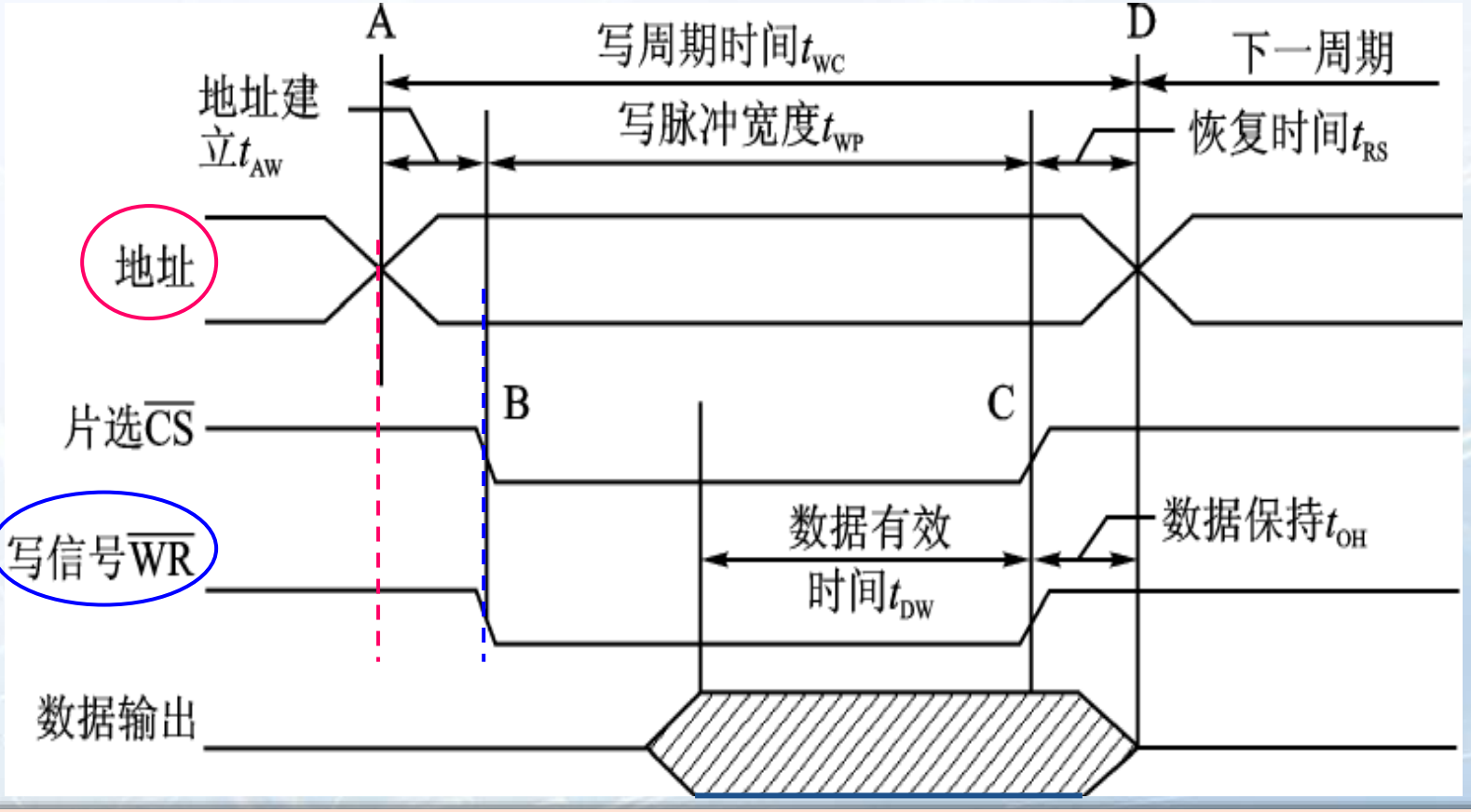
- 存储器读周期时间t_{RC}＝读取时间t_{AA}+读恢复时间t_{RS}。
- 两次读操作之间的时间间隔不应小于存储器读周期时间。



- CPU首先由地址总线给出地址信号，然后发出读操作控制信号，最后在数据总线上进行信息交流

SRAM写周期时序图

❖写周期时间t_{WC}＝地址建立时间t_{AW}+写入脉冲宽度t_{WP}+恢复时间t_{RS}

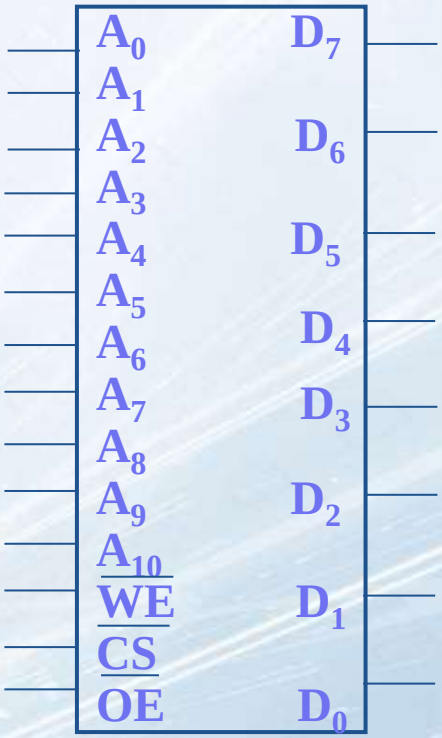


- 为保证在地址变化期间不会发生错误的写入而破坏存储器的内容， /WR在地址变化期间必须为高。只有在地址有效后再经过一段时间t_{AW}后，/WR才能有效。并且只有/WR变为高电平后再经过t_{RS}时间(恢复时间)，地址信号才能无效。

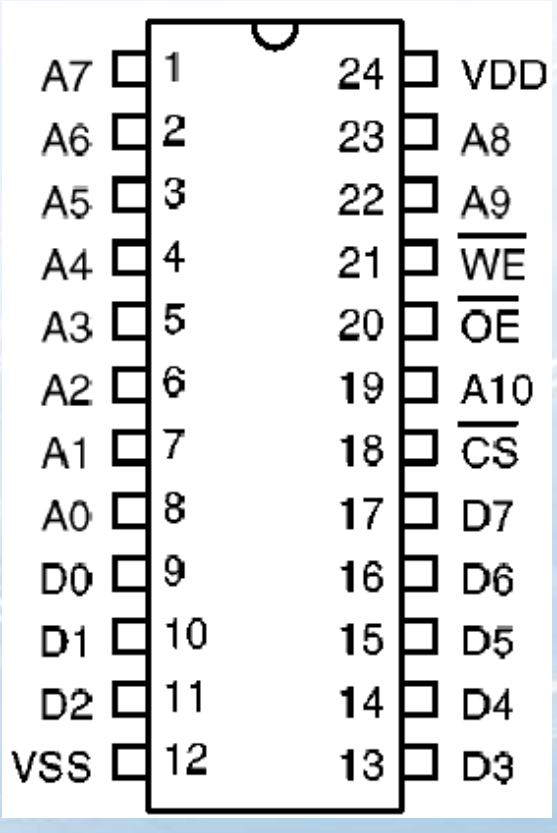
HM6116 SRAM芯片

- **HM6116（2K×8位）SRAM**
  - 存储容量：$2^{11}\times 8$ bit =2048字×8位
  - $\overline{WE}$
  - $\overline{CS}$
  - $\overline{OE}$: Output Enable，为0时读操作

- 写操作：$\overline{CS}$=0, $\overline{WE}$=0
- 读操作：$\overline{CS}$=0, $\overline{WE}$=1,$\overline{OE}$=0



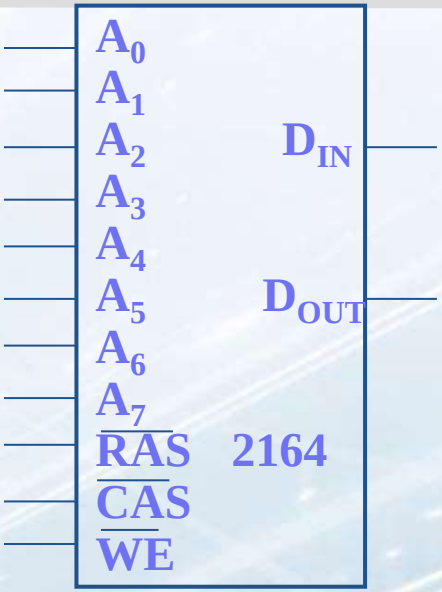
HM6116逻辑符号图



HM6116引脚图

DRAM典型芯片Intel 2164

- Intel 2164（64K×1位）DRAM
- 引脚名称
  - A_0～A_7　地址输入，2^{16}=64K个字
  - $\overline{CAS}$　列地址选通（Column Address Select）
  - $\overline{RAS}$　行地址选通（Row Address Select）
  - $\overline{WE}$　写允许（为0时写操作）

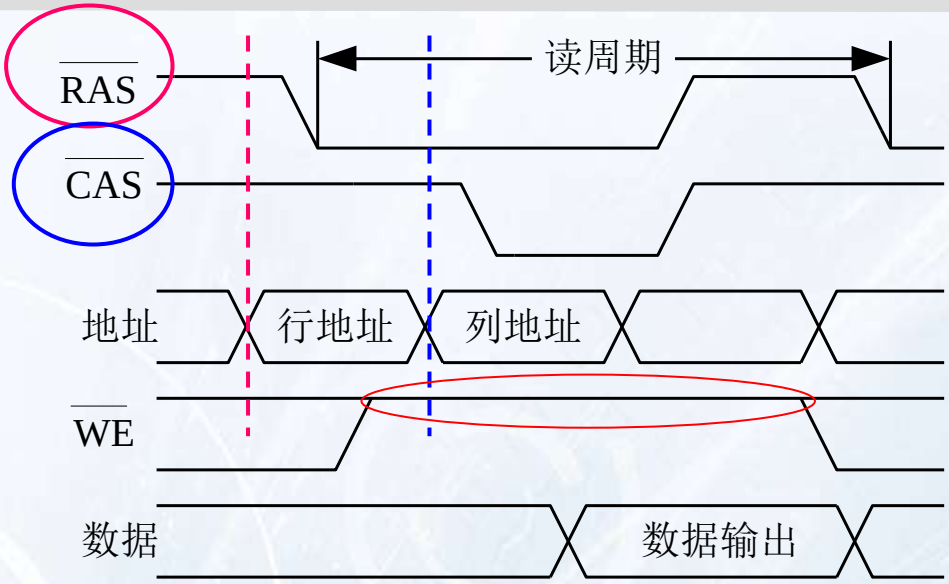


2164DRAM逻辑符号图

- 为了减少器件引脚，仅将8条地址线引到芯片外部。采用地址分时输入的方式，即地址代码分两次从同一组引脚输入。
  - 首先令/RAS=0，输入地址代码A_0~A_7，并锁存于行地址锁存器中
  - 然后令/CAS=0，输入地址代码A_8~A_{15}，并锁存于列地址锁存器中

2164每2ms需刷新一遍；每次刷新512单元，64K单元需128个刷新周期，每个刷新周期为15.625us

DRAM的读写时序控制（读周期）



- **读时序**

  行地址、$\overline{RAS}$有效

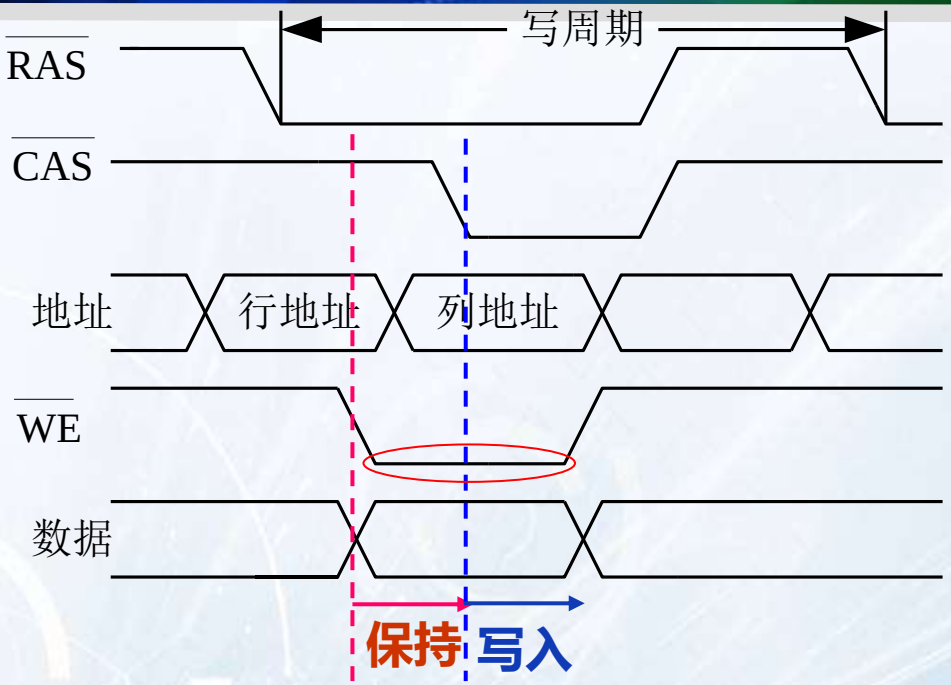
  写允许$\overline{WE}$无效(高)

  列地址、$\overline{CAS}$有效

  数据D_{OUT}有效

- 读周期（/WE=1时读操作）
  - 行地址必须在/RAS有效之前有效
  - 列地址也必须在/CAS有效之前有效
  - 且在/WE（为1）到来之前， /CAS必须为高电平——否则很可能对选中的存储单元进行的是写操作。

DRAM的读写时序控制（写周期）



- 写时序
  - 行地址、$\overline{RAS}$有效
  - 写允许$\overline{WE}$有效(低)
  - 数据D_{IN}有效
  - 列地址、$\overline{CAS}$有效
  - 数据被写入

- 写周期（/WE=0时写操作）
  - 当/WE有效（为0）之后，输入的数据必须保持到 /CAS变为低电平之后——数据提前有效
  - 在 /WE、 /RAS和/CAS全部有效时，数据被写入存储器

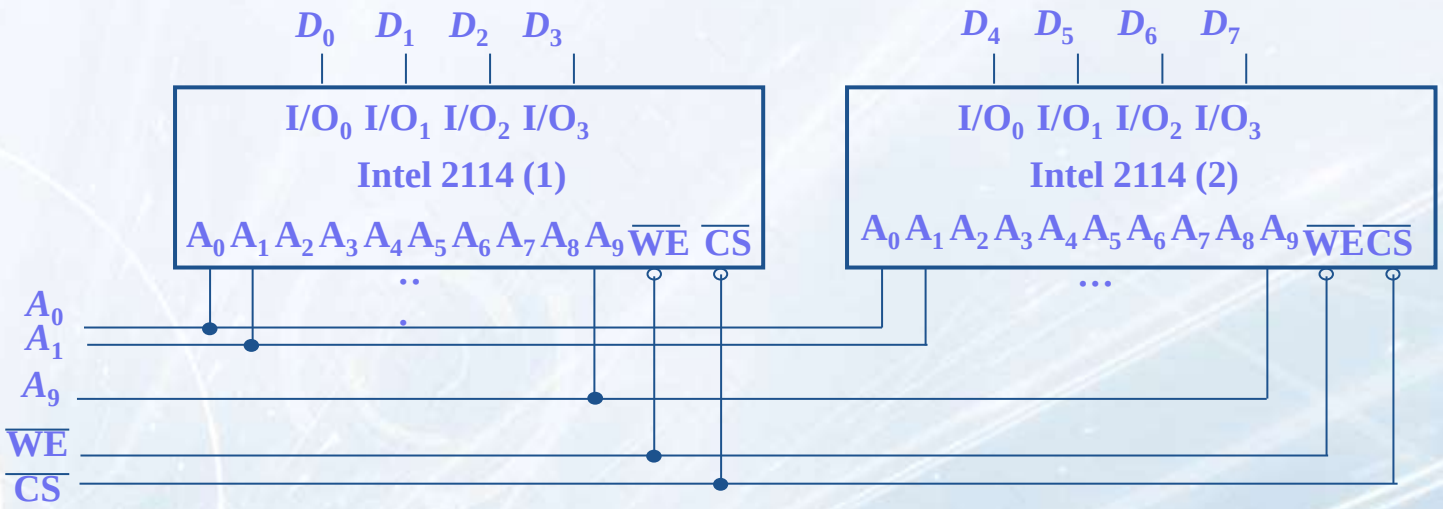
8.2.4 RAM芯片扩展

重点掌握！

- 存储器芯片的容量有限，当一片RAM或ROM不能满足对存储容量的要求时，需要对存储器进行扩展，即将若干片RAM或ROM组合起来，形成容量更大的存储器。
- 主要方法
  - 位扩展（增加字长）：若每一片存储器中的字数够用但每个字的位数不够用，应采用位扩展方式，增加存储器中每个字的位数
  - 字扩展（增加字数）：若每一片存储器中的数据位数够用但字数不够用，应采用字扩展方式，增加存储器字数
  - 字位扩展（扩展容量）：若一片存储器中的位数和字数都不够用，则应同时采用位扩展和字扩展方式，增加存储器的容量（位数和字数）

RAM芯片扩展（位扩展）

【例8.1】用2片Intel 2114（1K×4位）SRAM 扩展为1K×8位

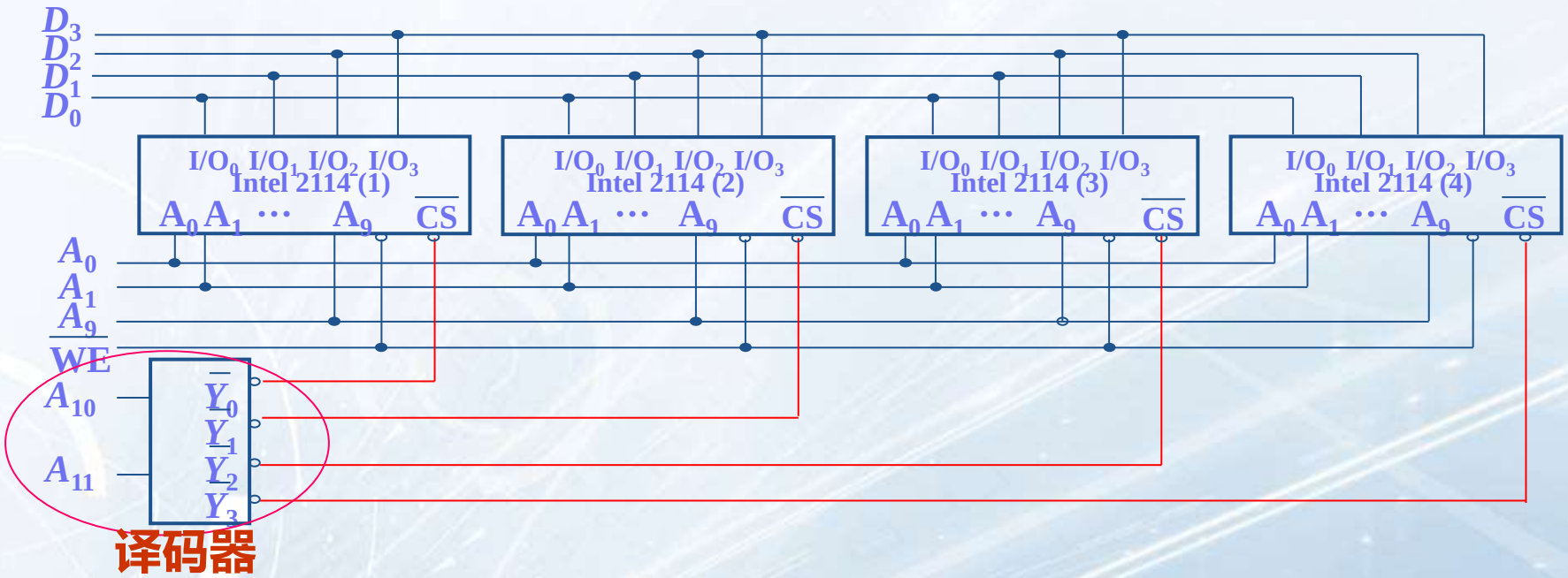


❖基本方法：

- 参与扩展的全部芯片的地址线、片选控制线$\overline{CS}$和写控制线$\overline{WE}$并接；
- 把低位芯片的数据线作为低位数据，高位芯片的数据线作为高位数据，数据位同时有效，实现I/O数据位数的增加。

