STM32 之八 在线升级（IAP）超详细图解 及 需要注意的问题解决

ba_wang_mao
于 2020-11-30 19:45:00 发布 2405 收藏 22

分类专栏： IAP STM32F407


IAP 同时被 2 个专栏收录

IAP 是啥

IAP（ In Application Programming）即在应用编程，也就是用户可以使用自己的程序对MCU的中的运行程序进行更新，而无需借助于外部烧写器。其实ST官网也给出了IAP的示例程序，感兴趣的可以直接去官网搜索。

这里有一点需要特殊注意，就是在MCU中，有一个特殊区域被称为 ***System memory***。在这块区域中存放了ST公司自己的bootloader 程序，它是在MCU出厂时，有ST固化到芯片中的，后续不能再更改。其中的 bootloader 程序也可以对MCU进行升级（DFU对芯片的编程应该就是用的这个Bootloader）。而且，芯片不同，BootLoader的功能也是有区别的。ST官网对于这些也是有详

细文档的，后续再写篇文章介绍这一块。下图为部分芯片BootLoader版本及功能

Table 3. Embedded bootloaders

STM32 series	Device		Supported serial peripherals	Bootloader ID		Bootloader (protocol) version
				ID	Memory location	
F0	STM32F05xxx/STM32F030x8 devices		USART1/USART2	0x21	0x1FFFF7A6	USART (V3.1)
	STM32F03xx4/6		USART1	0x10	0x1FFFF7A6	USART (V3.1)
	STM32F030xC		USART1/I2C1	0x52	0x1FFFF796	USART (V3.1) I2C1(V1.0)
	STM32F04xxx		USART1/USART2/ I2C1/ DFU (USB Device FS)	0xA1	0x1FFFF6A6	USART (V3.1) DFU (V2.2) I2C (V1.0)
	STM32F071xx/072xx		USART1/USART2/ I2C1/ DFU (USB Device FS)	0xA1	0x1FFFF6A6	USART (V3.1) DFU (V2.2) I2C (V1.0)
F0	STM32F070x6		USART1/USART2/ DFU (USB Device FS)/I2C1	0xA2	0x1FFFF6A6	USART (V3.1) DFU (V2.2) I2C (V1.0)
	STM32F070xB		USART1/USART2/ DFU (USB Device FS)/I2C1	0xA3	0x1FFFF6A6	USART (V3.1) DFU (V2.2) I2C (V1.0)
	STM32F09xxx		USART1/USART2/ I2C1	0x50	0x1FFFF796	USART (V3.1) I2C (V1.0)
F1	STM32F10xxx	Low-density	USART1	NA	NA	USART (V2.2)
		Medium-density	USART1	NA	NA	USART (V2.2)
		High-density	USART1	NA	NA	USART (V2.2)
		Medium-density value line	USART1	0x10	0x1FFFF7D6	USART (V2.2)
		High-density value line	USART1	0x10	0x1FFFF7D6	USART (V2.2)
	STM32F105xx/107xx		USART1 / USART2 (remapped) / CAN2 (remapped) / DFU (USB Device)	NA	NA	USART (V2.2[1]) CAN (V2.0) DFU(V2.2)
	STM32F10xxx XL-density		USART1/USART2 (remapped)	0x21	0x1FFFF7D6	USART (V3.0)

STM32 MCU启动配置

要实现IAP，首先要了解一下MCU是如何启动的。这一点在芯片的参考手册中都有详细的说明，不同的芯片手册所在位置可能不同，但是一般在第二章会有单独一节叫**Boot configuration**。如下图：

2.6 Boot configuration

In the STM32F37xxx, three different boot modes can be selected through the BOOT0 pin and nBOOT1 bit in the User option byte, as shown in the following table:

Table 2. Boot modes

Boot mode selection		Boot mode	Aliasing
BOOT1 (inverted nBOOT1)	BOOT0	-	-
x	0	Main Flash memory	Main flash memory is selected as boot space
0	1	System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM (on the DCode bus) is selected as boot space

