MM32 IAP Programming Manual V1.0

目录

目录

1.IAP 需求及基础介绍

在电子产品出厂前，可以通过离线烧录器，烧录夹具，或者用在线烧录器通过预留的烧录接口轻松将应用代码下载到 MCU 中。但是，如果产品已售出或不在研发端，又要怎样升级程序呢？因此，工程师在做产品的时候会有远程对产品进行升级的需求，对于没有开发过此功能的工程师会不知道从何下手，本文就以 MM32 M0 系列 MCU 为例介绍如何用 IAP 功能实现为单片机远程升级以及相关的基础知识点。

1.1 不同编程方式简介

电子工程师都知道，半导体技术发展迅猛，带动了各种芯片技术的不断升级。在数据存储方面，从最初的掩膜 ROM，发展到现在的 Flash 技术，存储技术的不断改进，相对应的编程技术也在不断发展。在这个发展过程中，也诞生了主要以下几种编程技术。

1.1.1 ISP

ISP:In System Programing，在系统编程。比如：使用 MM32 ISP Tool 对 MM32 MCU 编程。支持 ISP 的芯片一般在芯片内部固化了一段（用 ISP 升级的）Boot 程序，需要将 Boot 引脚跳成从系统空间启动。

1.1.2 ICP

ICP:In Circuit Programing，在电路编程。ICSP：In-Circuit Serial Programming，在电路串行编程。如：对 EEPROM 编程等。ISPICP 编程方式网上各有说法，从字面含义（在电路）来说，所有处于编程的芯片都需要上电，都处于电路中。不严格来说利用 J-Link、MM32-Link、U-Link 等工具进行编程也属于在电路编程（ICP）。在维基百科中，在系统编程(ISP)，也称为在电路串行编程(ICSP)。

1.1.3 IAP

IAP:In Applicating Programing，在应用编程。这里是本文说的重点，可以简单理解为：在程序运行的过程中进行编程（升级程序，更新固件）。IAP 是用户自己的程序在运行过程中对 User Flash 的部分区域进行烧写，目的是为了在产品发布后可以方便地通过预留的通信口对产品中的固件程序进行更新升级。

1.2 不同传输协议简介

串行通信的文件传输协议主要和常用的有：Xmodem、Ymodem、Zmodem 以及 KERMIT、Ymodem-G、ASCII 等。

- ASCII：这是最快的传输协议，但只能传送文本文件。
- Xmodem：这种古老的传输协议速度较慢，但由于使用了 CRC 错误侦测方法，传输的准确率可高达 99.6%。
- Ymodem：这是 Xmodem 的改良版，使用了 1024 位区段传送，速度比 Xmodem 要快。
- Zmodem：Zmodem 采用了串流式（streaming）传输方式，传输速度较快，而且还具有自动改变区段大小和断点续传、快速错误侦测等功能。这是目前最流行的文件传输协议。

1.2.1 Xmodem

XModem 是一种在串口通信中广泛使用的异步文件传输协议，分为 XModem 和 1k-XModem 协议两种，前者使用 128 字节的数据块，后者使用 1024 字节即 1k 字节的数据块。Xmodem 协议传输有接收程序和发送程序完成，先由接收程序发送协商字符，协商校验方式，协商通过之后发送程序就开始发送数据包，接收程序接收到完整的一个数据包之后按照协商的方式对数据包进行校验。校验通过之后发送确认字符，然后发送程序继续发送下一包；如果校验失败，则发送否认字符，发送程序重传此数据包。

信息包格式有如下两种：

```
-------------------------------------------------------------------------
|     Byte1     |    Byte2    |     Byte3      |  Byte4~Byte131|  Byte132   |
|-----------------------------------------------------------------------  |
|Start Of Header|Packet Number|~(Packet Number)| Packet Data   |  Check Sum |
-------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------
|    Byte1      |    Byte2    |     Byte3      |Byte4~Byte131|Byte132~Byte133|
|-----------------------------------------------------------------------|
|Start Of Header|Packet Number|~(Packet Number)| Packet Data | 16Bit CRC  |
```

帧字段定义如下：

```
SOH 01H（Xmodem 数据头）
STX 02H（Xmodem-1K 数据头）
EOT 04H（发送结束）
ACK 06H（应答）
NAK 15H（非应答）
CAN 18H（取消发送）
```

校验方式分为累加和与 CRC-16，具体操作如下：

在累加和方式中，所有的数据字节都将参与和运算，由于校验和只占一个字节，如果累加的和超过 255 将从零开始继续累加；对于发送方仅仅支持校验和的传输方式，接收方应首先发送 NAK 信号来发起传输，如果发送方没有数据发送过来，需要超时等待 3 秒之后再发起 NAK 信号来进行数据传输。对于数据传输正确，接收方需要发送 ACK 信号来进行确认，如果数据传输有误，则发送 NAK 信号，发送方在接收到 NAK 信号之后需要重新发起该次数据传输，如果数据已近传输完成，发送方需要发送 EOT 信号，来结束数据传输。当接收方发送 CAN 表示无条件结束本次传输过程，发送方收到 CAN 后，无需发送 EOT 来确认，直接停止数据的发送。

详细传输流程如下：

图 1 Xmodem 累加和方式传输流程

在 CRC-16 校验方式中，需要注意的是，在发送方，CRC 是高字节在前，低字节在后。和校验和方式不同的是，当接收方要求发送方以 CRC16 校验方式发送数据时以'C'来请求，发送方对此做出应答，流程就如上图所示。当发送方仅仅支持校验和方式时，则接收方要发送 NAK 来请求，要求以校验和方式来发送数据，如果仅仅支持 CRC16 校验方式，则只能发送'C'来请求。如果两者都支持的话，优先发送'C'来请求。最后，如果信息包中的数据如果不足 128 字节，剩余的部分要以 0x1A(Ctrl-Z)来填充。