RAM芯片扩展（字扩展）

【例8.2】用Intel 2114（1K×4位）SRAM 扩展为4K×4位



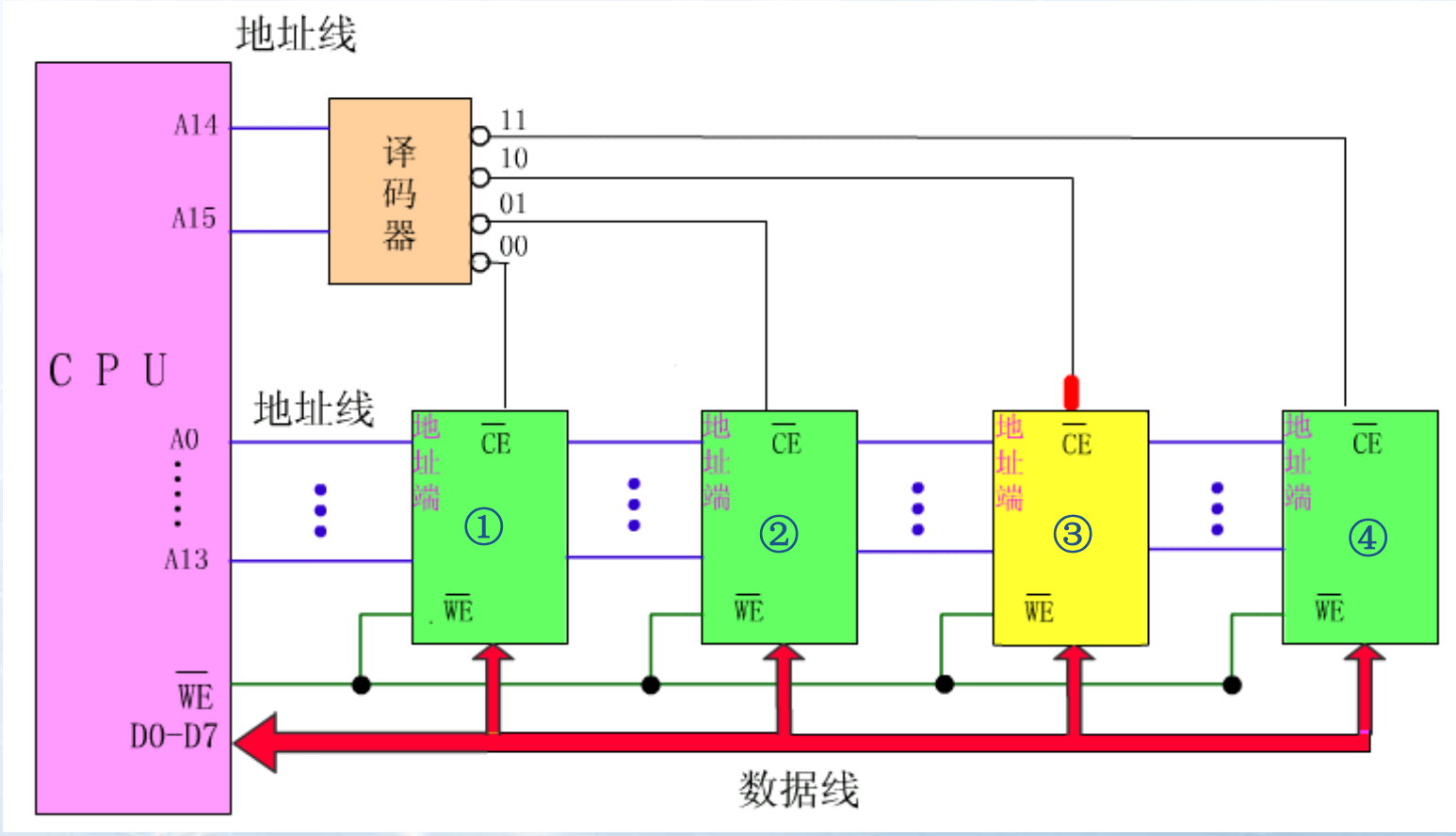
❖基本方法：

- 参与扩展的全部芯片的地址线、写控制线/WE和数据线分别并接；
- 把增加的高位地址线通过译码器产生译码信号来控制各芯片的片选控制/CS。

字扩展与CPU连接

【例8.3】用多片 SRAM （16K×8位）扩展为64K×8位，并与CPU相连接

- 分析：由于16K=$2^4 \times 2^{10}=2^{14}$，有14根地址线；64K=$2^6 \times 2^{10}=2^{16}$，有16根地址线，所以需要增加2根地址线；字数从16K扩展为64K，一片的字数为16K，则需要64K/16K=4片SRAM；需要一个2线-4线译码器将增加的高位地址信号译码后，输出分别接4片SRAM的/CS端。



RAM芯片扩展（字位扩展）

- 字位扩展（扩展容量）

【例8.4】用多片Intel 2114 （1K×4位）SRAM 扩展为4K×8位存储器

（1）计算扩展需要的芯片数 = 扩展后的存储器容量/芯片的容量

=（要求的字数×字长）/（芯片的字数×字长）

= (4K×8)/ (1K×4)=8（片）

（2）计算扩展需要增加的地址数

=扩展后的存储器地址线数 - 芯片的地址线数

=12-10=2（条）——送译码器产生2^2=4个片选/CS信号

（3）地址线、写控制线的连线

（4）画出与数据线的连线：分析芯片每几片为一组，进行位扩展？

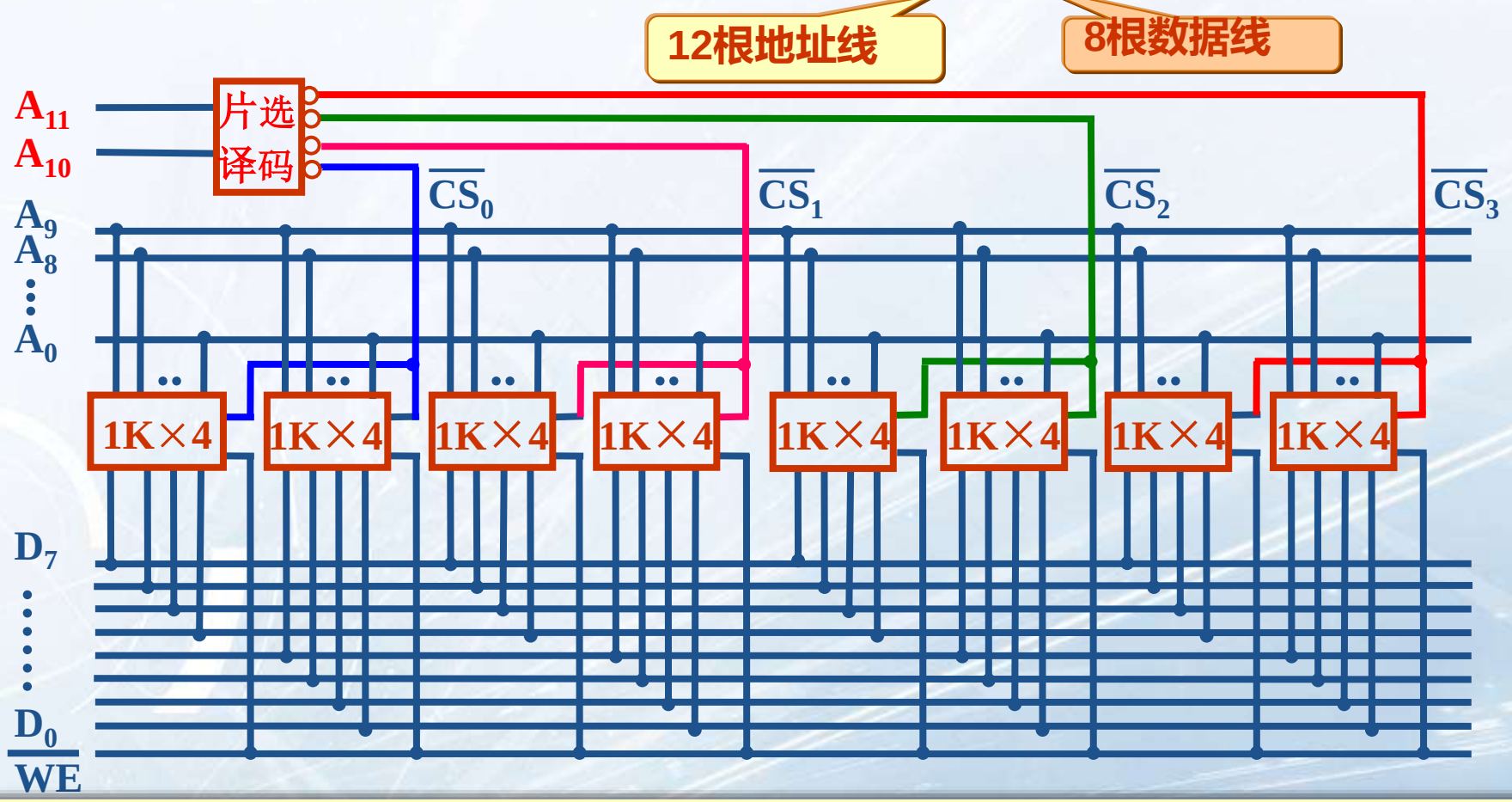
（5）将译码器的输出信号分别与这几组芯片的片选端相连，进行字扩展

？

- **如何用Intel 2114 （1K×4位）SRAM 扩展为8K×8位的存储器?**

【例8.4】1K×4位 SRAM 扩展为4K×8位存储器

用8片Intel 2114（1K×4）组成4K×8位的存储器



- **位扩展**：每2片为一组，公用一个片选信号，高位片的数据端接D7~D4，低位片的数据端接D3~D0，构成1K×8位存储器
- **字扩展**：译码器产生4个片选信号分别控制4组芯片，构成4K×8位存储器

8.3 只读存储器

内容概要

8.3.1 ROM的结构

8.3.2 ROM的扩展（同RAM的扩展）

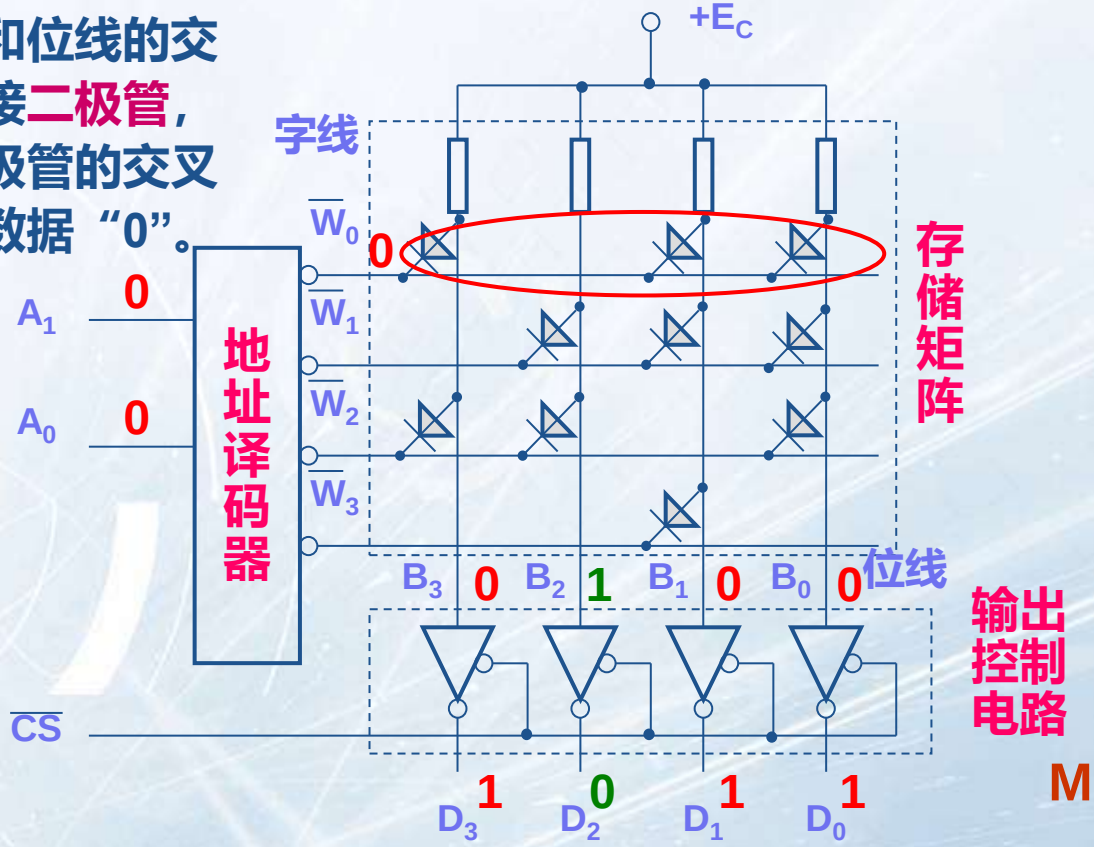
8.3.3 ROM的应用

8.3.1 ROM的结构

1、固定ROM

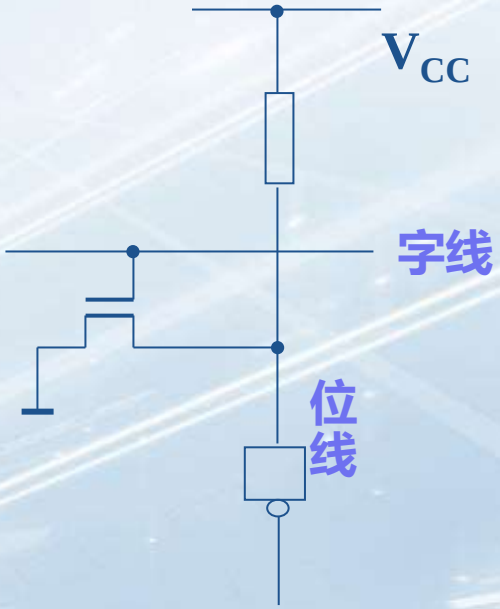
数据在工厂制作时写入，使用时不能更改。

需要写入“1”时，在字线和位线的交叉处连接二极管，不接二极管的交叉点表示数据“0”。



4×4位二极管固定ROM的结构

字线和位线的交叉处连接MOS管，表示存储的数据为“1”，不接MOS管的交叉点表示数据“0”。

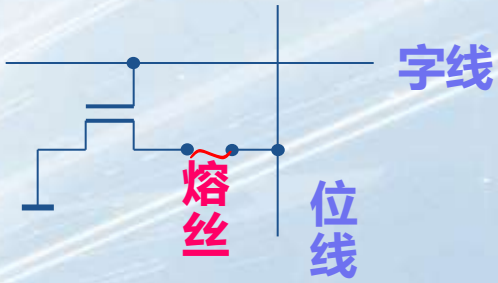


MOS管固定ROM存储单元

2、可编程ROM（PROM）

2、可编程ROM（Programmable ROM，PROM）

- 结构与固定ROM基本相同，区别在于PROM制造时，在每个字线和位线的交叉处都连接一个MOS管和一根熔丝，出厂时所有位均为“1”。
- 用户编程时（写入数据），对于要写0的单元加入特定的大电流，熔丝被烧断，数据为“0”；若保留熔丝，则数据仍为“1”。
- 由于熔丝不可恢复，所以数据由用户编程一次性写入，写入后的数据不能再更改。



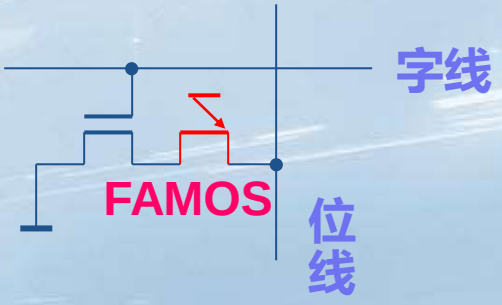
PROM存储单元

3、光可擦可编程ROM（EPROM）

3、光可擦可编程ROM（ Erasable PROM，EPROM）

- EPROM一般指紫外线擦除的可编程ROM（Ultra-Violet Erasable Programmable Read-Only Memory，简称UVE-PROM）
- 可以由用户通过专用的编程器多次编程写入数据，且可以用紫外光擦除后再改写，适于需要经常修改ROM中内容的场合。
- EPROM存储单元在每个字线和位线的交叉处都连接一个MOS管和一个FAMOS 管
- FAMOS：Floating-gate Avalanche-injection MOS，浮栅雪崩注入式MOS，为P沟道MOS管
- 擦除数据时，需要把EPROM芯片放在专门的擦除器中，紫外光下照射15~30分钟

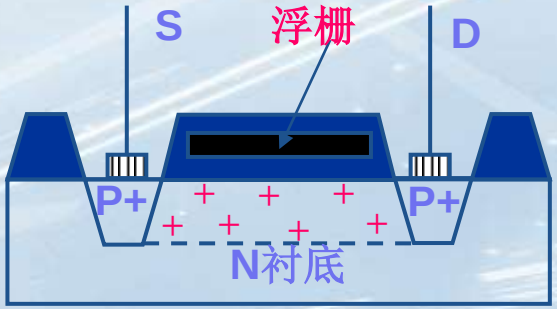
EPROM存储单元



EPROM的工作原理

- FAMOS与普通PMOS结构相似，但栅极没有被引出，被二氧化硅绝缘层包围，称为浮栅。
- 当浮栅没有电荷时，沟道不能形成，FAMOS截止，表示写入“0”。
- 当要写入“1”（注入电荷）时
  - 首先在漏极接上足够大的负电压（－30V左右），使部分自由电子获得高能量，以高速撞击其它电子，使高能自由电子数量越来越多，产生雪崩效应。一部分高能自由电子越过二氧化硅绝缘层，注入到浮栅中；同时在衬底留下大量的空穴，形成P沟道。
  - 当漏极电压消失后，由于浮栅周围都是绝缘层，注入浮栅的电荷基本不能泄漏，可以长期保存下来；电荷产生的电场，使P沟道形成，FAMOS导通。