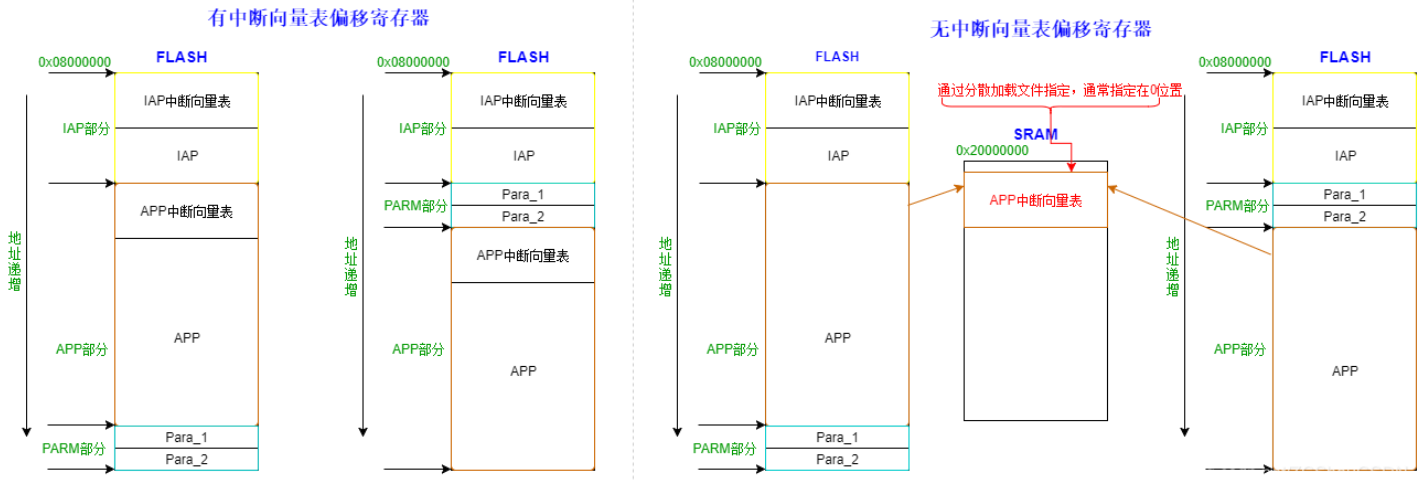
The values on both BOOT0 pin and nBOOT1 option bit are latched on the 4th rising edge of SYSCLK after a reset.

It is up to the user to set the nBOOT1 and BOOT0 to select the required boot mode. The BOOT0 pin and nBOOT1 option bit are also resampled when exiting from Standby mode. Consequently they must be kept in the required Boot mode configuration in Standby mode. After this startup delay has elapsed, the CPU fetches the top-of-stack value from address 0x0000 0000, then starts code execution from the boot memory at 0x0000 0004. Depending on the selected boot mode, main Flash memory, system memory or SRAM is accessible as

