

存储器2

程晨闻

东南大学电气工程学院



➤ 存储器的基本概念、种类和组成

- 存储体
- 控制电路
- 地址译码电路
- 数据缓冲

➤ SRAM的工作原理

- 结构
- 特点

➤ DRAM的工作原理

- 结构
- 特点



- 紫外线擦除可编程ROM (EPROM)
- 电可擦可编程只读存储器 (EEPROM)
- TM4C控制器中的存储器
- 字节顺序

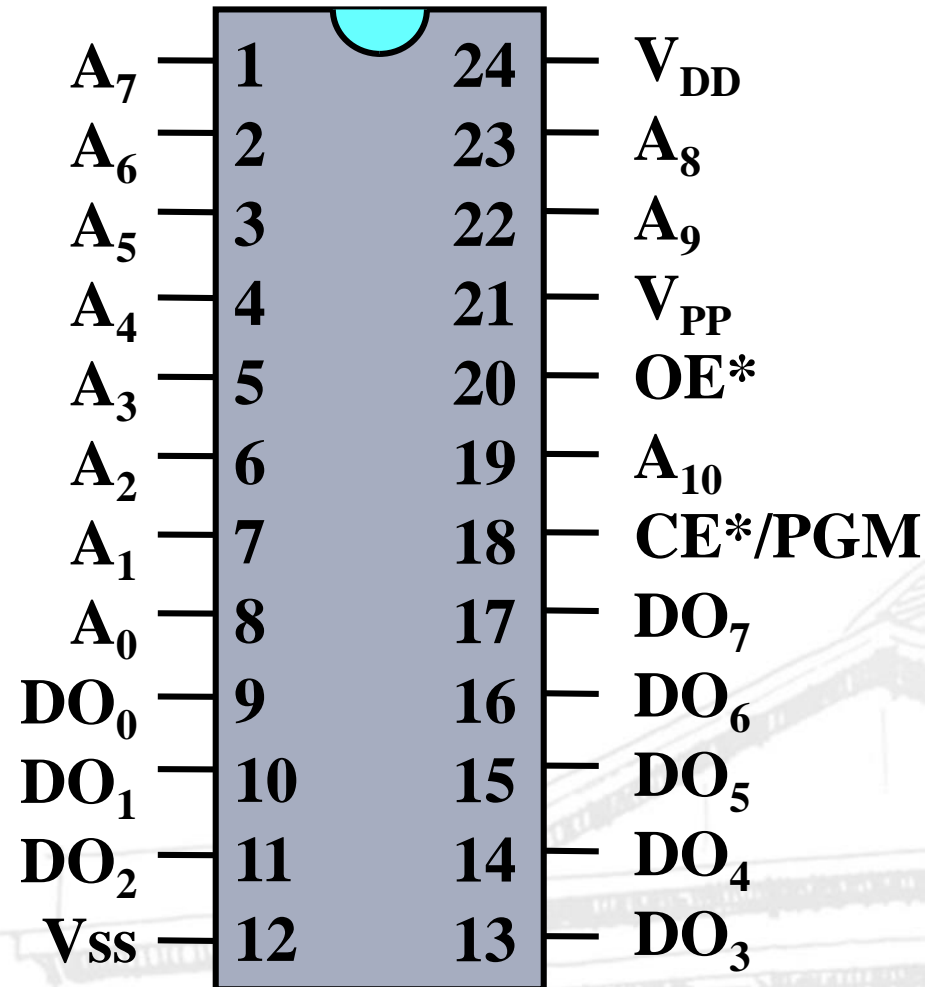


➤ EPROM

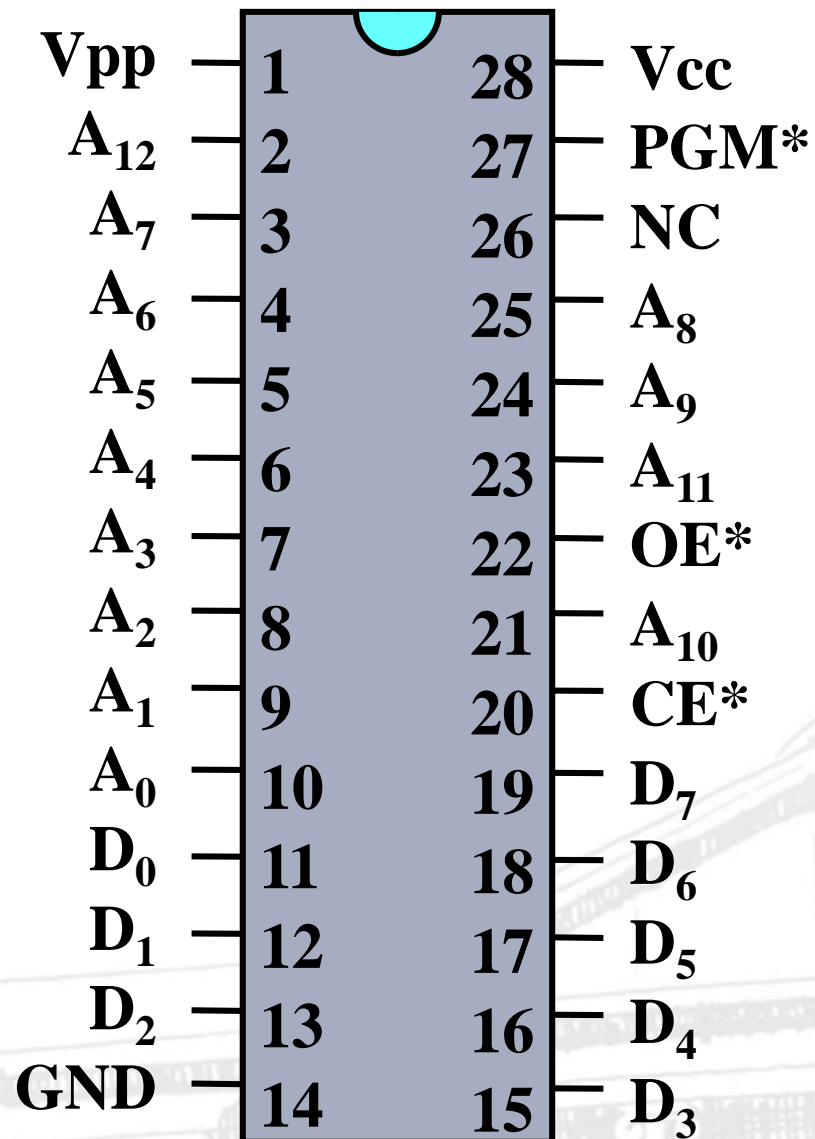
- 顶部开有一个圆形的**石英窗口**，用于紫外线透过擦除原有信息
- 一般使用**专门**的编程器（烧写器）进行编程
- 编程后，应该贴上不透光**封条**
- 出厂未编程前，每个基本存储单元都是信息**1**
- 编程就是将某些单元写入信息**0**



- 存储容量为**2K×8**
- 24个引脚：
 - 11根地址线 $A_{10} \sim A_0$
 - 8根数据线 $DO_7 \sim DO_0$
 - 片选/编程 CE^*/PGM
 - 读写 OE^*
 - 编程电压 V_{PP}



- 存储容量为 **8K×8**
- 28个引脚：
 - 13根地址线 $A_{12} \sim A_0$
 - 8根数据线 $D_7 \sim D_0$
 - 片选 CE^*
 - 编程 PGM^*
 - 读写 OE^*
 - 编程电压 V_{PP}



➤ EEPROM (Electrically Erasable Programmable read only memory)

- 即电可擦可编程只读存储器，是一种掉电后数据不丢失（不挥发）存储芯片

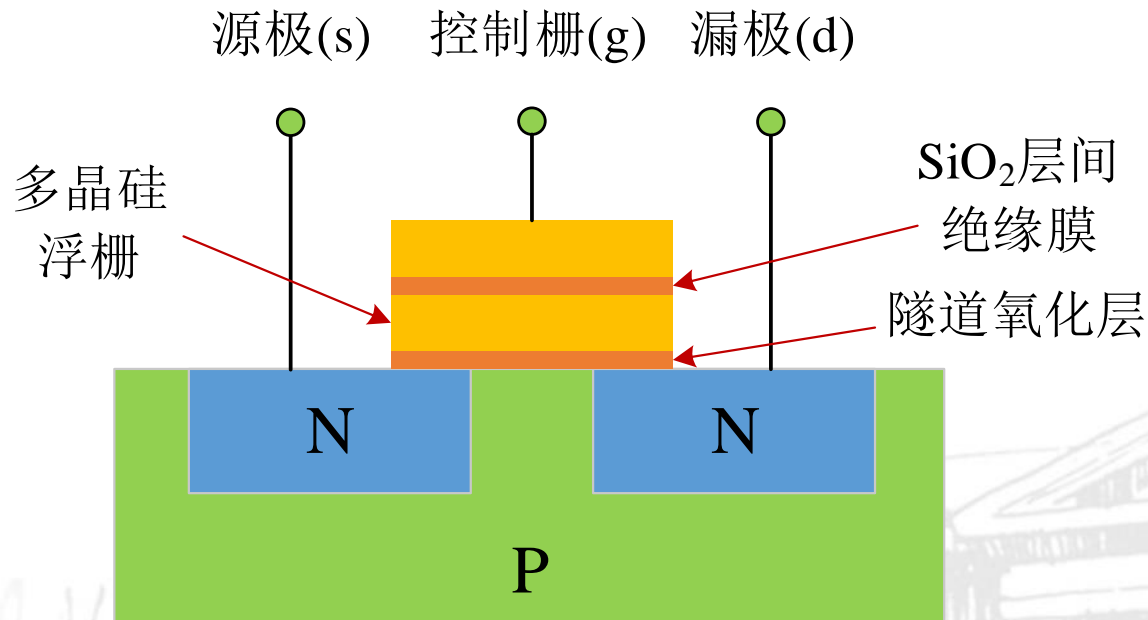
➤ 快闪存储器 (Flash Memory)

- 全名叫Flash EEPROM Memory，是一种电子式可清除程序化只读存储器的形式，允许在操作中被多次擦或写的存储器
- Flash又分为NAND flash和NOR flash二种