浮栅不注入电荷时，表示写入“0”；注入电荷表示写入“1”



FAMOS结构示意图

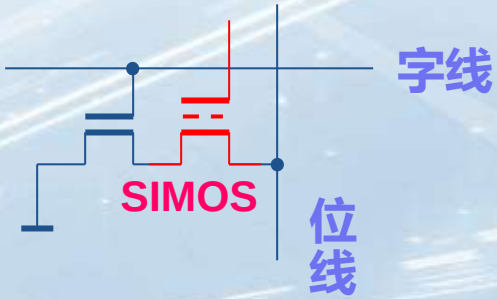
4、电可擦可编程ROM

4、电可擦可编程ROM（EEPROM, EAROM）

- 虽然紫外线擦除的EPROM具备可擦除重写的功能，但擦除操作复杂，擦除速度很慢（需15~30分钟）
- EEPROM（Electricity Erasable Programmable Read Only Memory）的数据可以用电擦除，并由用户多次编程写入。
- EAROM：Electrically Elterable ROM，电可修改ROM
- EEPROM存储单元在每个字线和位线的交叉处都连接一个MOS管和一个SIMOS 管
- SIMOS: Stacked-gate Injection MOS，重叠栅注入式MOS，为N沟道增强型MOS管

❖可擦除可编程ROM芯片

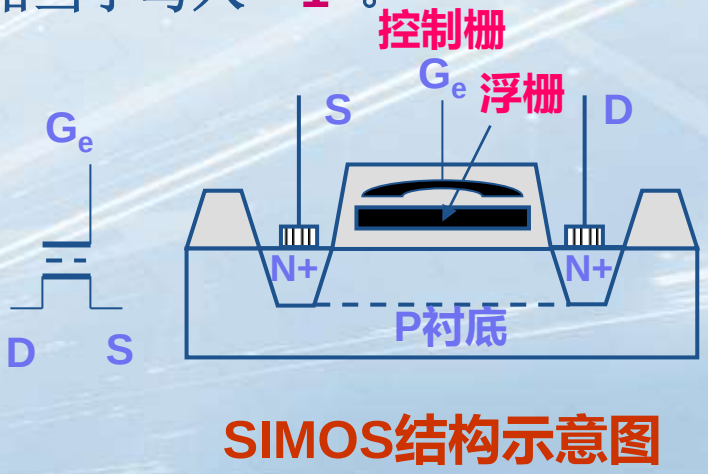
EPROM	EEPROM	容量
2716	27C16	2K×8
2732	27C32	4K×8
2764	27C64	8K×8



EEPROM存储单元

EEPROM的工作原理

- SIMOS有2个栅极：控制栅G_e（用于控制数据读出和写入）和浮栅G_f（用于长期保存注入电荷）
- 写入数据前，浮栅不带电，表示数据“0”。
- 在控制栅Ge加上正常高电平电压时，能够在漏、源极之间形成导电沟道，使SIMOS导通。
- 当要写入“1”（注入电荷）时
  - 在漏、源极间加上高电压（25V），同时在控制栅加上高压正脉冲（25V，25ms），在栅极电压作用下，自由电子穿过SiO_2绝缘层到达浮栅，被浮栅截获形成注入电子，相当于写入“1”。
- 写入的数据可以用紫外线或X射线擦除
- 也可以用电快速擦除
  - 在控制栅Ge上加一个高压负脉冲，可以在SiO_2层中感应出足够量的正电荷与浮栅中的负电荷中和，将原来存储的“1”擦除。



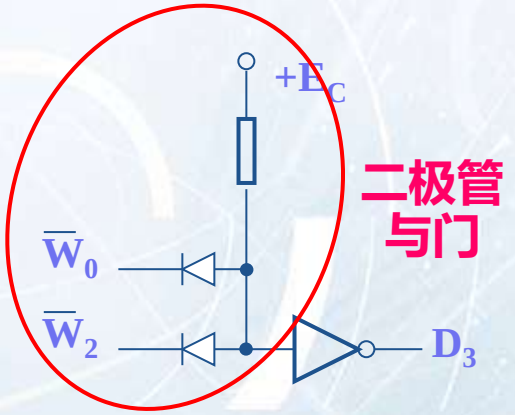
SIMOS结构示意图

8.3.3 ROM的应用

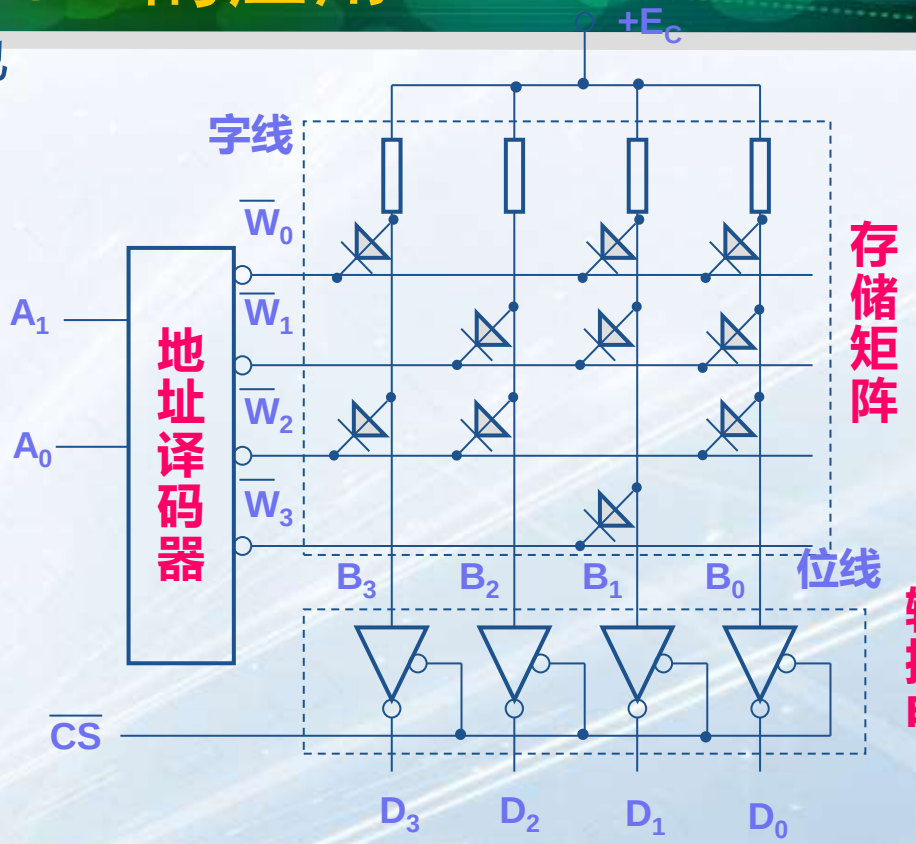
- 主要用途是存放数据和程序，也可实现组合逻辑电路

【例8.5 】分析右图数据输出与地址输入的逻辑关系

- 解：（1）单独画出D_3 输出的结构图，写出输出的逻辑表达式



$$D_3 = \overline{\overline{W_0} \cdot \overline{W_2}}$$



（2）写出译码器输出的逻辑表达式

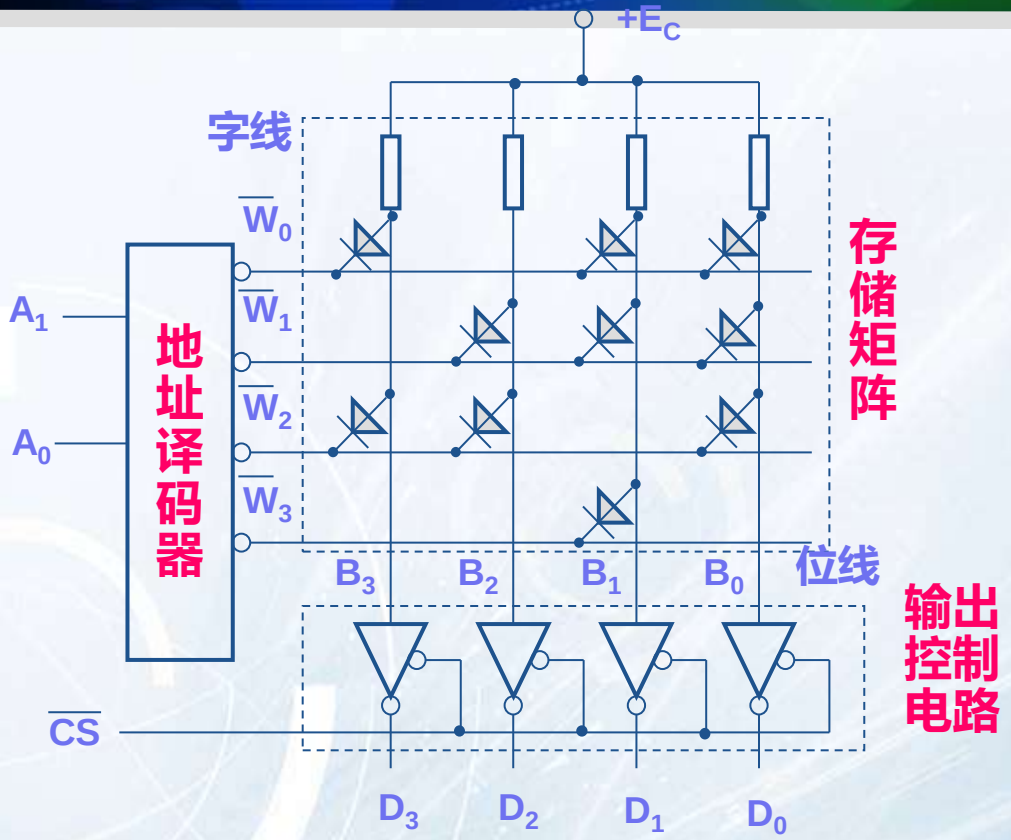
$$\overline{W}_0 = \overline{m}_0 = \overline{\overline{A}_1 \overline{A}_0}$$

$$\overline{W}_2 = \overline{m}_2 = \overline{A_1 \overline{A}_0}$$

（3）推导出数据输出与地址输入的逻辑关系

$$D_3 = \overline{\overline{W_0} \cdot \overline{W_2}} = \overline{\overline{\overline{A_1} \cdot \overline{A_0} \cdot \overline{A_1} \cdot \overline{A_0}}} = \overline{A_1} \cdot \overline{A_0} + A_1 \cdot \overline{A_0}$$

【例8.5 】数据输出与地址输入的逻辑关系



$$D_3 = \overline{\overline{W_0} \cdot \overline{W_2}} = \overline{\overline{A_1} \cdot \overline{A_0} \cdot A_1 \cdot \overline{A_0}} = \overline{A_1} \cdot \overline{A_0} + A_1 \cdot \overline{A_0}$$

译码器每个输出的逻辑表达式

$$\overline{W_0} = \overline{m_0} = \overline{\overline{A_1} \cdot \overline{A_0}}$$
$$\overline{W_1} = \overline{m_1} = \overline{\overline{A_1} \cdot A_0}$$
$$\overline{W_2} = \overline{m_2} = \overline{A_1 \cdot \overline{A_0}}$$
$$\overline{W_3} = \overline{m_3} = \overline{A_1 \cdot A_0}$$

$$D_2 = \overline{\overline{W_1} \cdot \overline{W_2}} = \overline{A_1} \cdot A_0 + A_1 \cdot \overline{A_0}$$
$$D_1 = \overline{\overline{W_0} \cdot \overline{W_1} \cdot \overline{W_3}} = \overline{A_1} \cdot \overline{A_0} + \overline{A_1} \cdot A_0 + A_1 \cdot A_0$$
$$D_0 = \overline{\overline{W_0} \cdot \overline{W_1} \cdot \overline{W_2}} = \overline{A_1} \cdot \overline{A_0} + \overline{A_1} \cdot A_0 + A_1 \cdot \overline{A_0}$$

将存储矩阵简化为点阵图

- 存储器的数据输出与地址译码器的输出构成与非逻辑关系 $D_3 = \overline{\overline{W_0} \cdot \overline{W_2}}$
- 与地址译码器输出的非构成或逻辑，将其结构图简化为点阵图 $D_3 = W_0 + W_2$

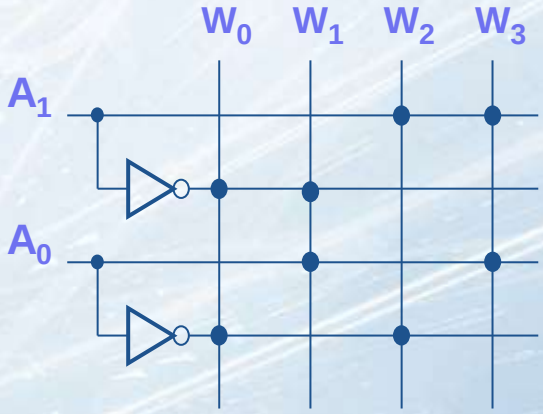
存储矩阵点阵图（或逻辑）

- 地址译码器的输出与输入也构成与非逻辑关系；

$\overline{W}_0 = \overline{m}_0 = \overline{\overline{A}_1 \overline{A}_0}$　　$\overline{W}_2 = \overline{m}_2 = \overline{A_1 \overline{A}_0}$

- 地址译码器输出的非与输入构成与逻辑关系

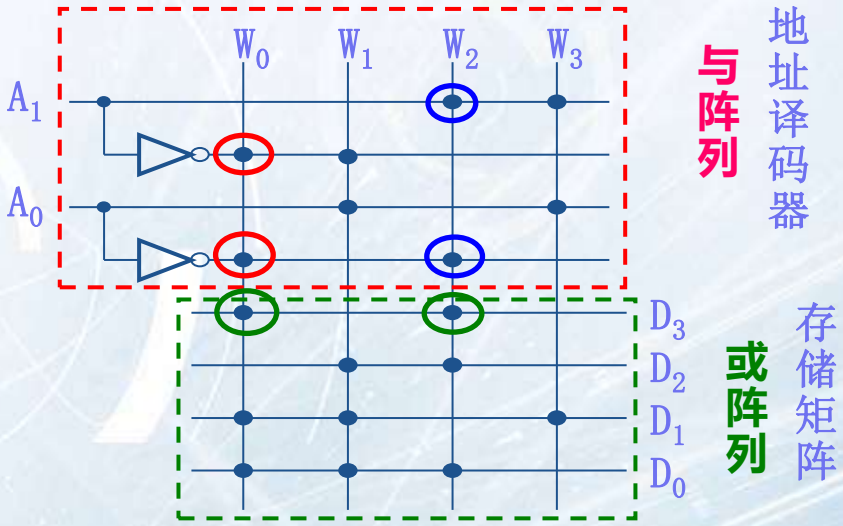
$W_0 = \overline{A}_1 \cdot \overline{A}_0$　　$W_2 = A_1 \cdot \overline{A}_0$



地址译码器点阵图（与逻辑）

固定ROM的点阵图

- 将存储矩阵点阵图与地址译码器点阵图组合起来，就得到整个固定ROM的点阵图。
- 把地址译码器实现的逻辑，等效为“与”逻辑，称为与阵列；把存储矩阵实现的逻辑，等效为“或”逻辑，称为或阵列。



固定ROM点阵图

- ROM的数据输出与地址输入构成与或逻辑关系

$$D_3 = \overline{A}_1 \cdot \overline{A}_0 + A_1 \cdot \overline{A}_0$$

- ROM的结构由一个与阵列和一个或阵列构成。

ROM实现任意组合逻辑函数的设计方法

- 任何一个组合逻辑都可以用“与 - 或”式（最小项表达式）来描述
- 从ROM的点阵图结构可知，地址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，因此ROM可以实现任意组合逻辑函数

❖设计方法

（1）列出真值表；

（2）采用最小项推导法写出输出信号的逻辑函数表达式；

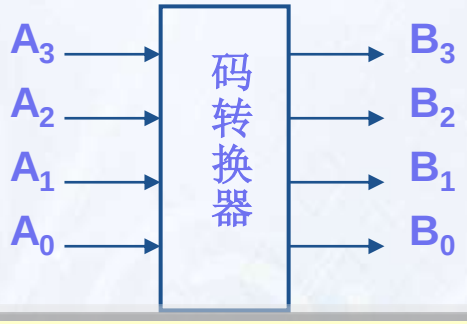
（3）先画出与阵列，行线为输入信号的原变量和反变量，列线是对应输入的各个最小项；

