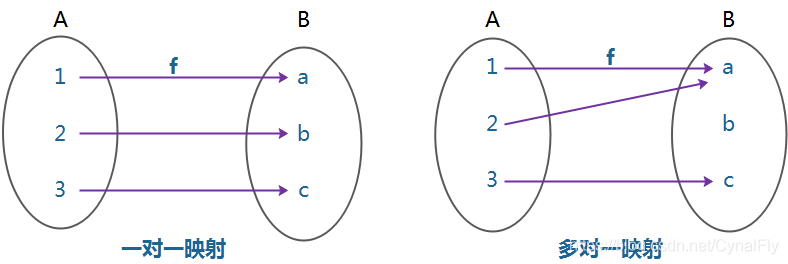
# STM32存储器映射

## 前言

在数学里，映射指两个元素的集之间元素相互“对应”的关系，为名词。



如图所示，两个非空集合A与B间存在着对应关系f，而且对于A中的每一个元素x，B中总有唯一的一个元素y与它对应，就这种对应为从A到B的映射，记作f：A→B。其中，y称为元素x在映射f下的像，记作：y=f(x)。x称为y关于映射f的原像。集合A中所有元素的像的集合称为映射f的值域，记作f(A)。

## 存储器映射

存储器本身不具有地址信息，它的地址是由芯片厂商或用户分配，把内核厂商（也就是ARM公司）定义的这个虚拟空间与芯片厂商（这里是ST）芯片内部外设进行对应，也就是给存储器分配地址的过程称为存储器映射，如果再分配一个地址就叫重映射

因为stm32的地址线是32位，也就是2的32次方，所以32根地址线可以传输4G个地址信号，每个地址信号访问一个字节，4G地址信号则可以访问4G个字节，对应4G的虚拟存储空间。所以理论上的可访问范围为4G。4个G的地址这么大，用不完没关系，可以保留。

STM32是指令和数据共用相同的地址空间，即程序存储器、数据存储器、寄存器和I/O端口在一个4GB的线性地址空间内，也就是从0x0000 0000~0xFFFF FFFF这一块空间，分为8个主要块，每个块为512MB。