➤ EEPROM使用浮栅场效应管(Floating Gate FET)作为基本存储单元来存储数据

- 传统MOS管栅极下插入一层**多晶硅浮栅**
- 浮栅周围氧化层和绝缘层与电极**隔离**
- 氧化物电阻高，势垒大，浮栅中**电子泄露速度慢**

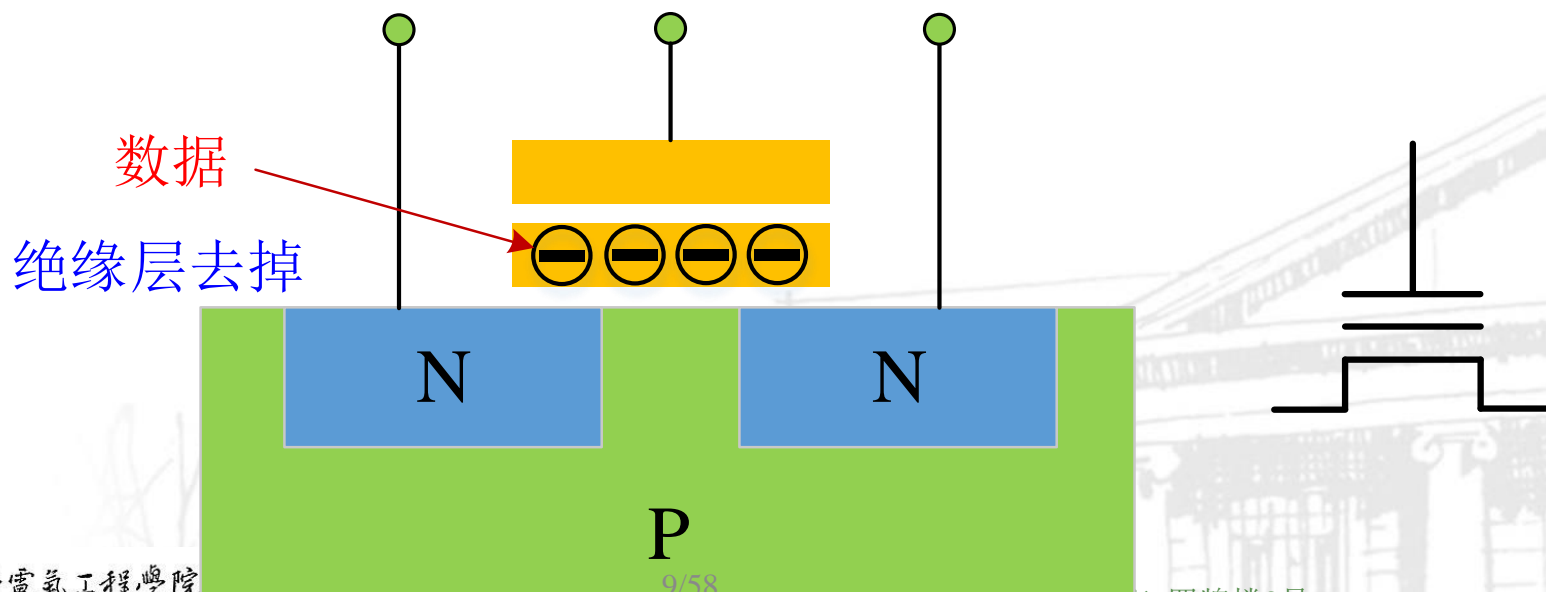


FLOTOX (Floating Gate Tunneling Oxide)结构

➤ 浮栅场效应管简化示意图和符号

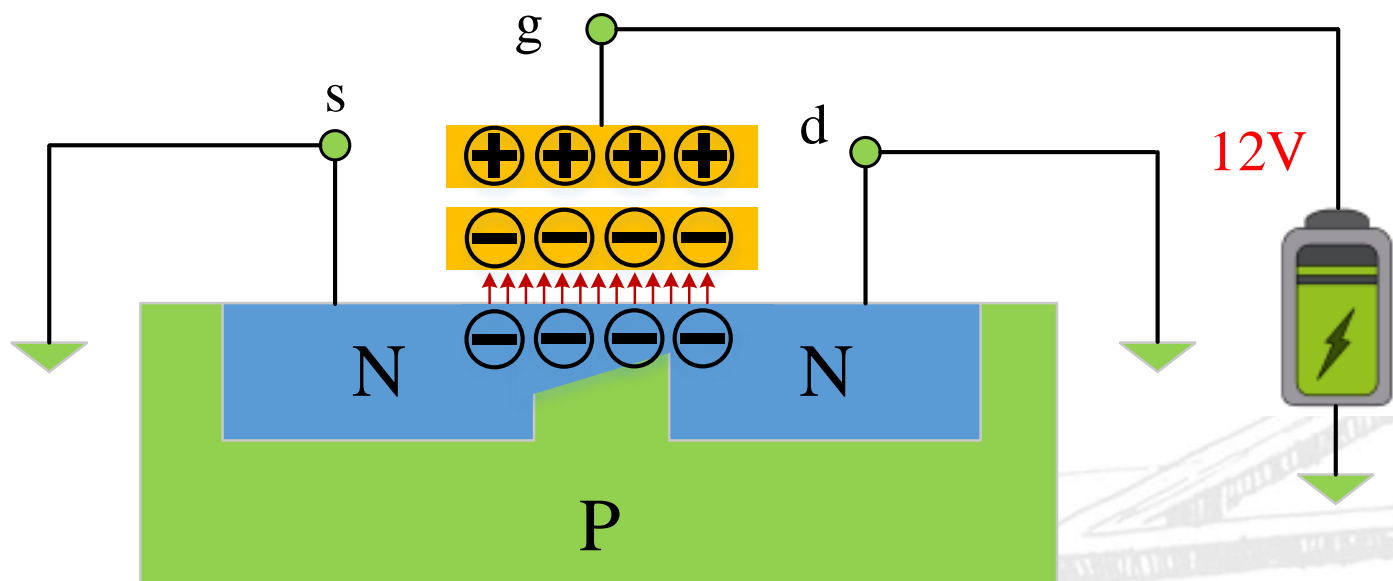
- 浮栅中的**电子**即为需要存储的数据
- **F-N隧道效应** (Fowler-Nordheim tunneling)
- 浮栅延长区的下方有个**薄氧区小窗口**，在外加**强电场**的作用下漏极与浮栅之间可以进行**双向电子流动**，从而达到对存储单元的“**擦除**”与“**写入**”操作

源极(s) 控制栅(g) 漏极(d)



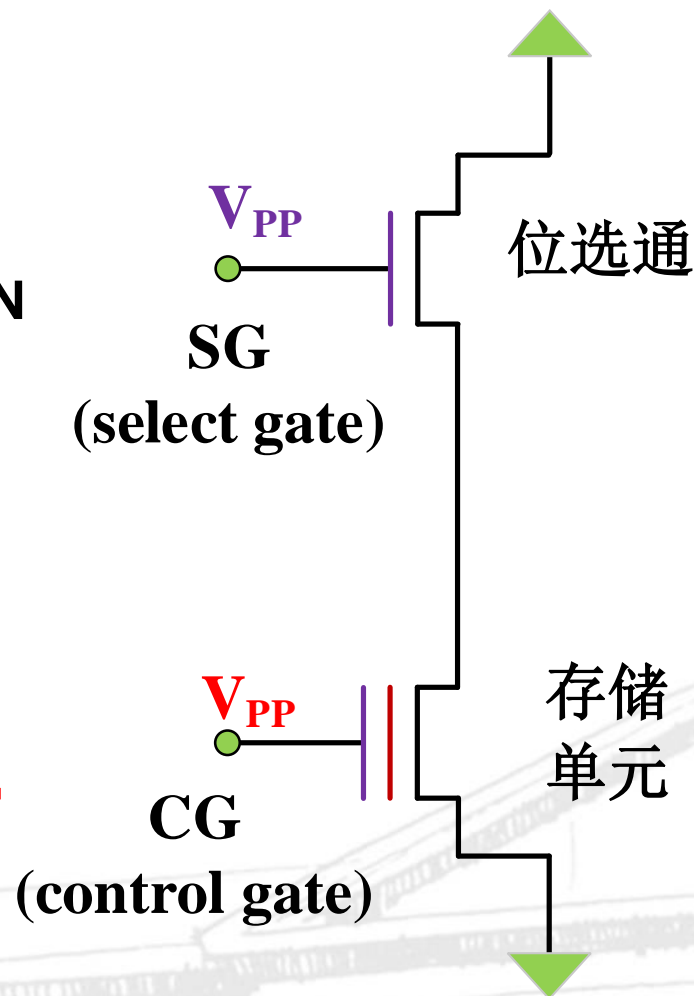
➤ 写入

- 控制栅极与漏极在强电场的作用下，衬底中的**电子**获得足够能量后，**穿过氧化层的禁带到达浮栅**
- 对存储单元进行“写入”操作，就是将**电子注入到浮栅**中的过程，即写入‘1’的过程，代表逻辑“0”



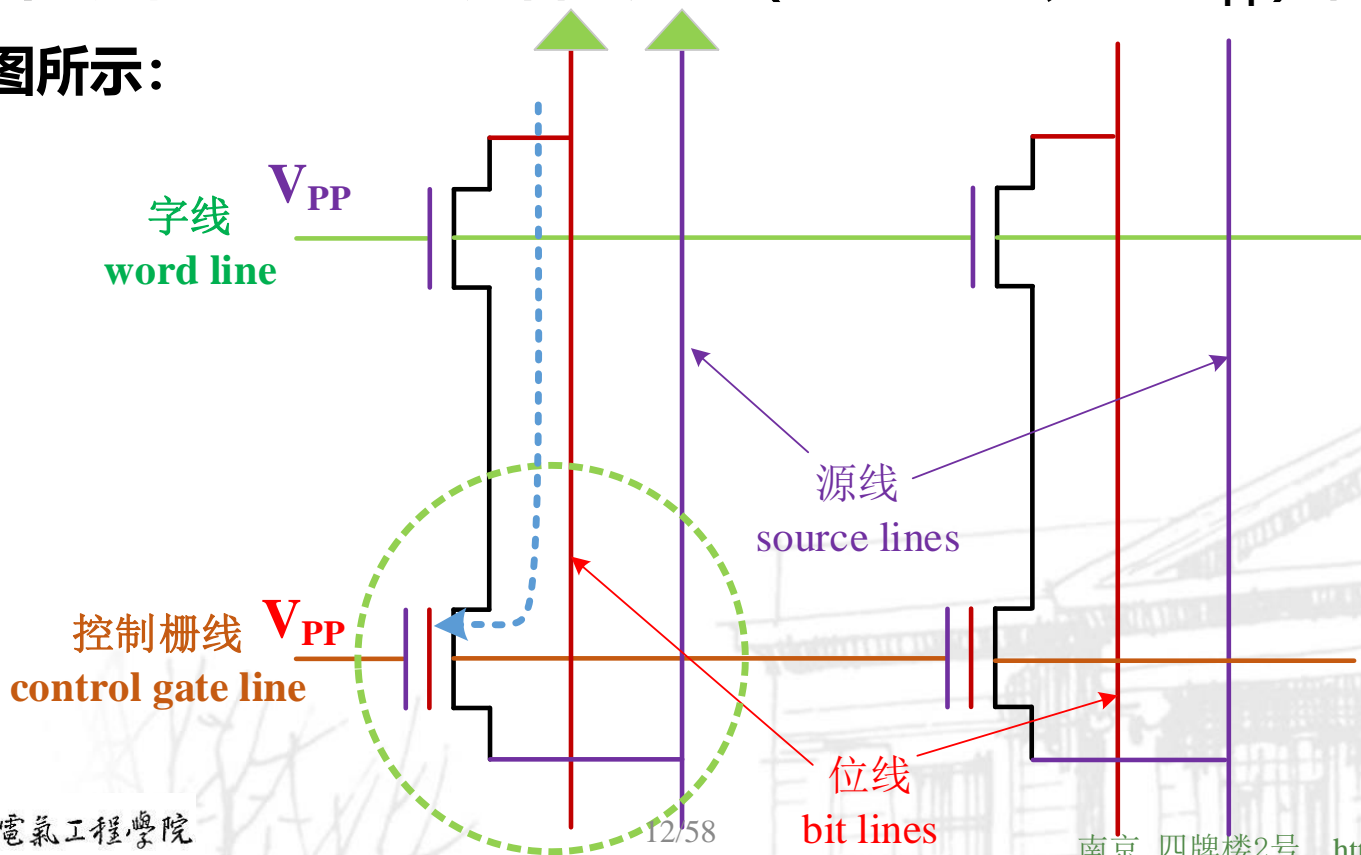
➤ 写入

- 为防止存储单元“写入”（或“擦除”）对其它单元产生影响，每个FLOTOX管均与一个**选通管**配对（以N管为例，P管是类似）
- FLOTOX管是**存储电子**的单元，而选通管用来选择相应存储单元的**控制位**
- 这种结构导致单位存储**面积比较大**，因此，EEPROM存储芯片的**容量通常都不会很大**



