

伟福 J.MasterIII 编程器

对 ARM 系列 CPU 附带外部存储器的编程方法

在很多 ARM 系列 CPU 的应用电路中，除了主程序存储器外，经常带有外部存储器（FLASH，EEPROM 等）用于存储数据，而这些存储器在生产过程中常常需要写入初始内容，通常的做法是利用探针接到存储器的管脚再接编程器进行烧写。利用伟福 J.MASTER 编程器，可以在烧写 CPU 的程序时，同时对这些存储器进行烧写，不用另外再接编程器。你所需要做的就是参照我们的程序例子，根据你电路的特性（哪些管脚接存储器，什么类型的存储器等）改写我们提供的程序例子，就可达到目的。以下详细说明如何改写外部存储器烧写的例子。

这里正好有一块 STM32F103RC 的演示电路板，带有 24C02 IIC EEPROM 及 W25X16 SPI FLASH 两种外部存储器，于是就针对这两种存储器的写了编程例子，方便用户参照借用。按照 ARM 系列 CPU 烧写程序的例子，要定义外部存储器的特性参数。见以下数据结构：

```

14 struct FlashDevice const FlashDevice = {
15     FLASH_DRV_VERSION, // 驱动程序版本，请勿改动！
16     "WaveTEST W25X16 SPI Flash", // 器件名称，在选择驱动程序时显示
17     EXTSPI, // 器件类型
18     0x00000000, // 器件存储起始地址
19     0x00200000, // 器件容量（字节）（w25x16有2M字节）
20     256, // 编程页容量（字节）
21     0, // 保留，必须为 0
22     0xFF, // 擦除后的初始内容
23     100, // 页编程超时时间（毫秒）
24     3000, // 块擦除超时时间（毫秒）
25     // 定义各块的大小及起始地址
26     0x10000, 0x000000, // 块大小及块起始地址
27     SECTOR_END // 块结束
28 };

14 struct FlashDevice const FlashDevice = {
15     FLASH_DRV_VERSION, // 驱动程序版本，请勿改动！
16     "WaveTEST 24C02 IIC EEPROM", // 器件名称，在选择驱动程序时显示
17     UNKNOWN, // 器件类型
18     0x00000000, // 器件存储起始地址
19     0x00000100, // 器件容量（字节）（24C02为256字节）
20     8, // 编程页容量（字节）
21     // Pagesize 8B for 02, 16B for 04/08/16, 32B for 32/64
22     0, // 保留，必须为 0
23     0xFF, // 擦除后的初始内容
24     2000, // 页编程超时时间（毫秒）
25     2000, // 块擦除超时时间（毫秒）
26     // 定义各块的大小及起始地址
27     0x8, 0x000000, // 块大小及块起始地址
28     SECTOR_END // 块结束
29 };

```

以上我们定义了 SPI 总线的 FLASH 存储器 W25X16 及 IIC 总线的 EEPROM 存储器 AT24C02 两种外部存储器的器件参数，编程器在调用烧写程序时会用到这些参数，在定义这些参数时，请参考存储器对应的说明书，否则可能会在烧写过程中出一些不可预见的错误。

为了能对外部存储器进行烧写，还需要定义几个可被调用外部函数，以便被编程器调用。几个函数如下，

```

// Flash Programming Functions (Called by FlashOS)
extern int Init (void);
extern int UnInit (unsigned long fnc); // De-initialize Flash
extern int BlankCheck (unsigned long adr, // Blank Check
    unsigned long sz,
    unsigned char pat);
extern int EraseChip (void); // Erase complete Device
extern int EraseSector (unsigned long adr); // Erase Sector Function
extern int ProgramPage (unsigned long adr, // Program Page Function
    unsigned long sz,
    unsigned char *buf);

extern int Verify (unsigned long adr, // Verify Function
    unsigned long sz,
    unsigned char *buf);

extern int Read (unsigned long adr,
    unsigned long sz,
    unsigned char *buf);

```

Init() 为初始化程序，对 CPU 的执行速度，对连接到存储器的 CPU 的 I/O 端口进行定义等。在初始化过程中尽量使用 8MHz 的 HSI 时钟，以免对其它程序造成干扰。初始化成功返回 0，不成功返回非 0。