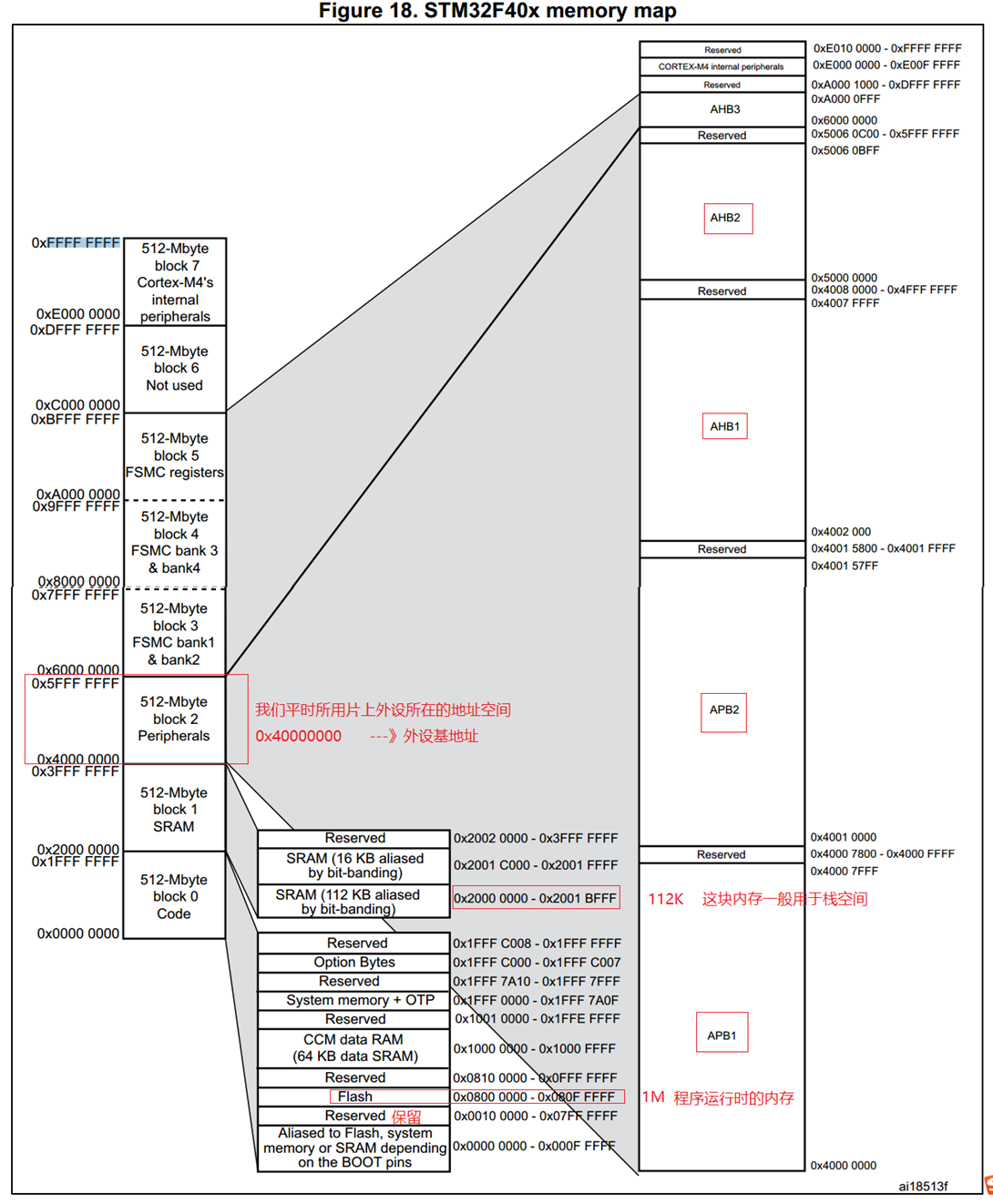
4G的地址空间即地址编码的范围，编码就是对每一个程序存储、数据存储器、寄存器和I/O端口分配唯一的地址编码，即“编址”或“地址映射”。

数据字节以小端格式存放在存储器中，一个字的最低地址字节被认为是该字的最低有效字节，而最高地址字节是最高有效字节。

图示

描述已自动生成

STM32存储器映射图



STM32F40X存储器映射图

### Block 0代码区

Aliased to flash ,system memory or SRAM depending on the Boot pins

开始运行，BOOT1、BOOT0这两个引脚的电平值选择0X0000 0000–0X001F FFFF映射到不同的存储器上，通过BOOT引脚选择启动模式。

Flash:用于保存数据的区域

CCM data RAM:Code区域是用I-Code和D-Code访问，作用是为了加快数据处理速度。

system memory :STM32在出厂时，已经固化了一段程序在System memory（medium-density devices的地址为：0x1FFF\_F000，大小为2KB）存储器中。这段程序就是一个固定好的，并且没法修改的Boot Loader

Options Bytes :可以按照用户的需要进行配置（如配置看门狗为硬件实现还是软件实现）；

### Block 1

SRAM运行时临时存放代码的地方。不同类型的STM32单片机的SRAM大小是不一样的，但是他们的起始地址都是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，用于程序运行的堆栈开销。设备断电后，SRAM中存储的数据就会丢失。

### Block 2

Block2 用于设计片内外设，根据总线速度的不同，Block2被分为了APB和AHB。在上图所示的stm32f40x的映射框图中可以看到，APB分为APB1和APB2,AHB分为AHB1和AHB2,AHB3(不在Block2的映射范围)