详细传输流程如下：

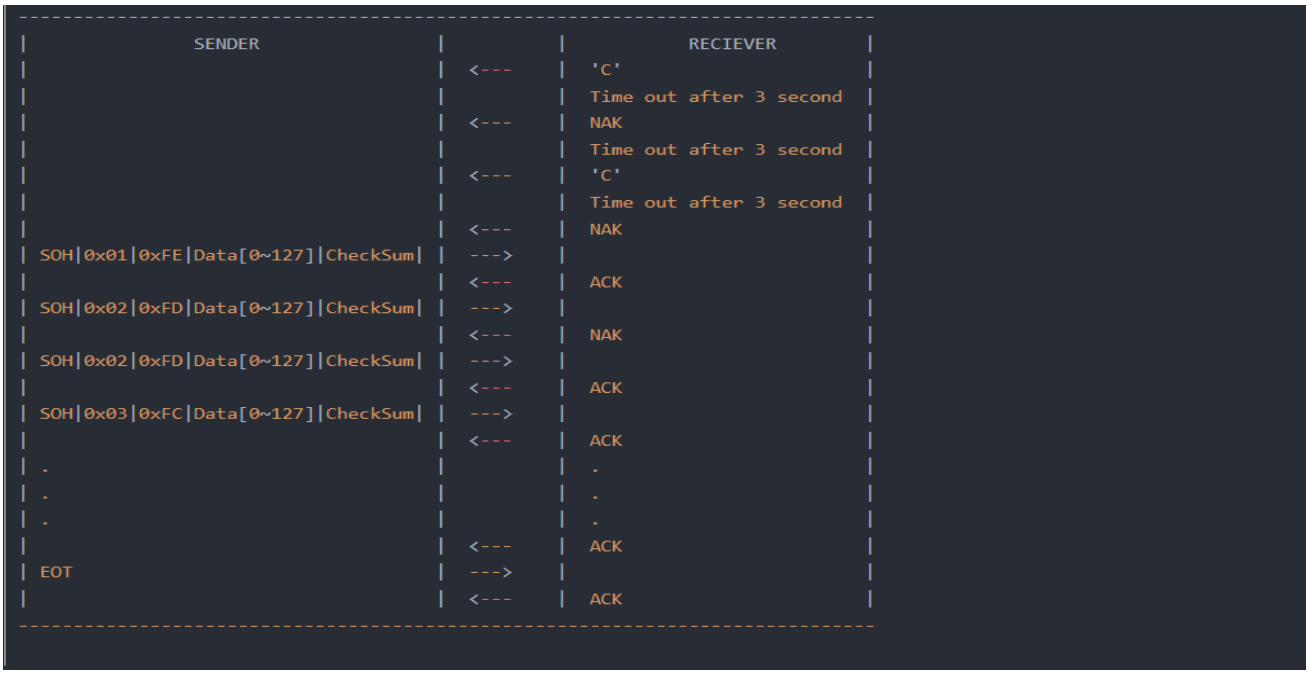

图 2 Xmodem CRC 方式传输流程

1k-XModem 协议同 XModem-CRC16 协议差不多，只是数据块长度变成了 1024 字节即 1k，同时每个信息报的第一个字节的 SOH 变成了 STX，STX 定义为 <STX> 0x02，能有效的加快数据传输速率。

1.2.2 Ymodem

YModem 协议是 XModem 的改进协议，它最常用于调制解调器之间的文件传输的协议，具有快速，稳定传输的优点。它的传输速度比 XModem 快，这是由于它可以一次传输 1024 字节的信息块，同时它还支持传输多个文件，也就是常说的批文件传输。

YModem 分成 YModem-1K 与 YModem-g。

YModem-1K 用 1024 字节信息块传输取代标准的 128 字节传输，数据使用 CRC 校验，保证数据传输的正确性。它每传输一个信息块数据时，就会等待接收端回应 ACK 信号，接收到回应后，才会继续传输下一个信息块，保证数据已经全部接收。

YModem-g 传输形式与 YModem-1K 差不多，只是它去掉了数据的 CRC 校验码，同时在发送完一个数据块信息后，它不会等待接收端的 ACK 信号，而直接传输下一个数据块。正是它没有涉及错误校验与等待响应，才使得它的传输速度比 YModem-1K 来得快。

YModem 的数据格式如下：

YModem 的起始帧用于传输文件名与文件的大小，注意该数据包号为 0，帧长=3 字节的数据首部+128 字节数据+2 个字节 CRC16 校验码 = 133 字节。数据结构为：

```
SOH 00 FF filename[ ] filezise[ ] NUL[ ] CRCH CRCL
```

YModem 的数据帧从第二包数据开始，注意该数据包号为 1。帧长 = 3 字节的数据首部+1024 字节数据+2 字节的 CRC16 校验码 = 1029 字节。数据结构为：

```
STX [num] [~num] data[ ] 1A …1A CRCH CRCL
```

其中的第二个字节为传输的数据包包号，第三个字节为数据包号取反组成。若文件数据的最后一包数据在 128~1024 之间，则数据部分剩余空间全部用 0x1A 填充。

注意，存在一种特殊情况：如果文件的大小小于或等于 128 字节或者文件数据最后剩余的数据小于 128 字节，则 YModem 会选择使用 SOH 数据帧，即用 128 字节来传输数据，如果数据不满 128 字节，剩余的数据用 0x1A 填充。这时数据帧的结构就变成了：

文件大小小于 128 字节：

```
SOH 01 FE data[ ] 1A …1A CRCH CRCL
```

当传输结束时，YModem 还会再传一包结束数据，只是数据内容为空。帧长=3 字节首部+128 字节的数据+2 字节 CRC16 校验码 = 133 字节，其数据帧结构为：

```
SOH 00 FF NUL[128] CRCH CRCL
```

特别注意的是，在文件传输结束时发送端发送了结束标识 EOT 之后待收到接收端的回复后，还会再发送一包空数据包以表示传输真正结束。

以下为 YModem 传输流程：

```
发送端                                                    接收端
 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< C
SOH 00 FF [55 53…6E 00]" "[32…30 00]'' NUL[96] CRC CRC >>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< C
STX 01 FE data[1024] CRC CRC>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK
STX 02 FD data[1024] CRC CRC>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK
STX 03 FC data[1024] CRC CRC>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK
STX 04 FB data[1024] CRC CRC>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

SOH 05 FA data[100]  1A[28] CRC CRC>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK
EOT >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< NAK
EOT>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  ACK
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< C
SOH 00 FF NUL[128] CRCCRC >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  ACK
```

1.2.3 Binary

Bin 文件是最纯粹的二进制机器代码，或者说是"顺序格式"。按照 assembly code 顺序翻译成 binary machine code，内部没有地址标记。Bin 是直接的内存映象表示，二进制文件大小即为文件所包含的数据的实际大小。严格意义上来说，Binary 不能称之为传输协议，它仅代表上位机对 code 编译出来的 bin 文件数据不进行任何协议处理，直接完整地将整个文件数据一下发送出去，中间没有任何超时、错误等帧处理机制。

