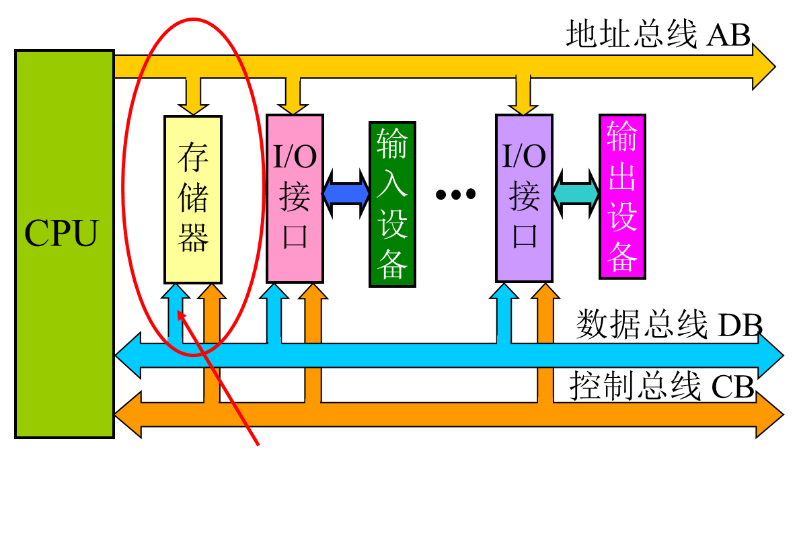
# 存储器的基本概念：

存储器是计算机及嵌入式系统中的记忆设备

存储器通过**总线**与CPU相连:



半导体存储器的分类：

存储器可以由半导体组成，也可以是磁带光盘等。

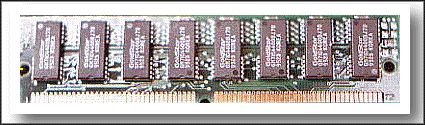
**半导体存储器**：

* 随机存取存储器（RAM）
  + 静态RAM（SRAM）

组成单元: 触发器；速度：快；集成度：低；应用：小容量系统

* + 动态RAM（DRAM）

组成单元: 极间电容；速度：慢；集成度：高；应用：大容量系统



* + 非易失RAM（NVRAM）

组成单元: 铁电材料；速度：慢；集成度：低；应用：小容量非易失

* 只读存储器（ROM）
  + 掩膜式ROM

信息制作在芯片中，不可更改

* + 一次性可编程ROM（PROM）

允许一次编程，此后不可更改

* + 紫外线擦除可编程ROM（EPROM）

用紫外光擦除，擦除后可编程；并允许用户多次擦除和编程

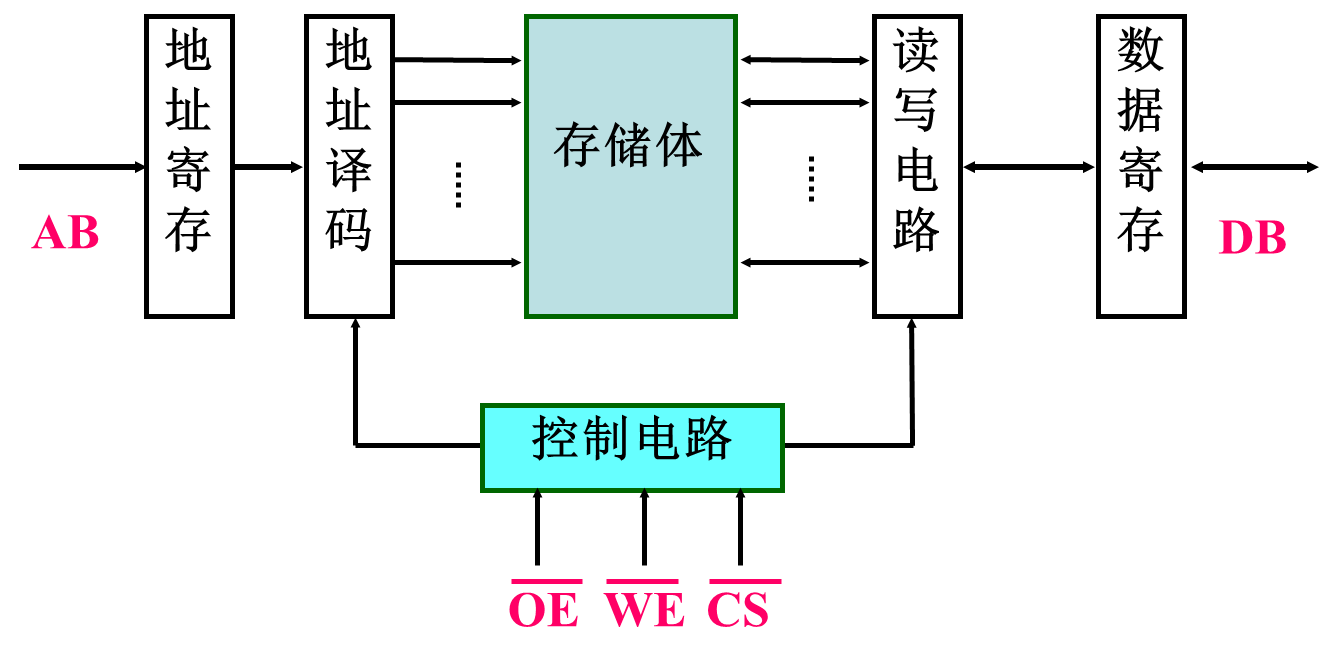
* + 电擦除可编程ROM（EEPROM）

采用加电方法在线进行擦除和编程，也可多次擦写

* + 闪速存储器（Flash Memory）

能够快速擦写的，但只能按块（Block）擦除

存储器的组成：



* 存储体：

存储器芯片的主要部分，用来存储信息

每个**存储单元**具有一个唯一的地址，可存储1位（位片结构）或多位（字片结构）二进制数据

但是一个**基本存储单元**，只能存储1位数据

芯片的存储容量：

＝2M×N bits

＝存储单元数×存储单元的位数

M：芯片的地址线根数

N：芯片的数据线根数

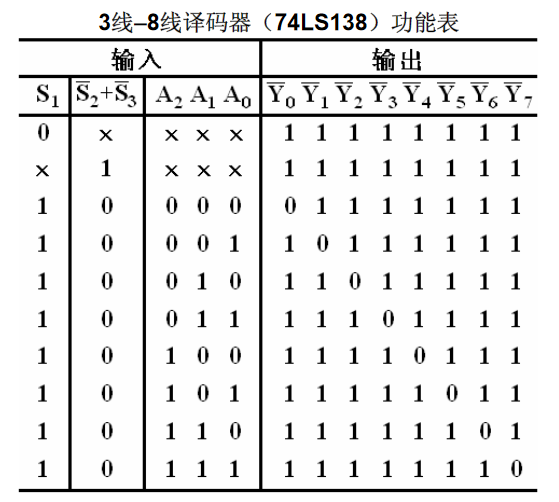
**寻址空间：**

寻址空间一般指的是CPU对于内存寻址的能力。通俗地说，就是能最多用到多少内存的一个问题。数据在存储器(RAM)中存放是有规律的，CPU在运算的时候需要把数据提取出来就需要知道数据存放在哪里 ，这时候就需要挨家挨户的找，这就叫做寻址，但如果地址太多超出了CPU的能力范围，CPU就无法找到数据了。CPU最大能查找多大范围的地址叫做寻址能力，CPU的寻址能力以字节为单位，如32位寻址的CPU可以寻址2的32次方大小的地址也就是4G，这也是为什么32位的CPU最大能搭配4G内存的原因 ，再多的话CPU就找不到了。