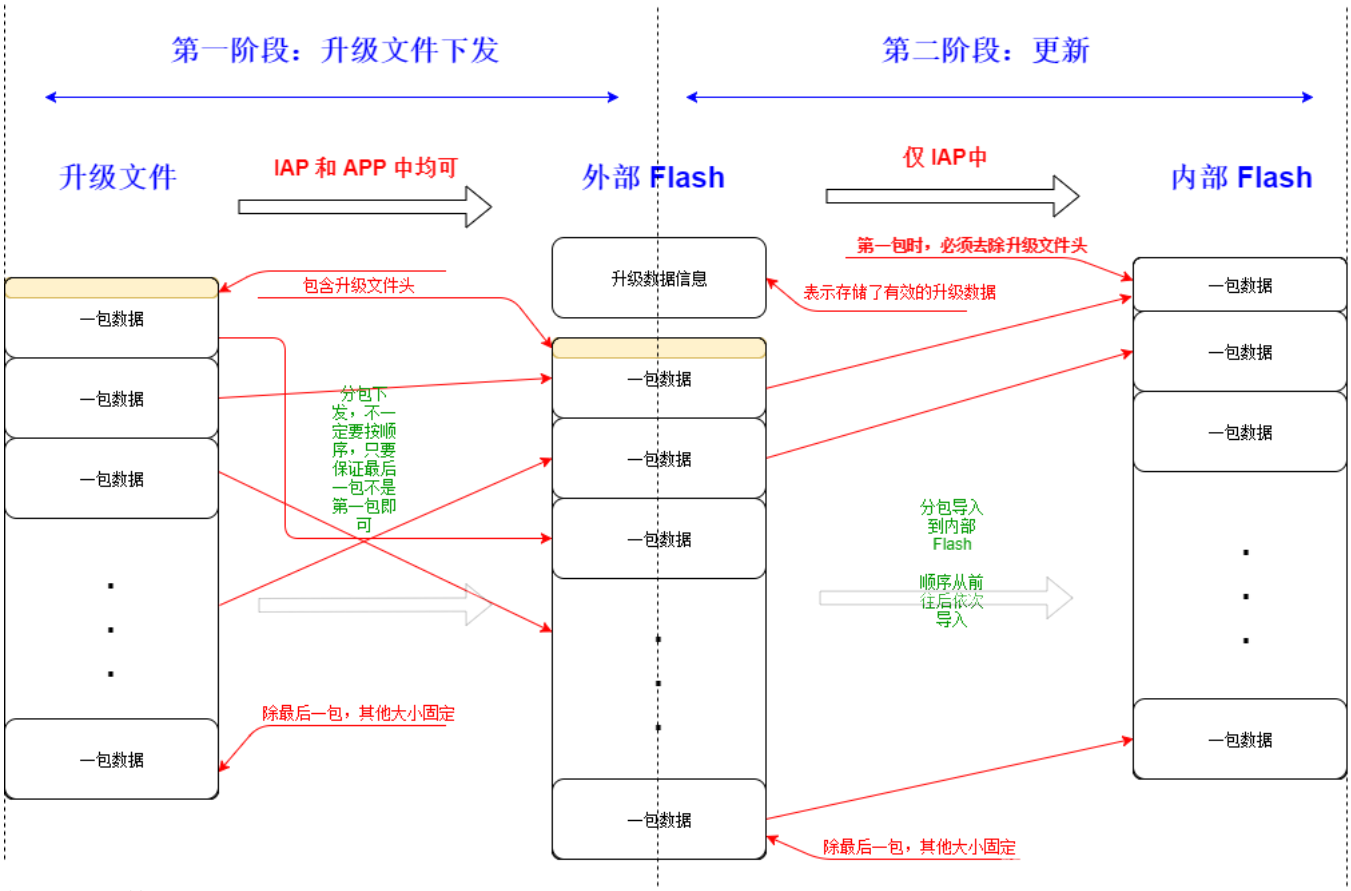
主要就是说，启动是通过管脚BOOT0和BOOT1的连接方式来控制的。这个是在硬件设计阶段设计好的。不同的配置决定了，MCU将何处映射到0x00000000。从这里又可以看到一点，MCU眼里只有0x00000000。至于为啥可以从Flash（0x08000000）启动，就是因为MCU内部做了映射。从其他位置启动时同理。

IAP 实现

要实现IAP，则整个程序实现分为大程序（APP）和小程序（IAP）两部分。其中，APP主要接收升级数据并存储，IAP处理擦除APP，并重新写入升级数据。此外，IAP还应该可以独立接收升级数据的情况。***但是，由于Cortex-M0核是没有中断向量表偏移寄存器的，这就导致了在Cortex-M0核的MCU上实现在线升级比较麻烦。***在实际产品中，整个程序的基本组成结构：