UnInit() 还原初始化程序对 CPU 的改变，成功返回 0，不成功返回非 0。

BlankCheck() 对存储器进行查空操作，存储器空返回 0，不空返回非 0。

EraseChip() 对存储器进行全片擦除，成功返回 0，不成功返回非 0。

EraseSector() 对存储器进行块擦除，成功返回 0，不成功返回非 0。（全片擦除和块擦除只需要有一个即可，如果选择块擦除，主程序会多次进行块擦除来完成全片擦除。）

ProgramPage() 对存储器进行页编程，成功返回 0，不成功返回非 0。

Verify() 对写入的内容进行校验，成功返回 0，不成功返回非 0。

Read() 读出存储器里的内容。成功返回 0，不成功返回非 0。

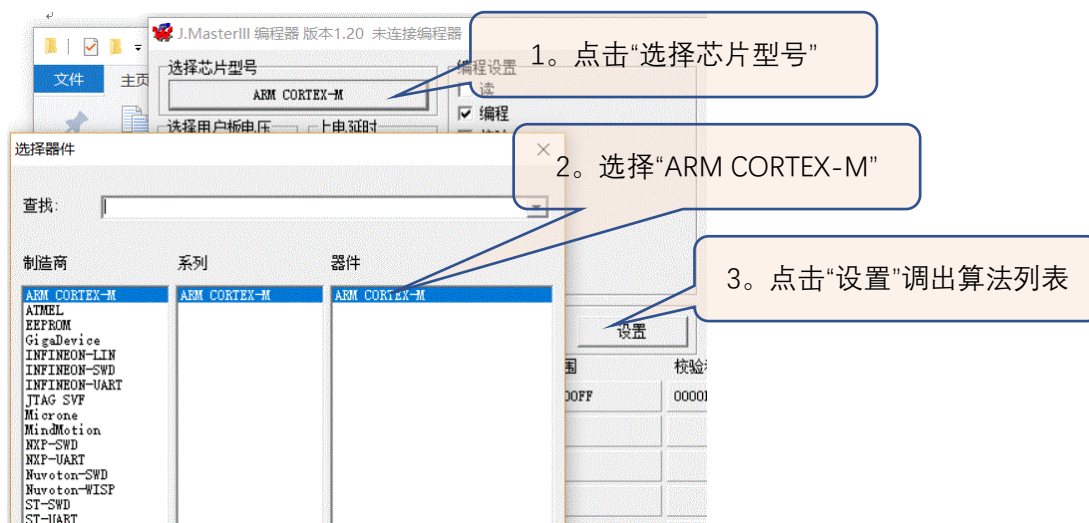
如果这些函数没有具体程序，那么在主程序的编程设置中就不会出现这功能的选择，例如如果没有写 **BlankCheck()** 的程序，那么在主程序的编程设置就不会有“查空”的功能选择。全片擦除和块擦除只需有一个即可，主控程序会利用块擦除函数完成全片擦除。有读函数就可以完成校验功能，即使 **Verify()** 没有定义。

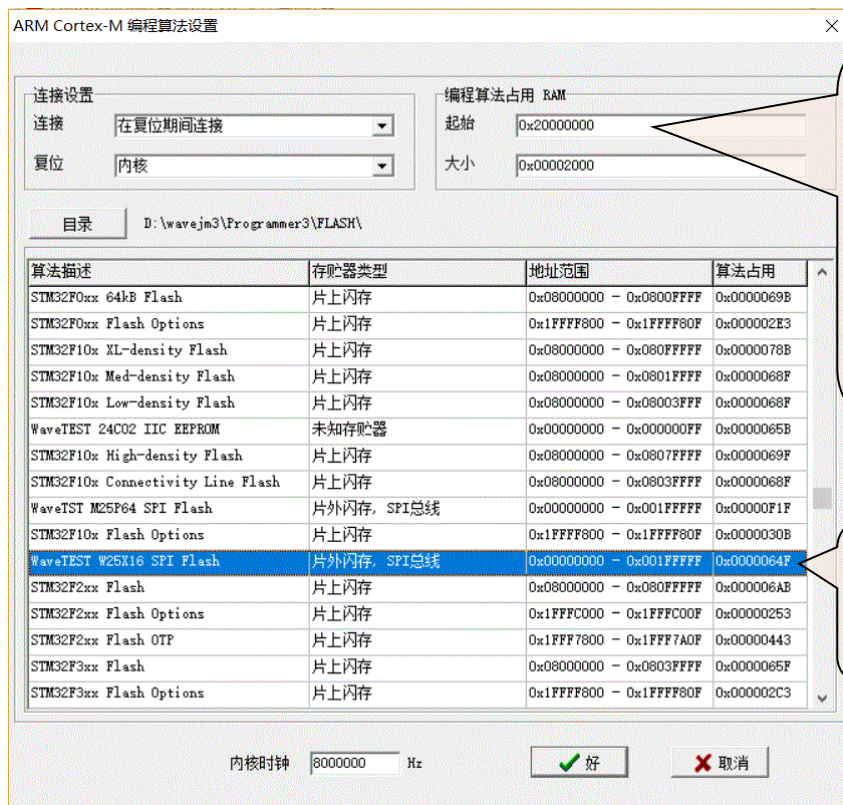
用户可以参照我们提供的例子，实现这几个函数功能。并编译生成代码，转换成 FLM 文件，将生成的 FLM 文件复制到 J. MasterIII 安装目录下的 FLASH 子目录下。