（4）再画出或阵列，列线是对应输入的各个最小项，行线是各个输出信号；

根据某输出包含的最小项，在与这些最小项的列线交叉处画上小圆点。

ROM的应用举例

【例8.6】用ROM设计一个码转换器，实现4位二进制码到4位循环码（格雷码3）的转换。



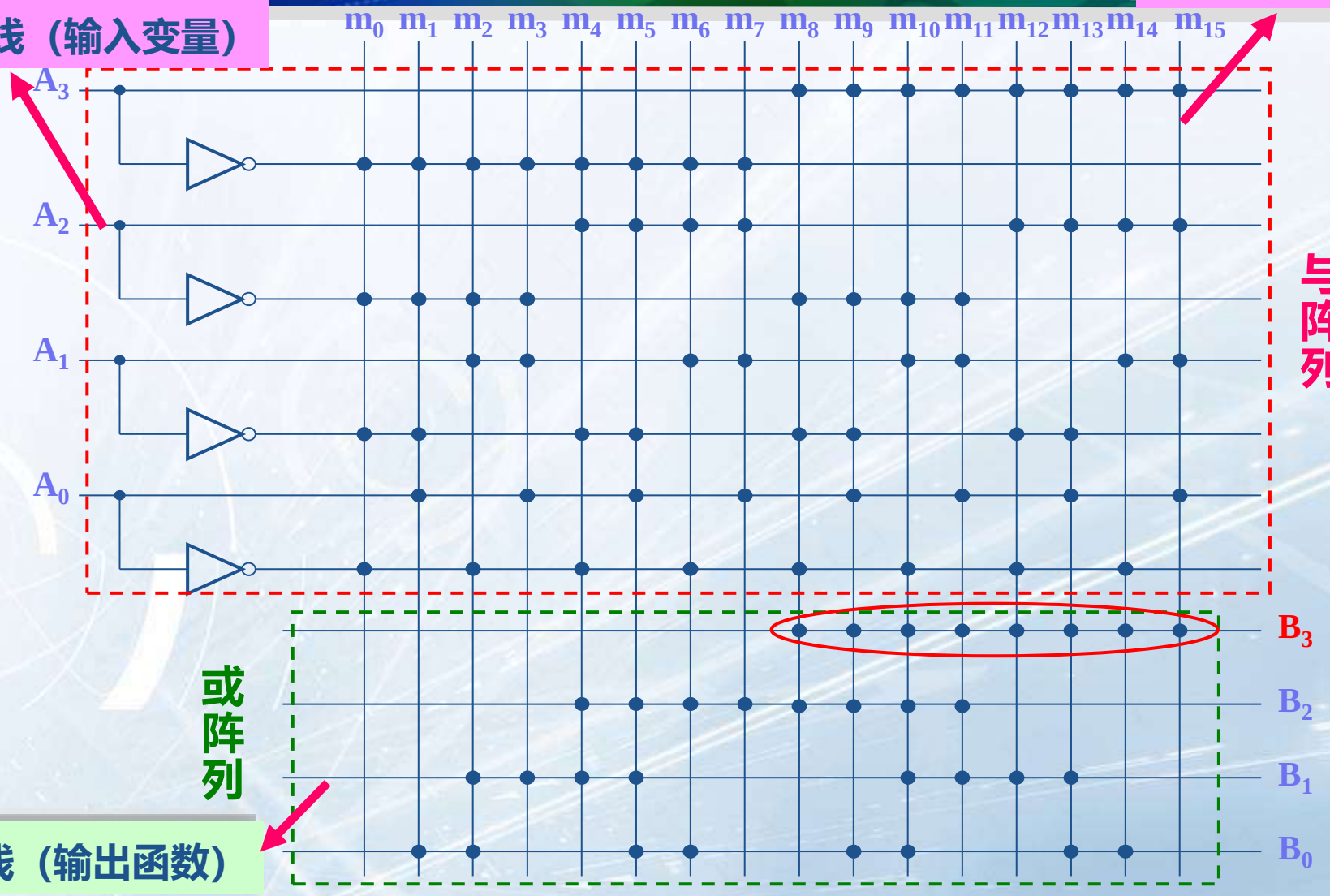
$$B_3=\sum_m(8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15)$$
$$B_2=\sum_m(4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11)$$
$$B_1=\sum_m(2,\ 3,\ 4,\ 5,\ 10,\ 11,\ 12,\ 13)$$
$$B_0=\sum_m(1,\ 2,\ 5,\ 6,\ 9,\ 10,\ 13,\ 14)$$

❖根据输出的逻辑函数表达式画出ROM点阵图

在接入存储器件的矩阵交叉点上画一个圆点，代表存储器件。图中接入存储器件表示存1，不接存储器件表示存0。

m_i	A_3 A_2 A_1 A_0	B_3 B_2 B_1 B_0
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

ROM实现的码转换器点阵图



Back

思考：哪种器件可以实现任意组合逻辑电路？

- 在第5章组合逻辑电路中我们学过，哪种器件可以实现任意组合逻辑电路?
- 如何做?

8.4 基于Verilog HDL的存储器设计

内容概要

8.4.1 RAM的HDL设计

8.4.2 ROM的HDL设计

8.4.1 RAM的HDL设计

- RAM的HDL设计
  - 若干个相同宽度的向量构成数组，reg（寄存器）型数组变量即为memory（存储器）型变量。
  - memory型变量定义语句如下：

```
reg[7:0]        mymemory[1023:0];
```

字长为8位

1024个字

  - 经定义后的memory型变量可以用下面的语句对存储器单元赋值（即写入）：

```
mymemory[7] = 75;  //存储器mymemory的第7个字被赋值75
```

- memory型变量相当于一个RAM

8×8位RAM的HDL设计

【例8.7】 8×8位RAM的Verilog HDL的源程序

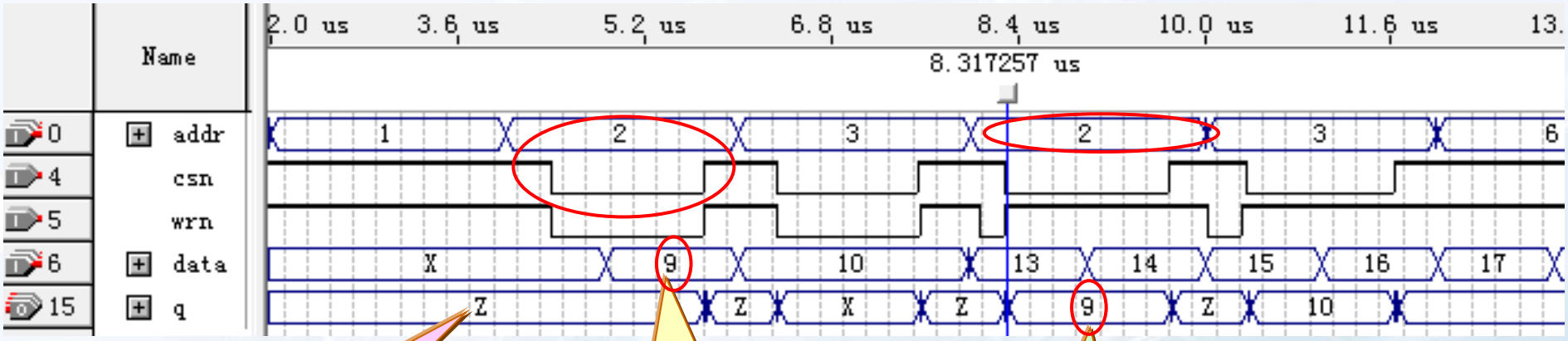
```
module ram8x8(addr,csn,wrn,data,q);
    input[2:0]   addr;              //字数为2^3=8
    input        csn,wrn;
    input[7:0]   data;              //输入数据
    output[7:0]  q;                 //RAM的输出端
    reg[7:0]     q;
    reg[7:0]     mymemory[7:0];
    always
      begin
        if (csn==1)  q =8'bzzzzzzzz;      // RAM禁止工作
        else if (wrn == 0)                //写操作
          mymemory[addr]= data;
        else if (wrn == 1) q = mymemory[addr]; //读操作
      end
endmodule
```

可以写成always @(posedge addr)吗? 为什么?

RAM的仿真波形阶段分析

- 第1阶段
  - 初始时csn=1， wrn=1，存储器处于禁止工作状态
  - 输出端为高阻状态（z）。
- 第2阶段
  - 使csn=0和wrn=0，存储器处于写操作状态
  - 存储器根据地址addr的变化（如地址2和3），将数据data写入存储器，此时的输出未具体赋值（wrn＝0，q高阻抗），输出q为未知（z）。
- 第3阶段
  - csn=0和wrn=1，存储器处于读操作状态
  - 存储器根据地址addr的变化（如地址2和3），将已写入该存储单元的数据送到q输出端。

8×8 RAM的仿真波形图



csn无效时禁止工作

将数据9写入第2个单元

将第2个单元中的数据（9）读出

- 一定要按照RAM的写周期和读周期的时序来编辑输入信号波形，否则仿真可能看不到预期的效果!

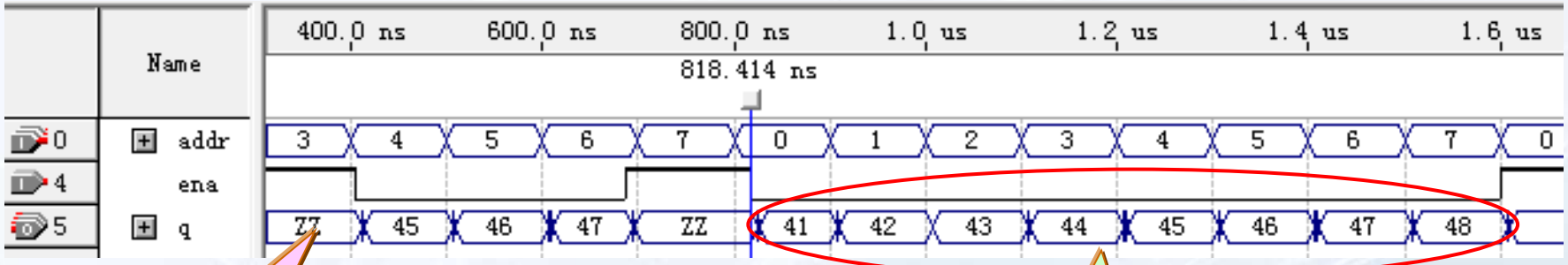
8.4.2 ROM的HDL设计

- 对于容量不大的ROM，可以用case语句实现

【例8.8】用case语句实现8×8位ROM（8个存储字中分别存储h41、h42、……、 h48）

```
module rom8x8(addr,ena,q);
    input [2:0]    addr;                  //字数为2^3=8
    input          ena;                   //使能控制信号，低有效
    output [7:0]   q;
    reg   [7:0]    q;
    always @(ena or addr)
      begin
        if (ena)   q =8' bzzzz_zzzz;  else
        case (addr)
            0:q =8'b0100_0001;  1:q =8'b0100_0010;
            2:q =8'b0100_0011;  3:q =8'b0100_0100;
            4:q =8'b0100_0101;  5:q =8'b0100_0110;
            6:q =8'b0100_0111;  7:q =8'b0100_1000;
           default :q =8'bzzzz_zzzz;
        endcase
      end
endmodule
```

rom8x8.v的仿真波形图



ena=1时禁止工作，ROM输出为高阻态

ena=0时读出ROM 中数据

- 若ROM中数据需要修改，可以通过修改case语句中对q的赋值来实现

用memory型变量实现8×8位ROM

【例8.9】用memory型变量实现8×8位ROM

```
module rom8x8_mem(addr,ena,q);
    input[2:0]      addr;        //字数为2^3=8
    input           ena;         //使能控制信号，低有效
    output [7:0]    q;
    reg[7:0]        q;
    reg[7:0]        ROM[7:0];// memory型变量，位宽为8位，8个存储字
   always @(ena or addr)
      begin
          ROM[0] = 8'b0100_0001;  ROM[1] = 8'b0100_0010;
          ROM[2] = 8'b0100_0011;  ROM[3] = 8'b0100_0100;
          ROM[4] = 8'b0100_0101;  ROM[5] = 8'b0100_0110;
          ROM[6] = 8'b0100_0111;  ROM[7] = 8'b0100_1000;
          if (ena)   q = 'bzzzz_zzzz; // ROM禁止工作
          else q=ROM[addr];       // 读操作
      end
endmodule
```

仿真波形同 rom8x8.v

利用lpm_ram_dq 实现256x8 RAM

【例8.10】 利用参数化的RAM模块lpm_ram_dq 实现256x8 RAM

```
module ram256x8 (data, address, we, inclock, outclock, q);
    input [7:0]data;
    input [7:0]address;
    input we, inclock, outclock;
    output [7:0]q;
    lpm_ram_dq myram (.q (q), .data (data), .address (address),
                      .we (we), .inclock (inclock), .outclock (outclock));
   defparam myram.lpm_width = 8;     //参数的传递，数据总线的宽度
   defparam myram.lpm_widthad = 8; //定义地址总线的宽度
endmodule
```

例化的元件名

实例的端口

模块的端口

信号名对应

被调用的模块名

- lpm_ram_dq的例化还可写为：

```
lpm_ram_dq # (8, 8) myram (q, data, address,we, inclock, outclock);
```

位置对应

参数的传递

建立存储器初始化文件

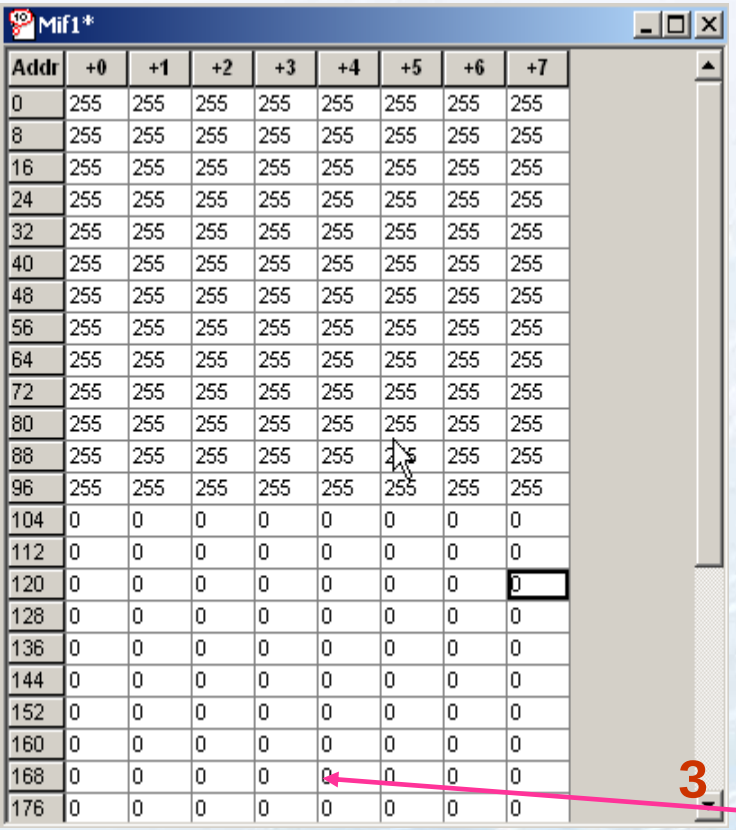
- 当在设计中使用了器件内部的存储器模块（RAM、ROM或双口RAM）时，需要对存储器模块进行初始化。
- 可利用存储器编辑器（Memory Editor）建立或编辑Intel Hex格式（.hex）或Altera存储器初始化格式（.mif）的文件。

- RAM中的内容可以存于.mif（Memory Initialization File）文件中。
- .mif文件可以在Quartus Ⅱ中使用File/New菜单命令，选择“Memory Initialization File”项来创建，然后编辑RAM中数据。

创建存储器初始化文件的方法

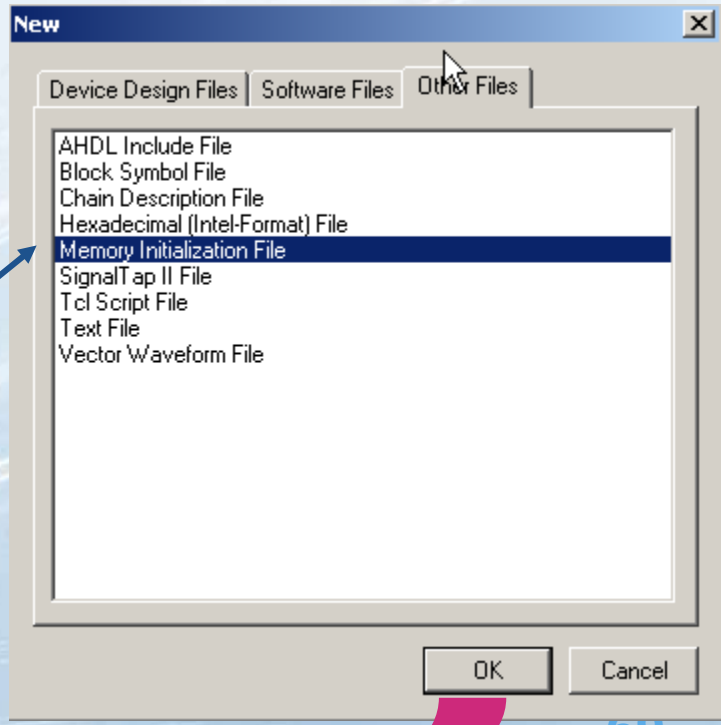
(1) 新建一个存储器初始化（.mif）文件

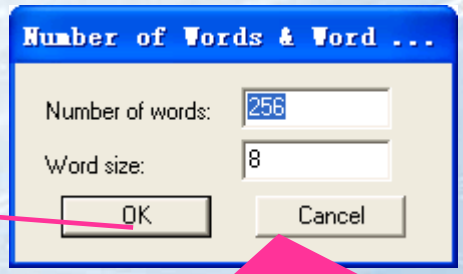
“File>New…” 命令→ “Other Files”标签→“Memory Initialization File” → 单击“OK” →在弹出的对话框中输入字数和字长→单击“OK” 。



1 File ⇒ New ⇒ Other Files 标签

2 MIF format





3

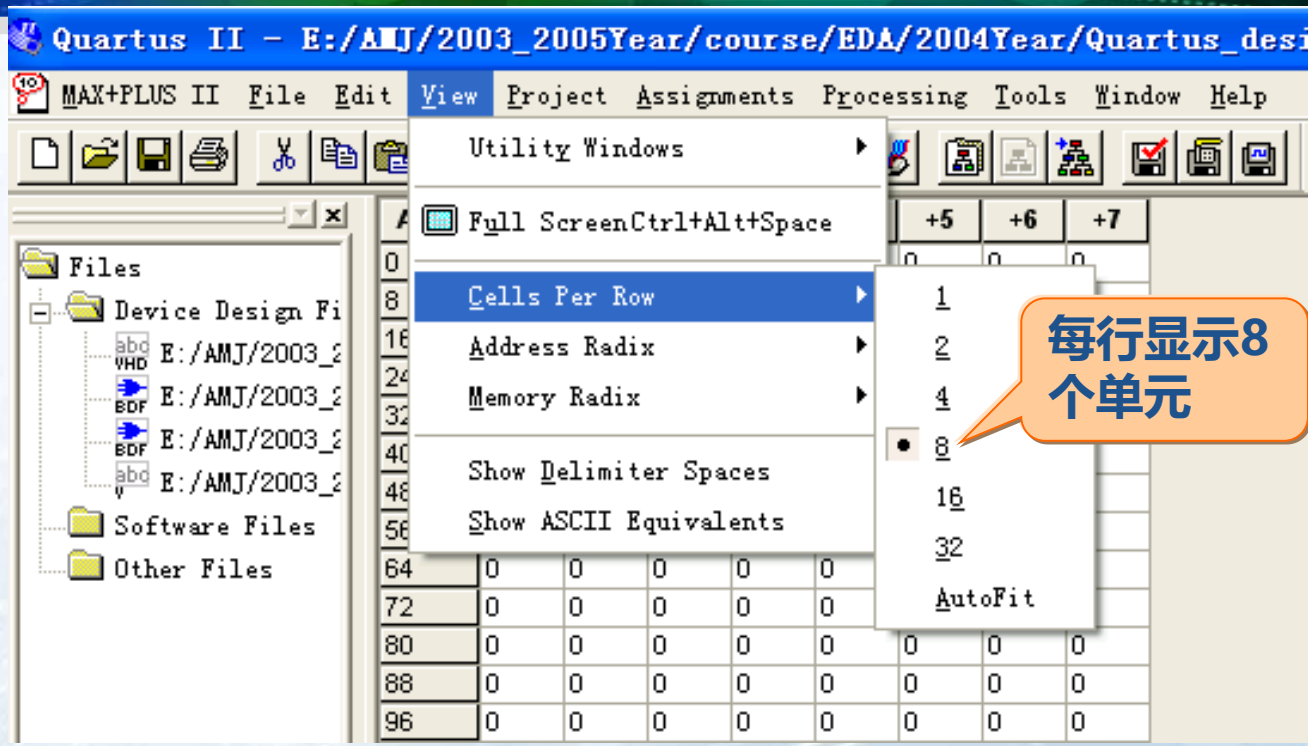
69

创建存储器初始化文件的方法（续）

（2）打开存储器编辑窗口

（3）改变编辑器选项

利用“View”菜单命令，改变地址或字的显示格式等



（4）编辑存储器内容

选择要编辑的字（反白显示），直接输入内容。

（5）保存文件

文件后缀为.mif（当选择“Memory Initialization File”时）。

课后练习

- 自己在Quartus II中利用lpm_ram_dq编程实现256x8 RAM
- 在Help中查看其 lpm_ram_dq的功能描述，并总结；
- 根据lpm_ram_dq的工作时序对ram256x8.v进行仿真，要求仿真所有的功能
- 对仿真波形进行分析