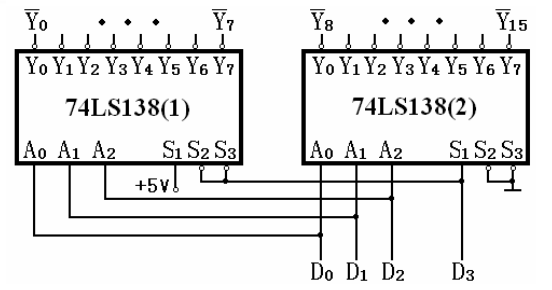
* 地址译码电路：

根据输入的地址编码来选中芯片内某个特定的存储单元

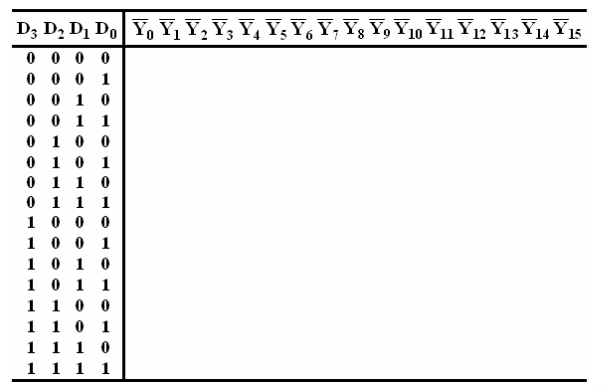
根据地址，选中其唯一对应的存储单元，典型的74LS138译码电路：

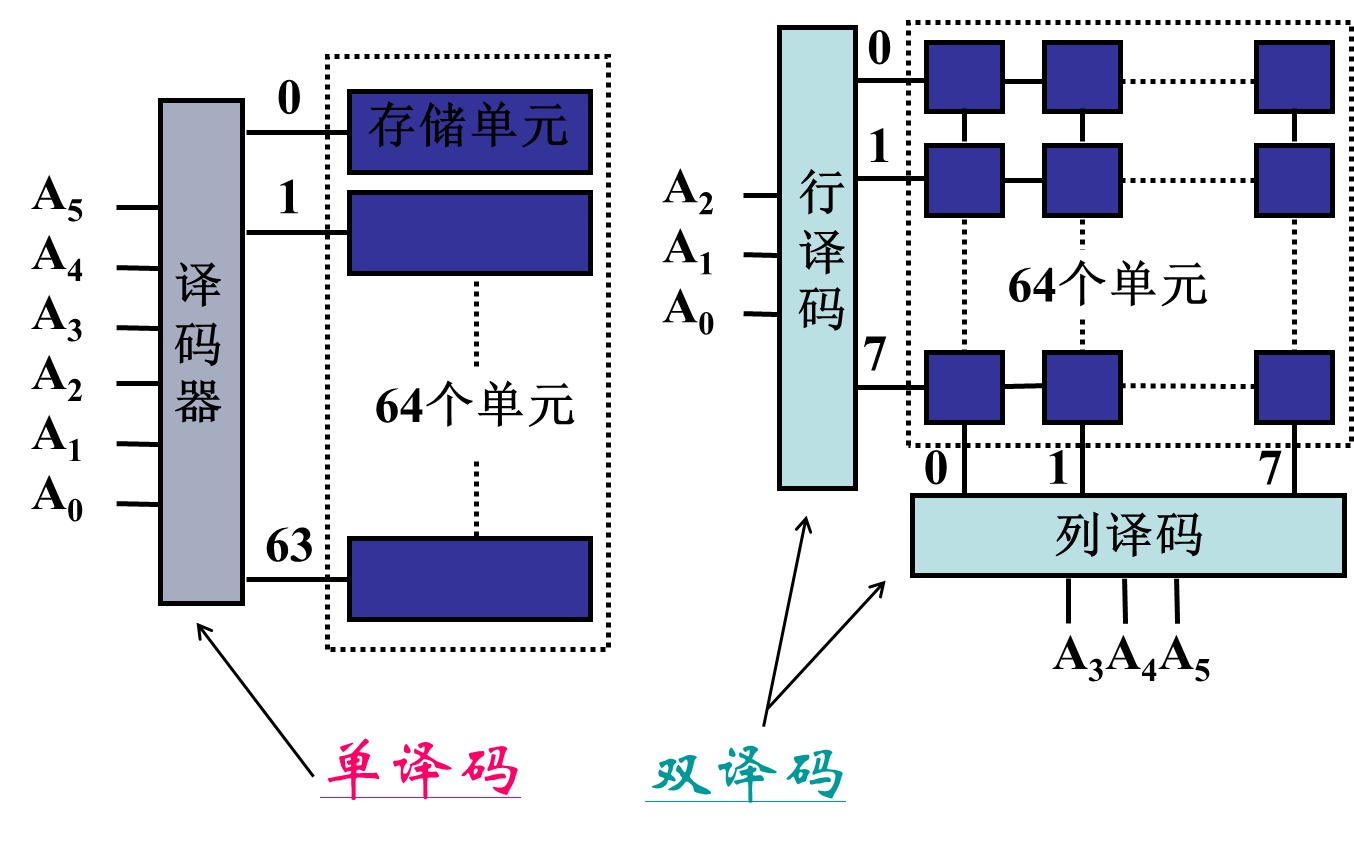


译码电路可以级联使用：



填写以上译码电路的真值表：

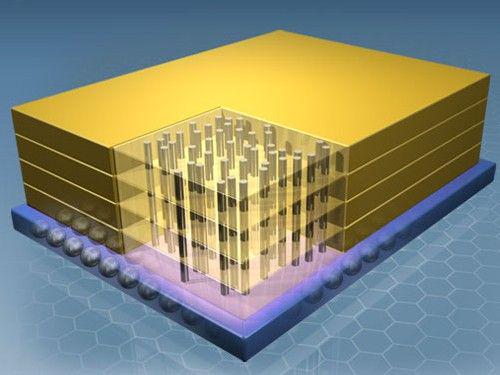




双译码可简化芯片设计，是主要采用的译码结构

3D存储技术：

普通的存储芯片多为平面结构，数据只能前后左右移动，而3D存储芯片可实现数据在三维空间中的存储和传递，将大幅提高存储设备的存储能力。



* 片选和读写控制电路：

选中存储芯片，控制读写操作

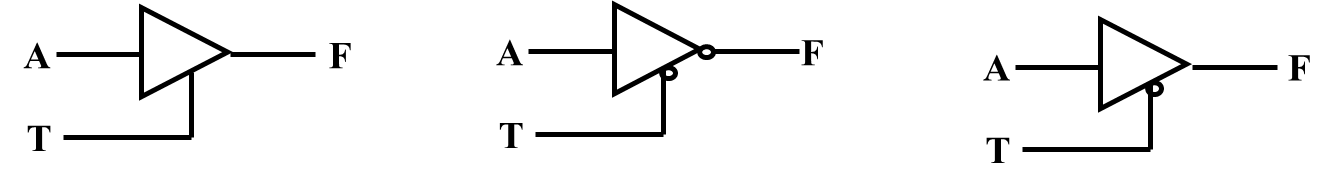
* 数据缓冲：

存储器数据与总线的接口，本存储器工作时，既要输入，又要输出。本存储器不工作时，要呈现高阻态，不影响其他存储器工作。

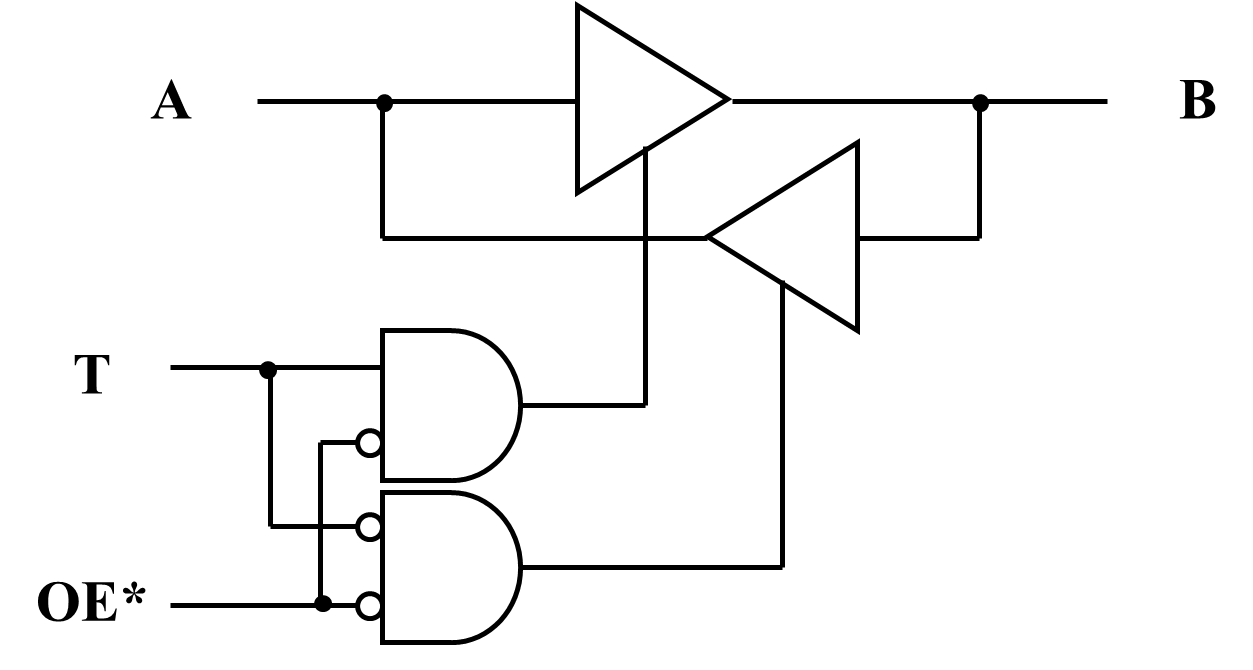
双向，三态

**三态：** 高电平、低电平、高阻态

三态门：



双向缓冲：具有双向导通和三态的特性



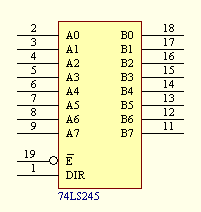
OE\*＝0，导通

T＝1 A→B

T＝0 A←B

OE\*＝1，不导通，高阻态

典型电路：



E\*＝0，导通

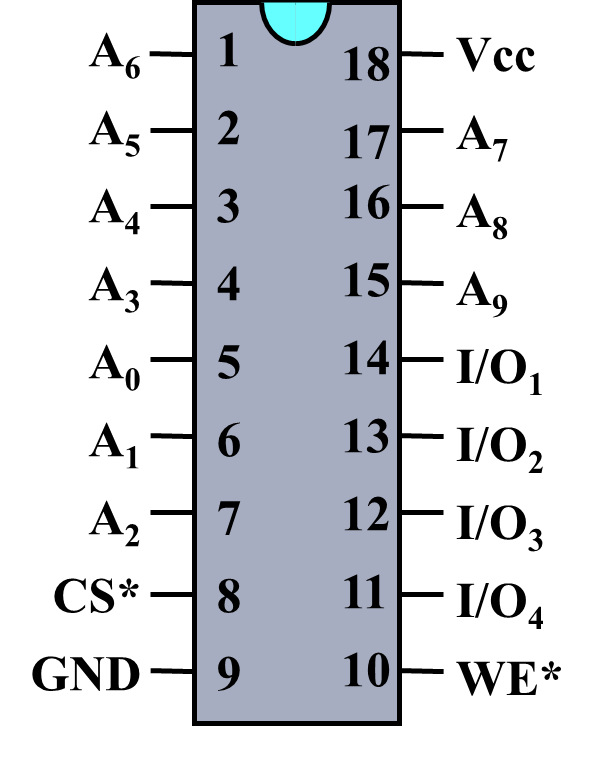
DIR＝1 A→B

DIR＝0 A←B

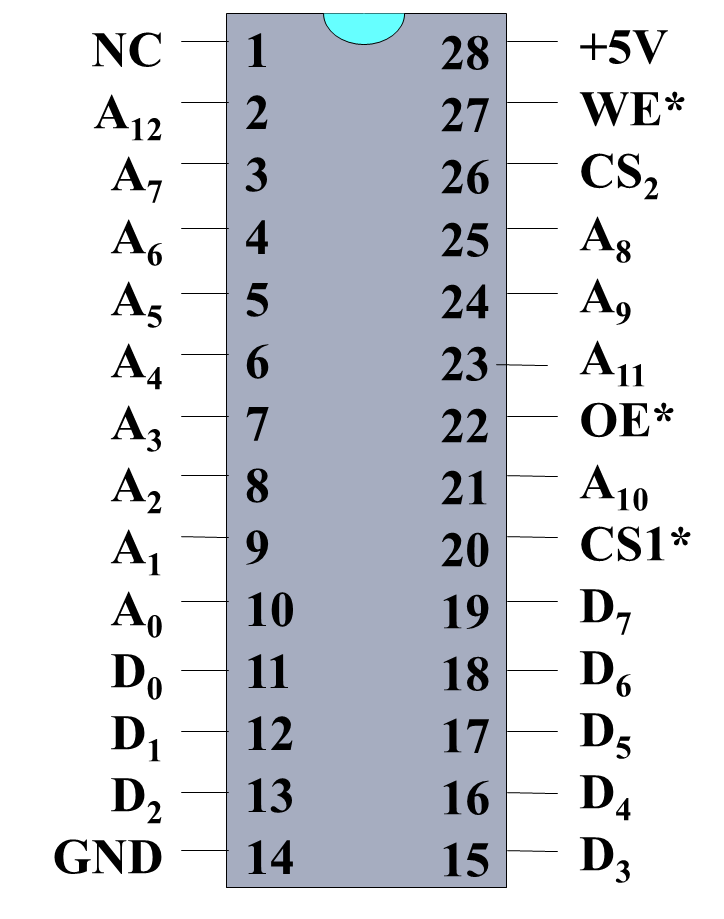
E\*＝1，不导通

根据芯片的数据线和地址线数量，判断芯片容量：

SRAM芯片2114: 存储容量为1024×4



SRAM芯片6264：存储容量为8K×8



# SRAM的工作原理

**SRAM：**

基本存储单元：触发器电路

每个基本存储单元存储二进制数一位，许多个基本存储单元形成行列存储矩阵

SRAM一般采用“字结构”存储矩阵：每个字存放多位（4、8、16等），每个字具有一个地址