### Block3

FSMC Bank 的四个分区的bank1~bank2

### Block4

FSMC Bank 的四个分区的bank3~bank4

### Block5

FSMC Bank 控制寄存器

### Block6

Reserved（保留）

### Block7

internal peripherals，cortex-m4 内核内部外设地址。

|  |  |  |
| --- | --- | --- |
| 序号 | 用途 | 地址范围 |
| Block0 | Code | 0x0000 0000~0x1FFF FFFF(512MB) |
| Block1 | SRAM | 0x2000 0000~0x3FFF FFFF(512MB) |
| Block2 | 片上外设 | 0x4000 0000~0x5FFF FFFF(512MB) |
| Block3 | FSMC的bank1~bank2 | 0x6000 0000~0x7FFF FFFF(512MB) |
| Block4 | FSMC的bank3~bank4 | 0x8000 0000~0x9FFF FFFF(512MB) |
| Block5 | FSMC寄存器 | 0xA000 0000~0xCFFF FFFF(512MB) |
| Block6 | 没有使用 | 0xD000 0000~0xDFFF FFFF(512MB) |
| Block7 | Cortex-M3内部外设 | 0xE000 0000~0xFFFF FFFF(512MB) |

STM32存储器映射

## 寄存器映射

给已经分配好地址的有特定功能的内存单元起别名的过程就叫寄存器映射，这个别名叫寄存器。

以存储器Block2这块区域为例，也就是地址从0x4000000—0x5FFFFFF这块区域，设计的是片上外设，它们以四个字节为一个单元，共32bit，每一个单元对应不同的功能，当我们控制这些单元时就可以驱动外设工作。可以找到每个单元的起始地址，然后通过C语言指针的操作方式来访问这些单元，如果每次都是通过这种地址的方式来访问，不仅不好记忆还容易出错，聪明的工程师就根据每个单元功能的不同，以功能为名给这个内存单元取一个别名，这个别名就是我们经常说的寄存器，这个给已经分配好地址的有特定功能的内存单元取别名的过程就叫寄存器映射。

### 对寄存器进行操作

我们知道寄存器就是一些有特定功能的内存单元，所以要访问STM32寄存器也就是操作STM32的内存单元，根据C语言指针的特点，可以使用指针来操作STM32的内存单元。假如我们要让STM32的GPIOC的第0管脚输出低电平，我们怎么使用C语言来处理？首先我们要知道GPIOC挂接在哪个总线上，需要知道其基地址，然后基地址加偏移地址计算寄存器地址，指针指向这个地址进行操作。

#### 方法一

表格

描述已自动生成

图示

低可信度描述已自动生成

例如：GPIOB的端口数据寄存器IDR的地址是0x40010C0C,并且IDR寄存器是32位的。那么我们可以用如下代码对寄存器进行操作：

\*(unsigned int\*)(0x40020410)=0xFFFFFFFF; //GPIOB端口数据寄存器IDR全为高电平

#### 方法二

在计算机中，一个字节（Byte）就有一个地址，一个字节有八个位（bit）。

也就是说0x4001 0C0C这个地址在存储器中，其实只是一个字节的地址。计算机能向四个字节写数据，是因为强制类型转换成了 unsigned int ，而 unsigned int 就是四个字节，0x4001 0C0C是只是这四个字节的首地址。

计算机知道首地址，然后知道要写几个字节，就能向4个字节里面写数据了，也就是写32位的数据。这其实就是数据类型的含义，即这种数据类型占用几个字节。

定义GPIO外设结构体，因为结构体成员在内存中是连续的，这种形式与寄存器组非常类似，所以用结构体能够很好的管理寄存器：

typedef struct

{

\_\_IO uint32\_t CRL;

\_\_IO uint32\_t CRH;

\_\_IO uint32\_t IDR;

\_\_IO uint32\_t ODR;

\_\_IO uint32\_t BSRR;

\_\_IO uint32\_t BRR;

\_\_IO uint32\_t LCKR;

} GPIO\_TypeDef;

#define \_\_IO volatile

定义GPIOA结构体指针，因为单单定义GPIO外设结构体，并不能确定其内存地址，因此用指针将其绑定到GPIOA外设基地址：

#define GPIOA ((GPIO\_TypeDef \*) GPIOA\_BASE)

/\*\*\*\*\*映射访问\*\*\*\*/

GPIOE->ODR=0<<5;

delay\_ms(300);

GPIOE->ODR=1<<5;