8.5 PLD的基本原理

内容概要

8.5.1 可编程逻辑器件的分类

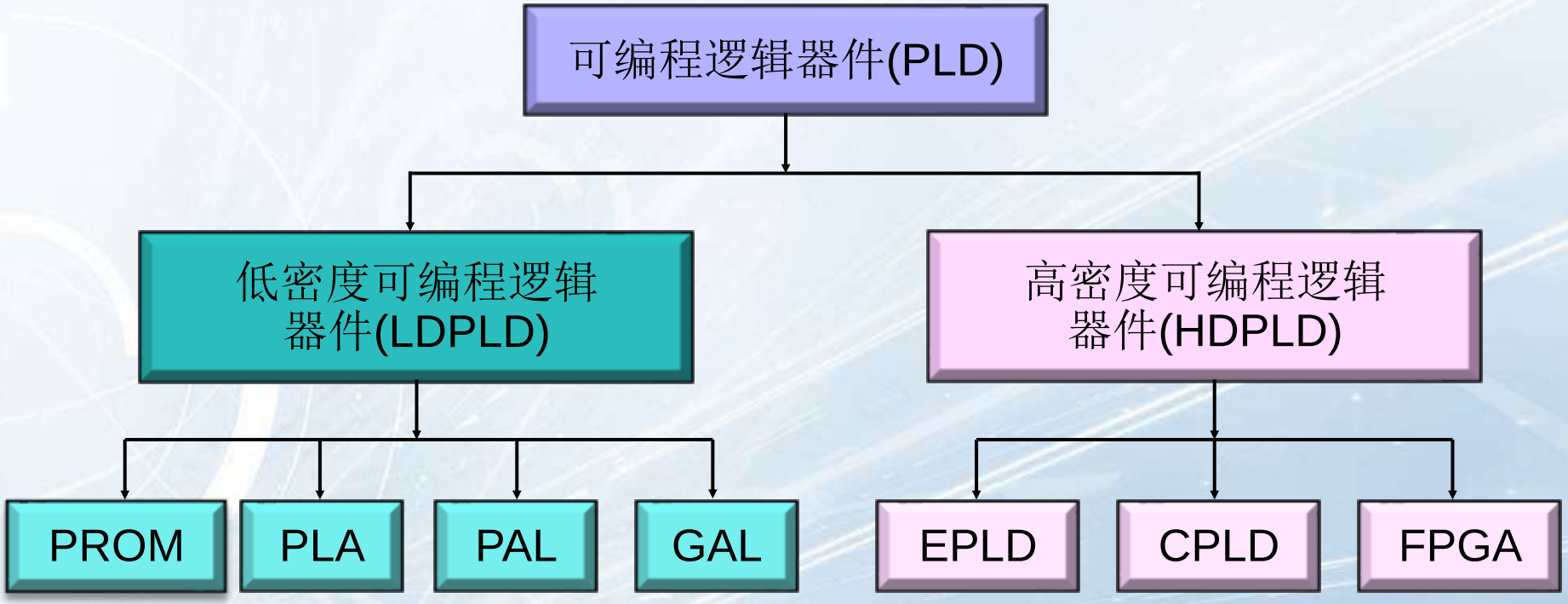
8.5.2 阵列型PLD

8.5.3 现场可编程门阵列FPGA

8.5.4 基于查找表的结构

8.5.1 可编程逻辑器件的分类

- 可编程逻辑器件（PLD， Programmable Logic Device）
  没有统一和严格的分类标准，通行的分类方法是按集成密度、编程方式、结构特点来分类。



可编程逻辑器件按集成密度分类

1、按集成密度分类

- 分为低密度可编程逻辑器件LDPLD（Low Density PLD）和高密度可编程逻辑器件HDPLD （High Density PLD）
  - LDPLD 通常是指早期发展起来的、集成密度小于1000门/片左右的PLD，如PROM、PLA、PAL和GAL等
    - PROM： Programmable Read Only Memory，可编程只读存储器
    - PLA ：Programmable Logic Array，可编程逻辑阵列
    - PAL：Programmable Array Logic，可编程阵列逻辑
    - GAL：Generic Array Logic，通用阵列逻辑

高密度可编程逻辑器件HDPLD

- HDPLD集成密度大于1000门/片
  - EPLD：Electrically Programmable Logic Device，电可擦除可编程逻辑器件
  - CPLD：Complex Programmable Logic Device，复杂可编程逻辑器件
  - FPGA： Field Programmable Gates Array，现场可编程门阵列器件
- 几类HDPLD芯片
  - Altera公司的EPM9560，密度12000门/片
  - Lattice公司的pLSI/ispLSI3320，密度14000门/片
  - Xilinx公司的Virtex-5 FPGAs, 330,000 logic cells,1200pins
  - Altera Stratix IV FPGA，680K 逻辑单元（LE）
  - 目前集成度最高的HDPLD可达400万门/片以上

2、按编程方式分类

（1） 一次性编程（OTP，One Time Programmable ）PLD

（2）可多次编程（MTP，Many Time Programmable） PLD

一次性编程PLD

- ✓ 采用熔丝工艺制造，采用熔丝或反熔丝编程元件。仅能一次性编程，不能重复编程和修改
- ✓ 优点：可靠性与集成度高，抗干扰性强
- ✓ 不适用于数字系统的研制、开发和实验阶段使用，而适用于产品定型后的批量生产

可多次编程PLD

- ✓ 大多采用场效应管作编程元件，控制存储器存储编程信息。通常采用**EPROM、EEPROM、FLASH或SRAM**工艺制造。
- ✓ 可重复编程和修改，适用于数字系统的研制、开发和实验阶段使用

3、按可编程元件的结构及编程方式分类（1/2）

- PLD是一种数字集成电路的半成品，芯片上集成了大量的门和触发器等基本逻辑元件，使用者可以利用EDA工具对它进行加工，把片内的元件连接起来，使之实现某个逻辑电路或系统功能。
- PLD实际上是通过对器件内部的基本可编程元件进行编程来实现用户所需的逻辑功能的。

- 基本可编程元件：
  - 熔丝型开关
  - 反熔丝型开关 { PLICE反熔丝；ViaLink元件 }
  - 基于浮栅编程技术的可编程元件 { 紫外光擦除EPROM；电擦除EPROM；闪速存储器Flash Memory }
  - 基于SRAM的可编程元件

3、按可编程元件的结构及编程方式分类（2/2）

类 型	存储编程信息的元件	擦除方式	掉电易失性	编程次数
采用熔丝型或反熔丝型开关的PLD	PROM	不可擦除	非易失性	一次
采用紫外光擦除、电可编程元件的PLD	EPROM	紫外光擦除	非易失性	多次
采用电擦除、电可编程元件的PLD	EEPROM或Flash Memory	电擦除	非易失性	多次
基于查找表技术、SRAM工艺的PLD	SRAM	电擦除	易失性	多次

- **非易失性器件**在编程后，配置数据将一直保持在器件内，掉电后数据也不会丢失，直至将它擦除或重写。
- **易失性器件**在编程后，每次掉电后数据会丢失，在每次上电时需要重新配置数据。

4、按结构特点分类

（1）阵列型PLD

- 基本结构：与阵列、或阵列
- 简单PLD（如PROM、PLA、PAL和GAL等）、EPLD（ Erasable Programmable Logic Device）和CPLD（Complex Programmable Logic Device，复杂可编程逻辑器件）

（2）单元型PLD器件

- 基本结构：逻辑单元
- 结构形式：门阵列的形式，它由许多可编程单元（或称逻辑功能块）排成阵列组成。
- 现场可编程门阵列FPGA（ Field Programmable Gates Array ）

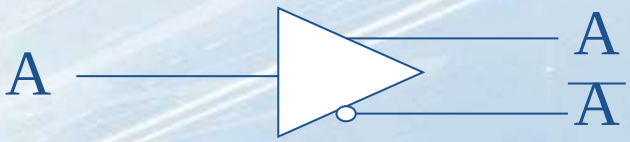
8.5.2 阵列型PLD

- 阵列型PLD包括PROM、PLA、PAL、GAL、EPLD及CPLD。
- EPLD和CPLD都是在PAL和GAL基础上发展而来。

1、SPLD的基本结构

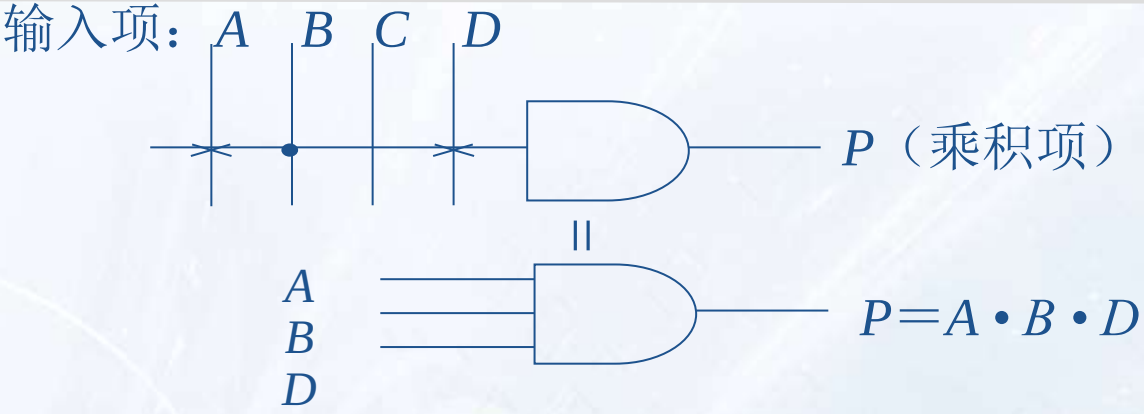
- **SPLD： PROM、PLA、PAL、GAL**
- **SPLD基本结构：与阵列、或阵列**

PLD的输入、输出缓冲器都采用互补的结构

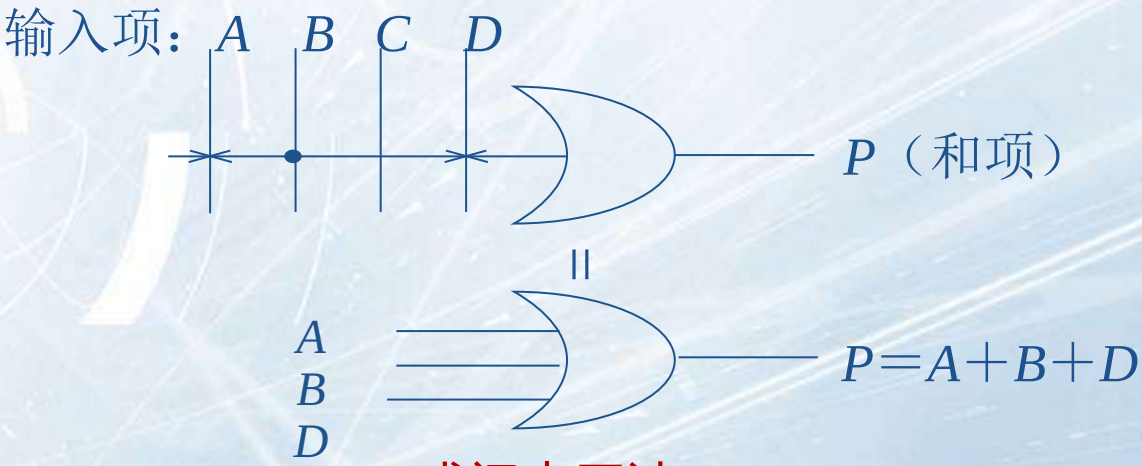


PLD的输入、输出缓冲器表示法

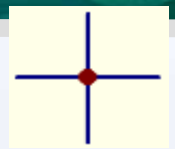
阵列型PLD的与门和或门表示



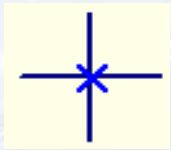
与门表示法



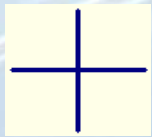
或门表示法



（a）固定连接

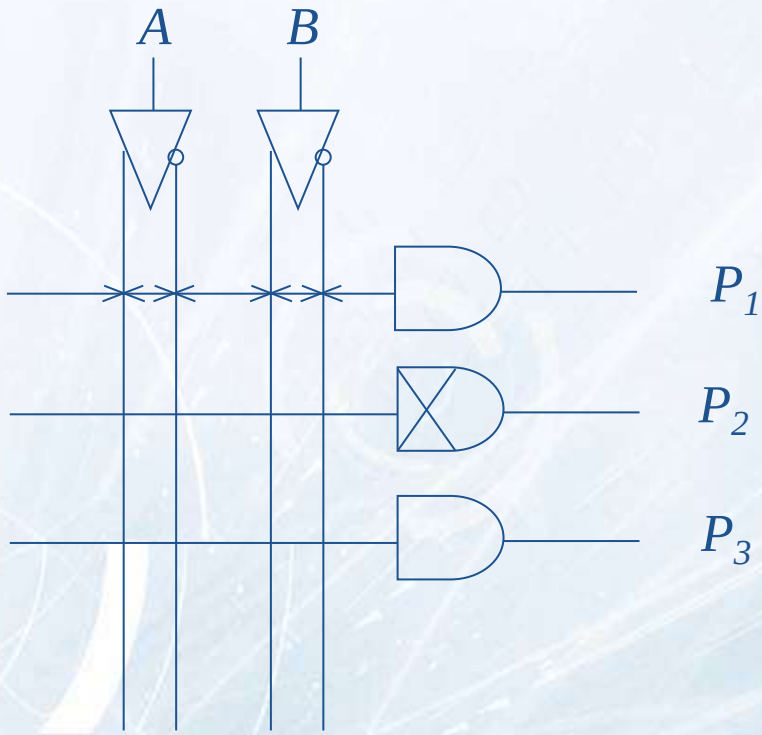


（b）可编程连接



（c）断开

PLD与门的简略表示

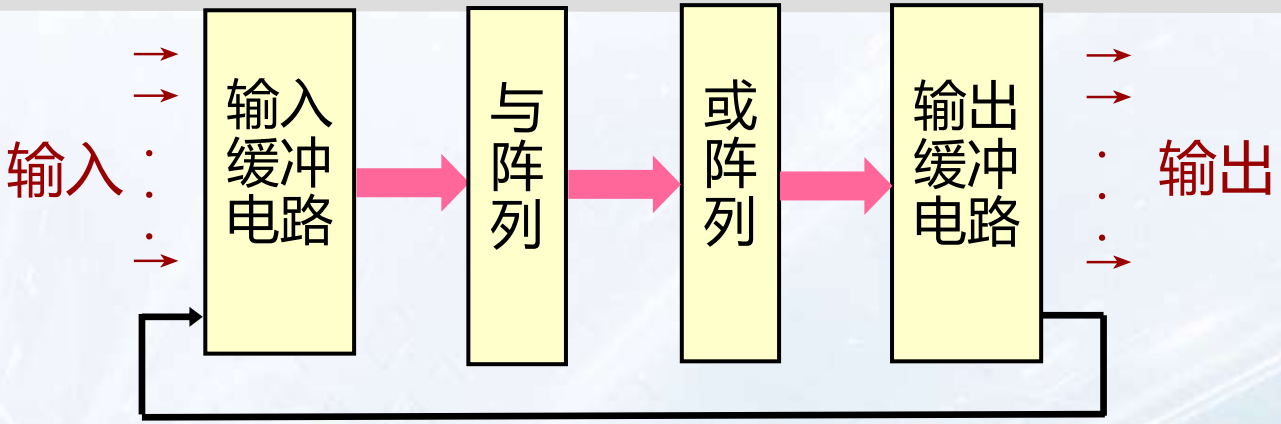


PLD与门的简略表示法

$$P_1 = A \cdot \overline{A} \cdot B \cdot \overline{B} = 0$$

- P1的全部输入项接通， P1=0，称为与门的**默认状态**，可用带“X”的与门符号表示
- P3的任何输入项都不接通，P3=1，称为**悬浮“1”状态**

简单PLD的基本结构



简单PLD的基本结构框图

- 与或阵列：PLD结构的主体，用来实现各种逻辑函数和逻辑功能。
- 输入缓冲电路：增强输入信号的驱动能力，产生输入信号的原变量和反变量；一般具有锁存器、甚至是可组态的宏单元。
- 输出缓冲电路：对将要输出的信号进行处理，既能输出纯组合逻辑信号，也能输出时序逻辑信号。一般有三态门、寄存器等单元，甚至是宏单元。

PROM、FPLA、PAL、GAL性能比较

分　类	出现时期	与阵列	或阵列	编程工艺	编程次数	输出电路
PROM	20世纪70年代初期	固定	可编程	熔丝开关	一次性	TS，OC
PLA	20世纪70年代中期	可编程	可编程	熔丝开关	一次性	TS，OC
PAL	20世纪70年代末期	可编程	固定	TTL型 CMOS型 ECL型	一次性 多次 一次性	TS，I/O，寄存器
GAL	20世纪80年代初期	可编程	固定	EEPROM	100次以上	可编程（用户定义）