1.2.4 Ascii

Ascii 传输协议实际上就是将整个 HEX 文件数据一下发送出去，类似 Binary 方式中途无任何传输管理机制。Hex 文件,一般是指 Intel 标准的十六进制文件。Intel HEX 文件是记录文本行的 ASCII 文本文件，在 Intel HEX 文件中，每一行是一个 HEX 记录，由十六进制数组成的机器码或者数据常量。Intel HEX 文件经常被用于将程序或数据传输存储到 ROM、EPROM，大多数编程器和模拟器使用 Intel HEX 文件。

HEX 格式文件以行为单位，每行由“：”（0x3a）开始，以回车键结束(0x0d,0x0a)。行内的数据都是由两个字符表示一个 16 进制字节，比如”01”就表示数 0x01；”0a”，就表示 0x0a。对于 16 位的地址，则高位在前低位在后，比如地址 0x010a，在 HEX 格式文件中就表示为字符串”010a”。下面为 HEX 文件中的一行：

```
:10000000FF0462FF051EFF0A93FF0572FF0A93FFBC
```

“：”表示一行的开始。

“：”后的第 1，2 个字符“10”表示本行包含的数据的长度，这里就是 0x10 即 16 个。

第 3，4，5，6 个字符“0000”表示数据存储的起始地址，这里表示从 0x0000 地址开始存储 16 个数据，其中高位地址在前，低位地址在后。

第 7，8 个字符“00”表示数据的类型。该类型总共有以下几种：

```
00 ----数据记录
01 ----文件结束记录
02 ----扩展段地址记录
04 ----扩展线性地址记录
```

这里就是 0x00 即为普通数据记录。

自后的 32 个字符就是本行包含的数据，每两个字符表示一个字节数据，总共有 16 个字节数据跟行首的记录的长度相一致。

最后两个字符表示校验码。

每个 HEX 格式的最后一行都是固定为：

```
:00000001FF
```

以上的信息其实就足够进行 HEX 转 BIN 格式的程序的编写。首先我们只处理数据类型为 0x00 及 0x01 的情况。0x02 表示对应的存储地址超过了 64K，由于我的编程器只针对 64K 以下的单片机，因此在次不处理，0x04 也是如此。

解析 HEX 文件的编程思路是从文件中一个一个读出字符，根据“：”判断一行的开始，然后每两个字符转换成一个字节，并解释其对应的意义。然后将数据从该行中剥离出来保存到缓冲区中，并最终输出到文件中。

1.3 IAP 实现原理

通常实现 IAP 功能时,在传输接口上，可以选择使用 UART/IIC/SPI/USB/CAN 以及 SD 卡方式，本文选用的是 UART 接口；在升级和跳转方向上，可以在 APP 中进行下载程序，完成后跳到 Boot 去执行更新升级，也可以直接在 dBoot 中进行下载、升级和跳转到用户程序，本文选用的后者，这样做可以固定 BootLoader 代码段，方便移植到用户的工程中去；在分区方式上可分为单 APP 和双 APP 区方式,单 APP 区适用于 Flash 资源较小的方案中，双 APP 区模式会预留一块备份的 APP 区域，虽然会浪费闪存空间，但增加了升级健壮性，本文工程中选用单 APP 模式。

IAP 升级方案中，整个程序实现分为大程序（APP）和小程序（BootLoader）两部分。其中，BootLoader 主要接收升级数据并存储，并处理擦除 APP 重新写入升级数据，校验成功完成后跳转到 APP 执行。而 APPP 程序除了用户应用层代码外还需要包括接收升级命令的接口，根据实际的升级请求擦写升级参数区域，成功后复位跳转到 BootLoader。

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，需要变换方式去处理。下面详细介绍基于实现 MM32 M0 UART IAP Demo 功能的工程源码。

2. IAP_UART 项目工程详解

MM32 M0 IAP_UART 项目适用于所有当前 MM32 M0 MCU 产品， 具体系列如下表：

Cortex 内核	产品型号

M0	MM32F031xxxx_n4 / MM32L0xxxx_n4 MM32SPIN2xxx_p5 MM32F031xxxx_q3 / MM32SPIN0xxx_q3 / MM32F003xx_q3 MM32F032xxxx_s2 / MM32L0xxxx_s2 / MM32SPIN0xxx_s2