实际的IAP流程如下：

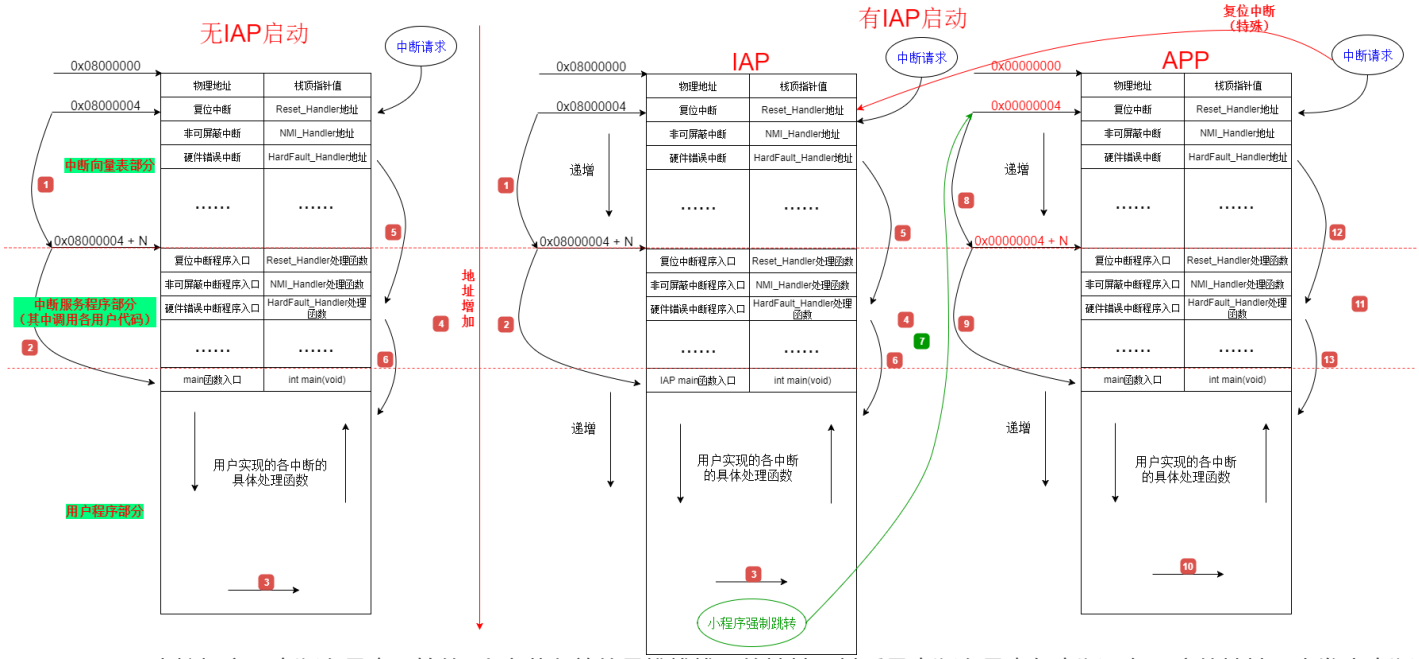


就是这么简单！

> 注意：
> （1）与 Cortex-M3 和 Cortex-M4 不同，Cortex-M0 没有中断向量表偏移寄存器（VTOR寄存器）
> （2）Cortex-M3 r2p0 及其之前版本，中断向量表只能位于SRAM或者CODE区域，但是Cortex-M3 r2p1及之后，Cortex-M4 没有该限制！
> （3）MCU根据Boot引脚配置将指定地址映射为0x地址！

IAP 启动

启动网上有很多文章介绍，但是或多或少不是很完善，我只做了一张相对来说比较详细的图，如下：



Cortex-M内核规定，中断向量表开始的4个字节存放的是堆栈栈顶的地址，其后是中断向量表各中断服务程序的地址。当发生中断后程序通过查找该表得到相应的中断服务程序入口地址，然后再跳到相应的中断服务程序中执行，中断服务程序中最终调用用户实现的各函数。例如：main函数就是复位中断服务函数中调用的！

在没有IAP时，上电后从0x08000004处取出复位中断向量的地址，然后跳转到复位中断程序的入口(标号①所示)，执行结束后跳转到main函数中(标号②所示)。通常main函数是个死循环，不会退出。在执行main函数的过程中发生中断，则STM32强制将PC指针指回中断向量表处(标号④所示)，从中断向量表中找到相应的中断函数入口地址，跳转到相应的中断服务函数(标号⑤所示)，执行完中断函