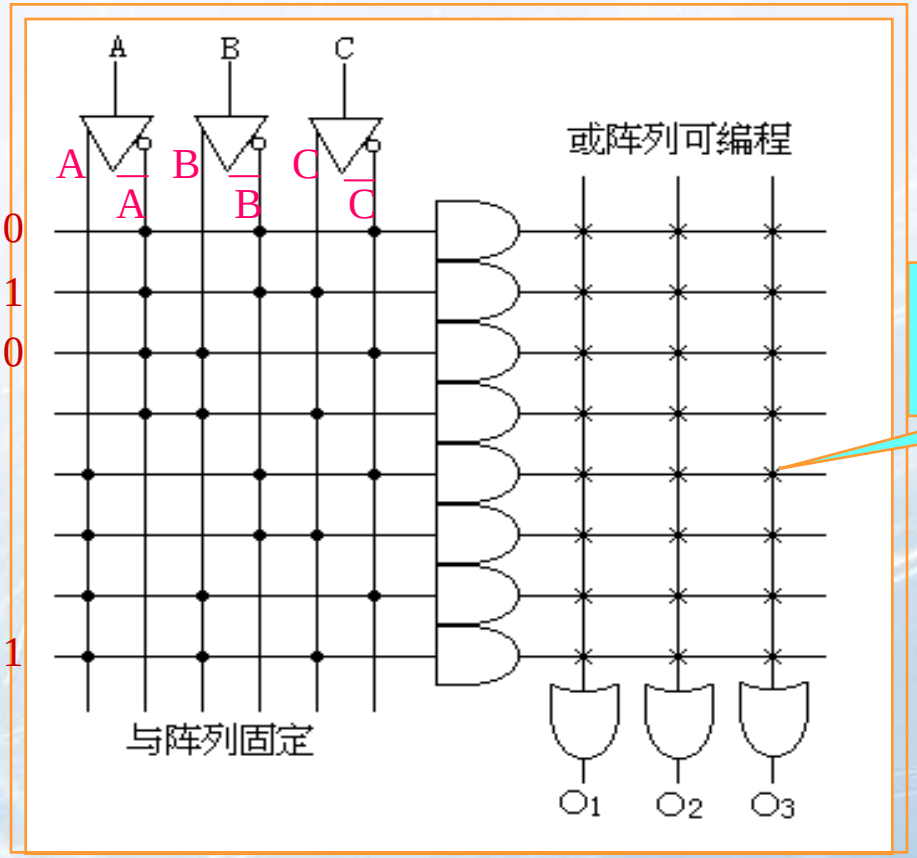
- PROM、PAL、GAL只有一种阵列可编程，称“半场可编程”逻辑器件
- PLA的与、或阵列均可编程，称“全场可编程”逻辑器件

PROM的阵列结构图

与固定、或编程：PROM

全译码

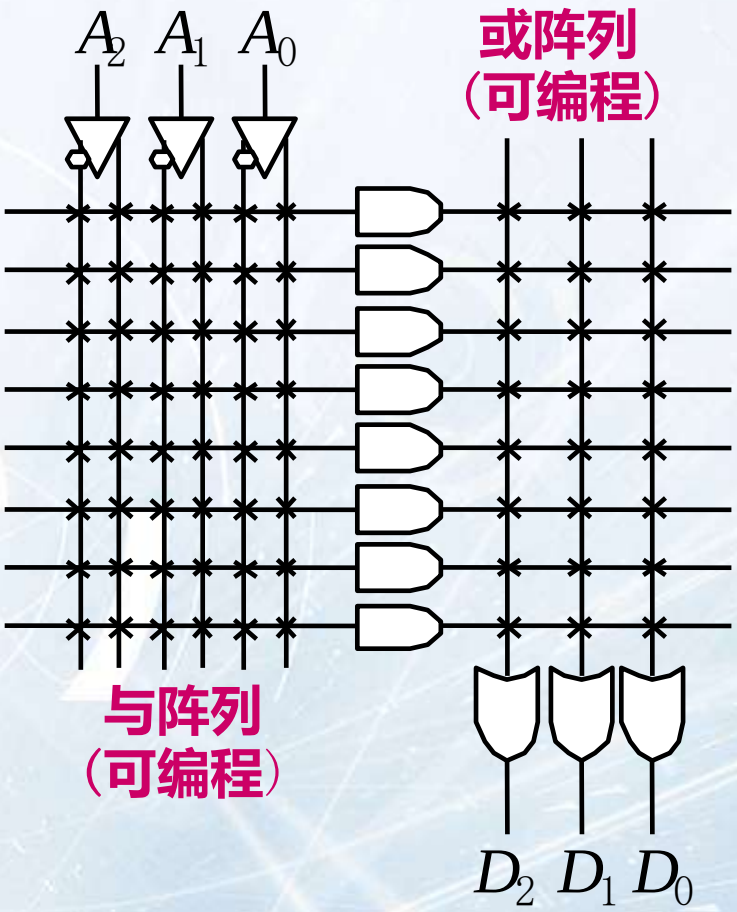
0 0 0
0 0 1
0 1 0
1 1 1



连接点编程时，需画一个叉。

PLA的阵列图

与编程、或编程： PLA（可编程逻辑阵列）

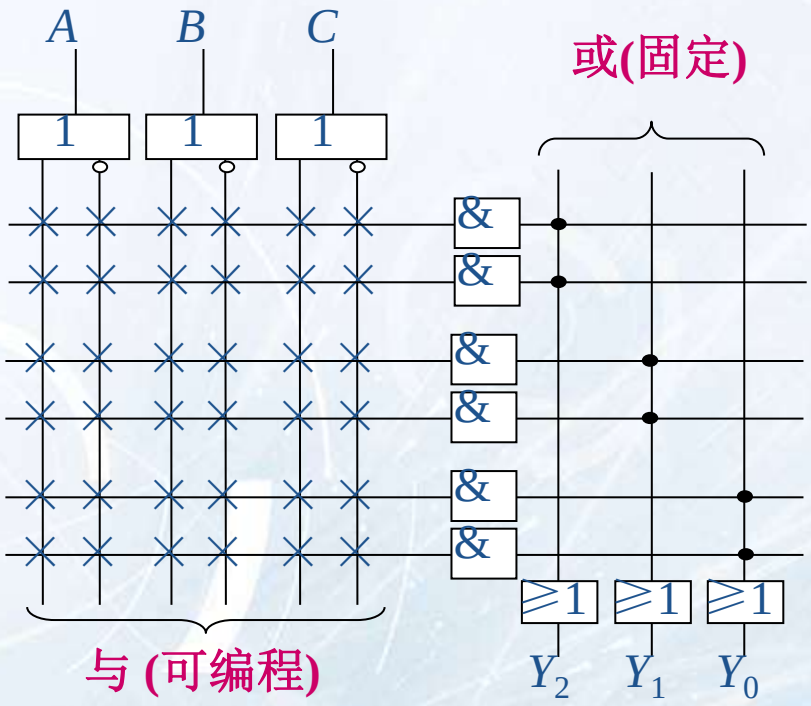


- PLA的编程工艺采用熔丝开关，为一次性编程器件
- 可以实现任意组合逻辑函数：将函数化简为最简与或式，将对应的与项或起来即可。

容量＝与门数×或门数

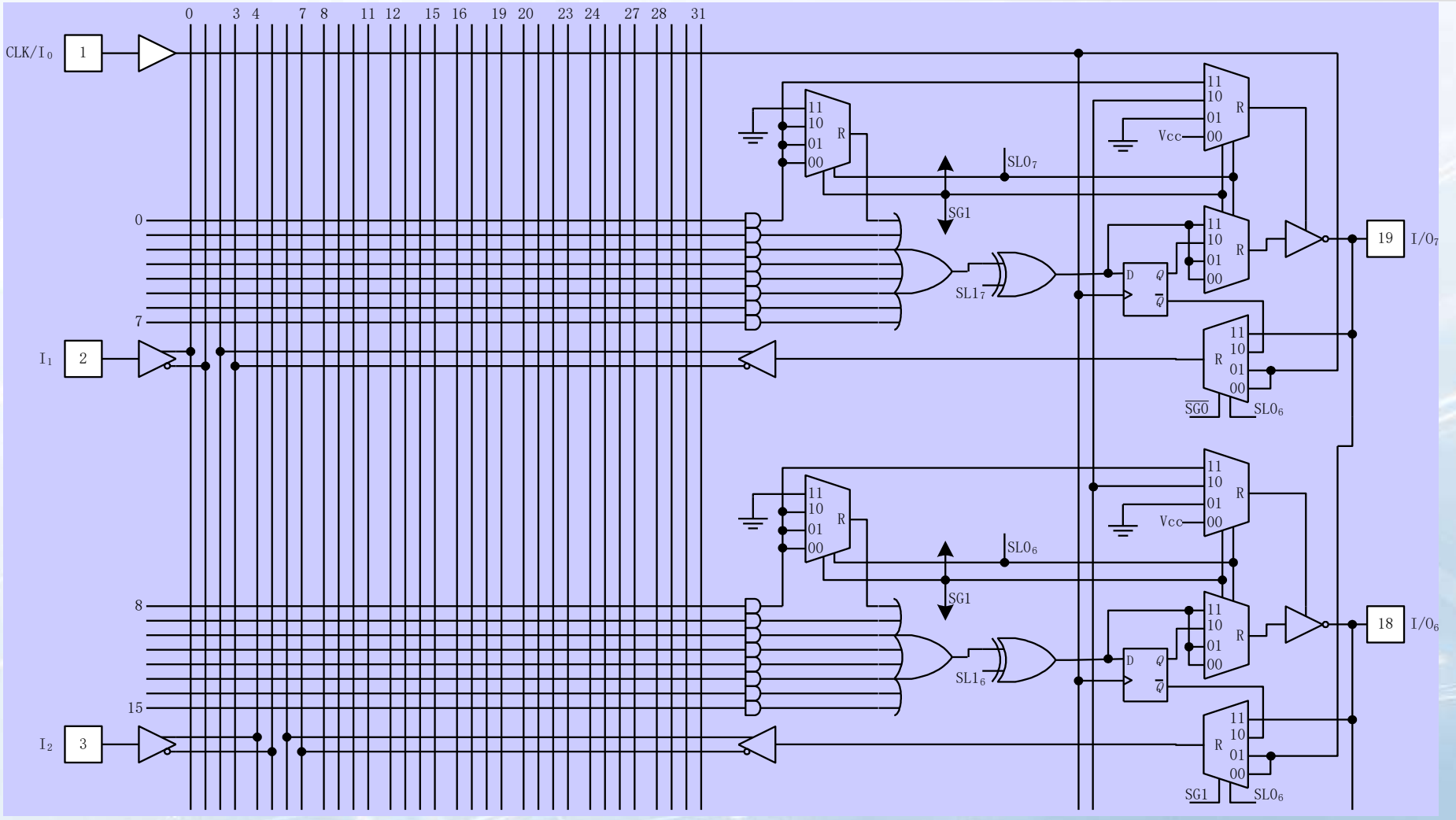
PAL、GAL的阵列结构图

与编程、或固定： PAL、GAL

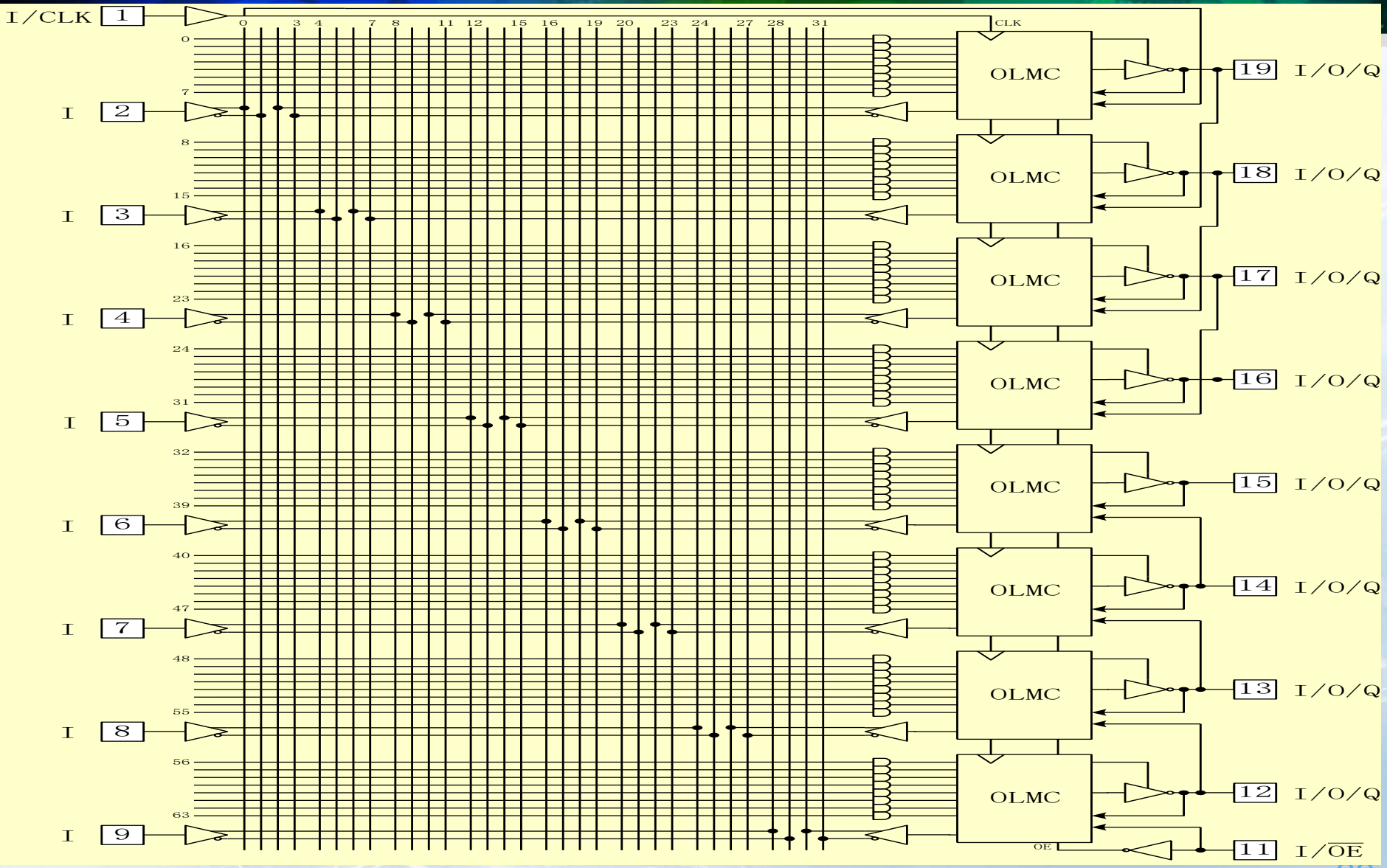


- 每个或门的输出是若干乘积项之和（最多可有8个），乘积项数目是固定的。
- GAL和PAL阵列结构相同，但输出电路不同，GAL比PAL器件功能更强，结构更灵活，可取代同型号的PAL器件。

PAL16V8的电路结构

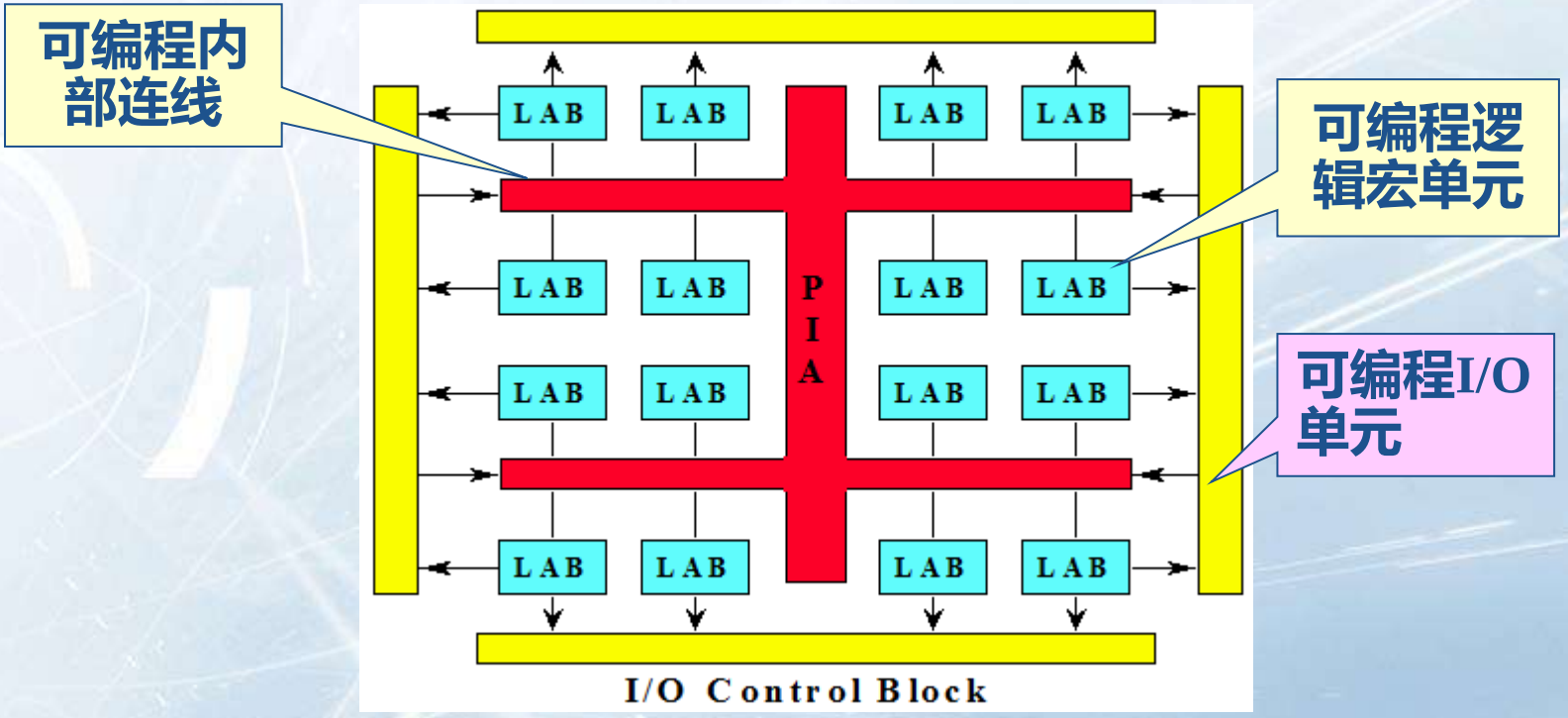


GAL16V8的电路结构



2、EPLD和CPLD的基本结构

- EPLD和CPLD都是在PAL和GAL基础上发展起来的阵列型HDPLD
- 采用CMOS EPROM、EEPROM、Flash Memory和SRAM等编程技术，构成了高密度、高速度和低功耗的PLD
- 大多由可编程逻辑宏单元、可编程I/O单元IOB和可编程内部连线PIA组成