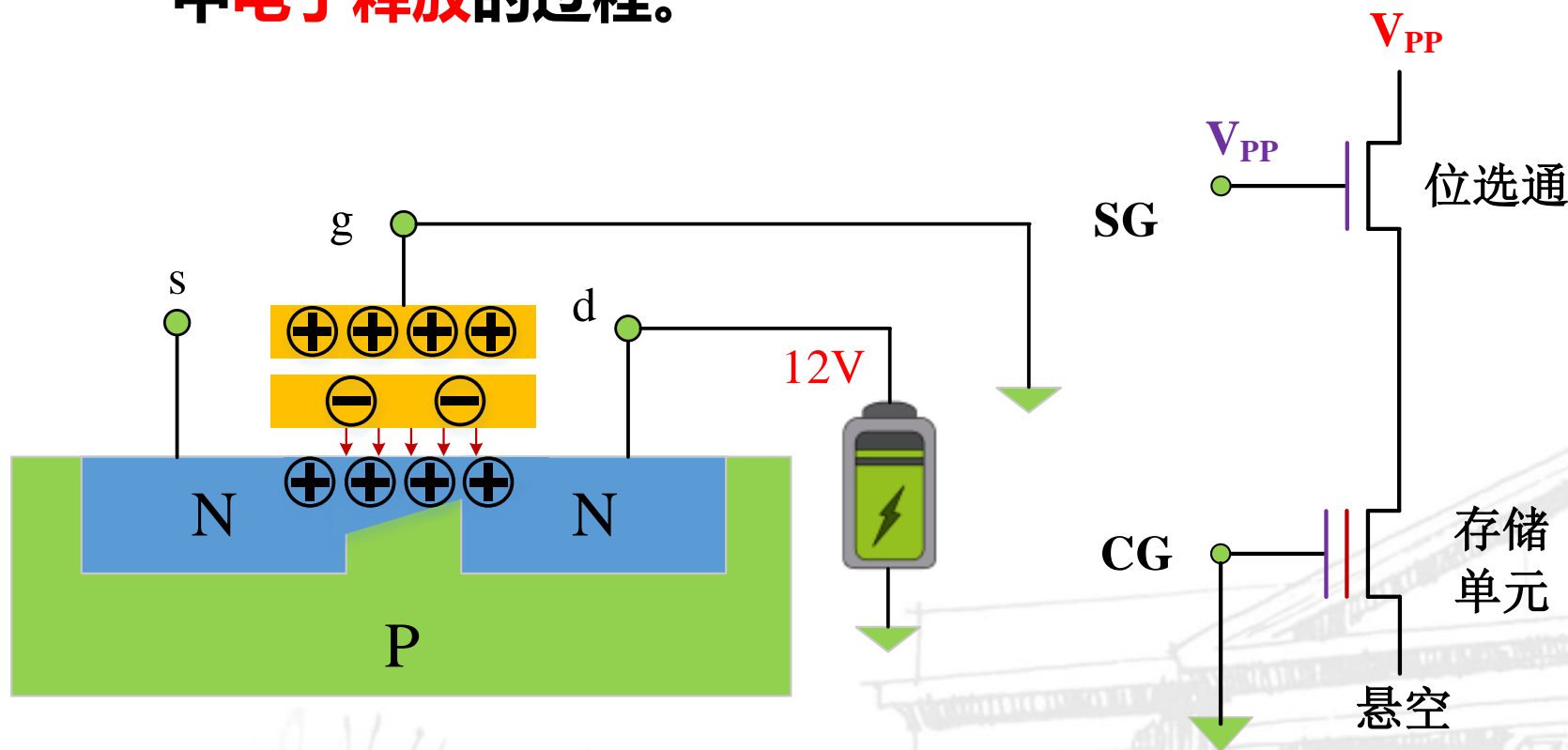
➤ 写入

- 源线 (Sources Lines, SL) 与位线 (Bit Lines, BL) 均为低电平
- 控制栅线 (CL) 为高电平 (不小于12V)
- 当对应存储单元的选通管打开时 (Word Line, $WL=V_{PP}$) , 如下图所示:



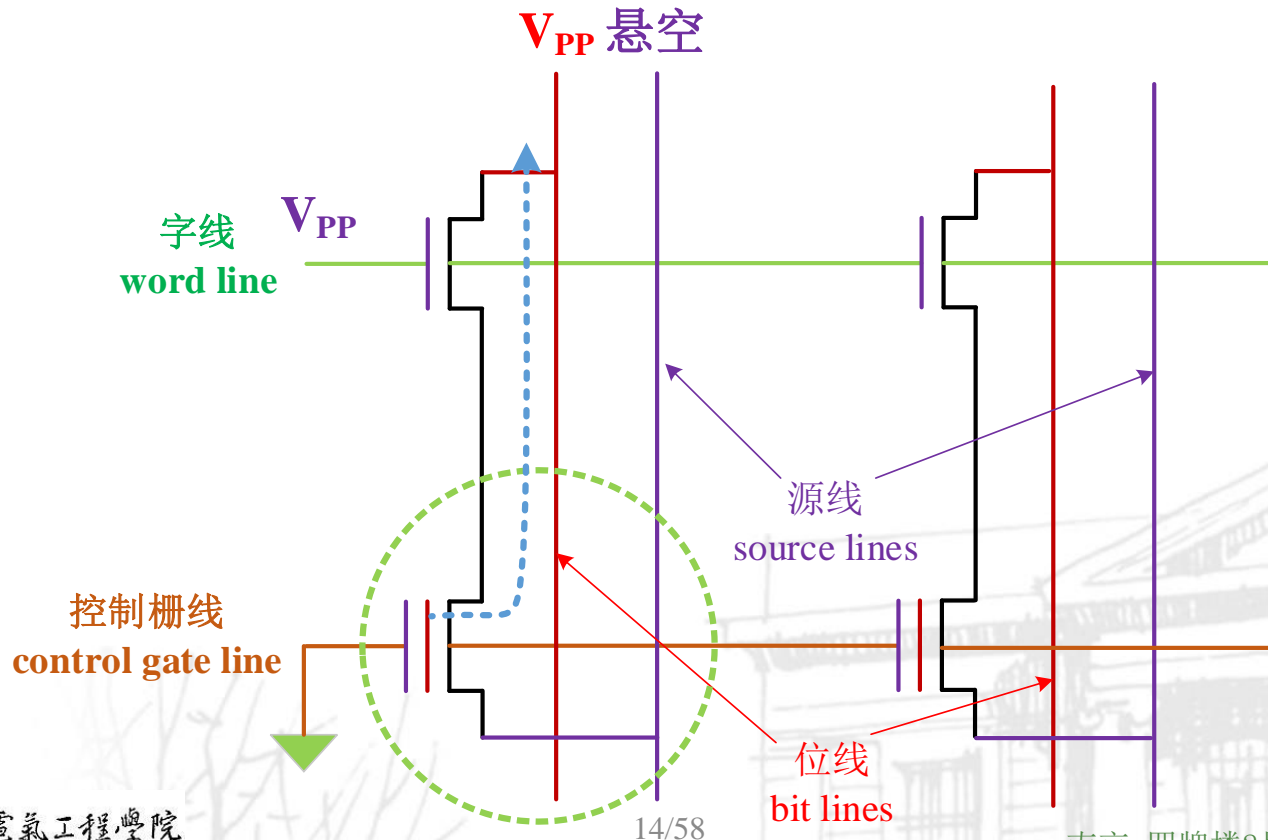
➤ 擦除

- 对EEPROM存储单元进行“擦除”操作，就是将浮栅中**电子释放**的过程。



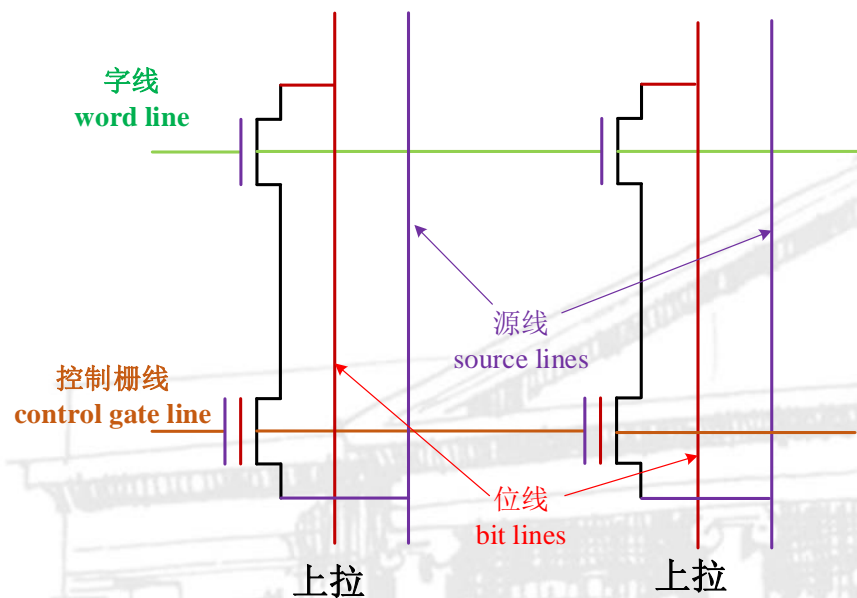
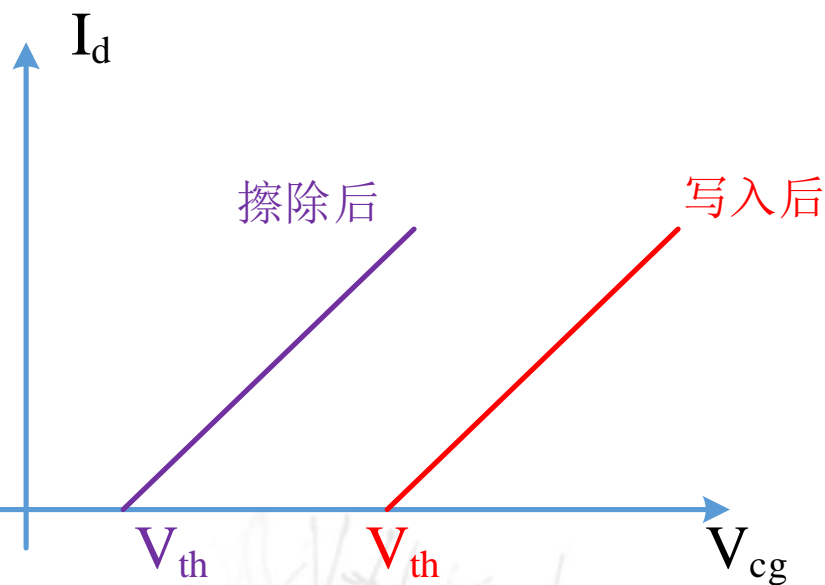
➤ 擦除