表 1 适用产品系列

2.1 项目目录结构

MM32 M0 IAP_UART 项目目录结构如下图所示：

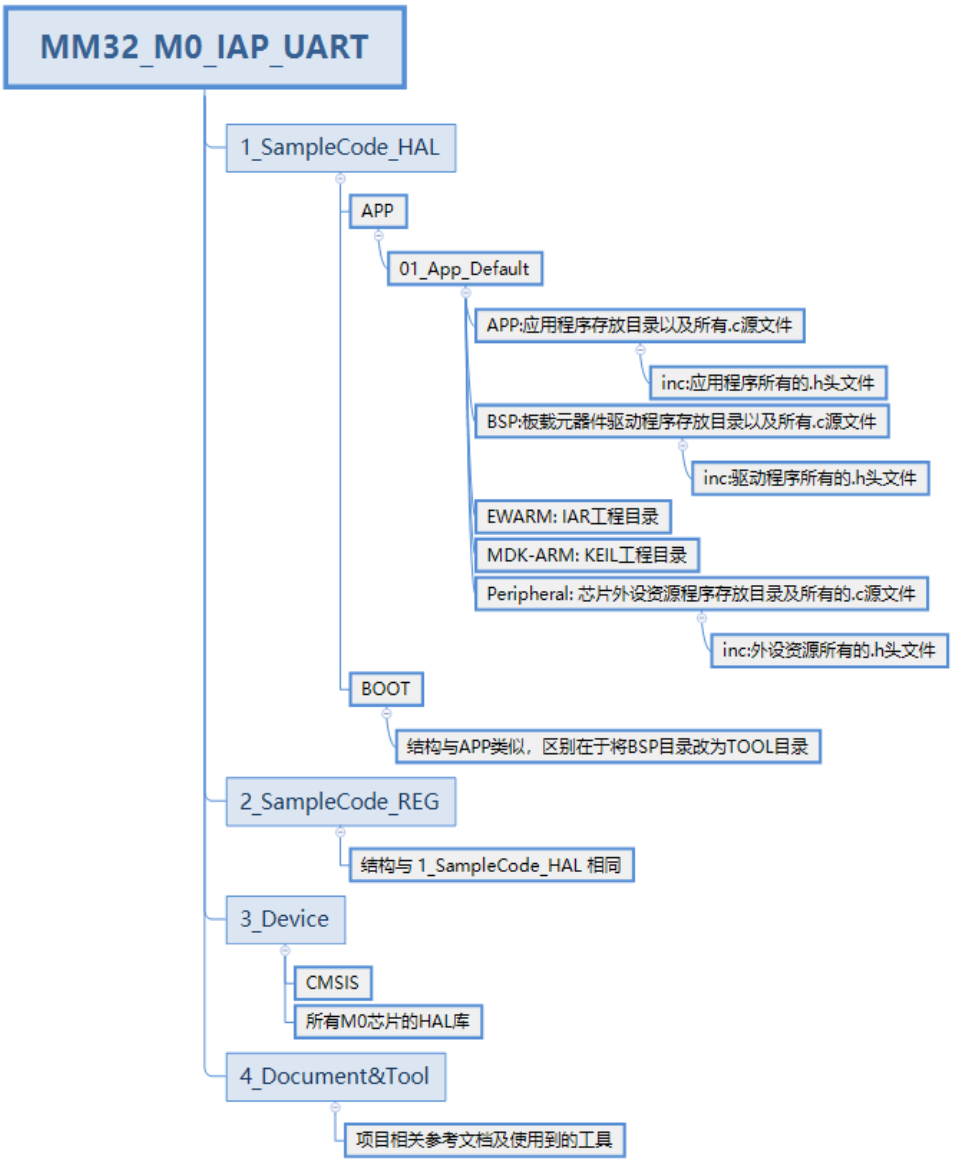

图 3 项目目录结构

整个项目中包含了 BootLoader 以及 APP 的 MDK/IAR 子工程以及所有 Device 的 workspace 工程，其中 BootLoader 还包括了以寄存器方式的实现，其作用主要是为了减小 code 占用 Flash 资源大小；还包括了参考文档以及升级助手工具。

2.2 所用外设资源

在本 Demo 中主要涉及到的 M0 外设资源为 Flash、UART、Timer，以下简单介绍这几个 IP 功能。

2.2.1 Flash

在 MM32 M0 所有系列产品中，片上 Flash 的 IP 设计保持一致，闪存空间由 64 位宽的存储单元组成，既可以存代码又可以存数据。以 128K 大小 MCU 为例，主闪存块按 128 页 (每 页 1K 字节) 或 32 扇区 (每扇区 4K 字节) 分块，以扇区为单位设置写保护，可以按照字节、半字以及字的数据结构对主闪存空间进行读数据，可以按照半字以及字的数据结构对主闪存空间进行写数据。写数据前必须先擦成 0xFF,且擦除分为按页擦除和整块擦除模式。读写以及擦除的流程详见 MM32 M0 MCU UM 手册。

在本 Demo 中，需要将主 Flash 划分为 3 个区域，BootLoader 代码区、升级参数区、APP 代码区，以寄存器方案的 Keil 工程为例，3 个区域划分情况如下，具体可以根据实际需要再另做划分：

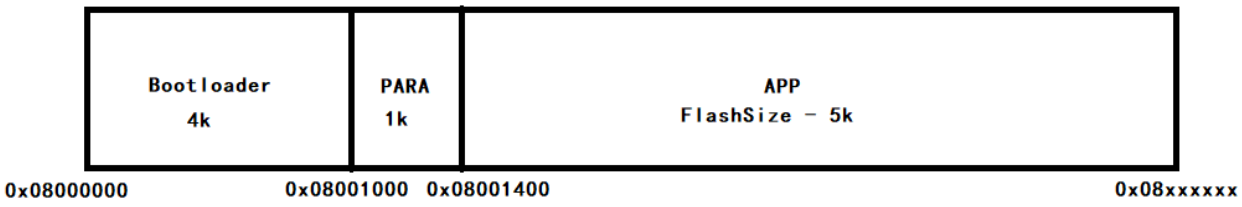

图 4 Flash 用户空间划分

2.2.2 UART

通用异步收发器 (UART) 提供了一种灵活的方法与使用工业标准 NRZ 异步串行数据格式的外部设备之间进行全双工数据交换。UART 利用分数波特率发生器提供宽范围的波特率选择。它支持同步单向通信和半双工单线通信，以及调制解调器 (CTS/RTS) 操作。