此电脑 > 数据盘 (D:) > wavejm3 > Programmer3 > Flash

名称	修改日期	类型	大小
STM32F10x_24c02.FLM	2019/6/5 12:50	FLM 文件	19 KB
STM32F10x_W25X16.FLM	2019/5/31 12:07	FLM 文件	18 KB
STM32F10x_M25P64.FLM	2019/5/28 14:27	FLM 文件	84 KB
STM32F10x_512.FLM	2015/3/18 18:16	FLM 文件	11 KB
LPC18xx43xx_S25FL032.FLM	2014/3/21 16:33	FLM 文件	35 KB

打开 J. MasterIII 控制程序，点击“选择芯片型号”为“ARM CORTEX-M”，然后点击“设置”在列表中找到要编程芯片的类型。





对于 STM32F103RC 内部 RAM 起始为 0x20000000，若应用电路选用其它型号的 ARM 芯片，请注意内部 RAM 的起始。占用 RAM 大小，要大于所选择的算法程序大小，程序才能正确运行。

此 SPI 编程算法程序占用空间大小

我们样例程序所设定的器件名称为“WaveTEST W25X16 SPI Flash”和“WaveTEST 24C02 IIC EEPROM”，两种存储器均接在 STM32F103RC 的应用电路上，前者为 SPI 总线存储器（NSS 接 PA2、SCK 接 PA5、MISO 接 PA6、MOSI 接 PA7），后者为 IIC 总线存储器（SDA 接 PC11、SCK 接 PC12）。



选择好要烧写的器件，返回主界面可以看到，我们在 W25X16 的 SPI 的烧写程序没有实现 BlankCheck() 功能，所以编程设置中没“空片检查”选项。而在 24C02 的程序中我们完成了 BlankCheck() 的功能，所以在编程设置选项中有“空片检查”。