- 位线为高电平（不小于12V）；
- 源线悬空且比Program Gate Line为低电平；
- 当对应存储单元的选通管打开时（ $SG=V_{PP}$ ），如下图所示：

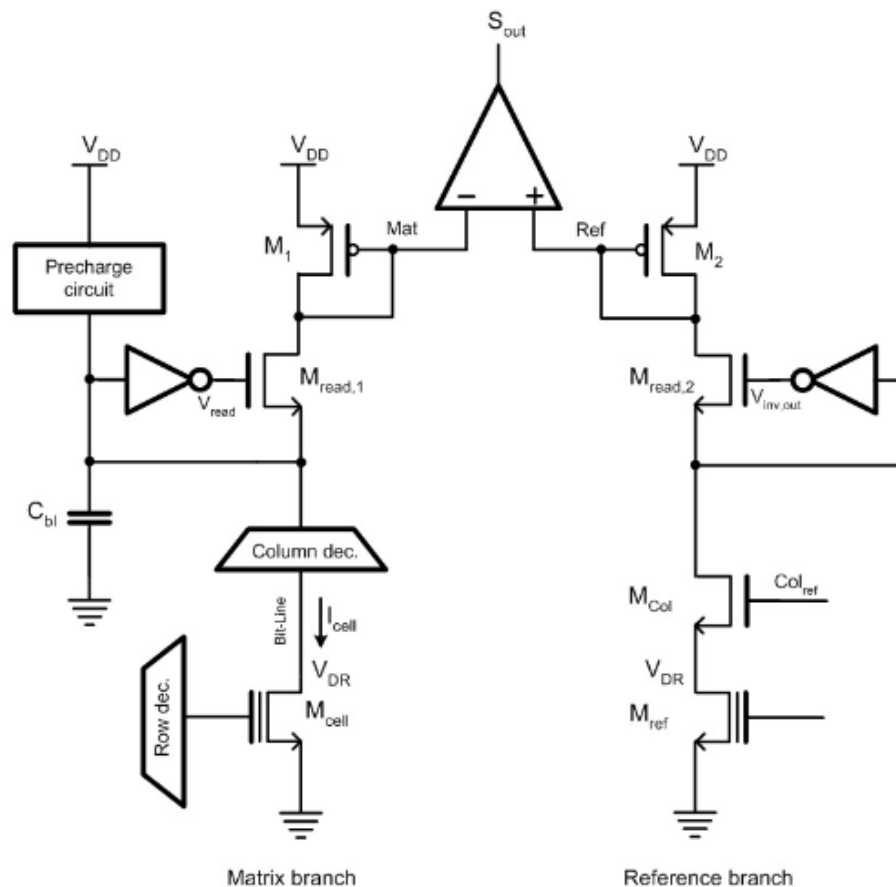


➤ 数据读取

- 浮栅注入（充电状态，写入）或释放电子（放电状态，擦除），存储单元**阈值电压**会发生改变；
- 读取时**位线被上拉**，给控制栅极加一个**中间电压**，如果浮动栅极有电荷，**DS关断**，位线读出**1**；否则位线读出**0**。



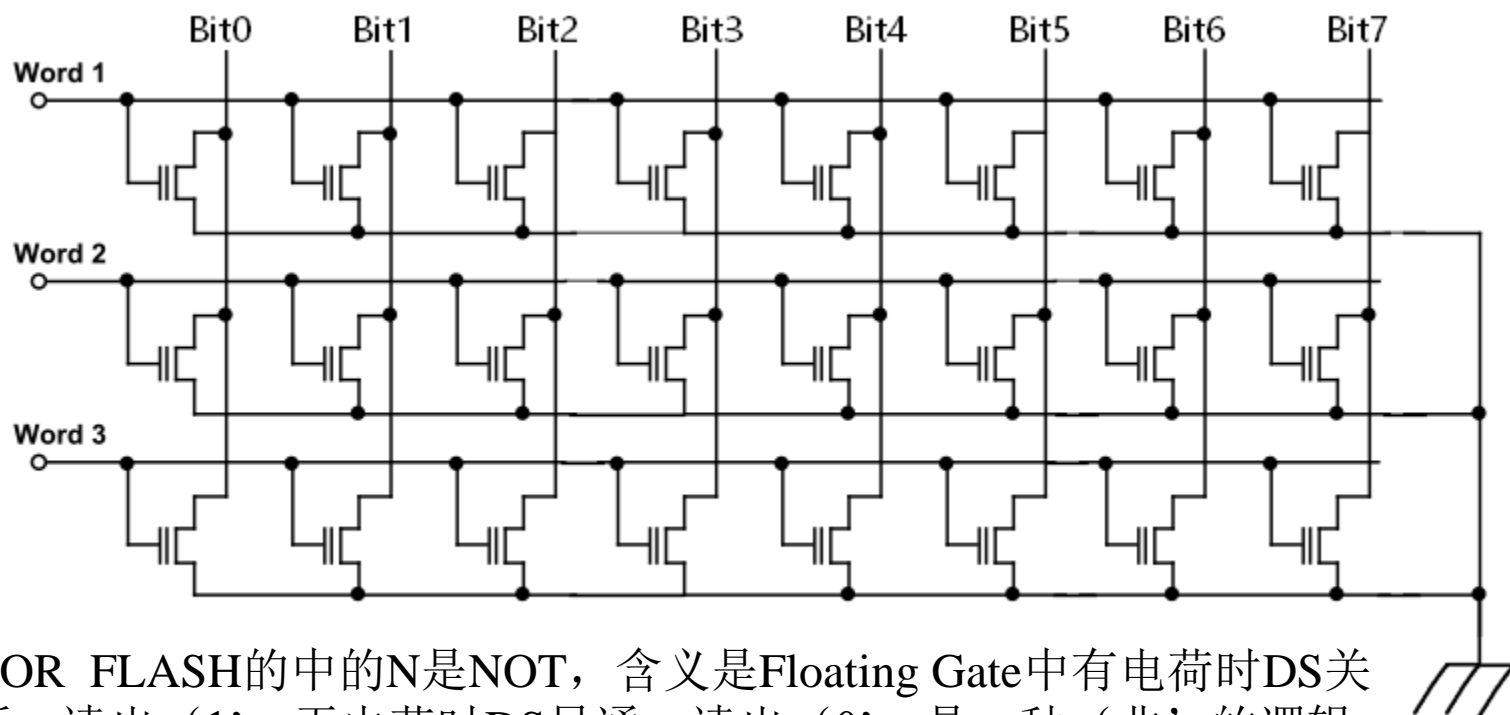
➤ 实际使用的感知放大和数据缓冲



擦除—FG不带电子—逻辑“1”—mos管容易导通，导通电流大—位线为0—数据总线位:1

写入（编程）—FG带电子—逻辑“0”—mos管难导通，导通电流小—位线为1—数据总线位:0

- 每个Bit Line下的基本存储单元是**并联**的，当某个Word Line被选中后，就可以实现对该Word的读取，也就是可以实现**位读取（即Random access）**，且具有**较高的读取速率**。

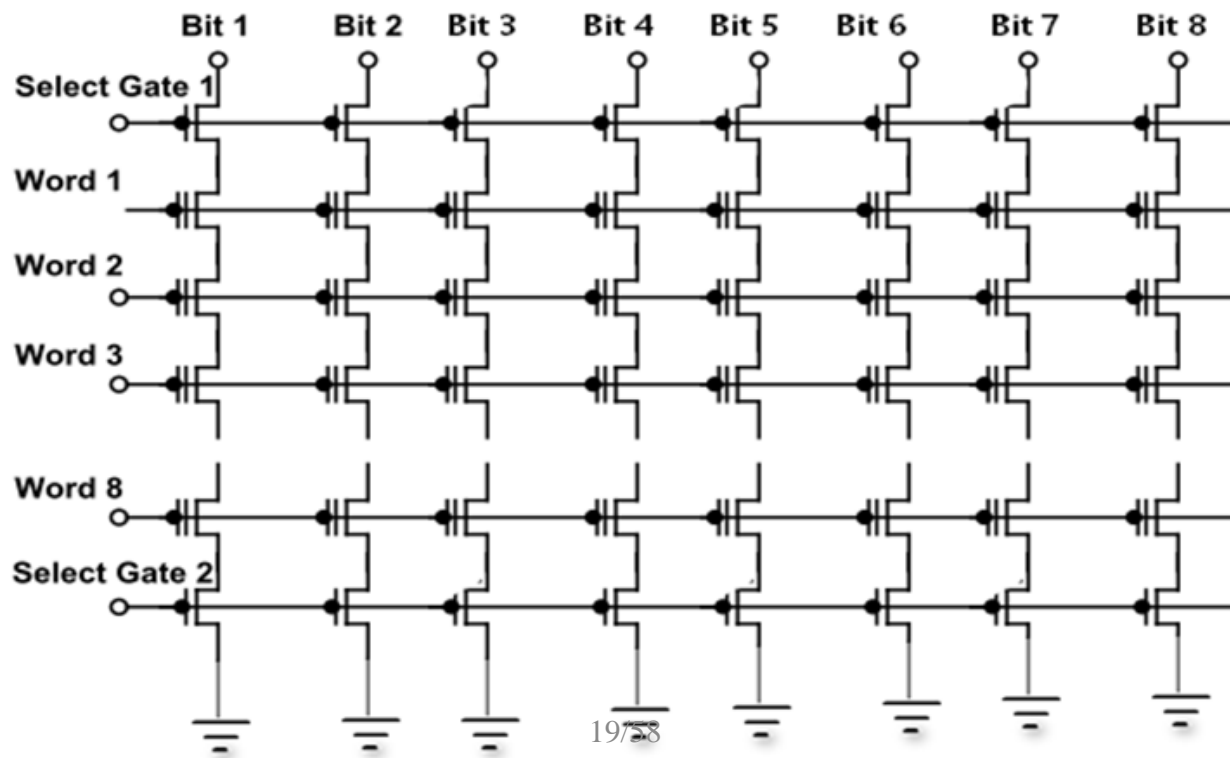


NOR FLASH中的N是NOT，含义是Floating Gate中有电荷时DS关断，读出‘1’，无电荷时DS导通，读出‘0’，是一种‘非’的逻辑；OR的含义是同一个Bit Line下的各个基本存储单元是并联的，是一种‘或’的逻辑，这就是NOR的由来。

- 基本存储单元的并联结构决定了金属导线占用很大的面积，因此NOR FLASH的**存储密度较低**，无法适用于需要大容量存储的应用场合，即适用于**code-storage**，不适用于data-storage；
- 基本存储单元的并联结构决定了NOR FLASH具有存储单元可**独立寻址且读取效率高**的特性，因此适用于code-storage，且程序可以直接在NOR 中运行（即具有RAM的特性）；
- NOR FLASH写入采用了**热电子注入**方式，**效率较低**，因此**NOR写入速率较低**，不适用于频繁擦除/写入场合。