在本 Demo 中，使用到 UART 作为预留数据接收、发送接口，可以使用查询、中断以及 DMA 方式进行数据传输。

2.2.3 Timer

在本 Demo 中，使用到 Timer 的更新中断搭配数据传输接口使用，作为超时判断的计数器。

2.3 程序执行流程

在正常情况下，芯片一上电后程序运行流程如下：

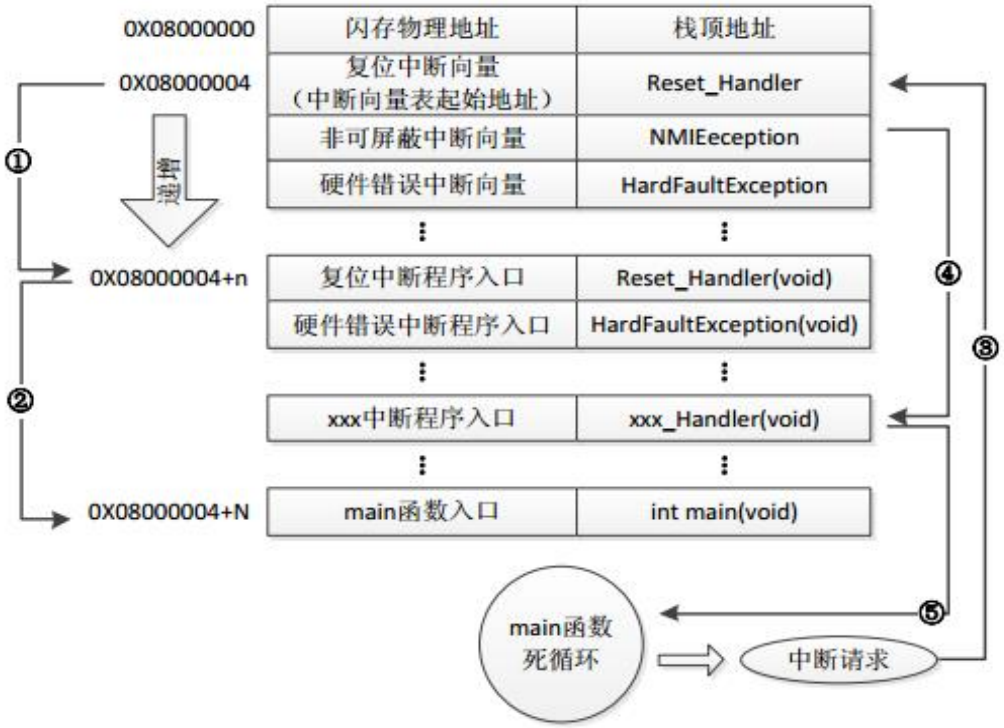

图 5 不加 BooLoader 程序运行流程示意

MM32 M0 MCU 内部主闪存地址起始于 0x08000000，一般情况下，程序文件就从此地 址开始写入，通过一张“中断向量表” 来响应中断，程序启动后，将首先从“中断向量表”取出复位中断向量，执行复位中断程序完成启动，而这张“中断向量表”的起始地址是 0x08000004，当中断来临，MCU 内部硬件机制亦会自动将 PC 指针定位到“中断向量表”处，并根据中断源取出对应的中断向量执行中断服务程序。

当复位后，先从 0x08000004 地址取出复位中断向量的地址，并跳转到复位中断服务程序，如图标号①所示；在复位中断服务程序执行完之后，会跳转到 main 函数，如图标号②所示；而我们的 main 函数一般都是一个死循环，在 main 函数执行过程 中，如果收到中断请求（发生重中断），此时 MCU 强制将 PC 指针指回中断向量表处，如图标号③所示；然后，根据中断源进入相应的中断服务程序，如图标号④所示；在执行完中断服务程序以后，程序再次返回 main 函数执行，如图标号⑤所示。

加入 BootLoader 程序之后，程序运行流程如下：

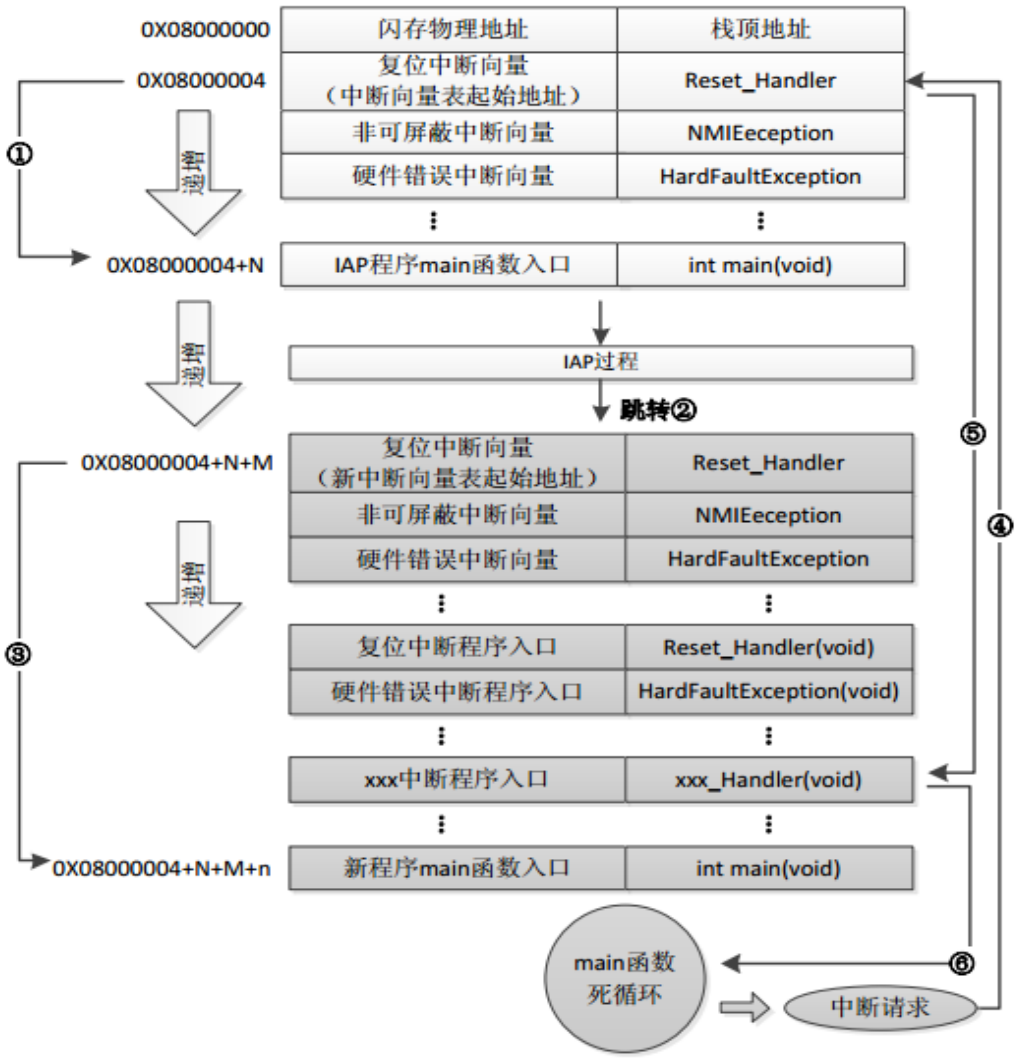

图 6 加入 BooLoader 程序运行流程示意

MCU 复位后，还是从 0x08000004 地址取出复位中断向量的地址，并跳转到复位中断服务程序，在运行完复位中断服务程序之后跳转到 BootLoader 的 main 函数，如图标号①所示，在执行完 BootLoader 以后，跳转至新写入程序的复位中断向量表（0X08000004+N+M--例如 9000），取出新程序的复位中断向量的地址，并跳转执行新程序的复位中断服务程 序，随后跳转至新程序的 main 函数，如图标号②和③所示，同样 main 函数为一个死循环，并且注意到此时 MCU 的 Flash，在不同位置上，共有两个中断向量表。