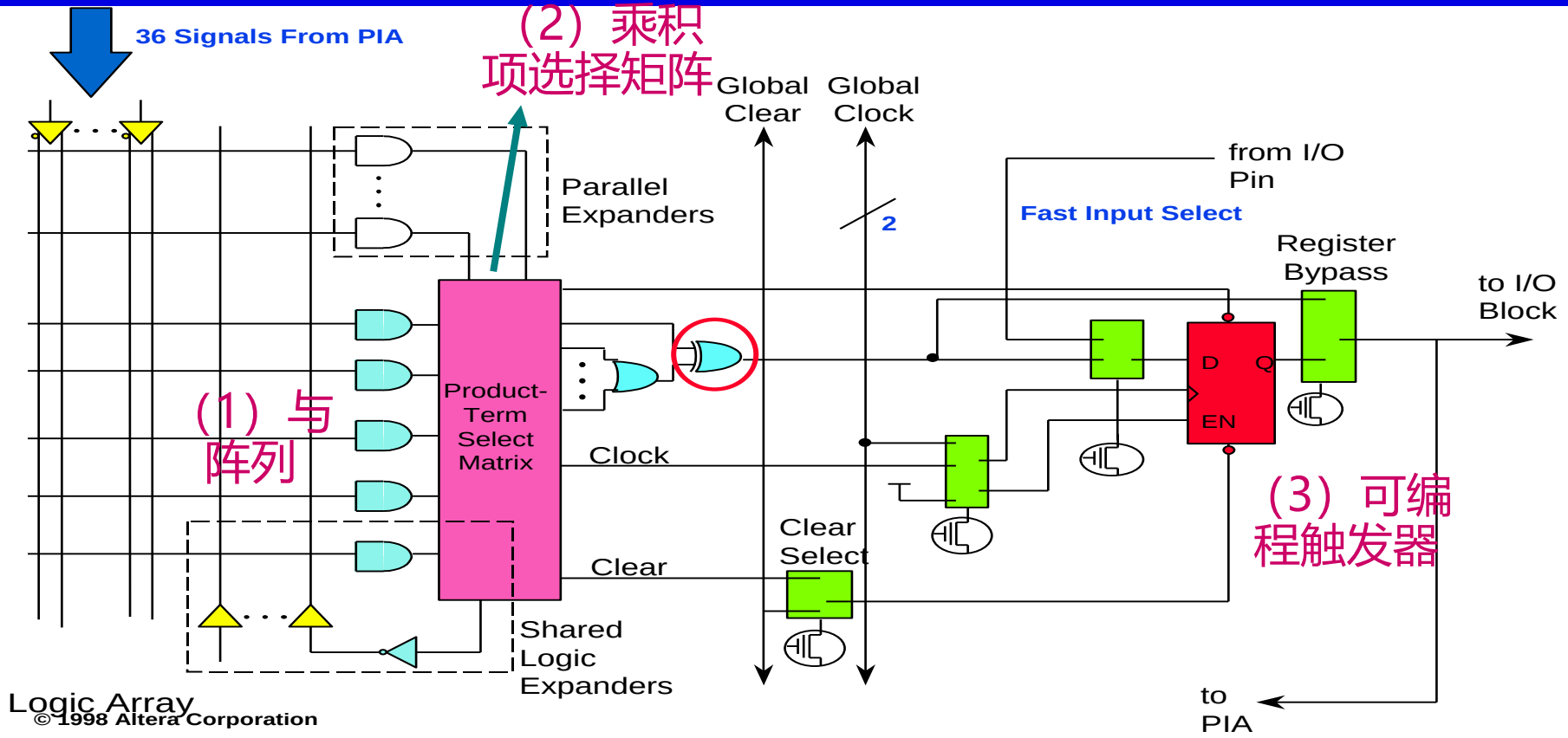


可编程逻辑宏单元

- 可编程逻辑宏单元：CPLD的基本单元，用于实现基本的逻辑功能。
- 主要包括与阵列、或阵列、可编程触发器、多路选择器等电路，能独立地配置为组合逻辑或时序逻辑工作方式。
  - 与阵列的每个交点都可编程，乘积项选择矩阵是一个或阵列，两者一起完成组合逻辑。
  - 可编程触发器的时钟和清零信号都可以编程选择，其类型可编程为D、T、J-K和R-S触发器。
- 大多数逻辑函数能够用每个宏单元中的乘积项实现。
- 但某些逻辑函数比较复杂，当输出表达式的与项较多，或门输入端不够用时，可以借助可编程开关将同一宏单元（或其他宏单元）中的其他或门联合起来使用，宏单元的输出也可以连到PIA，再作为另一个宏单元的输入，以实现复杂的逻辑。
- Altera公司的CPLD每16个宏单元组成一个LAB（Logic Array Block，逻辑阵列块）。

Altera公司MAX 7000S 逻辑宏单元结构

MAX7000S Macrocell Structure



可编程I/O单元及可编程连线阵列PIA

- I/O单元分布于器件的四周，提供器件外部引脚与内部逻辑之间的连接，负责输入/输出的电气特性控制。
- I/O单元要考虑以下要求：
  - 能够兼容TTL和CMOS多种接口电压和接口标准；
  - 可配置为输入、输出、双向I/O、集电极开路和三态门等各种组态；
  - 能提供适当的驱动电流，以直接驱动发光二极管等器件；
  - 降低功率消耗，防止过冲和减少电源噪声。
- I/O单元主要由触发器和缓冲器组成。
- 每个IOB控制一个外部引脚，可将其编程为输入、输出或双向I/O功能，或集电极开路、三态门等。

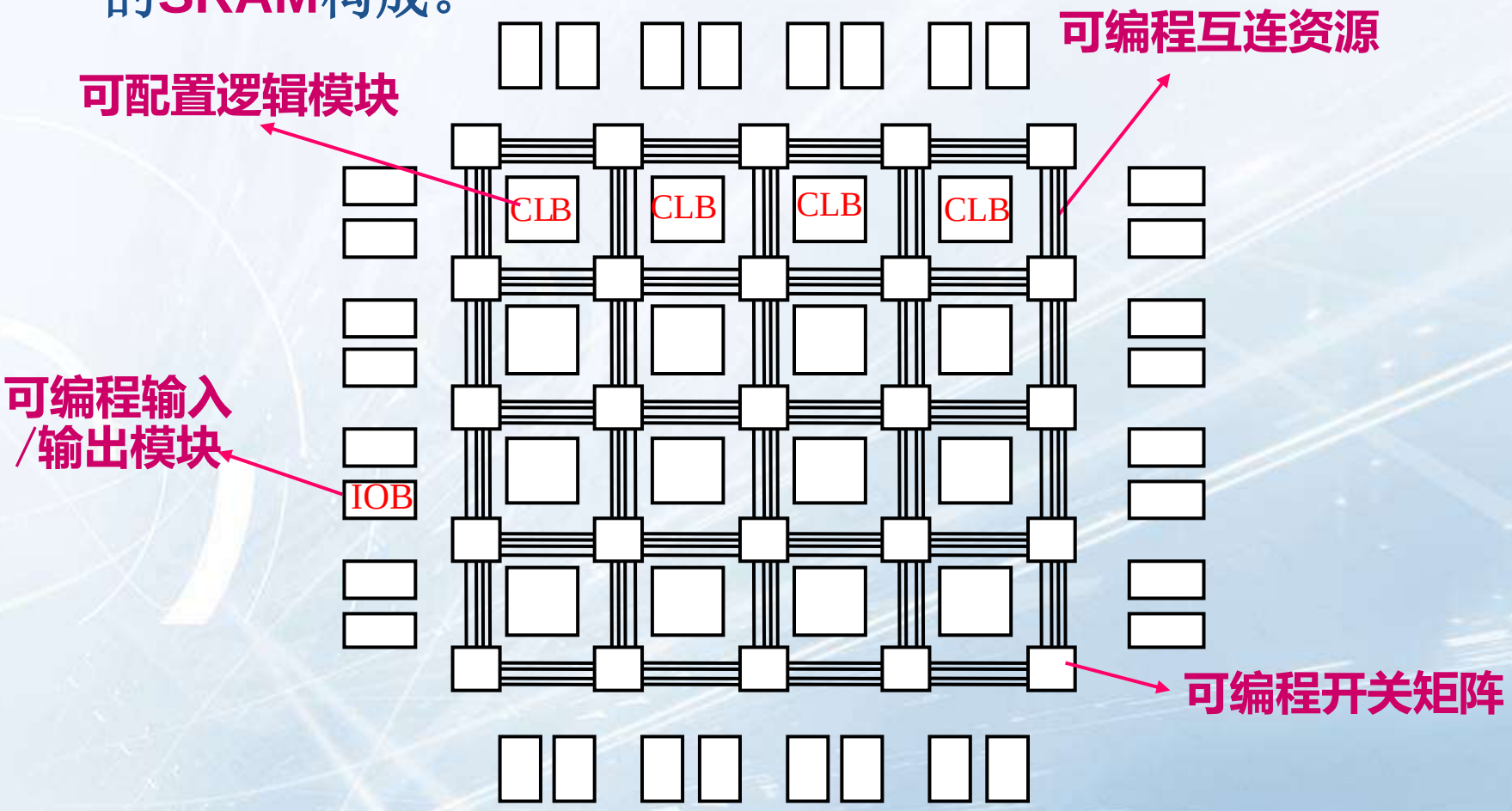
- **PIA，Programmable Interconnect Array。**
- **在各宏单元之间以及宏单元和I/O单元之间提供互连网络。**

8.5.3 现场可编程门阵列FPGA

- FPGA： Field Programmable Gates Array
  - 20世纪80年代中期出现的高密度PLD；
  - 采用类似于掩模可编程门阵列（MPGA）的通用结构，其内部由许多独立的可编程逻辑模块组成，用户可以通过编程将这些模块连接成所需要的数字系统；
  - 具有密度高、编程速度快、设计灵活和可再配置等许多优点
  - FPGA的功能由逻辑结构的配置数据决定。工作时，这些配置数据存放在片内的SRAM或熔丝图上；
  - 基于SRAM的FPGA器件，在工作前需要从芯片外部（EPROM、E^2PROM或计算机软、硬盘）加载配置数据；
  - 用户可以控制加载过程，在现场修改器件的逻辑功能，因此称为现场可编程

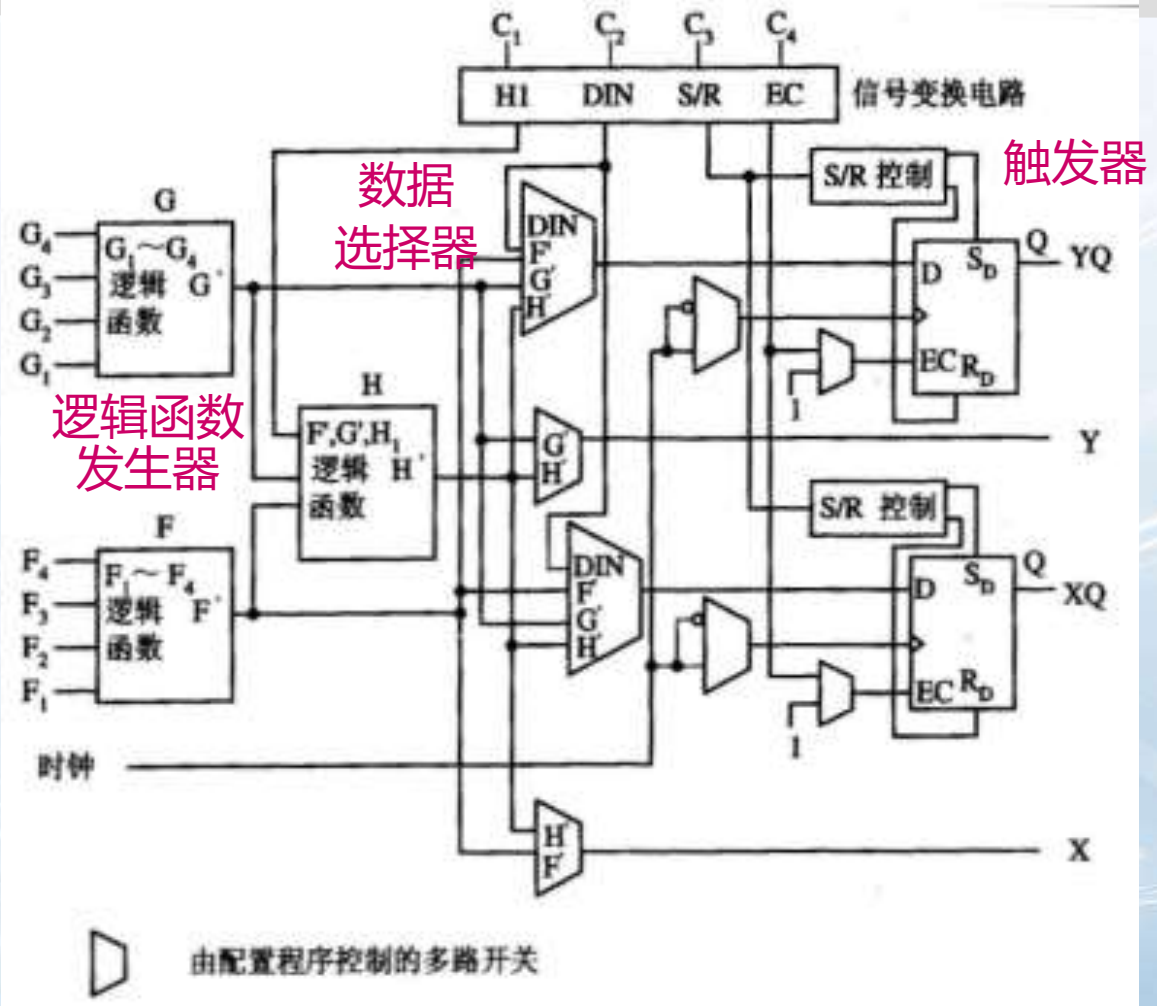
FPGA的基本结构

- 由3个可编程逻辑模块阵列和一个存储编程数据的可配置的SRAM构成。



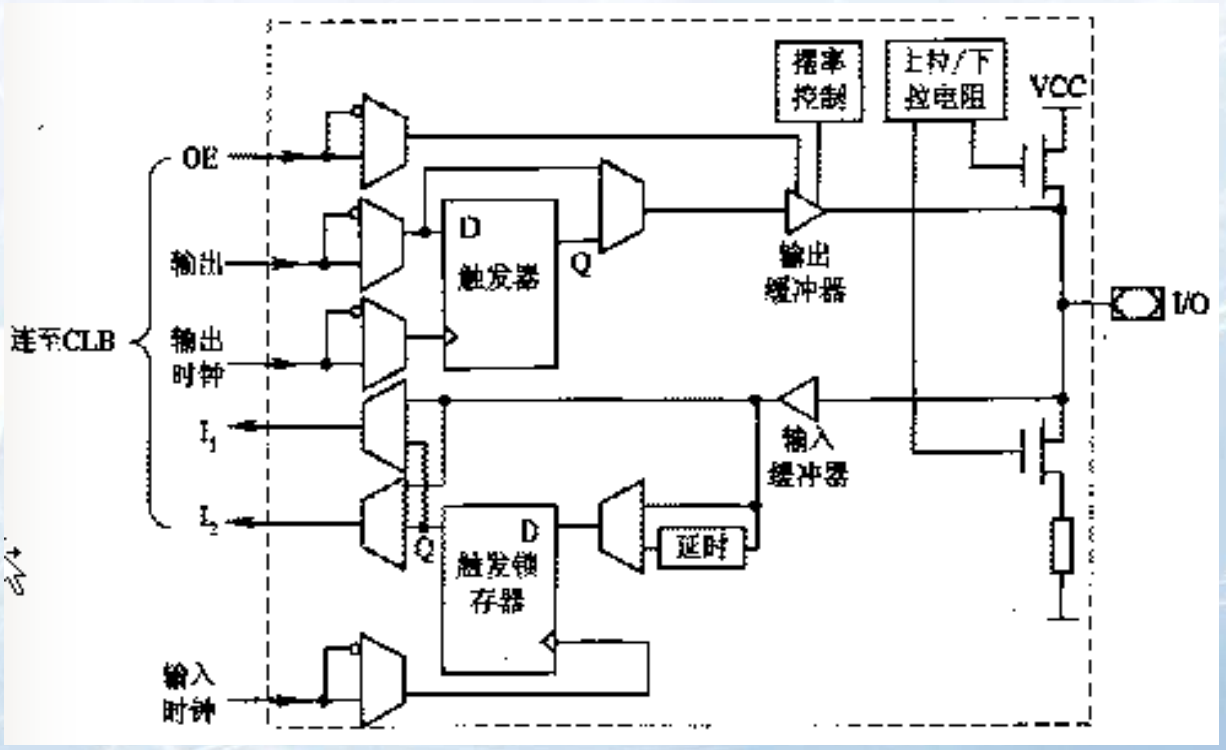
可配置逻辑模块CLB

- 可配置逻辑模块（**CLB，Configurable Logic Block**）
- **FPGA**的主要组成部分，实现逻辑功能的基本结构单元。
- 主要由逻辑函数发生器、触发器、数据选择器等电路组成。



输入/输出模块（IOB）

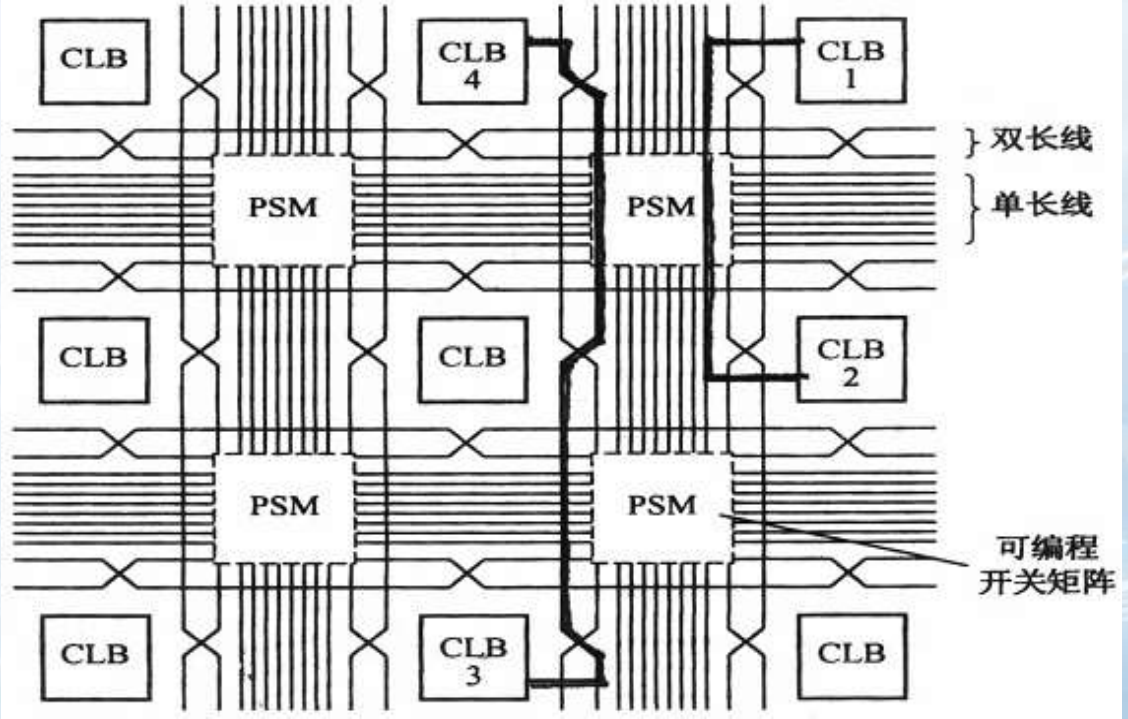
- 输入/输出模块（**IOB，Input/Output Block**）提供器件外部引脚与内部逻辑之间的连接
- 主要由触发器和（输入、输出）缓冲器组成
- 每个IOB控制一个引脚，可将其编程为输入、输出或双向I/O功能，或组合逻辑、寄存器逻辑、三态逻辑等。