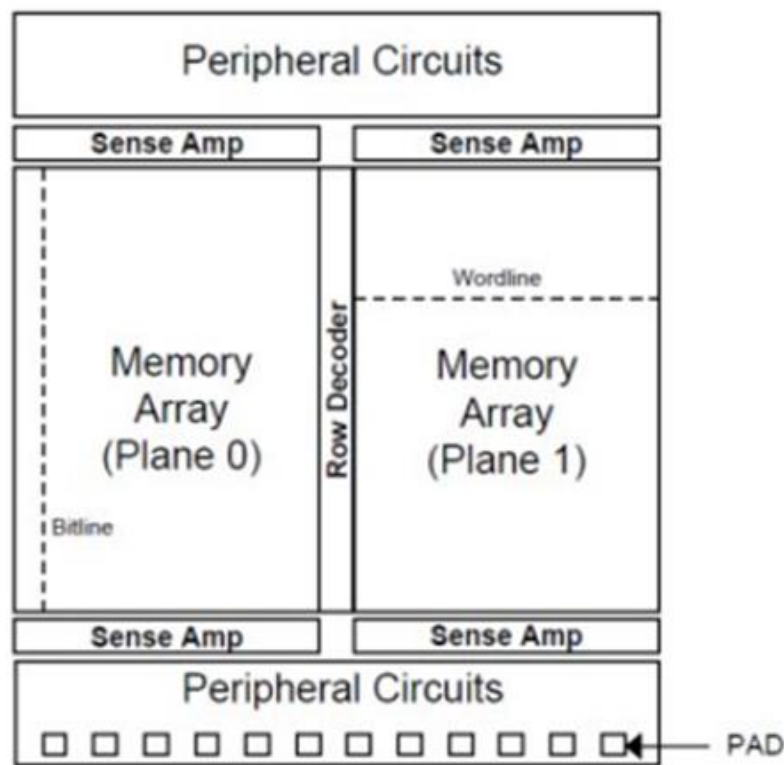
NAND型 FLASH

- 每个Bit Line下的基本存储单元是**串联**的，NAND读取数据的单位是**Page**；
- 读某个page时，对其他所有page的word line施加电压，让其D、S导通；要读取的Page的基本存储单元的D和S的导通/关断状态则取决于**Floating Gate**是否有电荷；
- NAND**无法实现位读取**（即Random access），程序代码也就无法在NAND上运行。


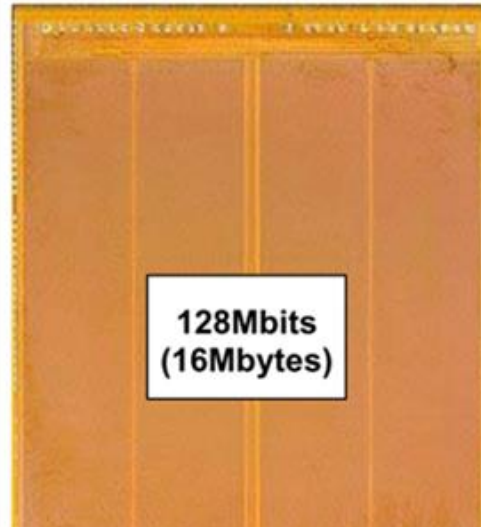


- 基本存储单元的串联结构减少了金属导线占用的面积，die的**利用率很高**，因此NAND FLASH**存储密度高**，适用于需要**大容量**存储的应用场合，即适用于**data-storage**；
- 基本存储单元的串联结构决定了NAND FLASH**无法进行位读取**，也就无法实现存储单元的独立寻址，因此程序不可以直接在NAND 中运行，因此**NAND是以Page为读取单位和写入单位，以Block为擦除单位**；
- NAND FLASH写入采用F-N隧道效应方式，**效率较高**，因此NAND擦除/写入**速率很高**，适用于频繁擦除/写入场合。同时NAND是以Page为单位进行读取的，因此**读取速率也不算低（稍低于NOR）**

➤ NAND Flash与NOR Flash对比



NAND Flash memory floorplan

32MBytes (256Mbits) DiskOnChip Millennium Plus (NAND-based)	16MBytes (128Mbits) NOR Flash
 <p>256Mbits (32Mbytes)</p>	 <p>128Mbits (16Mbytes)</p>
244% better than NOR	

- 256 KB bit-banded SRAM
 - 可以实现位操作，避免读取-修改-写入操作
- Internal ROM
 - 存储Boot Loader and vector table
 - 存储外设驱动程序
 - 存储加密标准表
 - 存储冗余校验功能
- 1024 KB Flash memory
 - 四个 256 bits预取寄存器
- 6KB EEPROM
 - 单个存储单元50万次写入

➤ 地址编排

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x000F.FFFF	On-chip Flash	621
0x0010.0000	0x01FF.FFFF	Reserved	-
0x0200.0000	0x02FF.FFFF	On-chip ROM (16 MB)	602
0x0300.0000	0x1FFF.FFFF	Reserved	-
0x2000.0000	0x2006.FFFF	Bit-banded on-chip SRAM	602
0x2007.0000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x2234.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	602
0x2235.0000	0x3FFF.FFFF	Reserved	-
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	1030
0x4000.1000	0x4000.1FFF	Watchdog timer 1	1030
0x4000.2000	0x4000.3FFF	Reserved	-

TM4C控制器中的存储器

0x4405.5000	0x5FFF.FFFF	Reserved	-
0x6000.0000	0xDFFF.FFFF	EPI0 mapped peripheral and RAM	-
Private Peripheral Bus			
0xE000.0000	0xE000.0FFF	Instrumentation Trace Macrocell (ITM)	82
0xE000.1000	0xE000.1FFF	Data Watchpoint and Trace (DWT)	82
0xE000.2000	0xE000.2FFF	Flash Patch and Breakpoint (FPB)	82
0xE000.3000	0xE000.DFFF	Reserved	-
0xE000.E000	0xE000.EFFF	Cortex-M4F Peripherals (SysTick, NVIC, MPU, FPU and SCB)	146
0xE000.F000	0xE003.FFFF	Reserved	-
0xE004.0000	0xE004.0FFF	Trace Port Interface Unit (TPIU)	83
0xE004.1000	0xE004.1FFF	Embedded Trace Macrocell (ETM)	82
0xE004.2000	0xFFFF.FFFF	Reserved	-

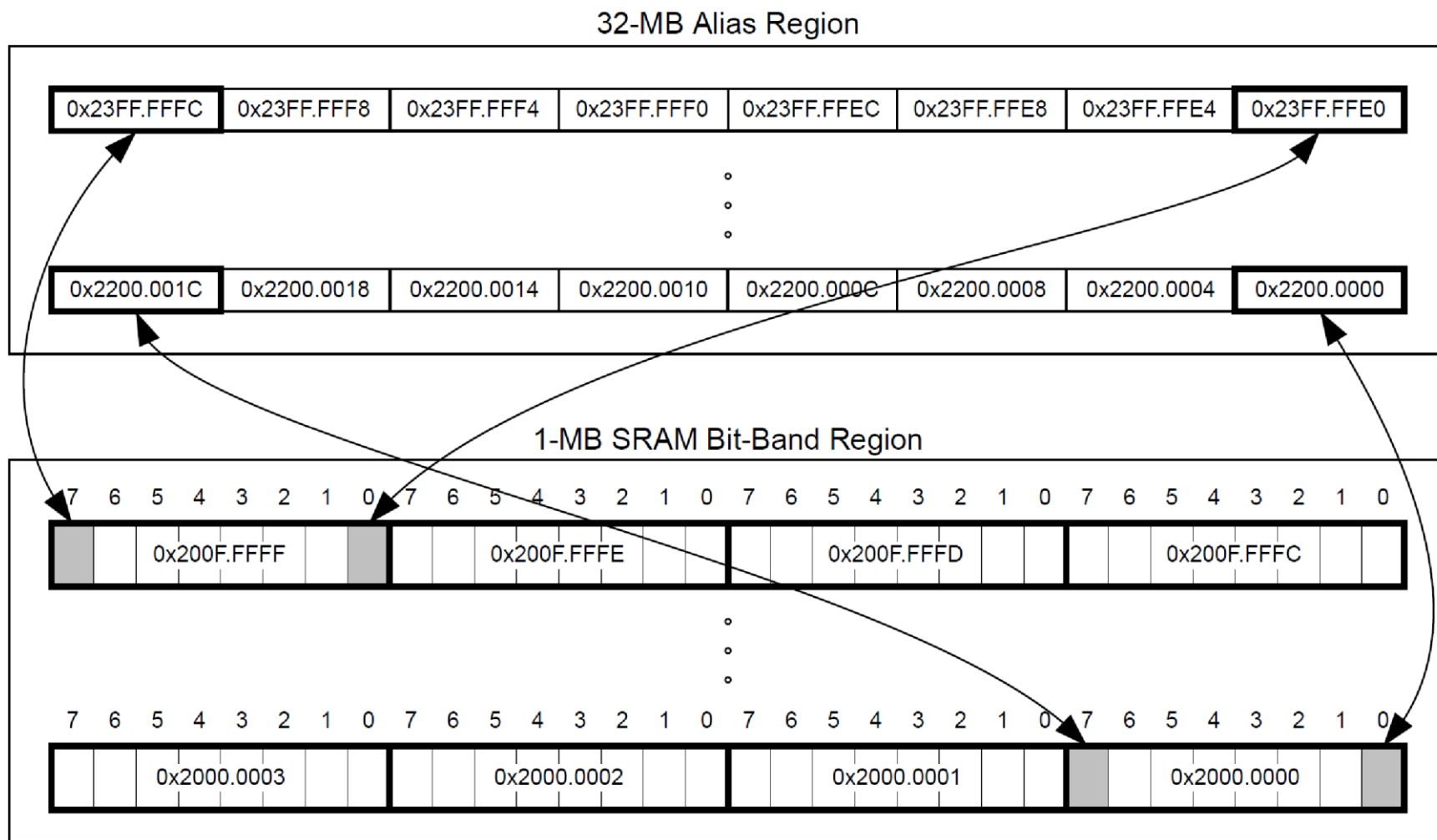


➤ Bit-banding

- SRAM起始地址0x2000.0000
- Bit-band 别名映射: 0x2200.0000
- Bit-band 别名区域的每个字映射到bit-band 区域的每个位

Address Range		Memory Region	Instruction and Data Accesses
Start	End		
0x2000.0000	0x2006.FFFF	SRAM bit-band region	Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias.
0x2200.0000	0x2234.FFFF	SRAM bit-band alias	Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped.