举例：

常用的SRAM：IS61LV25616AL

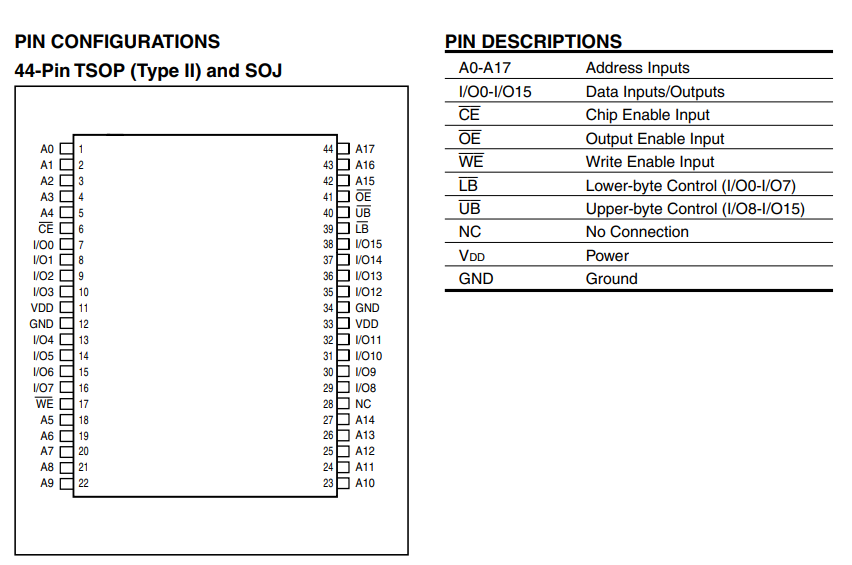
共有 4,194,304-bit 个存储单元 262,144 ×16 bits

地址线：A0 - A17，共18个，地址空间：218=262,144

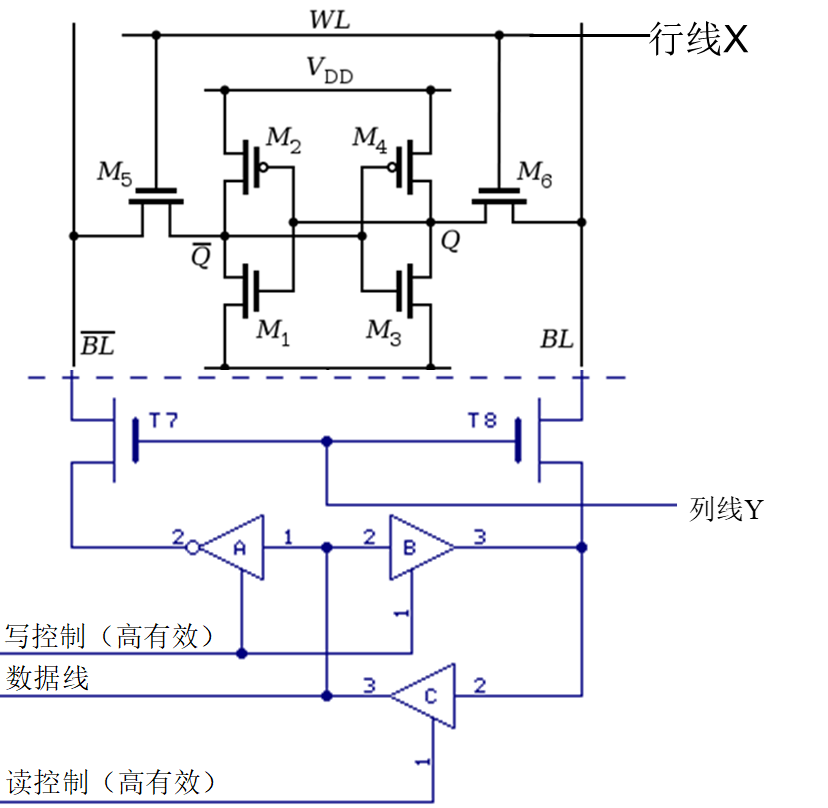
数据线：I/O0 - I/O15 共16个，存储单元数量：218×16 bits=4,194,304-bit

控制线：CE、OE、WE、LB、UB

电源线：VDD、GND



SRAM六管结构的工作原理：



* 保持（Standby）

如果字线（Word Line）没有被选为高电平, 那么作为控制用的M5与M6两个晶体管处于断路，把基本单元与位线隔离。由M1 – M4组成的两个反相器继续保持其状态，只要保持与高、低电平的连接。数据线呈高阻态。

* 读（Reading ）

行线和列线被选中，读使能。写无效。

M5，M6，M7，M8导通，Q中保存的值传送的数据线上。

* 写（Writing）

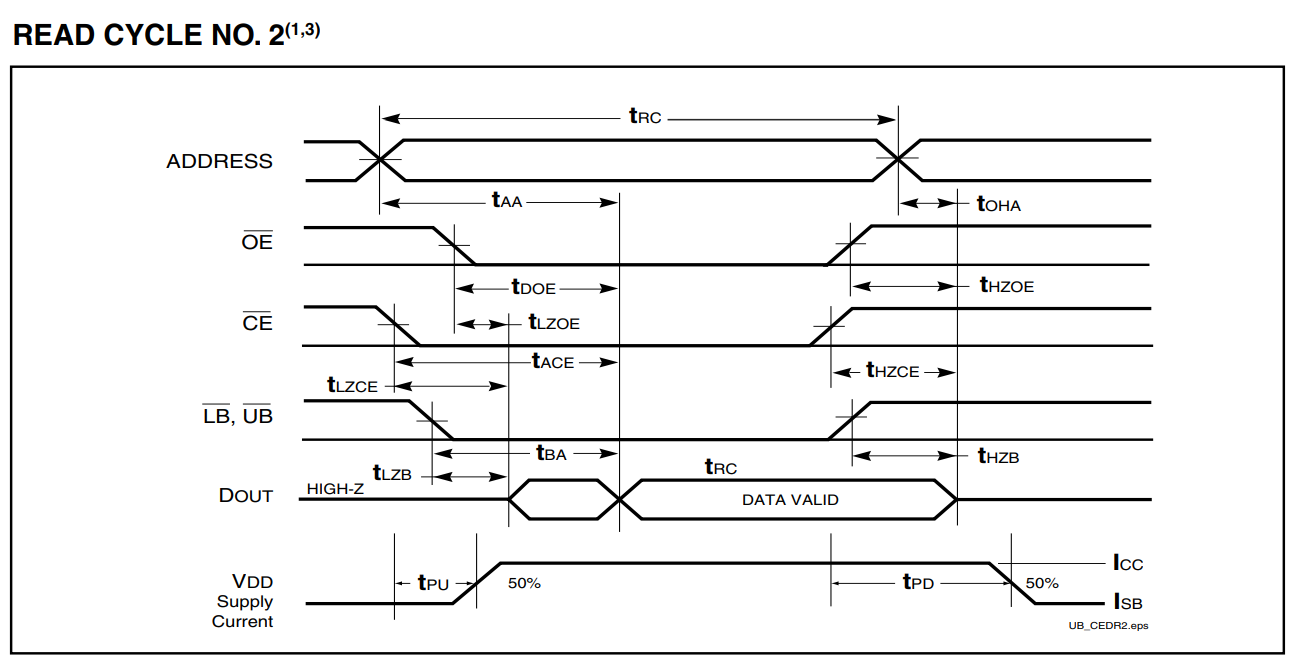
行线和列线被选中，写使能。读无效。

M5，M6，M7，M8导通，数据线上为需要写入的数据，且具有较强的驱动能力，强制存储单元改变状态。

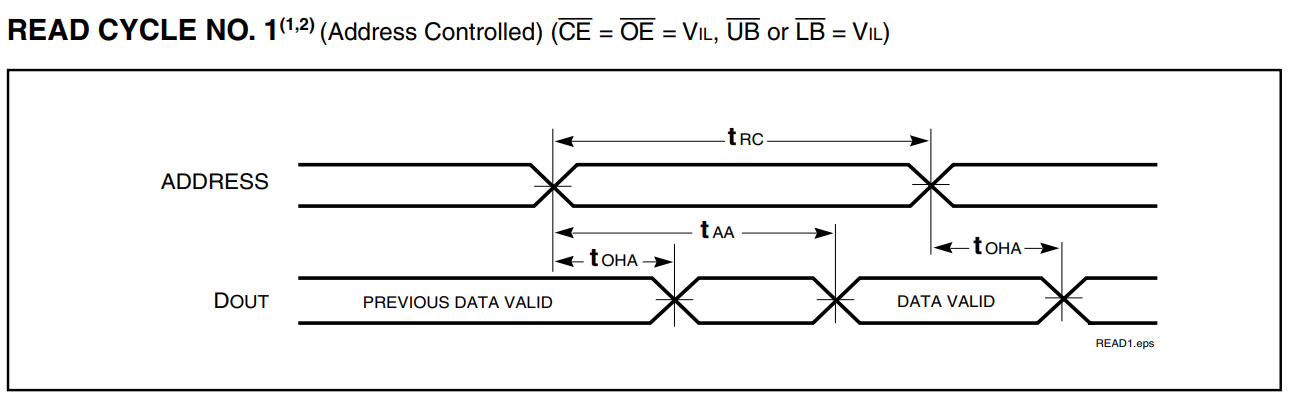
根据SRAM的工作原理，读写过程非常简单，以IS61LV25616AL为例：

**读**：

方式1： 控制使能和片选，给出地址，数据就会出现在地址线上

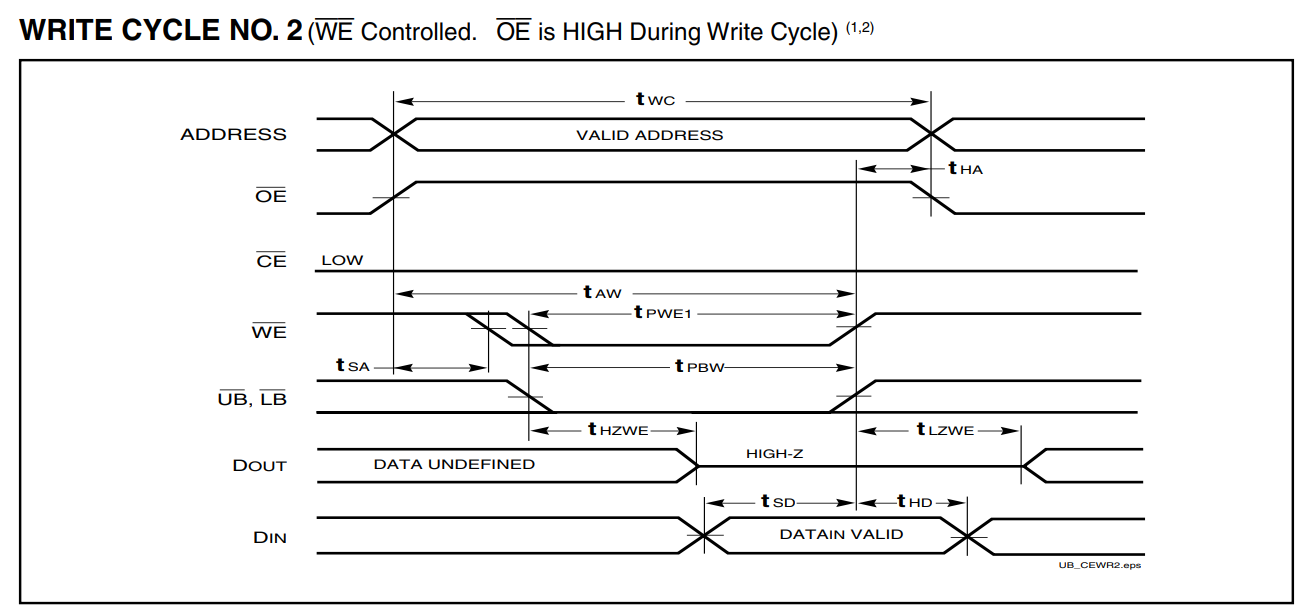


方式2：小规模简单系统中，也可以将使能和片选永久有效，给出地址，即得到数据：



写：

地址、数据准备好，WE信号有效后，数据即可写入。

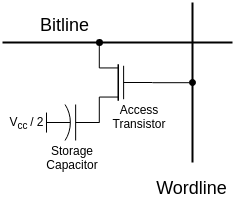


# DRAM的工作原理

**DRAM:**

DRAM（Dynamic Random Access Memory），即动态随机存储器，也就是我们常说的计算机内存，在现代计算机系统和SOC系统中有很重要的作用。