## STM32存储空间

芯片能访问的存储空间有多大，是由芯片内CPU的地址总线的数量决来定的，STM32芯片内部的地址总线为32根。

1根地址线：可以传输的地址为0和1的，那么理论上就可以访问2个字节。

2根地址线：可以传输地址为00、01、10、11，理论上可以访问4个字节。

3根地址线：可以传输的地址为000、001、010、011、100、101、110、111，理论上可以访问8个字节。

32根地址线：可以产生00000000 00000000 00000000 00000000 — 11111111 11111111 11111111 11111111的2^32个地址，范围刚好为4G

STM32的32根地址线，理论上可以访问4G字节的存储器空间。

STM32地址是从0x00000000到0xFFFFFFFF，这就是4GB的存储空间。

但是STM32真的有4GB的存储空间吗？答案当然不是，我们的PC电脑也才4GB的内存。一个小小的单片机怎么可能有4GB的存储空间！这个4GB的是STM32理论分配的地址空间。也就是说实际上并不是有这么大的存储单元。其中有很多预留的地址，这些地址并没有给他分配存储单元。

所有的存储器都是与地址线连着的，但是实际上如果你只接了一个10M的存储器，而且是从0地址开始映射的，那么32根地址线所产生的0~10M的地址信号其实才是有意义的，因为这些地址信号才有对应真实的存储器，而所产生的10M以上地址信号其实并无意义，因为并不对应真实的存储器。

举个例子，李云龙建立了三个营，只有三营是全员满编制的，一营和二营只有番号，人员待补充。所以一营二营是不能用的。

STM32中的32是32根地址线的意思吗？

答：不是，STM32的32不是32根地址线的意思，而是表示MCU芯片内部CPU在处理数据时，每次可以处理的数据位宽为32个bit。正是由于这个原因，STM32内部的寄存器大小都是32位的，刚好等于位宽。

某个芯片是32位的，但是它的地址线完全可以只有16根、或者8根，对于STM32来说，刚好碰巧的是，CPU能够处理的数据位宽与地址线数量恰好都是32。

## 存储器的重映射

通常系统启动都是从0地址处开始，但是为了支持不同的存储介质，不同的存储介质被分配了一个非0地址区域。这就是为什么要进行重映射。

因此重映射主要发生在两种情况下，一是系统启动的过程中；二是如果中途遇到需要在不同的存储器之间进行切换的时候也需要进行重映射

我们一般的单片机自举（启动）单片机地址，都是从0开始运行的，STM32启动需要重映射地址，F4xx的0X0000 0000~0x001F FFFF地址映射了到什么存储器上，那么就从该存储器上读取指令，开始运行。

自举(bootstrap)计算机设备使用硬件加载的程序，用于初始化足够的软件来查找并加载功能完整的操作系统。也用来描述加载自举程序的过程。什么是单片机的自举，单片机的自举就是单片机的启动。

至于说0x00000000-0x001FFFFF到底映射在了什么存储器上，这个要看芯片BOOT1、BOOT0这两个引脚的电平值，说白了就是，通过BOOT1和BOOT0引脚的电平值，可以选择将0x00000000-0x001FFFFF映射到不同的存储器上。

STM32片内的FLASH分成两部分：主存储块、信息块。主存储块(主Flash)用于存储程序，我们写的程序一般存储在这里。信息块又分成两部分：系统存储器(系统FLASH)、选项字节。系统存储器存储用于存放在系统存储器自举模式下的启动程序(BootLoader)，当使用ISP方式加载程序时，就是由这个程序执行。这个区域由芯片厂写入BootLoader，然后锁死，用户是无法改变这个区域的。选项字节存储芯片的配置信息及对主存储块的保护信息。

## 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。

1. STM32 学习记录四：存储器寄存器映射

原文链接：https://blog.csdn.net/wangyanan52121/article/details/126519524

1. STM32的存储器映射中的指针操作

原文链接：https://blog.csdn.net/C1664510416/article/details/127063065

1. stm32单片机基础知识总结（四）——存储器映射

原文链接：https://blog.csdn.net/weixin\_42506096/article/details/123625114

1. STM32学习笔记（2）：存储器映射和存储器重映射（Memory Map and Memory Remap）

原文链接：https://blog.csdn.net/Cacti\_one/article/details/70157261

1. STM32F103的存储器映射&寄存器映射

原文链接：https://blog.csdn.net/sinat\_37826100/article/details/115443141