➤ Bit-band 示意图



➤ 使用方法

- bit-band alias = bit-band base + (byte offset * 32) + (bit number * 4)
- 例： 0x2000.1000 bit 3
 - $0x2200.0000 + (0x1000 * 32) + (3 * 4) = 0x2202.000C$

```
#define HWREG(x)  
    (*((volatile uint32_t *) (x)))
```

```
#define HWREGBITW(x, b)  
    HWREG(((uint32_t)(x) & 0xF0000000) | 0x02000000 |  
          (((uint32_t)(x) & 0x000FFFFF) << 5) | ((b) << 2))
```

➤ ROM函数

- 存储Boot Loader and vector table
- 存储外设驱动程序
- 存储加密标准表
- 存储冗余校验功能

➤ 将函数保持在Flash中

```
// Run from the PLL at 120 MHz.  
//  
ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |  
                                     SYSCTL_OSC_MAIN |  
                                     SYSCTL_USE_PLL |  
                                     SYSCTL_CFG_VCO_480), 120000000);
```

➤ 使用ROM中预先存储好的函数

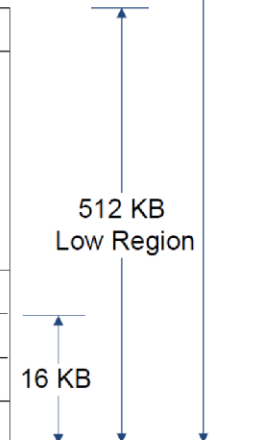
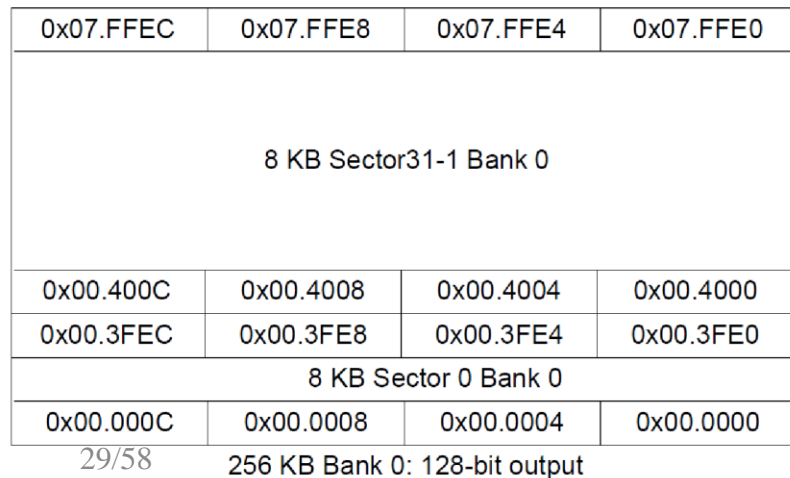
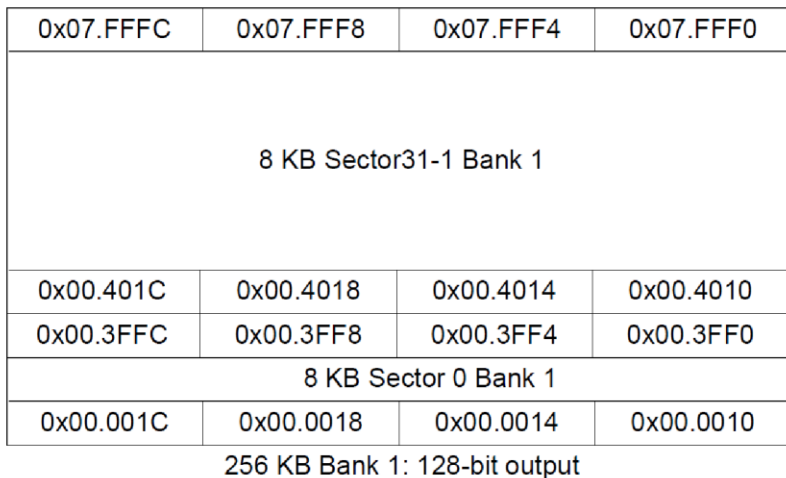
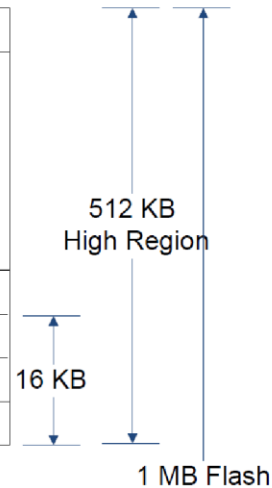
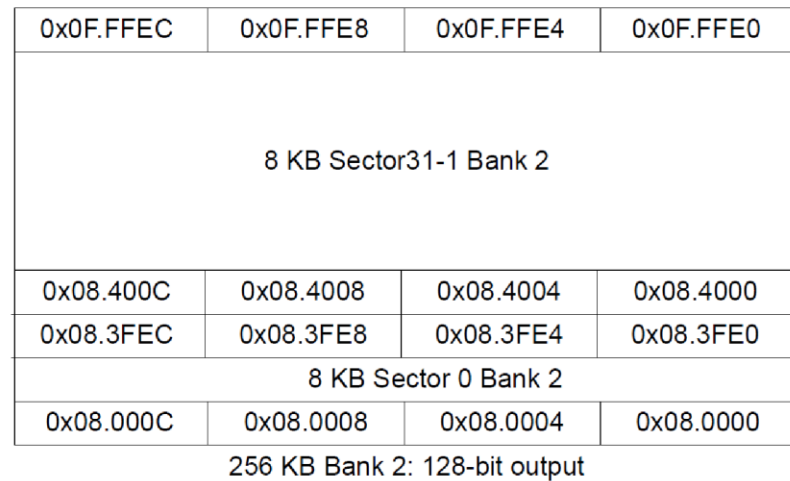
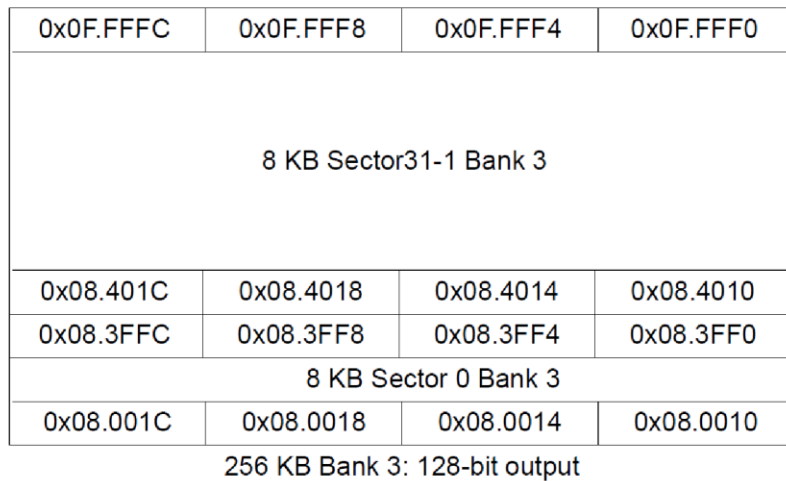
```
#include "driverlib/rom.h"  
#include "driverlib/rom_map.h"
```

```
g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |  
                                           SYSCTL_OSC_MAIN |  
                                           SYSCTL_USE_PLL |  
                                           SYSCTL_CFG_VCO_480), 120000000);
```



Flash memory

- 4个bank, 双向交错并联
- 16k擦除
- 32位存储结构, 4个256 bit 的预取寄存器



Flash memory

Prototype:

```
int32_t  
ROM_FlashErase(uint32_t ui32Address)
```

Parameters:

ui32Address is the start address of the flash block to be erased.

Description:

This function will erase a 16 kB block of the on-chip flash. After erasing the block is filled with 0xFF bytes. Read-only and execute-only blocks cannot be erased.

This function does not return until the block has been erased.

Prototype:

```
int32_t  
ROM_FlashProgram(uint32_t *pui32Data,  
                 uint32_t ui32Address,  
                 uint32_t ui32Count)
```

Description:

This function programs a sequence of words into the on-chip flash. Because the flash is programmed one word at a time, the starting address and byte count must both be multiples of four. It is up to the caller to verify the programmed contents, if such verification is required.

This function does not return until the data has been programmed.

```
1  for(i=0; i < CodeSize; i+=FlashBlockSize)  
2  {  
3      FlashProtectSet(BackupAddress+i,FlashReadWrite);  
4      FlashErase(BackupAddress+i);  
5      if(FlashProgram((uint32_t*)(CurrentAddress+i), BackupAddress+i, FlashBlockSize) == 0) { blocks++; }  
6      //FlashProtectSet(BackupAddress+i,FlashReadOnly);  
7  }
```



➤ EEPROM

- 用于保存程序配置数据
- 6K bytes
- 共有1536 32-bit words
- 分为96 blocks, 每个block有16 words (64 bytes)。每个block可以单独设置保护
- 随机读写, 8/16/32 bits读, 32bits写

Prototype:

```
void  
EEPROMRead(uint32_t *pui32Data,  
            uint32_t ui32Address,  
            uint32_t ui32Count)
```

读函数

Parameters:

pui32Data is a pointer to storage for the data read from the EEPROM. This pointer must point to at least *ui32Count* bytes of available memory.

ui32Address is the byte address within the EEPROM from which data is to be read. This value must be a multiple of 4.

ui32Count is the number of bytes of data to read from the EEPROM. This value must be a multiple of 4.

Description:

This function may be called to read a number of words of data from a word-aligned address within the EEPROM. Data read is copied into the buffer pointed to by the *pui32Data* parameter.

➤ 写函数

Prototype:

```
uint32_t  
EEPROMProgram(uint32_t *pui32Data,  
               uint32_t ui32Address,  
               uint32_t ui32Count)
```

Parameters:

pui32Data points to the first word of data to write to the EEPROM.

ui32Address defines the byte address within the EEPROM that the data is to be written to.
This value must be a multiple of 4.

ui32Count defines the number of bytes of data that is to be written. This value must be a multiple of 4.

Description:

This function may be called to write data into the EEPROM at a given word-aligned address. The call is synchronous and returns only after all data has been written or an error occurs.



➤ 示例

```
#include "driverlib/eeprom.h"
```

```
uint32_t pui32Data[2];  
uint32_t pui32Read[2];
```

```
// Program some data into the EEPROM at address 0x400.  
pui32Data[0] = 0x12345678;  
pui32Data[1] = 0x56789abc;  
EEPROMProgram(pui32Data, 0x400, sizeof(pui32Data));
```

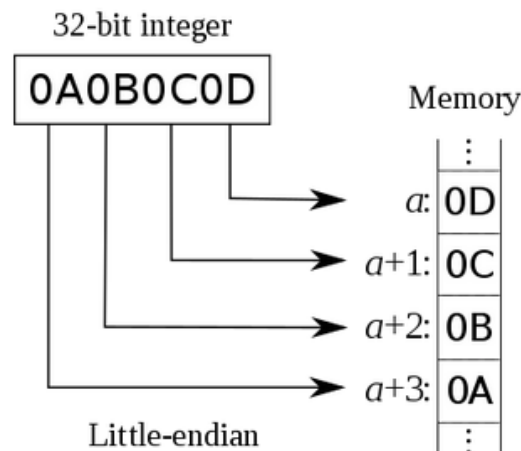
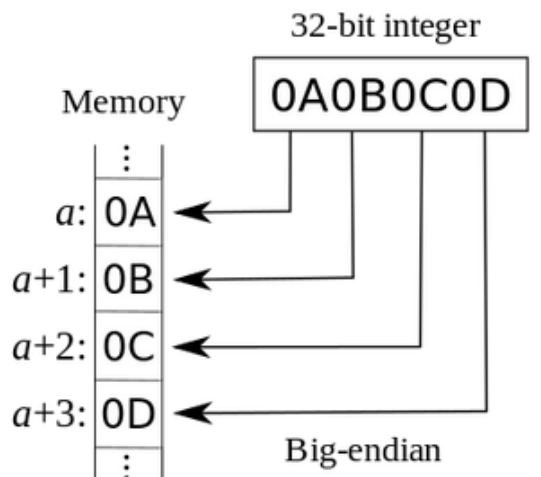
```
// Read it back.  
EEPROMRead(pui32Read, 0x400, sizeof(pui32Read));
```

➤ 字节顺序

- 又称端序或尾序（Endianness），指存储器中或在数字通信链路中，组成字的字节排列顺序
- Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端
- Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端

2.4.6 Data Storage

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. **Data is stored in little-endian format**, with the least-significant byte (lsbyte) of a word stored at the lowest-numbered byte, and the most-significant byte (msbyte) stored at the highest-numbered byte. Figure 2-5 on page 112 illustrates how data is stored.