结构：主要有存储电容、访问晶体管（Access Transistor）、字线（Wordline）和位线（Bitline）



**存储电容(Storage Capacitor)：**

它通过存储在其中的电荷的多和少，或者说电容两端电压差的高和低，来表示逻辑上的 1 和 0。存储电容的 Common 端接在 Vcc/2。

当存储电容存储的信息为 1 时，另一端电压为 Vcc，此时其所存储的电荷

Q = +Vcc/2 / C

当存储电容存储的信息为 0 时，另一端电压为 0，此时其所存储的电荷

Q = -Vcc/2 / C

**访问晶体管(Access Transistor)：**

它的导通和截止，决定了允许或禁止对存储电容所存储的信息的读取和改写。

**字线(Wordline)：**

它决定了访问晶体管的导通或者截止。

**位线(Bitline)：**

它是外界访问存储电容的唯一通道，当访问晶体管导通后，外界可以通过位线对存储电容进行读取或者写入操作。

**DRAM读写原理及其问题：**

* **读数据时，字线设为逻辑高电平，打开访问晶体管，然后读取位线上的状态**
* **写数据时，先把要写入的电平状态设定到位线上，然后打开访问晶体管，通过位线改变存储电容内部的状态。**
* **问题：**
  + **外界的逻辑电平与 Storage Capacitor 的电平不匹配**

由于位线的电容值比存储电容要大的多（通常为 10 倍以上），当访问晶体管导通后，如果存储电容存储的信息为 1 时，位线电压变化非常小。外界电路无法直接通过位线来读取存储电容所存储的信息。

* + **进行一次读取操作后，存储电容存储的电荷会变化**

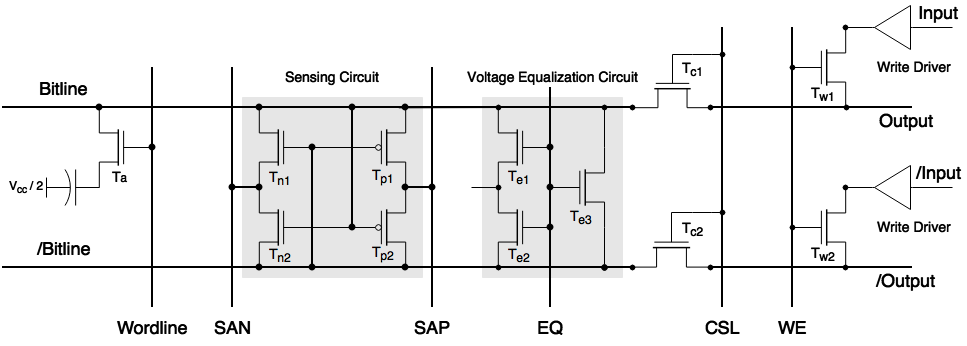
在进行一次读取操作的过程中，访问晶体管导通后，由于位线存储电容端的电压不一致，会导致存储电容中存储的电荷量被改变。最终可能会导致在下一次读取操作过程中，无法正确的判断存储电容内存储的信息。

* + **由于存储电容的物理特性，即使不进行读写操作，其所存储的电荷都会慢慢变少**

这个特性要求 DRAM 在没有读写操作时，也要主动对存储电容进行电荷恢复的操作。

**DRAM读写电路：**

差分读取放大器



差分读取放大器（Differential Sense Amplifier）包含 **读出电路**（Sensing Circuit） 和**电压均衡电路**（Voltage Equalization Circuit） 两个主要部分。它主要的功能就是

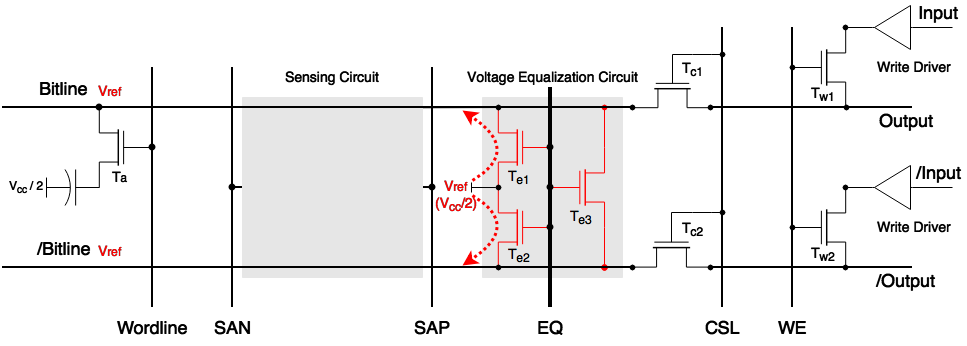
1. 将存储电容存储的信息转换为逻辑 1 或者 0 所对应的电压，并且呈现到位线上。
2. 同时，在完成一次读取操作后，通过位线将存储电容中的电荷恢复到读取之前的状态。

* 读周期：

一个完整的 Read Operation 包含了，**Precharge**、**Access**、**Sense**、**Restore** 四个阶段。

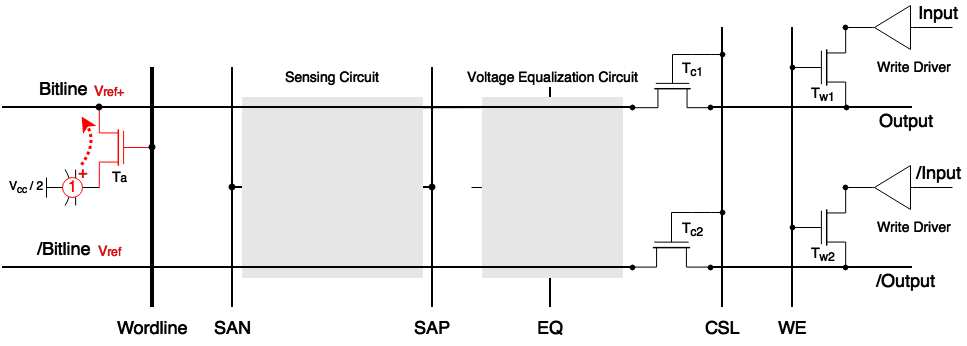
**Precharge:**

在这个阶段，首先会通过控制 EQ 信号，让 Te1、Te2、Te3 晶体管处于导通状态，将 Bitline 和 /Bitline 线上的电压稳定在 Vref 上, Vref = Vcc/2。然后进入到下一个阶段。



**Access：**

经过 Precharge 阶段， Bitline 和 /Bitline 线上的电压已经稳定在 Vref 上了，此时，通过控制 Wordline 信号，将 Ta 晶体管导通。Storage Capacitor 中存储正电荷会流向 Bitline，继而将 Bitline 的电压拉升到 Vref+。然后进入到下一个阶段。

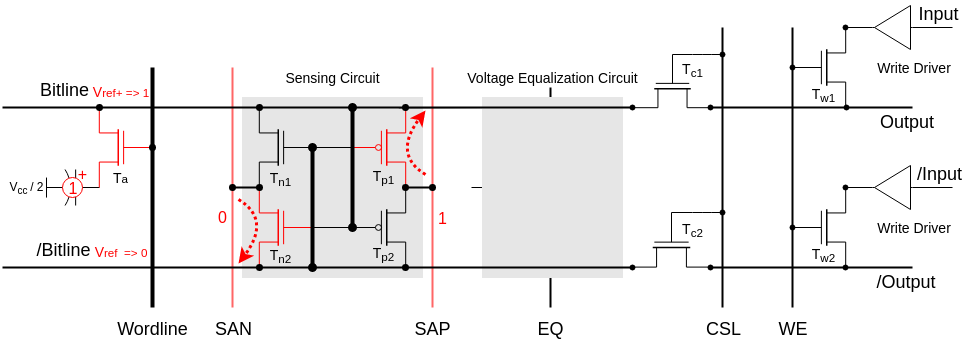


**Sense：**

由于在 Access 阶段，Bitline 的电压被拉升到 Vref+，Tn2 会比 Tn1 更具导通性，Tp1 则会比 Tp2 更具导通性。

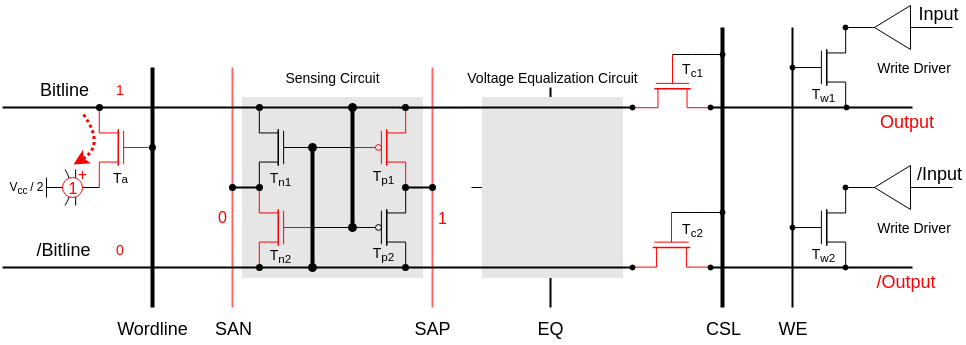
此时，SAN (Sense-Amplifier N-Fet Control) 会被设定为逻辑 0 的电压，SAP (Sense-Amplifier P-Fet Control) 则会被设定为逻辑 1 的电压，即 Vcc。由于 Tn2 会比 Tn1 更具导通性，/Bitline 上的电压会更快被 SAN 拉到逻辑 0 电压，同理，Bitline 上的电压也会更快被 SAP 拉到逻辑 1 电压。接着 Tp1 和 Tn2 进入导通状态，Tp2 和 Tn1 进入截止状态。