可编程互连线（PI）

- 可编程互连线（**PI，Programmable Interconnect**）或互连资源（**ICR，Interconnect Capital Resource**）提供高速可靠的内部连线，将**CLB**之间、**CLB**和**IOB**之间连接起来，构成复杂的逻辑连接

- 主要由纵横分布在**CLB**阵列之间的金属线网络和位于纵横交叉点上的可编程开关矩阵（**PSM，Programmable Switch Matrix**）组成。

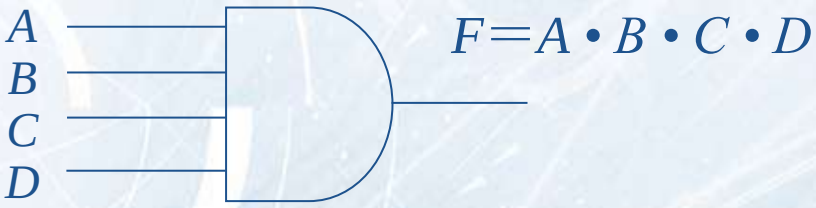


8.5.4 基于查找表的结构

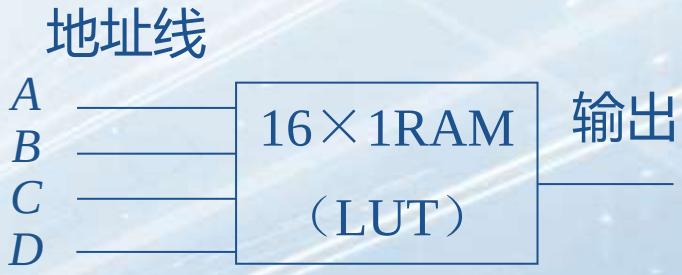
- 基于查找表（Look Up Table，LUT）结构的PLD也称之为FPGA

1、LUT原理

- 本质上就是一个RAM
- 目前FPGA中多使用4输入的LUT，相当于16×1位的RAM



4输入与门



LUT的实现方式

用LUT实现与门的实例

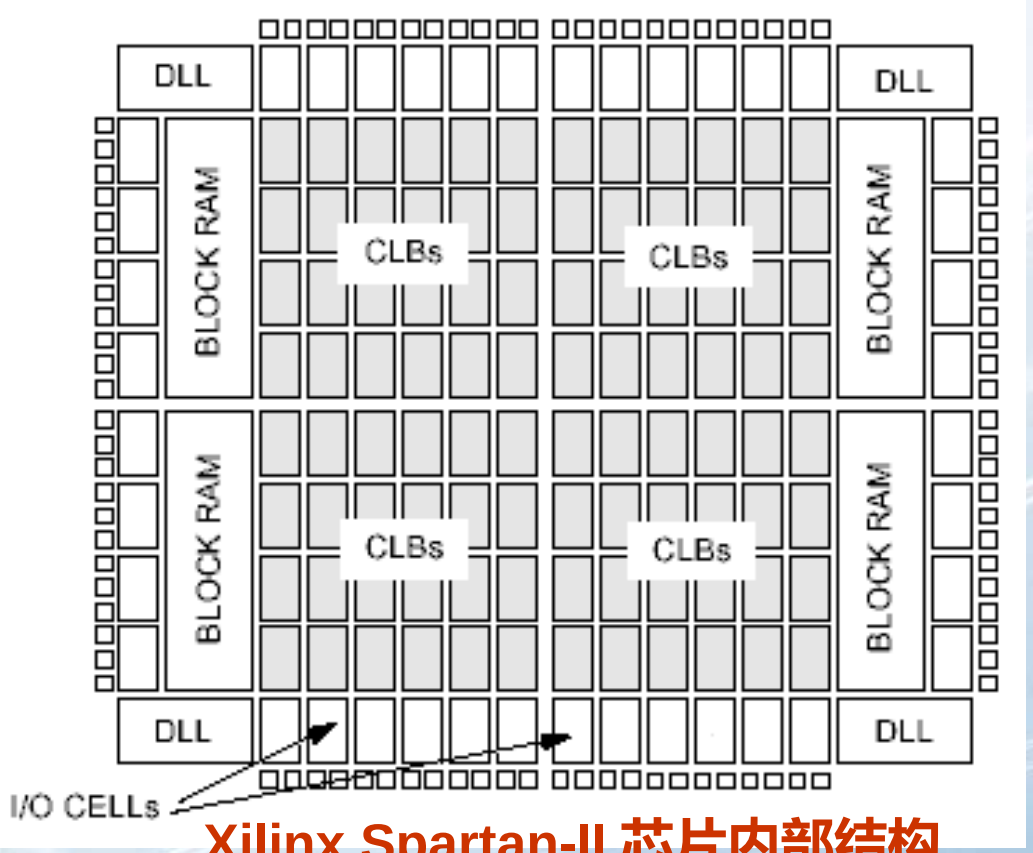
用LUT实现与门的实例

- 对于用户设计的4输入端与门，EDA软件自动计算其所有可能的结果，并把结果事先写入RAM。
- 用户每输入一组信号，就等于输入一个地址进行查表，找出地址对应的内容，然后输出，得到逻辑运算的结果。

A B C D	RAM地址（10进制）	数据
0 0 0 0	0	0
0 0 0 1	1	0
0 0 1 0	2	0
0 0 1 1	3	0
……	……	0
1 1 1 0	14	0
1 1 1 1	15	1

2、基于查找表的FPGA 的结构

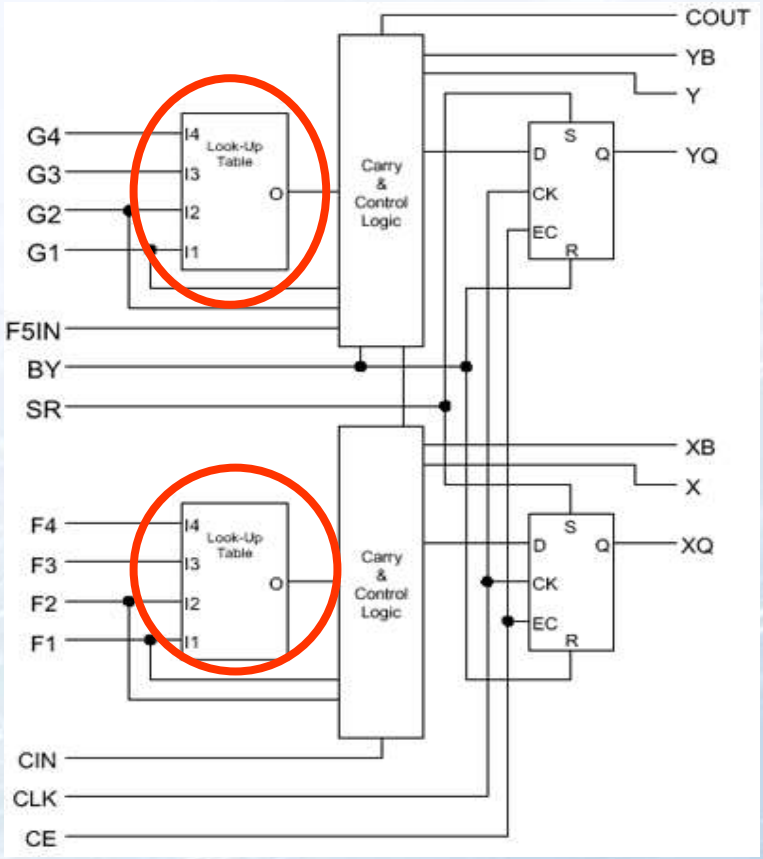
- Xilinx公司的Spartan-II即是基于LUT的FPGA
  - Spartan－II由CLBs、I/O块、RAM和可编程连线构成。一个CLB由2个Slice构成。



Xilinx Spartan-II 芯片内部结构

Spartan的Slices结构

- 每个Slice由2个LUT、2个触发器和相关逻辑构成。



Spartan的Slices结构

8.6 PLD的设计技术

内容概要

8.6.1 PLD 的设计方法

8.6.2 PLD的设计流程

8.6.1 PLD 的设计方法

❖过去的“积木”式方法

- 用于PLD没有出现之前的数字系统传统设计；
- 通过标准集成电路器件搭建成电路板来实现系统功能，即先由器件搭成电路板，再由电路板搭成系统——“自底向上”（Bottom-Up）的设计；
- 数字系统的“积木块”就是具有固定功能的标准集成电路器件。

❖特点

- 设计中，设计者没有灵活性可言；
- 搭成的系统需要的芯片种类多且数目大；
- 用户只能根据需要选择合适的集成电路器件，并按照此种器件推荐的电路搭成系统，如TTL的74/54系列、CMOS的4000/4500系列芯片和一些固定功能的大规模集成电路等。

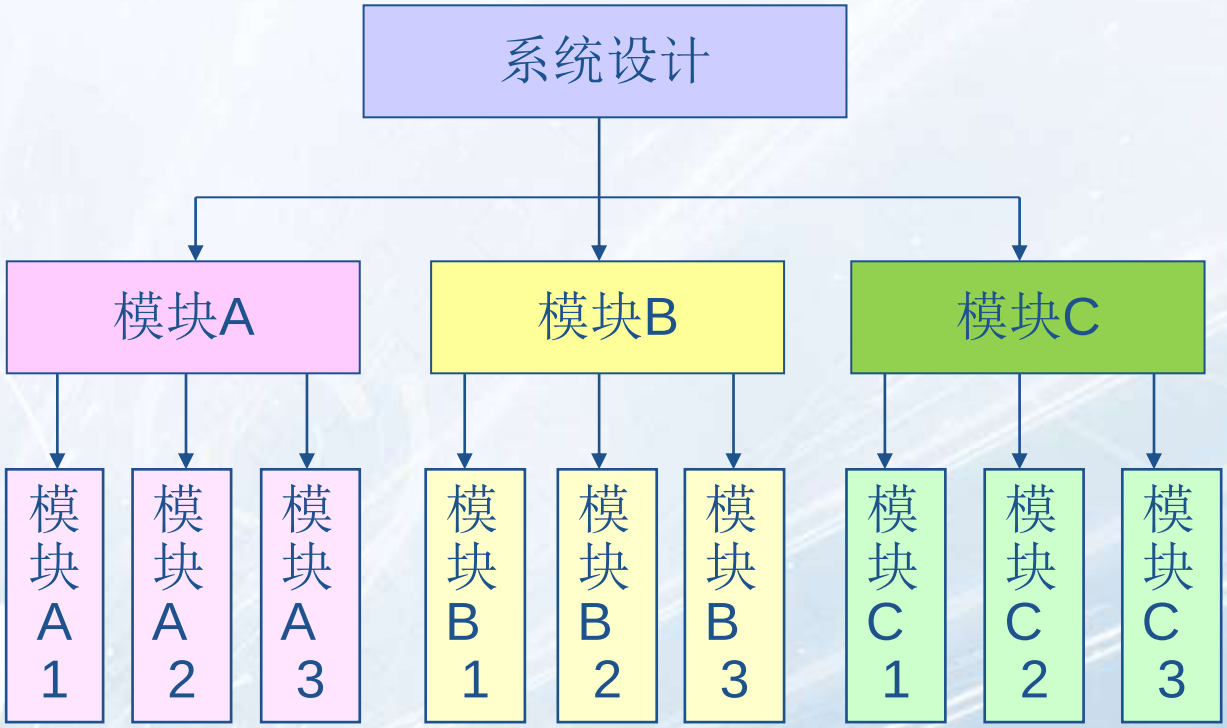
PLD 的设计方法

- 采用PLD进行的数字系统设计
  - 基于芯片的设计——“自顶向下”（Top-Down）设计法
  - 跟传统的积木式设计有本质的不同
- 设计方法特点
  - 设计者可以根据实际情况和要求定义器件的内部逻辑关系和引脚
  - 可直接通过设计PLD芯片来实现数字系统功能，将原来由电路板设计完成的大部分工作放在PLD芯片的设计中进行。
  - 引脚定义的灵活性，大大减轻了系统设计的工作量和难度，提高了工作效率，减少芯片数量，缩小系统体积，降低功耗，提高系统稳定性和可靠性。

“自顶向下”的数字系统设计

- “自顶向下”（Top-Down）设计法
  - 目前最为常用；
  - 采用功能分割的方法从顶向下逐次将设计内容进行分块和细化；
  - 在设计过程中采用层次化和模块化将使系统设计变得简洁和方便。
- 层次化设计
  - 分层次、分模块地进行设计描述；
  - 描述器件总功能的模块放在最上层，称为顶层设计；
  - 描述器件某一部分功能的模块放在下层，称为底层设计；
  - 底层模块还可以再向下分层，直至最后完成硬件电子系统电路的整体设计。

自顶向下设计法的示意图



“自顶向下”设计法的示意图

8.6.2 PLD的设计流程

- 设计流程包括四个步骤和三个设计验证过程
- 四个步骤
  - 设计准备
  - 设计输入
  - 设计处理
  - 器件编程
- 三个设计验证过程
  - 功能仿真
  - 时序仿真
  - 器件测试

PLD 的设计流程


设计准备
设计输入
电路图
程序
波形图
元件库
功能仿真
设计过程
设计处理
编译优化、综合
适配、分割
布局、布线
时序仿真
器件编程（下载）
器件测试
设计完成

设计准备与设计输入

❖ 设计准备

- 分析设计要求，预估电路形式与规模，选择合适的PLD。一般所设计电路需用的I/O端数量和GLB数量不要超过所选芯片所能提供数量的 80% 。
- 根据选定的PLD确定应采用何种设计开发工具。

❖设计输入

- 设计输入在软件开发工具上进行。
- 对于低密度 PLD，可采用象 ABEL这样的简单开发软件，可采用逻辑方程输入方式。
- 对于高密度 PLD，可采用图形（逻辑电路图）方式、文本（HDL语言）方式和波形图等输入方式。
- 设计输入时，应尽量调用设计软件中所提供的元件。

本章小结

1、程序逻辑电路的结构及特点

- 结构
  - 控制器: 系统的控制指挥中心
  - 运算器: 完成算数逻辑计算
  - 存储器: 存放程序和数据的器件
  - 输入电路: 完成外部信息和指令、程序的输入
  - 输出电路: 完成处理结果信息和数据的输出
- 特点
  - 软硬结合，用一块相同的硬件电路，通过改变存储器中的程序或数据，完成多种功能的操作

2、半导体存储器的结构

- **存储矩阵**：存放数据的主体
  - **字**：若干个存储单元构成的存储组，有共同的地址，共同用来代表某种信息，并共同写入存储器或从存储器中读出。
  - **字长**：构成存储器字中的二进制位数。字长有1、4、8、16、32位等，一般把8位字长称为1字节（Byte）。
  - 存储器的**容量**：字数M×字长N（位bit）(1B=8b)

    1024=1K；1024K=1M；1024M=1G；1024G=1T
- **地址译码器**：产生到存储器“字”的地址
  - 译码方式（若地址线有i条）
    - 线译码：i线-2^i线译码器，译码线数＝ 2^i
    - 矩阵译码：若行地址线和列地址线各i/2条，则译码线数＝$2 \times 2^{i/2}$
- **输入/输出控制电路**：控制数据的流向和片选使能

3、半导体存储器的分类

- 随机存取存储器——RAM（Random Access Memory）
  - 静态RAM（SRAM）——使用中可读可写，不需要刷新。
    - 优点：存取速度比DRAM快；
    - 缺点：集成度不如DRAM高
  - 动态RAM（DRAM）——使用中可读可写，但需要刷新。
    - 优点：结构非常简单，集成度远高于SRAM
    - 缺点：存取速度不如SRAM快
- 只读存储器——ROM（Read Only Memory）
  - 固定ROM（掩膜MROM）
  - 可编程ROM（PROM）
  - 光可擦可编程ROM（EPROM）
  - 电可擦可编程ROM（EEPROM）
  - Flash Memory

4、随机存储器RAM

- 静态随机存储器SRAM
  - 工作原理
  - 6管NMOS静态存储单元的结构
  - SRAM读写周期时序图
- 动态随机存储器DRAM
  - 工作原理
  - 动态存储单元结构
    - 四管MOS动态存储单元
    - 单管MOS动态存储单元
  - DRAM的状态
    - 写入，读出，保持，刷新
  - DRAM读写周期时序图
- RAM典型芯片
  - Intel 2114 （1K×4位）SRAM
  - HM6116 （2K×8位） SRAM
  - Intel 2164（ 64K×1位）DRAM

RAM芯片的扩展方法

- 位扩展（扩展字长）
  - 参与扩展的全部芯片的地址线、片选控制线/CS和写控制线/WE并接
  - 把低位芯片的数据线作为低位数据，高位芯片的数据线作为高位数据，数据位同时有效，实现I/O数据位数的增加
- 字扩展（扩展字数）
  - 参与扩展的全部芯片的地址线、写控制线/WE和数据线并接;
  - 把增加的高位地址线通过译码器产生译码信号来控制各芯片的片选控制/CS。
- 字位扩展（扩展容量）
  - 计算扩展需要的芯片数
  - 计算扩展需要增加的地址数
  - 确定几片芯片为一组，进行位扩展
  - 将译码器的输出信号分别与这几组芯片的片选端相连，进行字扩展

5、只读存储器ROM

- 4种ROM结构
  - 固定ROM
  - 可编程ROM（PROM）
  - 光可擦可编程ROM（EPROM）
  - 电可擦可编程ROM（EEPROM, EAROM）
- ROM的扩展（与RAM扩展相同）
  - 位扩展
  - 字扩展
  - 字位扩展
- ROM的应用
  - 实现任意组合逻辑函数
  - 掌握ROM实现任意组合逻辑函数的设计方法

Go

6、基于Verilog HDL的存储器设计

- RAM的HDL设计
  - 利用memory型变量进行设计
  - RAM的仿真波形编辑一定按照RAM的读写时序
- ROM的HDL设计
  - 容量不大的ROM可以用case语句实现
  - 也可用memory型变量实现ROM
- 利用Quartus II的LPM宏单元库实现大容量ROM/RAM
  - 模块元件例化的端口对应有两种方式
    - 信号名对应的方式，位置对应方式
  - 创建存储器初始化文件的方法