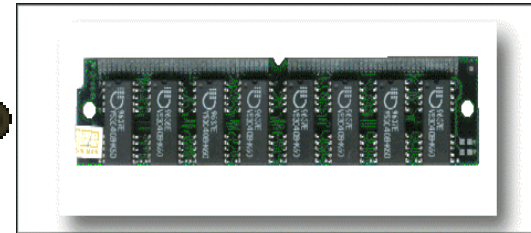
- `uint32_t a=100000, &a=0x20000010`, 那么, 存储变量a是如何存储的? 存储变量a所用的存储单元的地址分别是哪些? 这些存储单元里各存储了什么? (存储器以字节为单位编址)

`a = 0x0001 86A0`

地址	0x2000 0010	0x2000 0011	0x2000 0012	0x2000 0013
数据	A0	86	01	00



- **SRAM CPU的连接**
- **译码方法同样适合I/O端口**



- 存储芯片的数据线
- 存储芯片的地址线
- 存储芯片的片选端
- 存储芯片的读写控制线



- 若芯片的数据线正好与CPU数据线数量相同
 - 一次可从芯片中访问到8/16/32位数据
 - 全部数据线与系统的数据总线相连
- 若芯片的数据线与CPU数据线数量不同
 - 一次不能从一个芯片中访问到全部数据
 - 利用多个芯片扩充数据位
 - 这个扩充方式简称 **“位扩充”**



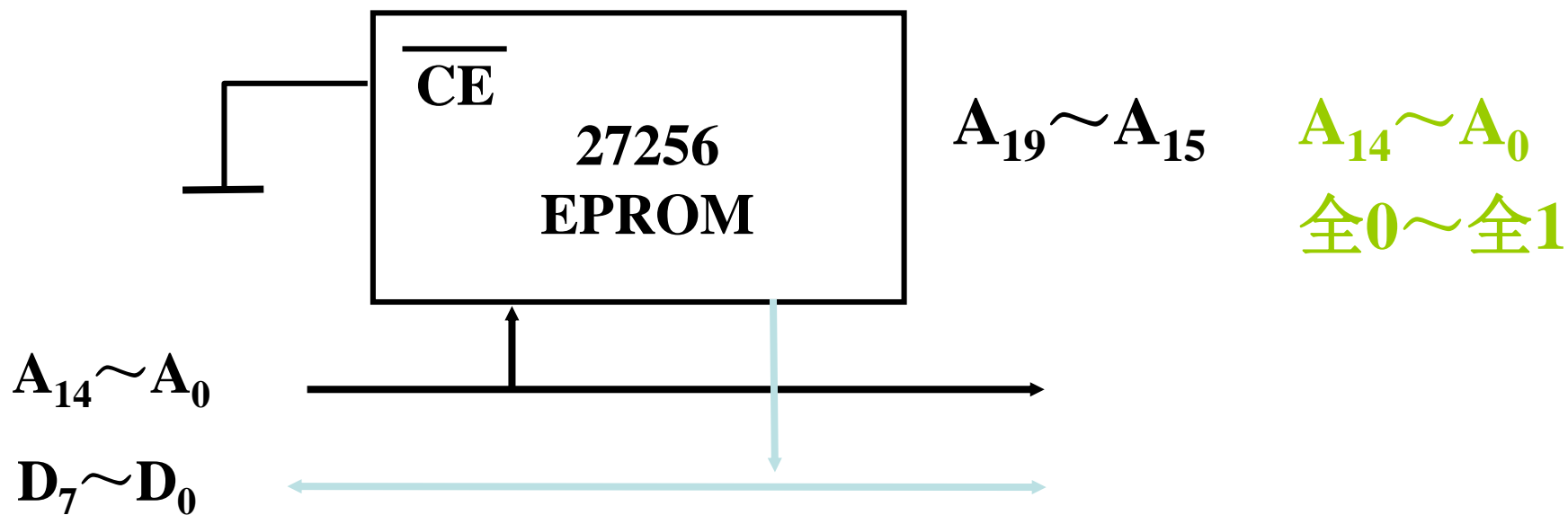
$A_9 \sim A_0$

- 多个位扩充的存储芯片的数据线连接于系统数据总线的不同位数
- 其它连接都一样
- 这些芯片应被看作是一个整体
- 常被称为“芯片组”

$D_3 \sim D_0$

- 芯片的地址线通常应全部与系统的**低位地址总线**相连
- 寻址时，这部分地址的译码是在存储芯片内完成的，我们称为“**片内译码**”

$A_9 \sim A_0$		范围（16进制）
00...00	全0 ┌ │ │ │ └ 全1	000H
00...01		001H
00...10		002H
...		...
11...01		3FDH
11...10		3FEH
11...11		3FFH



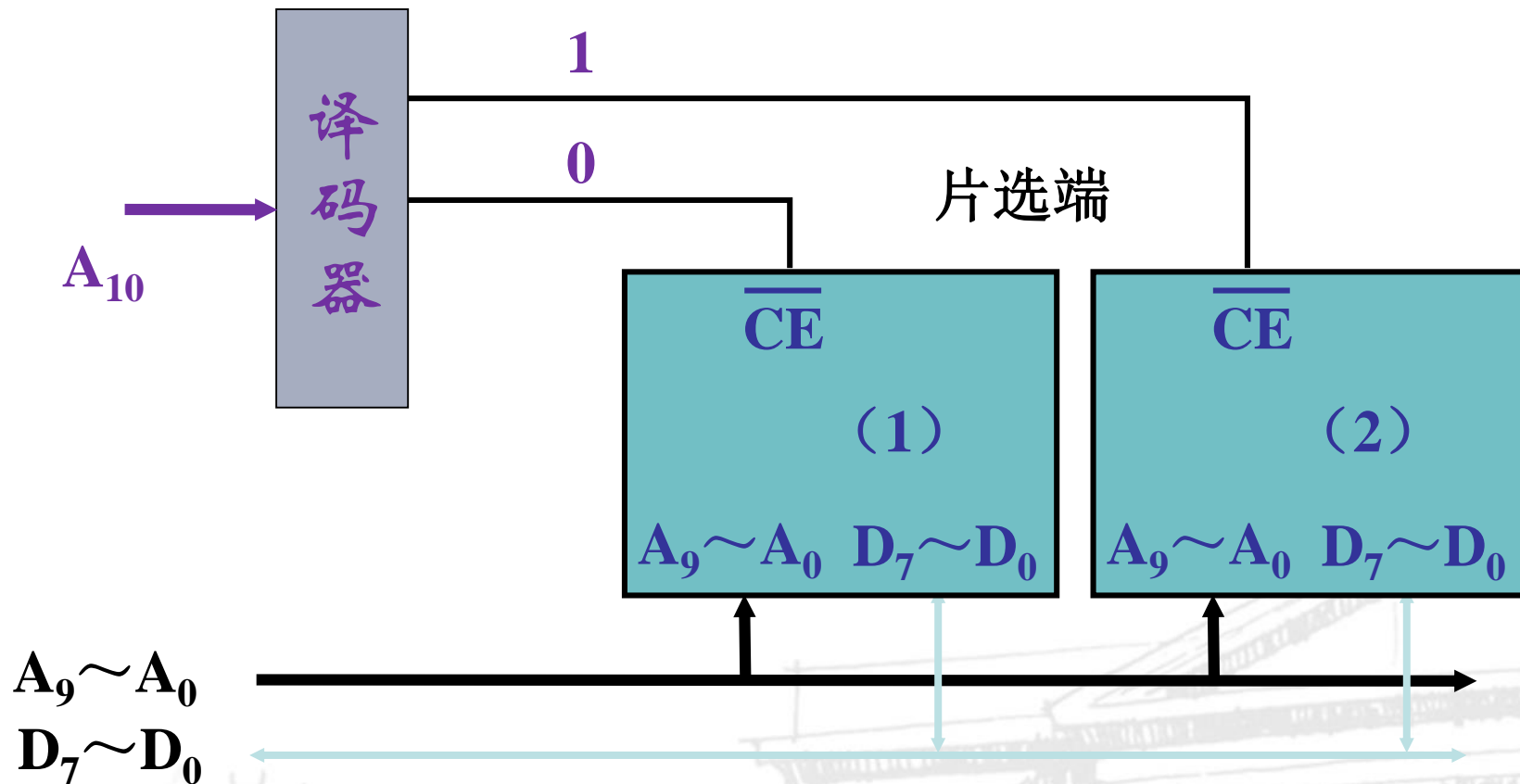
- 令芯片（组）的片选端常有效
- 不与系统的高位地址线发生联系
- 芯片（组）总处在被选中的状态
- 虽简单易行、但无法再进行地址扩充，会出现“地址重复”

- 存储系统常需利用多个存储芯片**扩充**容量
- 也就是扩充了存储器地址范围
- 进行“地址扩充”，需要利用存储芯片的**片选**端对多个存储芯片（组）进行寻址
- 这个寻址方法，主要通过将存储芯片的**片选**端与系统的**高位**地址线相关联来实现
- 这种扩充简称为“**地址扩充**”或“**字扩充**”

- **译码：将某个特定的“编码输入”翻译为唯一“有效输出”的过程**
- **译码电路可以使用门电路组合逻辑**
- **译码电路更多的是采用集成译码器**
 - **常用的2:4译码器：74LS139**
 - **常用的3:8译码器：74LS138**
 - **常用的4:16译码器：74LS154**

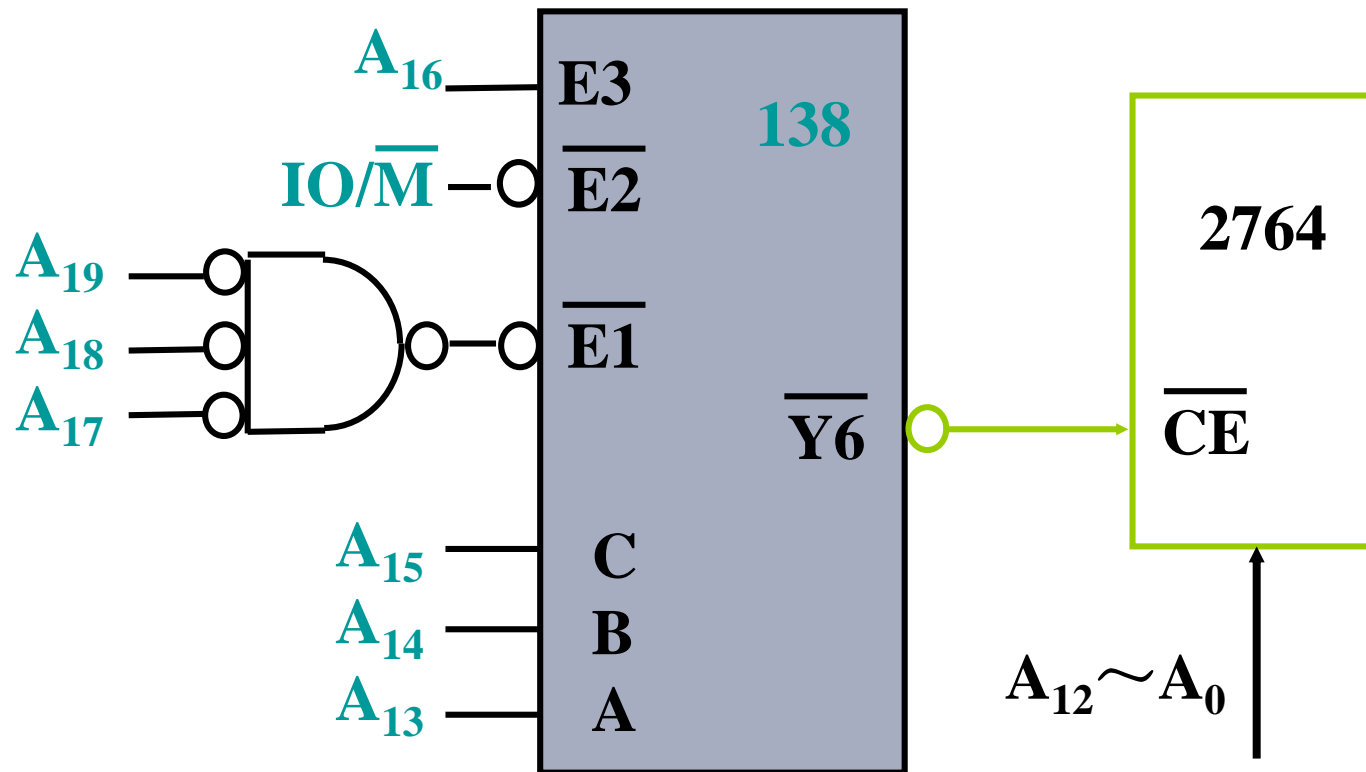


地址扩充 (字扩充)