最后，Bitline 和 /Bitline 的电压都进入稳定状态，正确的呈现了 Storage Capacitor 所存储的信息 Bit。



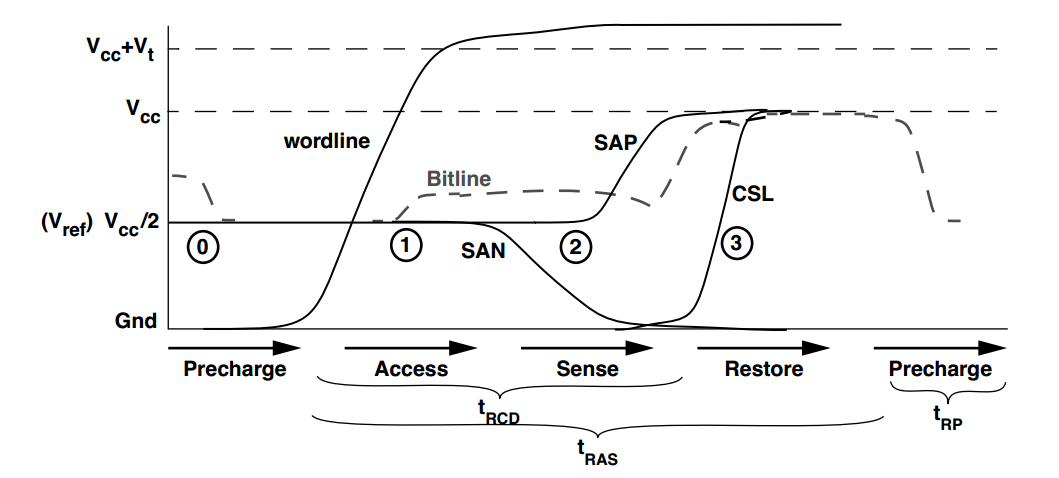
Restore：

在完成 Sense 阶段的操作后，Bitline 线处于稳定的逻辑 1 电压 Vcc，此时 Bitline 会对 Storage Capacitor 进行充电。经过特定的时间后，Storage Capacitor 的电荷就可以恢复到读取操作前的状态。

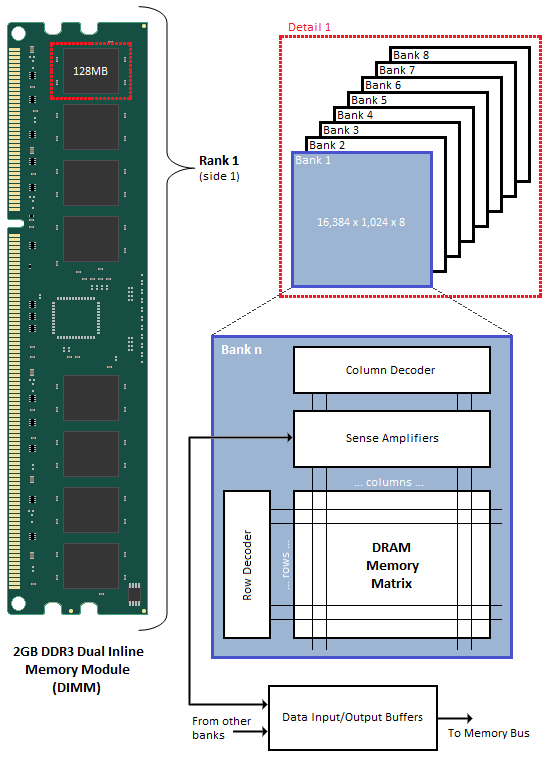


**DRAM的读时序：**

从地址选定到数据输出有一定的延迟



DRAM存储矩阵的结构：



再生放大电路位于列线上。

一个DRAM Array只能一次取出一个比特的数据，一个DRAM 颗粒可以取出一般2，4，8，16位的都有，这是怎么做到的呢？方法就是在一个DRAM颗粒中封装多个DRAM array，让他们同时接受行地址，同时接受列地址，同时给出一个位的数据，如果有8个dram array，就可以做成一个8位的DRAM颗粒了。

由于再生放大电路位于列线上，因此行线响应比列线慢，所以尽量访问同一行的数据

**DRAM的数据刷新：**

由于存储单元的访问是随机的，有可能某些存储单元长期得不到访问，不进行存储器的读/写操作，其存储单元内的原信息将会慢慢消失，为此，必须采用定时刷新的方法，它规定在一定的时间内，对动态RAM的全部基本单元电路必作一次刷新，一般取2ms，即刷新周期（再生周期）。

* 集中刷新

在刷新周期内，对全部存储单元集中一段时间进行刷新（逐行进行），此时必须停止读写操作。

* 异步刷新

在刷新周期内对所有行各刷新一次，即每隔（刷新周期/总行数）us，刷新一次。能充分利用刷新周期，提高刷新频率。

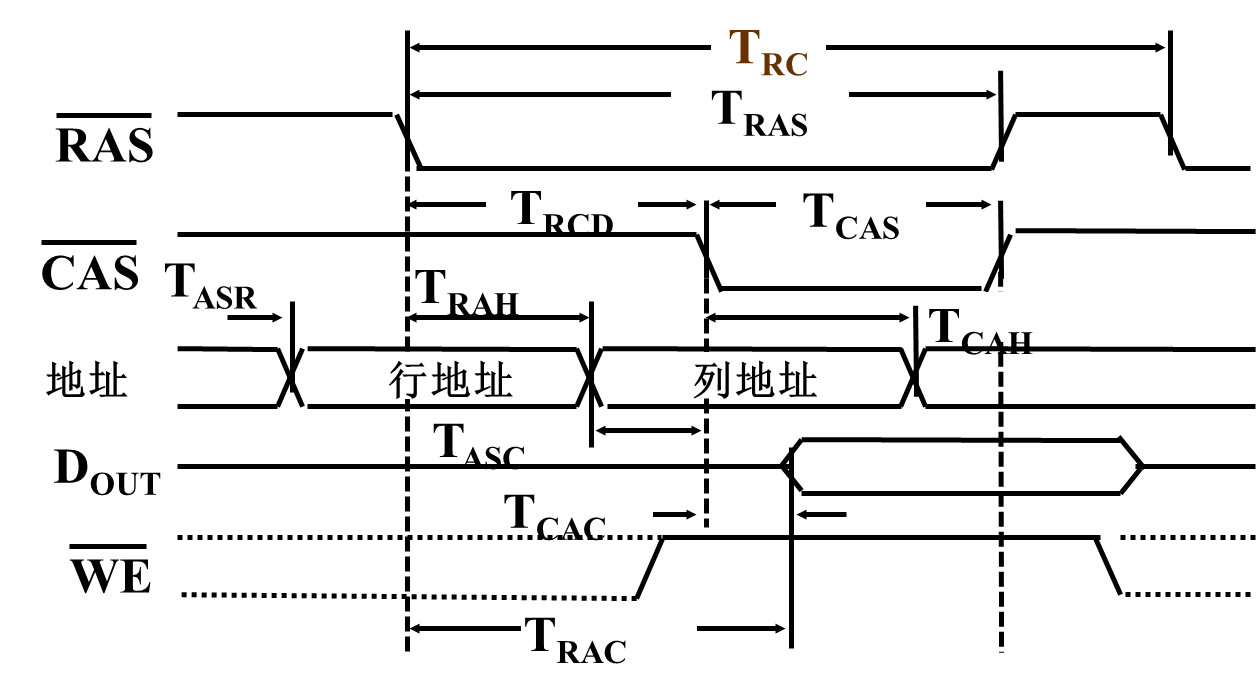
DRAM与SRAM的对比：

* 与SRAM相比，DRAM所用的mos管少，占硅面积小，因而功耗小，集成度高
* 但是因为采用了电容存储电荷的原理来寄存信息，会发生漏电现象，所以要保持状态不变，需要定时刷新，因为读操作会使得状态发生改变，故需要读后再生。且速度比SRAM慢。
* 但是由于其功耗小，集成度高，被广泛应用于计算机中。
* 需要有专门的控制器与CPU连接

**DRAM芯片：**

|  |  |
| --- | --- |
|  | 16个引脚：  7根地址线A6～A0  1根数据输入线DIN  1根数据输出线DOUT  行地址选通RAS\*  列地址选通CAS\*  读写控制WE\*  存储容量为214×1=16K×1 |
|  | 16个引脚：  8根地址线A7～A0  1根数据输入线DIN  1根数据输出线DOUT  行地址选通RAS\*  列地址选通CAS\*  读写控制WE\*  存储容量为216×1=64K×1 |

DRAM的存储地址需要分两批传送



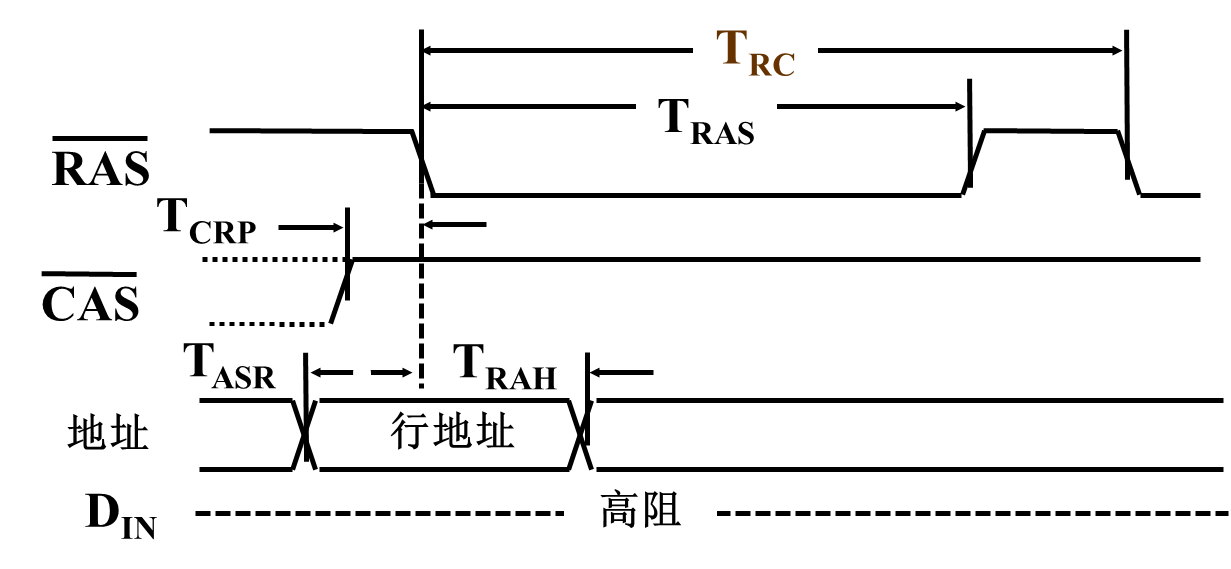
行地址选通信号RAS\*有效，开始传送行地址

随后，列地址选通信号CAS\*有效，传送列地址，CAS\*相当于片选信号

读写信号WE\*读有效

数据从DOUT引脚输出

**DRAM芯片的刷新：**



**采用“仅行地址有效”方法刷新**

**行地址选通RAS\*有效，传送行地址**

**列地址选通CAS\*无效，没有列地址**

**芯片内部实现一行存储单元的刷新**

**没有数据输入输出**

**存储系统中所有芯片同时进行刷新**

**DRAM必须每隔固定时间就刷新**

# 紫外线擦除可编程ROM（EPROM）

**紫外线擦除可编程ROM（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

|  |  |
| --- | --- |
|  | 24个引脚：  11根地址线A10～A0  8根数据线DO7～DO0  片选/编程CE\*/PGM  读写OE\*  编程电压VPP  存储容量为211×1=2K×8 |