在 main 函数执行过程中，如果 CPU 得到一个中断请求，PC 指针仍强制跳转到地址 0x08000004 中断向量表处，而不是新程序的中断向量表，如图标号④所示；程序再根据我们设置的中断向量表偏移量，跳转到对应中断源新的中断服务程序中，如图标号⑤所示；在执行完 中断服务程序后，程序返回 main 函数继续运行，如图标号⑥所示。

BootLoader 和 APP 结合起来，实际的工作流程如下：

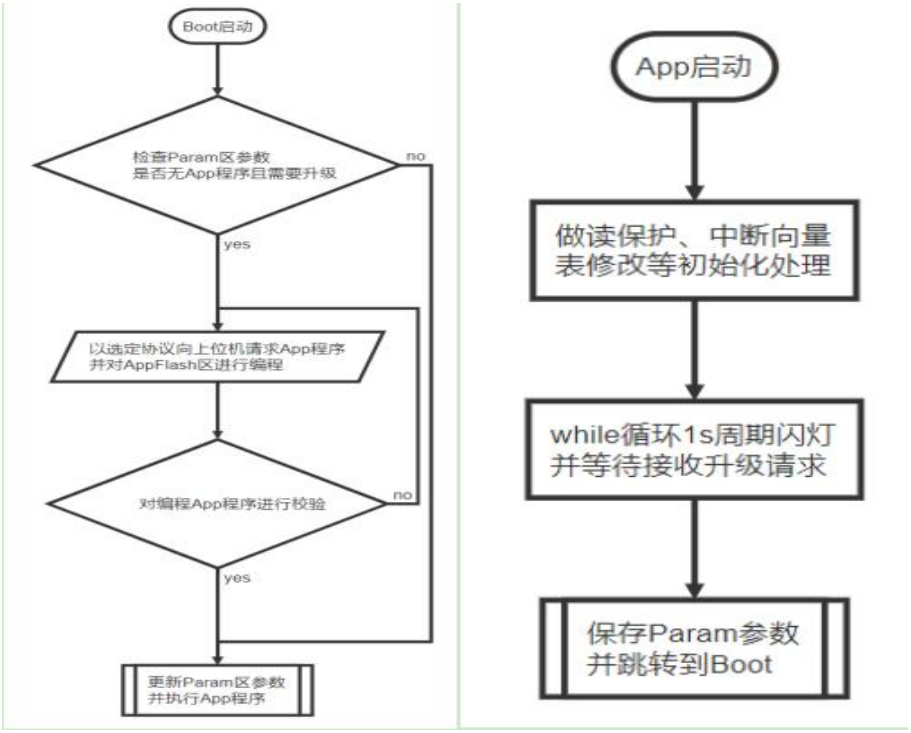

图 7 IAP_UART Demo 程序运行流程示意

2.4 工程源码及配置

本小节主要介绍 Keil 及 IAR 工程设置、中断向量表处理代码以及跳转代码的实现、Flash 全片读保护设置，以及对其它.c 文件中重要函数作说明。

2.4.1 BootLoader 工程

以寄存器方式为例进行说明。其中，Keil 工程包括了不同系列的 MM32 M0 MCU 的多个不同 Target，在进行编译前需要先选定实际应用到 MCU 目标，类似地，IAR 工程是将所有不同 MCU 的子工程集合成了 workspace 工作站，需要将想要编译的工程设置为 Active。

ROM 设置起始地址为 0x08000000，RAM 设置起始地址为 0x2000000+VectSize*4（中断向量表大小*4 字节）。Keil 工程中的 IROM1 和 IRAM1 需要做如下设置：