- 所有的系统地址线均参与对存储单元的译码寻址
- 包括低位地址线对芯片内各存储单元的译码寻址（片内译码），高位地址线对存储芯片的译码寻址（片选译码）
- 采用全译码，每个存储单元的地址都是唯一的，不存在地址重复
- 译码电路可能比较复杂、连线也较多

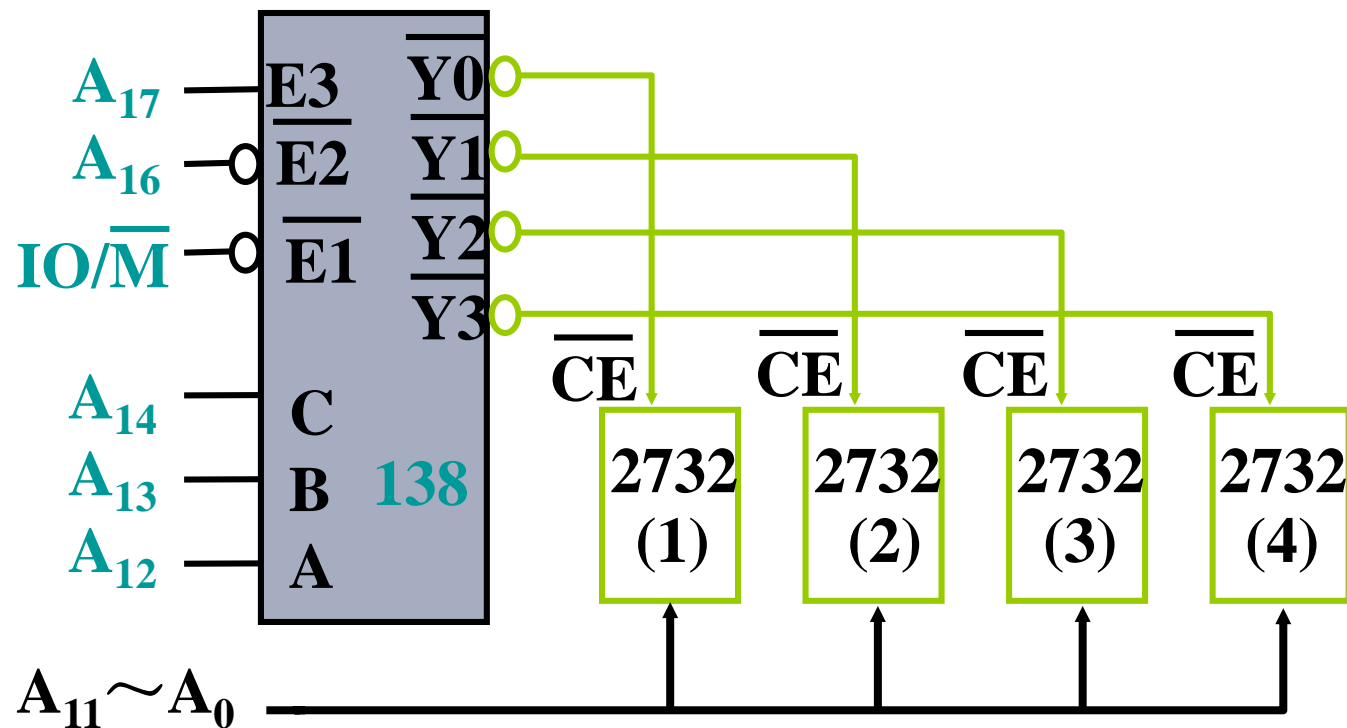
全译码示例



$A_{19}A_{18}A_{17}A_{16}A_{15}A_{14}A_{13}$	$A_{12} \sim A_0$	地址范围
0 0 0 1 1 1 0	全0	1C000H
0 0 0 1 1 1 0	全1	1DFFFH

- 只有部分（高位）地址线参与对存储芯片的译码
- 每个存储单元将对应多个地址（地址重复），需要选取一个可用地址
- 可简化译码电路的设计
- 但系统的部分地址空间将被浪费

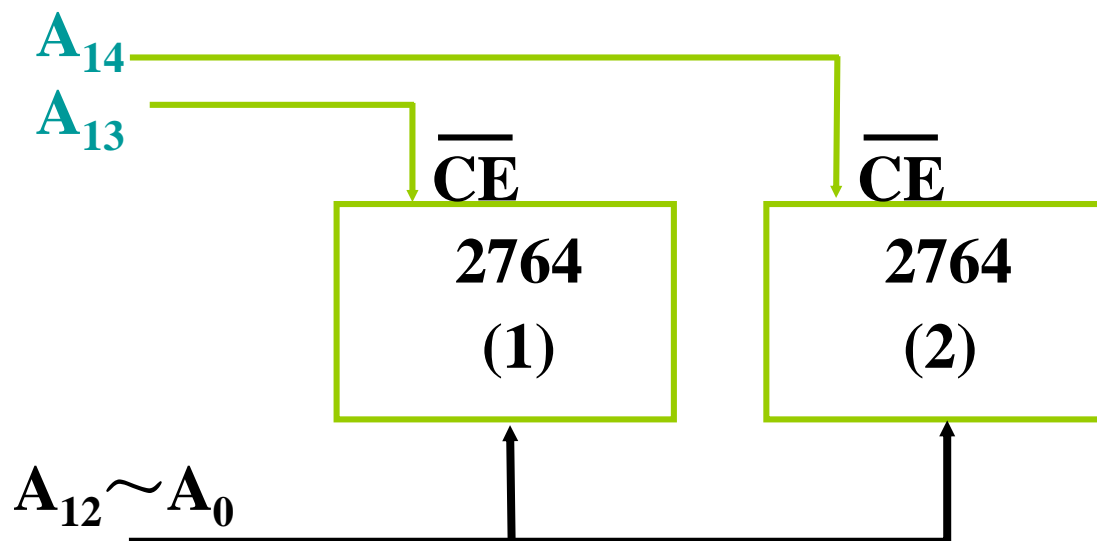
部分译码示例



	$A_{19} \sim A_{15}$	$A_{14} \sim A_{12}$	$A_{11} \sim A_0$	一个可用地址
1	$\times \times 10 \times$	000	全0~全1	20000H~20FFFH
2	$\times \times 10 \times$	001	全0~全1	21000H~21FFFH
3	$\times \times 10 \times$	010	全0~全1	22000H~22FFFH
4	$\times \times 10 \times$	011	全0~全1	23000H~23FFFH

- 只用少数几根高位地址线进行芯片的译码，且每根负责选中一个芯片（组）
- 虽构成简单，但地址空间**严重浪费**
- 可能会出现**地址重复**
- 一个存储地址会对应多个存储单元
- 多个存储单元共用的存储地址不应使用

线选译码示例



	$A_{19} \sim A_{15}$	$A_{14} A_{13}$	$A_{12} \sim A_0$	一个可用地址
1	×××××	1 0	全0~全1	04000H~05FFFH
2	×××××	0 1	全0~全1	02000H~03FFFH

切记： $A_{14} A_{13} = 00$ 的情况不能出现
00000H~01FFFH的地址不可使用



- 存储芯片的片选控制端可以被看作是一根最高位地址线
- 地址空间的选择和高位地址的译码选择
(与系统的高位地址线相关联)
- 对一些存储芯片通过片选无效可关闭内部的输出驱动机制，起到降低功耗的作用



➤ 芯片/OE与系统的读命令线相连

- 当芯片被选中、且读命令有效时，存储芯片将开放并驱动数据到总线

➤ 芯片/WE与系统的写命令线相连

- 当芯片被选中、且写命令有效时，允许总线数据写入存储芯片



➤ 两个问题

– CPU的总线**负载能力**

- CPU能否带动总线上包括存储器在内的连接器件
- CPU的总线驱动**能力有限**
- 单向传送的地址和控制总线，可采用**三态锁存器**和**三态单向驱动器**等来加以锁存和驱动
- 双向传送的数据总线，可以采用三态双向驱动器来加以驱动



– 存储芯片与CPU总线时序的配合

- CPU能否与存储器的存取速度相配合
- 分析存储器的存取速度是否满足CPU总线时序的要求
- 如果不能满足：考虑更换芯片，或总线周期中插入等待状态 T_w

切记：时序配合是连接中的难点

➤ 作业

- 1: 十进制数10, -100, -1000, 10000转换为16进制(补码, 32位存储)。
- 2: 定义两个变量 `int8_t m=100`, `n=-100`;他们在计算机中是以_____进制的形式存储的, 具体存储的数据为:
m:_____, n:_____。
- 3: 定义一个字符串: `char hello[] = "Hello"`;则存储器 `hello` 地址处存放的数据是_____, `hello+1` 处存放的数据是_____, `hello+2` 处存放的数据是_____, `hello+3` 处存放的数据是_____, `hello+4` 处存放的数据是_____, `hello+5` 处存放的数据是_____。
(为方便书写, 可用16进制填写, ASCII码)。
- 4: 定义一个变量 `int16_t k=1230`。把这个变量, 从一个嵌入式系统传到另一个嵌入式系统, 实际上传递的数据是_____。(为方便书写, 可用16进制填写)。



➤ 作业

- 5: 存储器一般由____、____、____、____四部分组成。
- 6: SRAM的基本存储单元是____、DRAM的基本存储单元是____、U盘的基本存储单元是____。
- 7: TM4C1294的程序存储器是____类型的FLASH。
- 8: TM4C1294的FLASH中，一个uint32_t类型的数据存放在地址0x0000 0000处，的内容是 0x2000 036C，那么0x0000 0000处存放的字节是____，0x0000 0001处存放的字节是____，0x0000 0002处存放的字节是____，0x0000 0003处存放的字节是____。



谢谢!