# 电可擦可编程只读存储器

**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（英语：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**

**浮栅场效应管：**

在传统的MOS管控制栅下插入一层多晶硅浮栅，浮栅周围的氧化层与绝缘层将其与各电极相互隔离，这些氧化物的电阻非常高，而且电子从浮栅的导带向周围氧化物导电带的移动需要克服较高的势叠，因此，浮栅中的电子泄漏速度很慢，在非热平衡的亚稳态下可保持数十年。

**浮栅场效应管简化示意图及符号**

**写入数据（编程）：**

通常利用F-N隧道效应（Fowler-Nordheim tunneling）对EEPROM存储单元进行“擦除”或“写入”操作，简单地说，即FLOTOX管的控制栅极与漏极在强电场的作用下（正向或负向），浮栅中的电子获得足够的能量后，穿过二氧化硅层的禁带到达导带，这样电子可自由向衬底移动。

对存储单元进行“编程”操作，就是将电子注入到浮栅中的过程，记作**逻辑“0”**。

进行存储单元的“编程”操作时，源线（Sources Lines, SL）与位线（Bit Lines, BL）均为低电平，而控制栅线（CL）为高电平，当对应存储单元的选通管打开时（Word Line, WL=VPP），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，记作**逻辑“1”**。

当进行存储单元的“擦除”操作时，位线为高电平，而源线悬空且控制线CL为低电平，当对应存储单元的选通管打开时（SG=VPP），浮栅中的电子通过隧道效应释放掉，如下图所示：

**读操作：**

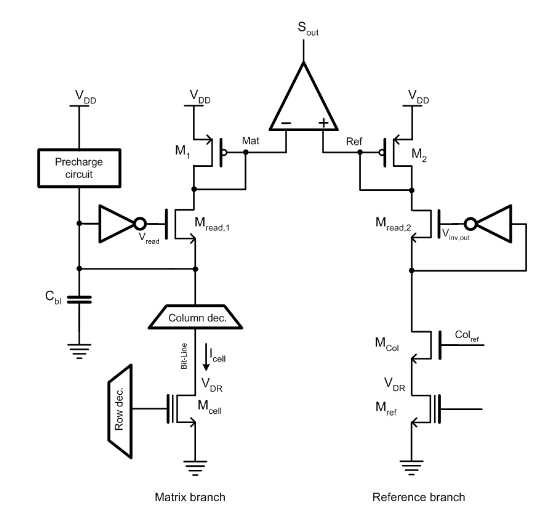
擦出和写入后，控制栅极的开启电压发生变化，利用这个特性，读出数据

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不导通，位线为1。否则位线为0。

擦除----FG不带电子----逻辑“1”----mos管容易导通----位线为0

写入（编程）----FG带电子----逻辑“0” ----mos管难导通----位线为1

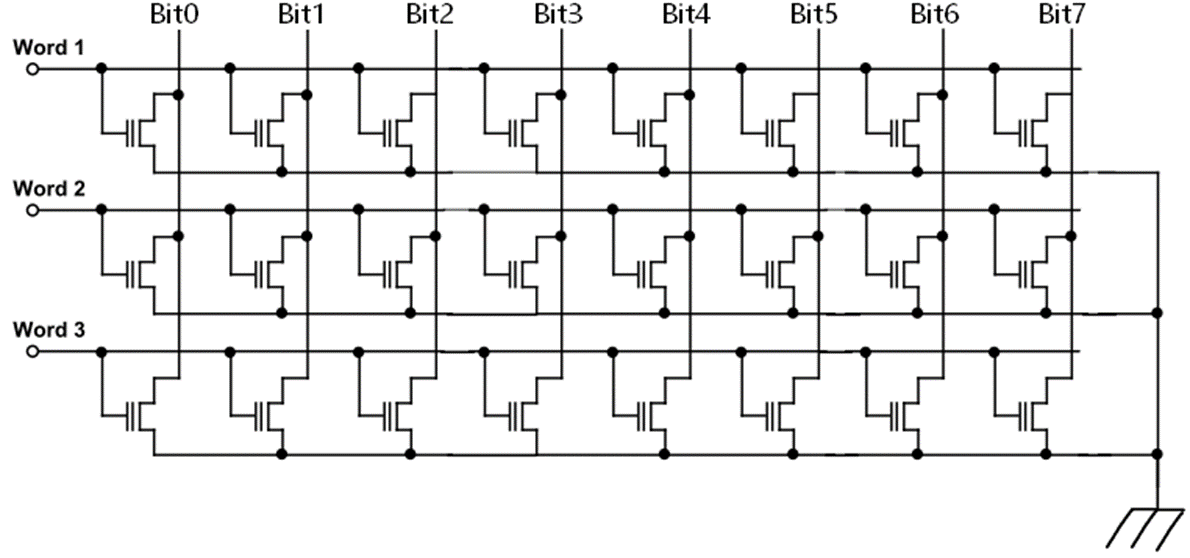
位线上的电压，不能直接输出到数据总线上。为了正确的读取数据，并与逻辑电平匹配，还需要差分感知放大电路和输出缓冲电路，差分感知放大电路或输出缓冲电路中有反相器，所以数据总线上的数据，与位线上的数据相反。



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，导通电流大----位线为1----数据总线位:0

**NOR型Flash，EEPROM**



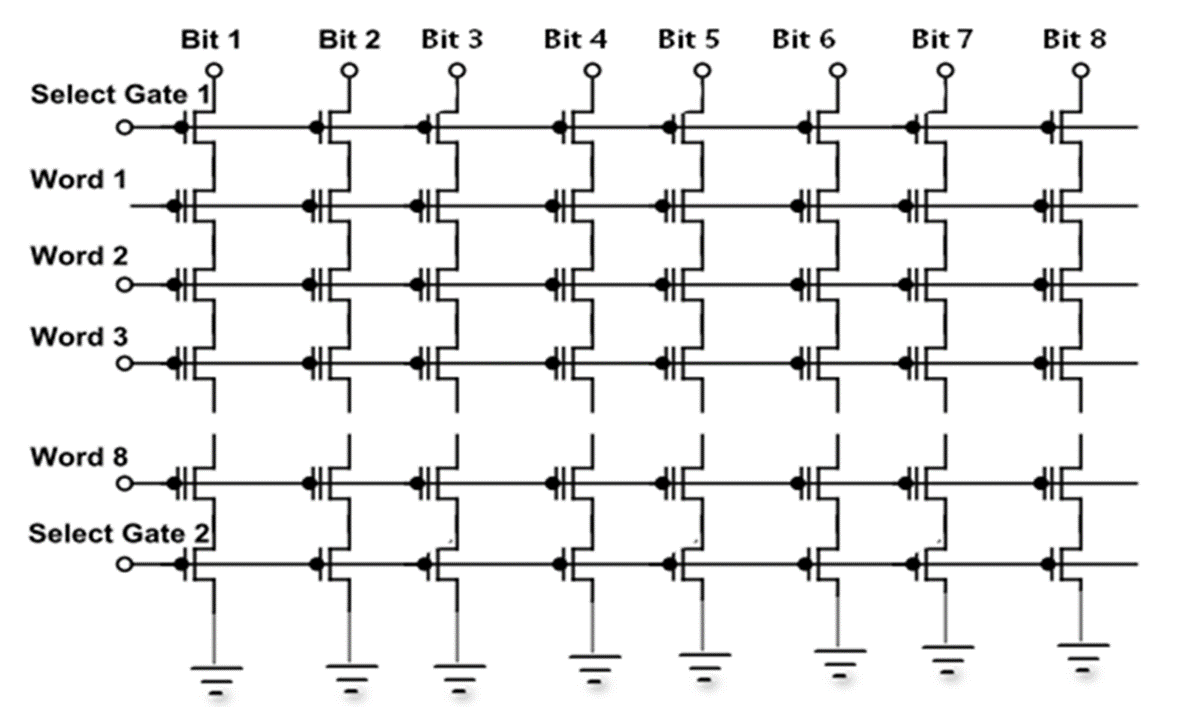
每个Bit Line下的基本存储单元是并联的，当某个Word Line被选中后，就可以实现对该Word的读取，也就是可以实现位读取（即Random access），且具有较高的读取速率。

NOR  FLASH的中的N是NOT，含义是取反，擦除后，FG不带电，数据总线读出1；写入后，FG带电，数据总线读出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。

**缺点：**

* 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。

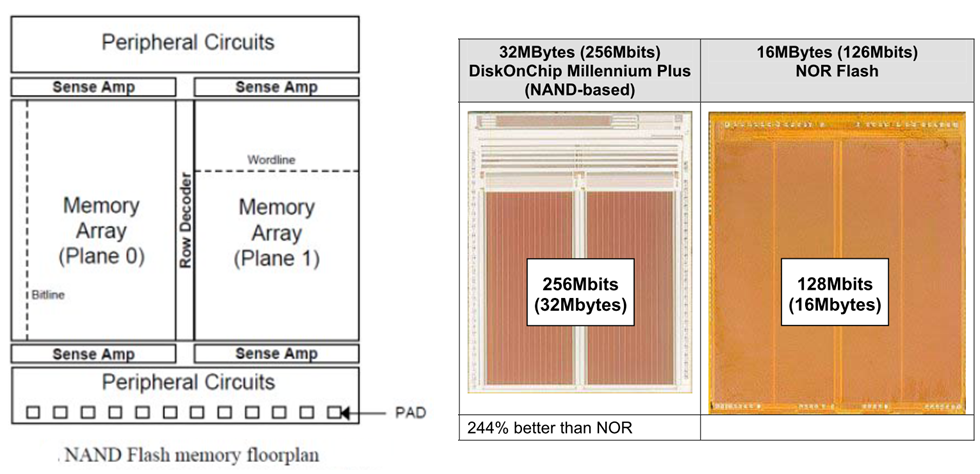
NAND无法实现位读取（即Random access），程序代码也就无法在NAND上运行。

当需要读取某个Page时，FLASH 控制器就不在这个Page的Word Line施加电压，而对其他所有Page的Word Line施加电压（电压值不能改变Floating Gate中电荷数量），让这些Page的所有基本存储单元的D和S导通，而我们要读取的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，有电荷时，Bit Line为‘1’，数据总线读出0，无电荷Bit Line为‘0’，总线读出1，实现了Page数据的读出，