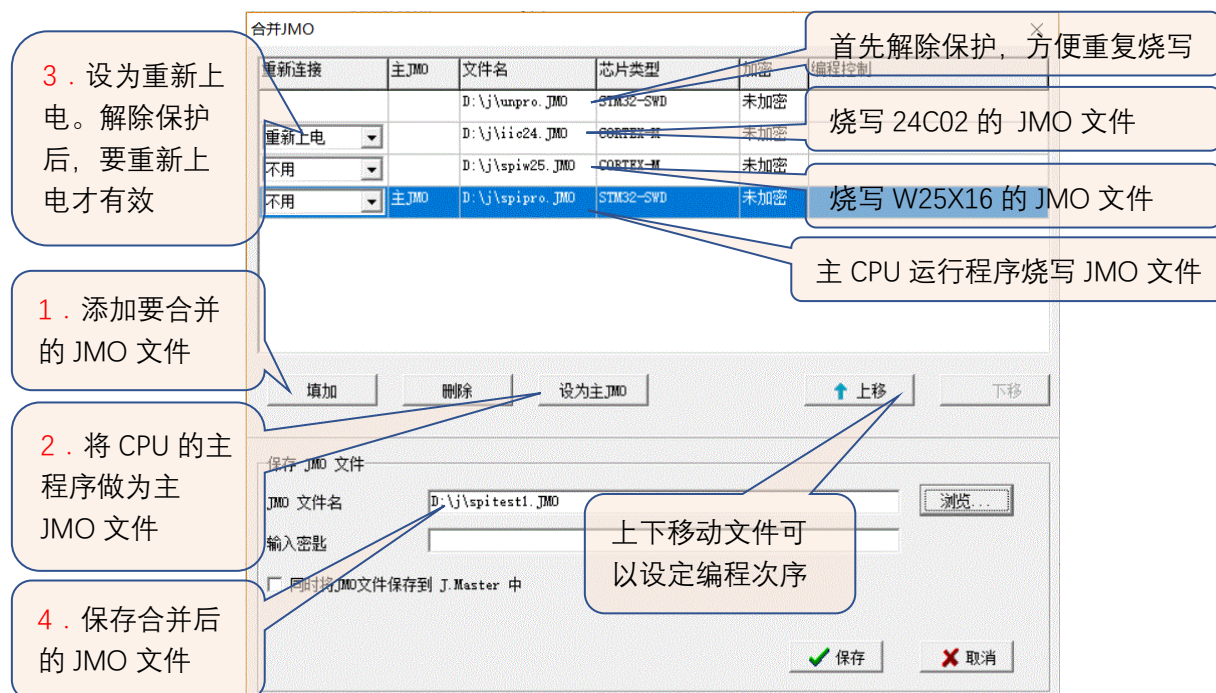
选择好器件后，就可以从磁盘上读入要烧写的数，也可手工直接在数据缓冲区里填写。

J. MasterIII 编程器以 SWD 方式接在应用电路的编程端口上，如果用户板单独供电，则 VCC 不用接，如果是单线仿真，编程器的 PIN9 做为 SPD 信号使用。

J. MasterIII 编程器	STM32F103RC 应用电路
VCC (PIN1)	VCC
GND (PIN8 或 PIN10)	GND
SWCLK (PIN7)	TCK/SWCLK (PA14)
SWDIO (PIN9)	TMS/SWDIO (PA13)

点“编程 P1”或“编程 P2”进行编程操作，如果无误可以点“写文件”将要烧写的内容及操作保存成 JMO 文件，以便合成一个大的 JMO，完成全部烧写过程。

点主界面上的“合并 JMO”调出合并 JMO 界面。



按以下步骤合并 JMO 文件：1. 添加要合并的 JMO 文件、2. 最后加上 CPU 的主程序，并设为主 JMO 文件、3. 将解除保护后的 JMO 文件连接方式设为“重新上电”，重新上电后解除保护才有效果、4. 保存合并后 JMO 文件。如果添加文件次序不是理想编程次序，可以点“上移”“下移”来移动文件调整编程次序。

在主界面点“读文件”装载进刚才合并生成的 JMO 文件，点“编程 P1”或“编程 P2”进行编程，就可以依次自动完成 1. 解除保护、2. 烧写 24C02、3. 烧写 W25X16、4. 烧写主 CPU 程序(主程序 JMO 附带保护操作)多个过程。

合并 JMO 有以下限制：

1. 必须是用仿真口编程(SWD、SPD、JTAG)方式连接到用户应用电路，串口下载方式不可以合并(可以合并才会显示合并按钮)
2. 最多合并 8 个独立的 JMO 文件，已合并过的 JMO 文件不能再合并
3. 用户需要指定一个主 JMO 文件，编程参数由主 JMO 决定。
限定编程次数、限定编程器号、自动编号只能在主 JMO 文件中使用，不能在非主 JMO 文件中设定。
4. 被合并的 JMO 文件可以是加密的，也可以不是加密的，
如果含有加密的 JMO，所有加密 JMO 的密码必须是同一个，
并且在生成合并 JMO 文件时也必须再次输入这个密码。
5. 主 JMO 一般是 ARM 芯片，外围器件不要做为主 JMO。
6. 如果 ARM 芯片是已加密的芯片，第一个 JMO 必须是解密芯片，否则编程算法不能正常运行。
第二个 JMO 算法重新连接必须是“重新上电”，(通常加密的 ARM 芯片必须重新上电才能脱离加密态)