数后再返回到main函数中来(标号⑥所示)。

在添加IAP后，上电后仍然从0x08000004处取出复位中断向量的地址，然后跳转到复位中断程序的入口(标号①所示)，执行结束后跳转到小程序的main函数中(标号②所示)。在执行小程序main函数的过程中发生中断，则STM32强制将PC指针指回中断向量表处(标号④所示)，从中断向量表中找到相应的中断函数入口地址，跳转到相应的中断服务函数(标号⑤所示)，执行完中断函数后再返回到main函数中来(标号⑥所示)。而想要大程序执行，则必须在小程序中显示强制跳转（标号⑦）。

在大程序的main函数的执行过程中，如果CPU得到一个中断请求，由于我们设置了中断向量表偏移量为N+M，因此PC指针被强制跳转到0x08000004+N+M处的中断向量表中得到相应的中断函数地址，再跳转到相应新的中断服务函数，执行结束后返回到main函数中来。

需要注意的是，复位中断比较特殊。产生复位后，PC的值会被硬件强制置为0x08000004。因为，在发生复位后，负责中断向量偏移的寄存器VTOR变为了0，因此，复位后的中断就变为了0x08000004。而其他中断发生时，VTOR为已经设置好的终端向量表偏移。

程序实现

有了上面的介绍，实现就比较简单了！其实我有设计了一套适用于全部STM32芯片的IAP模板，但是属于公司产品，不方便对外公布！简单说几个重点：

1. 使用 **分散加载文件** 实现起来会比较方便
2. 对于没有中断向量表偏移寄存器的MCU（主要是Cortex-M0核），一般采用将中断向量表复制到指定位置的内存中的方式实现：
    1. 使用分散加载文件在内存中指定一块区域：

```
#if   (defined ( __CC_ARM ))
  __IO uint32_t VectorTable[48] __attribute__((section("SECTION_APP_VECTOR")));
#elif (defined (__ICCARM__))
#pragma location = 0x20000000
  __no_init __IO uint32_t VectorTable[48];
#elif defined   (  __GNUC__  )
  __IO uint32_t VectorTable[48] __attribute__((section(".RAMVectorTable")));
#elif defined ( __TASKING__ )
  __IO uint32_t VectorTable[48] __at(0x20000000);
#endif
```

    1. 将APP的终端向量表复制到以上位置，设置中断向量表重映射

```
static void SetVectorTable(void)
{
    int i;

    /*!< At this stage the microcontroller clock setting is already configured,
    this is done through SystemInit() function which is called from startup
    file (startup_stm32f0xx.s) before to branch to application main.
    To reconfigure the default setting of SystemInit() function, refer to
    system_stm32f0xx.c file
    */

    /* Relocate by software the vector table to the internal SRAM at 0x20000000 ***/
    /* Copy the vector table from the Flash (mapped at the base of the application load address 0x08003000) to the base addre
    for(i = 0; i < 48; i++)
    {
        VectorTable[i] = *(__IO uint32_t*)(APP_SPACE_ADDR + (i<<2));
    }

    /* Enable the SYSCFG peripheral clock */
    RCC_APB2PeriphResetCmd(RCC_APB2Periph_SYSCFG, ENABLE);  /* 注意：ST官方例程使用 RCC_APB2PeriphResetCmd是不对的 */
    /* Remap SRAM at 0x00000000 */
    SYSCFG_MemoryRemapConfig(SYSCFG_MemoryRemap_SRAM);
}
```

3. 在由 IAP 跳转到 APP 时，***一定注意把 IAP 中开启的外设全部关闭，否则在刚进入 APP中时，如果产生中断将导致死机等问题。*** 包括 SysTic 中断！！！包括 SysTic 中断！！！包括 SysTic 中断！！！这里可以做测试：
    1. 测试一：IAP 中开启串口，然后用上位机不停的发送数据，在发送数据过程中执行 IAP 跳转 APP
    2. 将 SysTick 中断 配置时间很短（微秒级别），当程序跳转到 APP 后，会出现 先产生 SysTick 中断，然后才会到 main 函数。此时如果 SysTick 中断中有相关代码，将导致出现错误！

4. STM32 的 back SRAM 在 IAP 中和 APP 中都初始化时，将导致 APP中的初始化不起作用。如果 IAP 中有使用，则在跳转 APP前必须反初始化。

(2条消息) STM32 之八 在线升级（IAP）超详细图解 及 需要注意的问题解决_EXP-CSDN博客