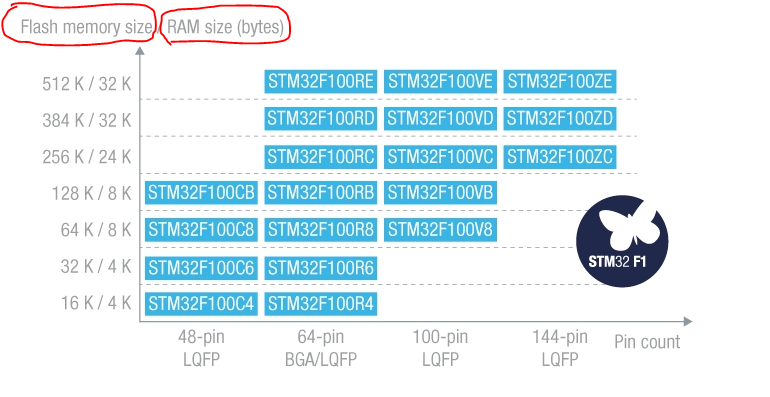
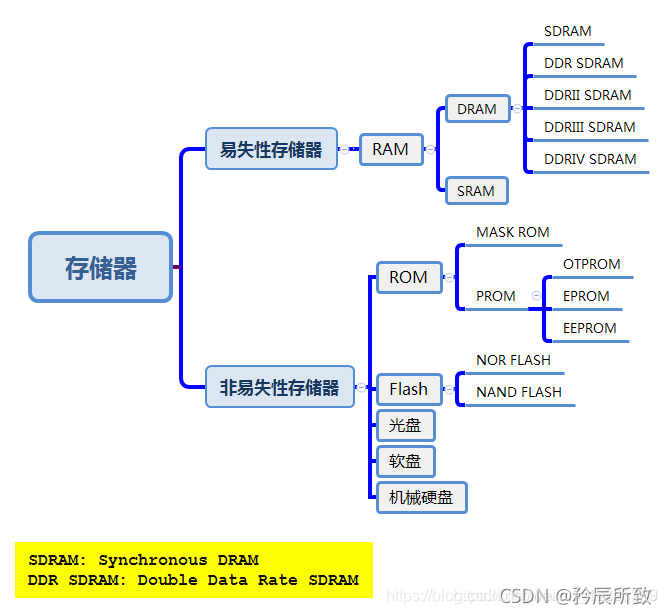
# stm32内存资源

## 前言

STM32的数据在物理上分别储存在RAM和Flash中，STM32芯片的内存有两个直观的指标RAM大小，FLASH大小，比如STM32F10X系列（其他系列相同）。

RAM可读可写，掉电清零。Flash可读可写，但是读写时间很长，能掉电储存，并且一般空间比RAM大很多。





## FLASH基本概念

FLASH属于非易失性存储器：

扩展一点说，FLASH又称为闪存，不仅具备电子可擦除可编程(EEPROM)的性能，还不会断电丢失数据同时可以快速读取数据，U盘和MP3里用的就是这种存储器。在以前的嵌入式芯片中，存储设备一直使用ROM(EPROM)，随着技术的进步，现在嵌入式中基本都是FLASH，用作存储Bootloader以及操作系统或者程序代码或者直接当硬盘使用(U盘)。

然后Flash主要有两种NOR Flash和NADN Flash。（对于这两者的区别，下面的话供参考，因为这些介绍都是基于早些年的技术了）

NOR Flash的读取和我们常见的SDRAM的读取是一样，用户可以直接运行装载在NOR FLASH里面的代码，这样可以减少SRAM的容量从而节约了成本。

NAND Flash没有采取内存的随机读取技术，它的读取是以一次读取一块的形式来进行的，通常是一次读取512个字节，采用这种技术的Flash比较廉价。用户不能直接运行NAND Flash上的代码，因此好多使用NAND Flash的开发板除了使用NAND Flah以外，还作上了一块小的NOR Flash来运行启动代码。