**特点：**

* 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**



# TM4C控制器中的存储器与地址编排、SRAM的bitband，ROM函数的使用，FLASH的组织和操作，EEPROM的介绍与使用

**TM4C1294的存储器：**

* **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

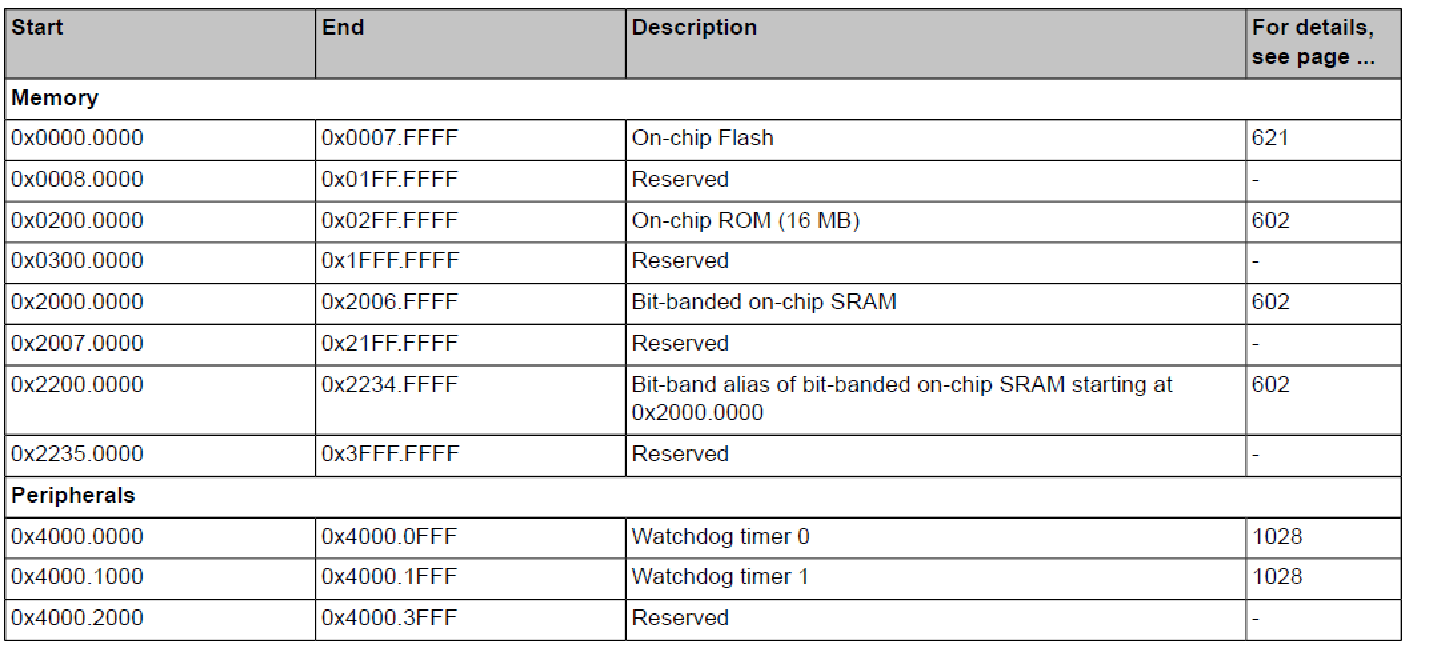
存储冗余校验功能

* **1024 KB Flash memor**y

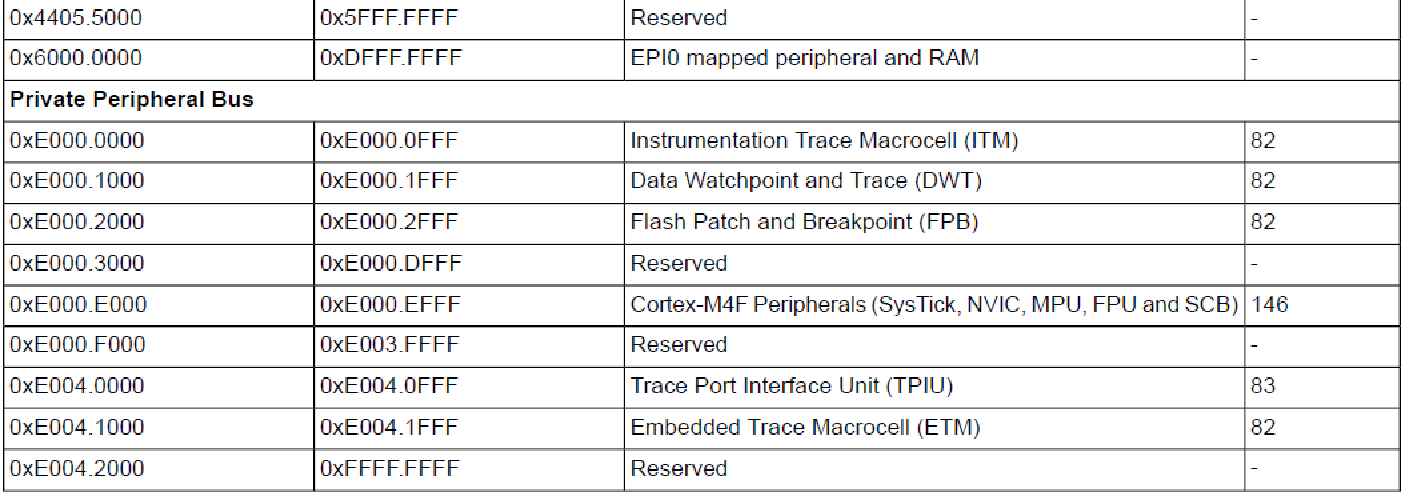
四个 256 bits预取寄存器

* **6KB EEPROM**

单个存储单元50万次写入

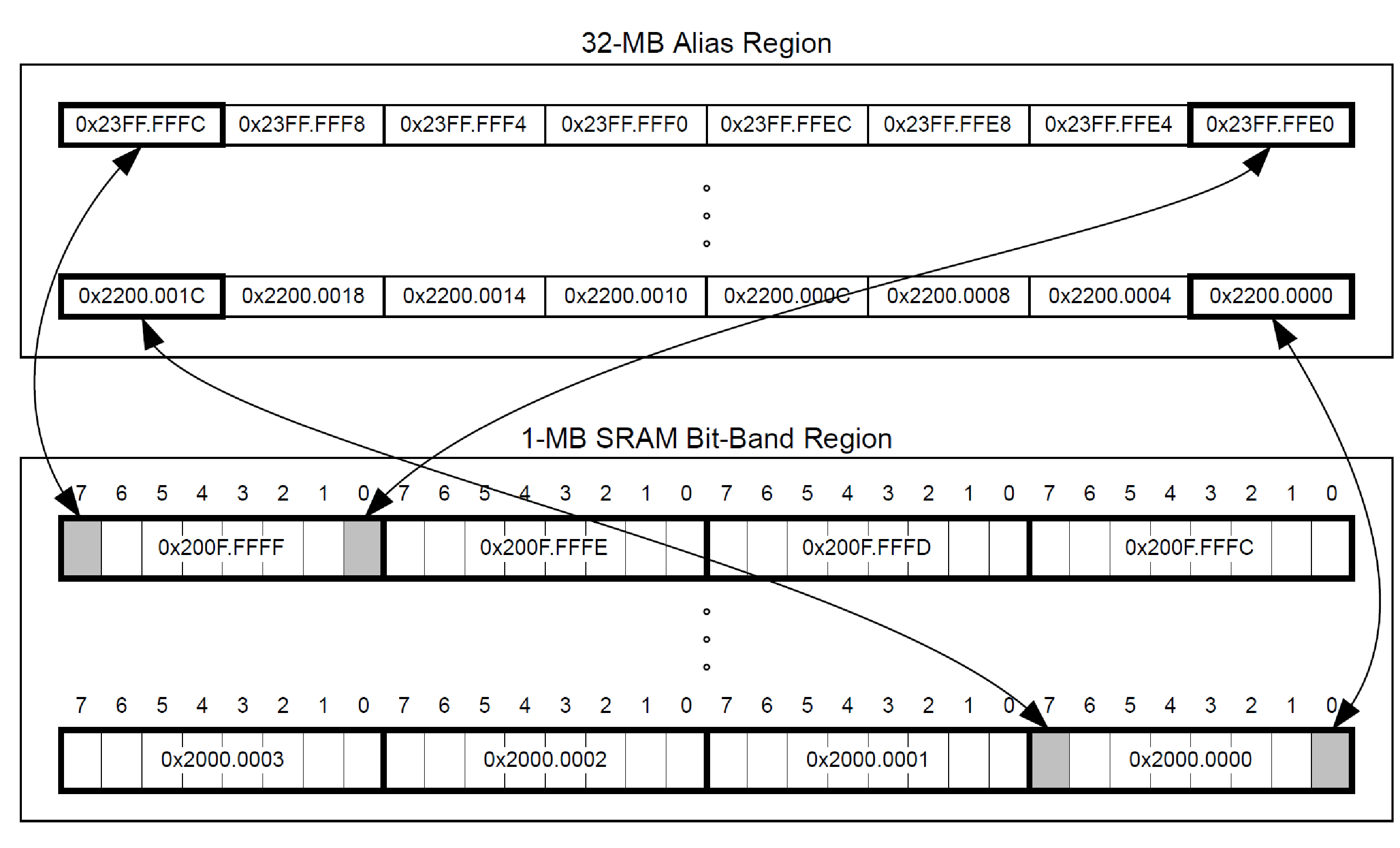


……



**Bit-band：提供了快速读写一个地址空间中某个位的方法:**

SRAM起始地址0x2000.0000，Bit-band 映射的起始地址0x2200.0000



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

库函数中提供了宏定义，用于方便的计算想要修改的位的映射地址：

修改整个字：



修改字中的某一位：



**ROM函数：**

存储Boot Loader and vector table

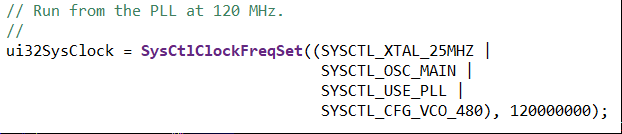
存储外设驱动程序

存储加密标准表

存储冗余校验功能

使用方法：

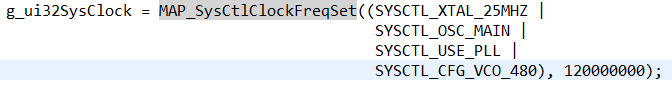
将函数保持在Flash中：



使用ROM中余先存储好的函数：

#include "driverlib/rom.h"

#include "driverlib/rom\_map.h"

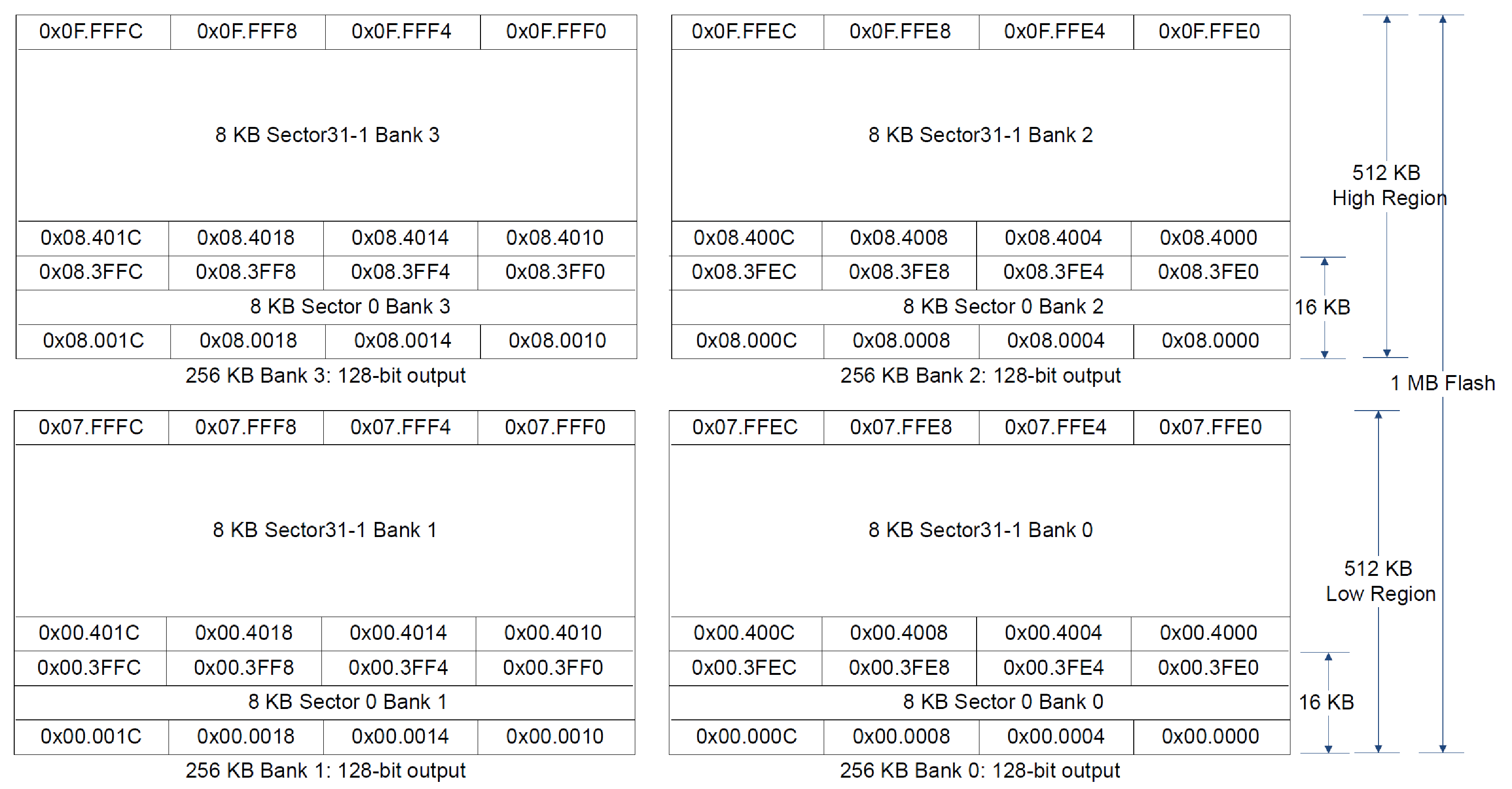


**Flash memory：**

4个bank，双向交错并联

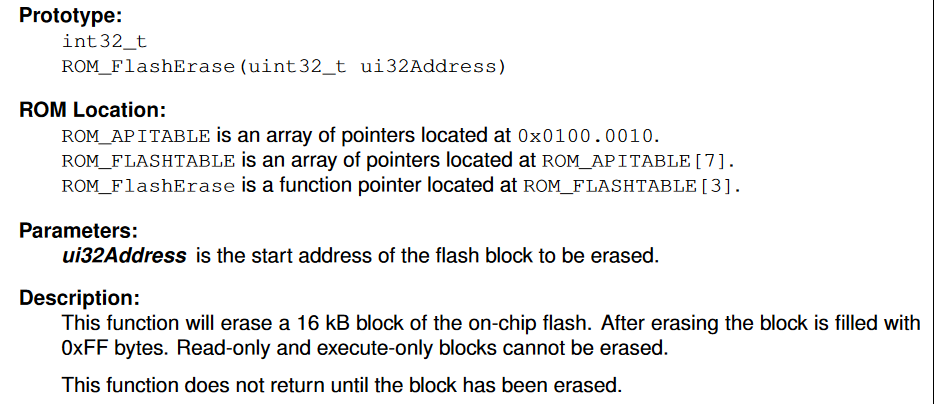
16k擦除

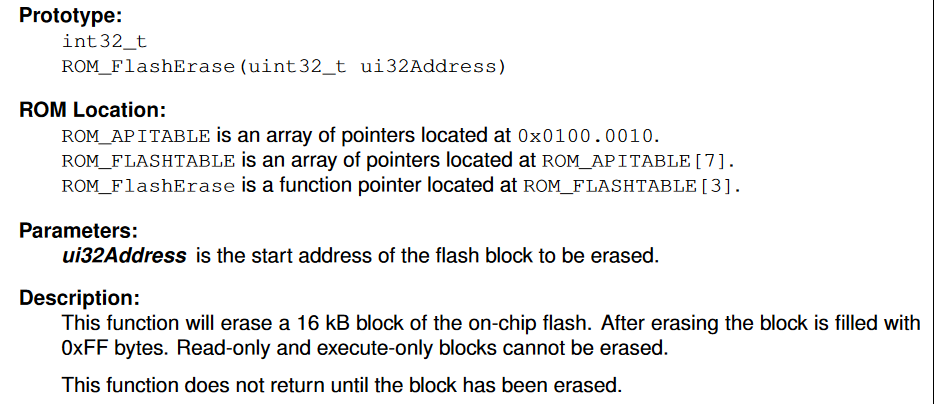
32位存储结构，4个256 bit 的预取寄存器



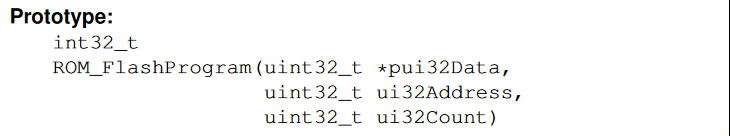
提供了专用的擦除函数和编程函数：

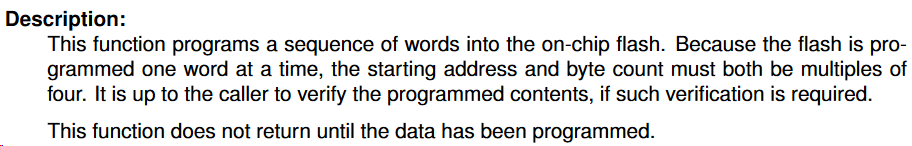
擦除函数：



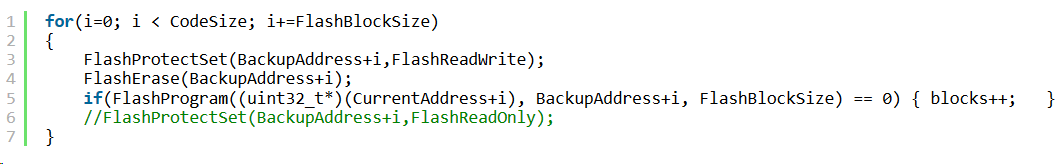


编程函数：





编程示例：



EEPROM：

用于在程序运行过程中，保存程序的重要配置数据。即使程序重新烧写，FLASH被擦除，这些数据也不会丢失。

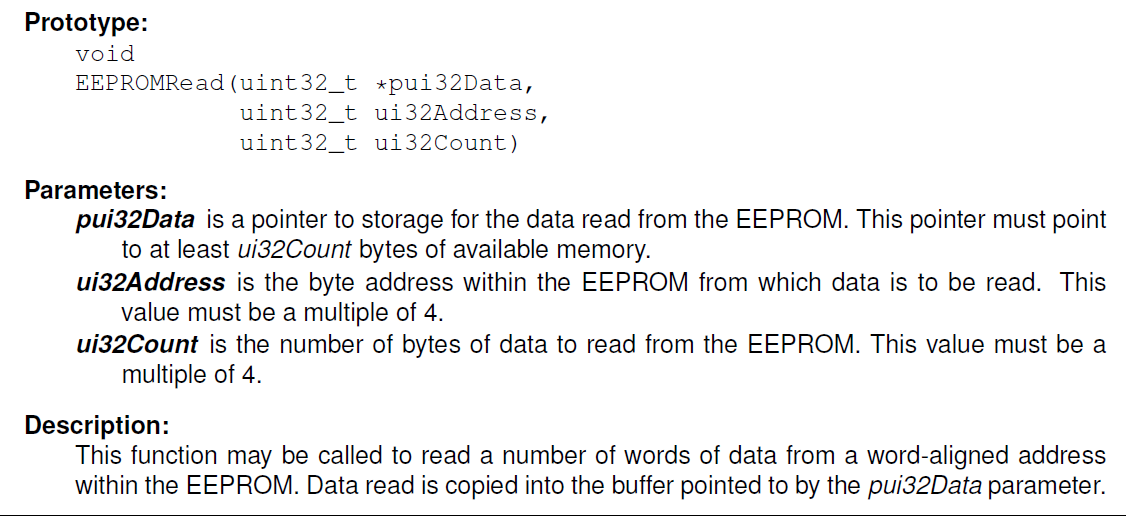
共有1536个32-bit words

分为96 blocks，每个block有16 words (64 bytes)。每个block可以单独设置保护

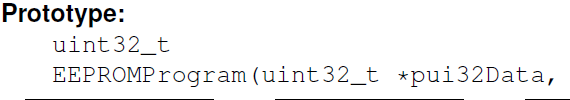
随机读写，8/16/32 bits读，32bits写

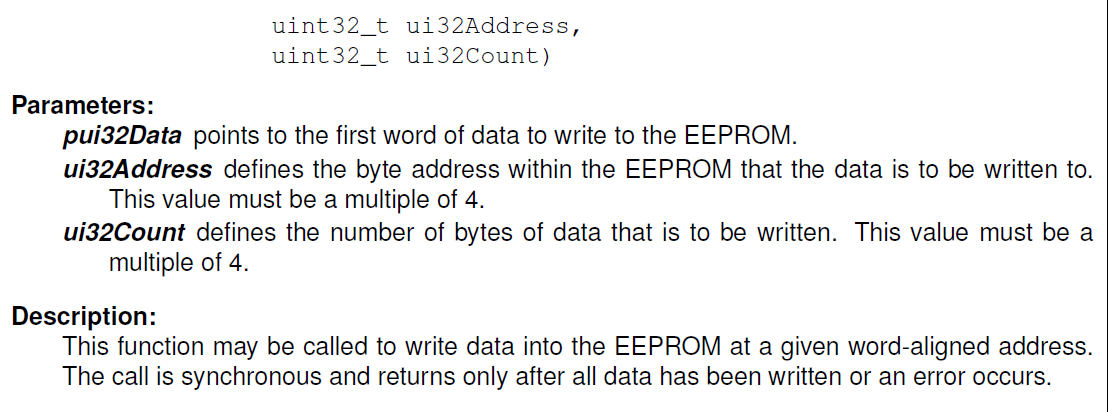
没有主存地址，EEPROM的存储单元，单独从0x000开始编址，库函数提供了专用的读写函数：

读函数：



写函数：





示例：

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));

# 小端模式，大端模式

**字节顺序**

**定义：**

字节顺序，又称端序或尾序（Endianness），在计算机科学领域中，指存储器中或在数字通信链路中，组成多字节的字的字节的排列顺序。

**分类：**

1. 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。
2. 2) 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