在STM32单片机中的FLASH为NOR FLASH。

Flash相对容量大，掉电数据不丢失，主要用来存储代码，以及一些掉电不丢失的用户数据。

## RAM基本概念

RAM属于易失性存储器：

RAM随机存储器（Random Access Memory）表示既可以从中读取数据，也可以写入数据。当机器电源关闭时，存于其中的数据就会丢失。比如电脑的内存条。

RAM有两大类，一种称为静态RAM(Static RAM/SRAM)，SRAM速度非常快，是目前读写最快的存储设备了，但是它也非常昂贵，所以只在要求很苛刻的地方使用，譬如CPU的一级缓冲，二级缓冲。另一种称为动态RAM(Dynamic RAM/DRAM)，DRAM保留数据的时间很短，速度也比SRAM慢，不过它还是比任何的ROM都要快，但从价格上来说DRAM相比SRAM要便宜很多，计算机内存就是DRAM的。

DRAM分为很多种，常见的主要有FPRAM/FastPage、EDORAM、SDRAM、DDR RAM、RDRAM、SGRAM以及WRAM等，这里介绍其中的一种DDR RAM。

DDR RAM(Date-Rate RAM)也称作DDR SDRAM，这种改进型的RAM和SDRAM是基本一样的，不同之处在于它可以在一个时钟读写两次数据，这样就使得数据传输速度加倍了。这是目前电脑中用得最多的内存，而且它有着成本优势，事实上击败了Intel的另外一种内存标准-Rambus DRAM。在很多高端的显卡上，也配备了高速DDR RAM来提高带宽，这可以大幅度提高3D加速卡的像素渲染能力。

为什么需要RAM，因为相对FlASH而言，RAM的速度快很多，所有数据在FLASH里面读取太慢了，为了加快速度，就把一些需要和CPU交换的数据读到RAM里来执行（注意这里不是全部数据，只是一部分需要的数据，这个在后面介绍STM32的内存管理中会提到）。

STM32单片机内部的RAM为SRAM。

RAM相对容量小，速度快，掉电数据丢失，其作用是用来存取各种动态的输入输出数据、中间计算结果以及与外部存储器交换的数据和暂存数据。

## STM32的Flash

STM32的Flash，严格说，应该是Flash模块。该Flash模块包括：Flash主存储区（Main memory）、Flash信息区（Informationblock），以及Flash存储接口寄存器区（Flash memory interface）。三个组成部分分别在0x0000 0000——0xFFFF FFFF不同的区域，如下表所示。



STM32的闪存模块由：主存储器、信息块和闪存储器块3部分组成。

主存储器，该部分用来存放代码和数据常数（如加const类型的数据）。对于大容量产品，其被划分为256页，每页2K，注意，小容量和中容量产品则每页只有1K字节。主存储起的起始地址为0X08000000，B0、B1都接GND的时候，就从0X08000000开始运行代码。

信息块，该部分分为2个部分，其中启动程序代码，是用来存储ST自带的启动程序，用于串口下载，当B0接3.3V，B1接GND时，运行的就这部分代码，用户选择字节，则一般用于配置保护等功能。

闪存存储器块（Flash存储接口寄存器区），该部分用于控制闪存储器读取等，是整个闪存储器的控制机构。

对于主存储器和信息块的写入有内嵌的闪存编程管理；编程与擦除的高压由内部产生。

在执行闪存写操作时，任何对闪存的读操作都会锁定总线，在写完成后才能正确进行，在进行读取或擦除操作时，不能进行代码或者数据的读取操作。

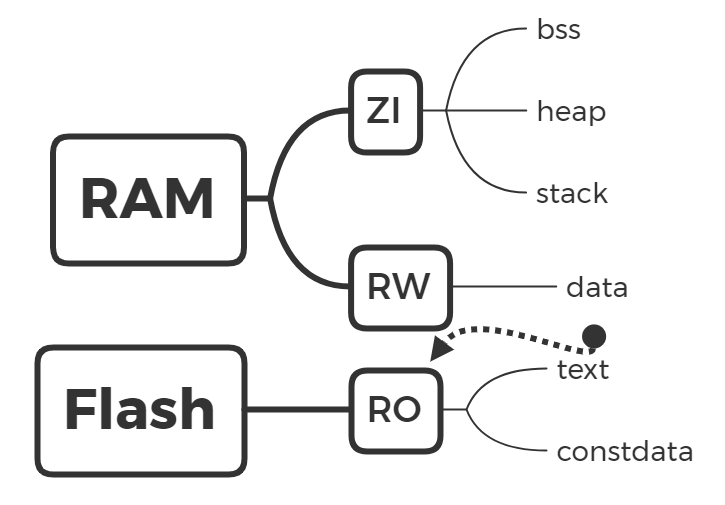
## STM32的SRAM

不同类型的STM32单片机的SRAM大小是不一样的，但起始地址都是0x2000 0000，终止地址都是0x2000 0000+其固定的容量大小。

SRAM的理解比较简单，其作用是用来存取各种动态的输入输出数据、中间计算结果以及与外部存储器交换的数据和暂存数据。设备断电后，SRAM中存储的数据就会丢失。

## 内存分配

在关于如何使用RAM和Flash的问题上，STM32的内存又有了６个储存数据段和３种储存属性区的概念。



### flash存储

#### .text（CodeSegment/Text Segment）

Code代码

代码段，储存程序代码。也就是存放CPU执行的机器指令(machineinstructions)。这部分区域的大小在程序运行前就已经确定，并且内存区域通常属于只读(某些架构也允许代码段为可写，即允许修改程序)。

#### .constdata

RO-data:程序定义的常量

储存只读常量。const修饰的常量，不管是在局部还是全局都放在Flash里。所以为了节省RAM，把常量的字符串，数据等用const声明。

### RAM存储

#### .data

数据段，储存已初始化且不为0的全局变量和静态变量（全局静态变量和局部静态变量）。

static声明的变量放在data段。

数据段属于静态内存分配，所以放在RAM里，准确来说，是在程序运行的时候需要在RAM中运行。

#### .BSS

Block Started by Symbol。储存未初始化的，或初始化为0的全局变量和静态变量。

BSS段属于静态内存分配，所以放在RAM里。

#### heap（堆）

堆是用于存放进程运行中被动态分配的内存段。他的大小并不固定，可动态扩张或者缩减，由程序员使用malloc()和free()函数进行分配和释放。当调用malloc等函数分配内存时，新分配的内存就被动态添加到堆上（堆被扩张）；当利用free等函数释放内存时，被释放的内存从堆中被剔除（堆被缩减）。

堆放在RAM里，其可用大小定义在启动文件startup\_stm32fxx.s中。

#### stack（栈）

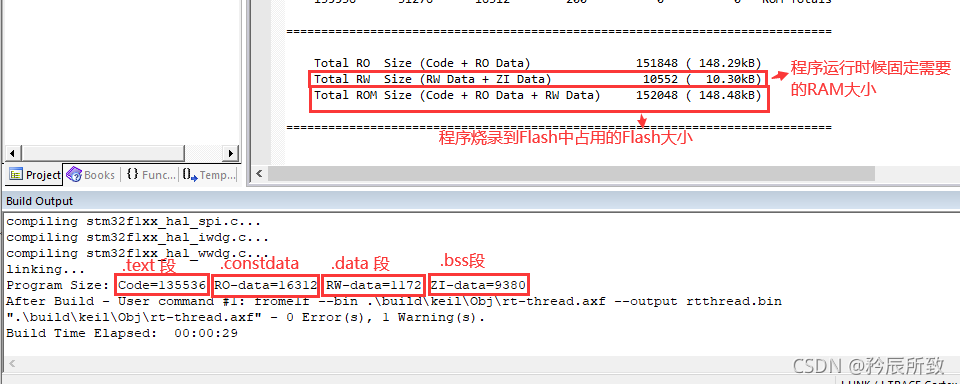
栈又称堆栈，是用户存放程序临时创建的局部变量，由系统自动分配和释放。可存放局部变量、函数的参数和返回值（但不包括static声明的变量，static意味着放在data数据段中）。

除此以外，在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放回栈中。

栈特别方便用来保存/恢复调用现场。

从这个意义上讲，我们可以把堆栈看成一个寄存、交换临时数据的内存区。

栈放在RAM里，其大小定义在启动文件startup\_stm32fxx.s中。



## 参考资料

版权声明：本文参考了其他CSDN博主的文章，遵循CC 4.0 BY-SA版权协议，现附上原文出处链接及本声明。

版权声明：本文为CSDN博主「矜辰所致」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_42328389/article/details/120622384

————————————————

版权声明：本文为CSDN博主「行稳方能走远」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/zhuguanlin121/article/details/119799860

————————————————

版权声明：本文为CSDN博主「矜辰所致」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_42328389/article/details/120622384

————————————————

版权声明：本文为CSDN博主「行稳方能走远」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/zhuguanlin121/article/details/119799860