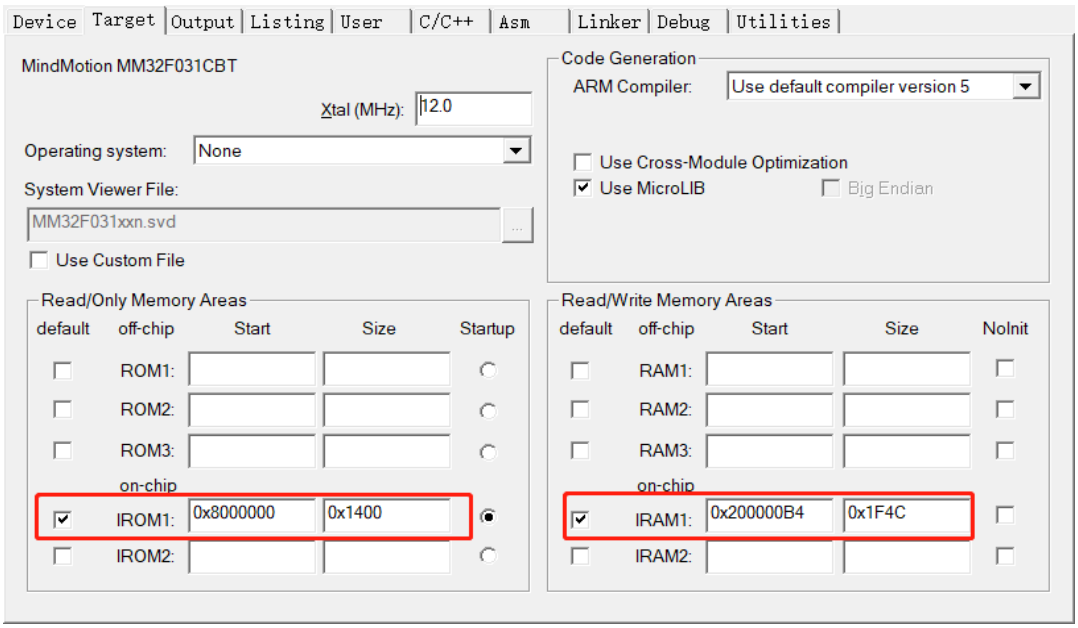

图 8 Keil 工程设置

IAR 工程中的.icf 文件需要做如下修改设置：

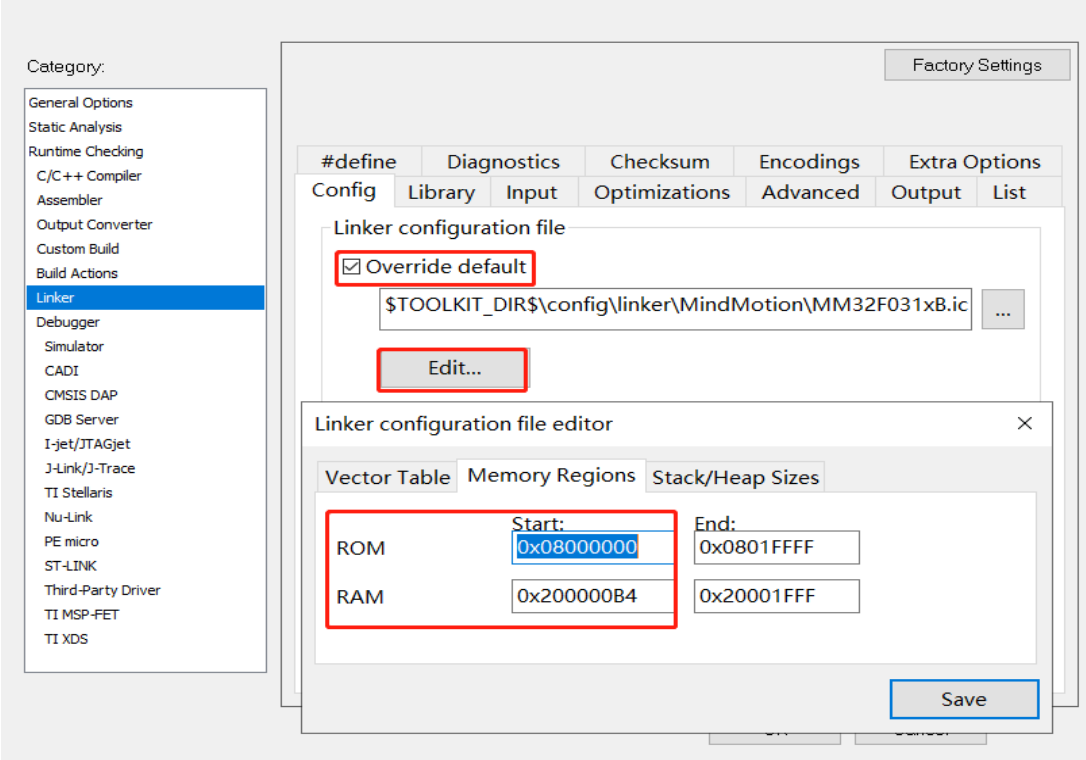

图 9 IAR 工程设置

单个 BootLoader 工程目录结构以及源文件功能说明如下：

- BOOT_startup.s: 启动文件，中断向量表的定义和处理在其中进行，不同于官网例程库中普通的.s 文件
- main.c: BootLoader 功能运行入口，从.s 启动后运行到 main 函数
- m0_isr_redefine.c: 中断服务函数重新指向新的入口函数功能定义，实现中断服务函数的跳转

- xmodem.c/ymodem.c/binary.c/ascii.c: 传输协议实现，接口函数为 xxxx_Receive()
- check.c/queue.c: 传输协议实现过程用到的工具函数定义，例如：CRC-16 校验、累加和校验、缓冲队列
- uart.c: 以串口中断、轮训的方式接收和发送数据
- tim3.c: 定义频率为 2kHz 的接收超时计数时钟
- flash.c: 读写和擦除内部 Flash 的接口函数定义

中断向量表的处理是 IAP 升级的关键，其处理方式以及关键代码包括以下几点：

1) 改写官网库例程中的.s 文件为 startup_BOOT_MM32xx.s，其包含了定义 APP 起始地址的 asm_config.h，在工程设置中需要设置 Asm 选项卡中的 Include 路径地址，Keil 中如下:

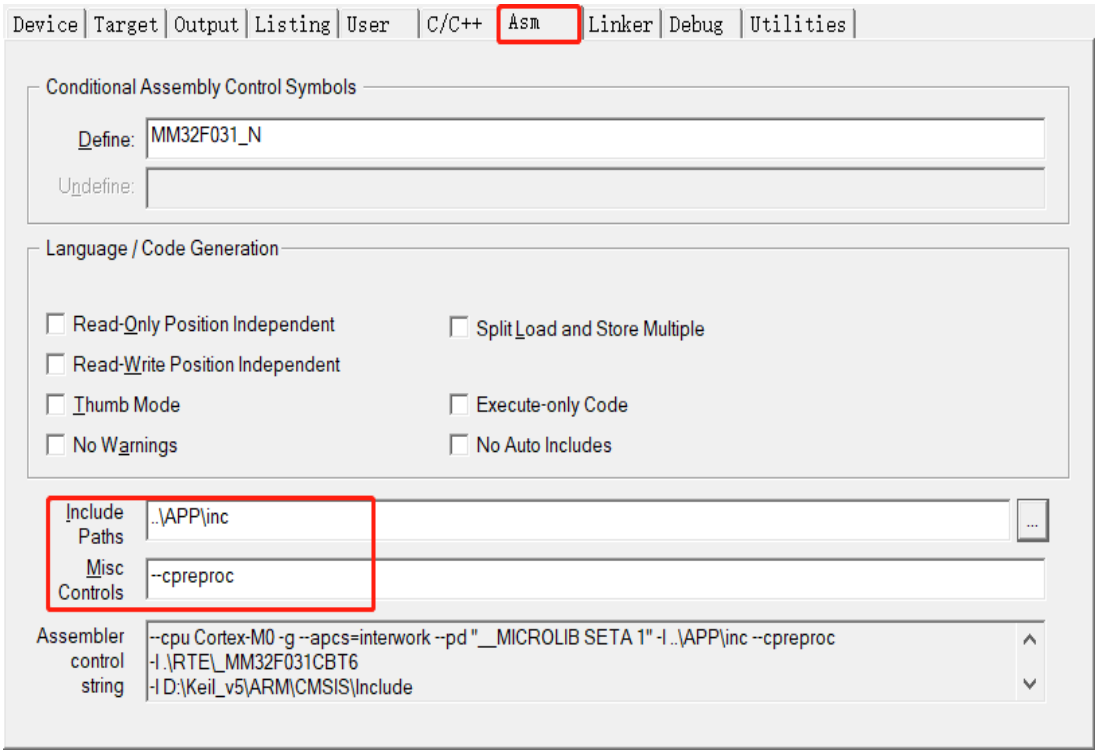

图 10 Keil 工程 Asm 设置

```
.s 文件修改如下：

#include "asm_config.h"

;         IMPORT  SystemInit      ;屏蔽 SystmInit

ApplicationStart

            LDR     R0,=0x4002101C

            LDR     R1,=0x04000000

            STR     R1, [R0]

            LDR     R0,=0x40006818

            LDR     R1,=0x00000012

            STR     R1, [R0]
```

```
            LDR     R0,=0x4000681C
            LDR     R1,=0x00000012
            STR     R1, [R0]
            LDR     R0,=0x4002101C
            LDR     R1,=0x00000000
            STR     R1, [R0]
;           LDR     R0, =SystemInit
;           BLX     R0
            LDR     R0, =__main
            BX      R0
            ENDP
NMI_Handler     PROC
                EXPORT  NMI_Handler                 [WEAK]
                LDR R0, = APPADDR_NMI_Handler
                LDR R1,[R0]
                BX      R1
                ;B       .
                ENDP
HardFault_Handler\
                PROC
                EXPORT  HardFault_Handler           [WEAK]
                LDR R0, = APPADDR_HardFault_Handler
                LDR R1,[R0]
                BX      R1
                ;B       .
                ENDP
PendSV_Handler  PROC
                EXPORT  PendSV_Handler              [WEAK]
                LDR R0, = APPADDR_PendSV_Handler
                LDR R1,[R0]
```

```
                BX      R1
                ;B       .
                ENDP
SysTick_Handler PROC
                EXPORT  SysTick_Handler                [WEAK]
                LDR R0, = APPADDR_SysTick_Handler
                LDR R1,[R0]
                BX      R1
                ;B       .
                ENDP
```

2) 包含重新定义了中断向量表入口地址的.h 文件，以函数指针方式可以重新定位到对应中断服务函数。

```
#include "m0_interrupt_table_redefine.h"
```

3) 将重新定义的中断向量表固定于 RAM 起始位置 0x20000000 处。

```
#ifndef USE_IAR
NVIC_TABLE_t tNVIC_Table __attribute__((at(0x20000000)));
#else
NVIC_TABLE_t tNVIC_Table __attribute__((section(".ARM.__at_0x20000000")));
#endif
```

4) 在 m0_isr_redefine.c 中实现在 ISR 中重新跳转到真正意义上的服务函数中去，如：

```
void TIM3_IRQHandler(void)
{
    tNVIC_Table.pTIM3_IRQHandler();
}
```

5) 最后在外设初始化时需要指定新的中断服务函数入口地址，以及定义具体实现函数，如：

```
tNVIC_Table.pTIM3_IRQHandler = TIM3_Processing;
/*****************************************************************************
 * @brief  Update interrupt handler
 * @param  None
 * @retval  None
 * @attention  This is the actual TIM3_UP interrupt processing entry
```

```
*****************************************************************************/

void TIM3_Processing(void)

{

    if ((TIM3->SR & 0x01U) != RESET)

    {

        /* Clear the IT pending Bit */

        TIM3->SR = (uint16_t)~(0x01U);

        g_recTimeOutFlag = UART_TIME_OUT;

    }

}
```

实现从 BootLoader 跳转到 APP 的接口函数为：

```
void Iap_Jump_To_Address(uint32_t wUserFlashAddr) ;  //只需要传入 APP 起始地址
```

2.4.2 APP 工程

ROM 设置起始地址为 APPLICATION_ADDRESS，RAM 设置起始地址为 0x2000000+VectSize*4（中断向量表大小*4 字节）。Keil 工程中的 IROM1 和 IRAM1 需要做如下设置：

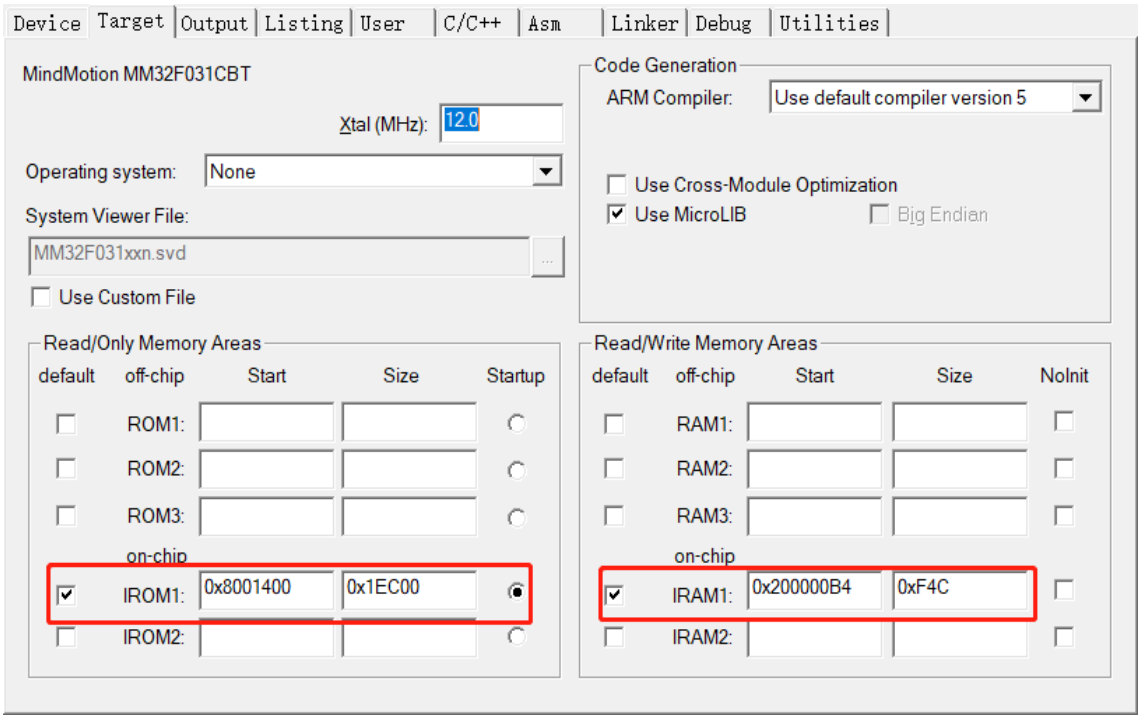

图 11 Keil 工程设置

IAR 工程中的.icf 文件需要做如下修改设置，其中 Vector Table 也需要设置为 ROM 设置起始地址：

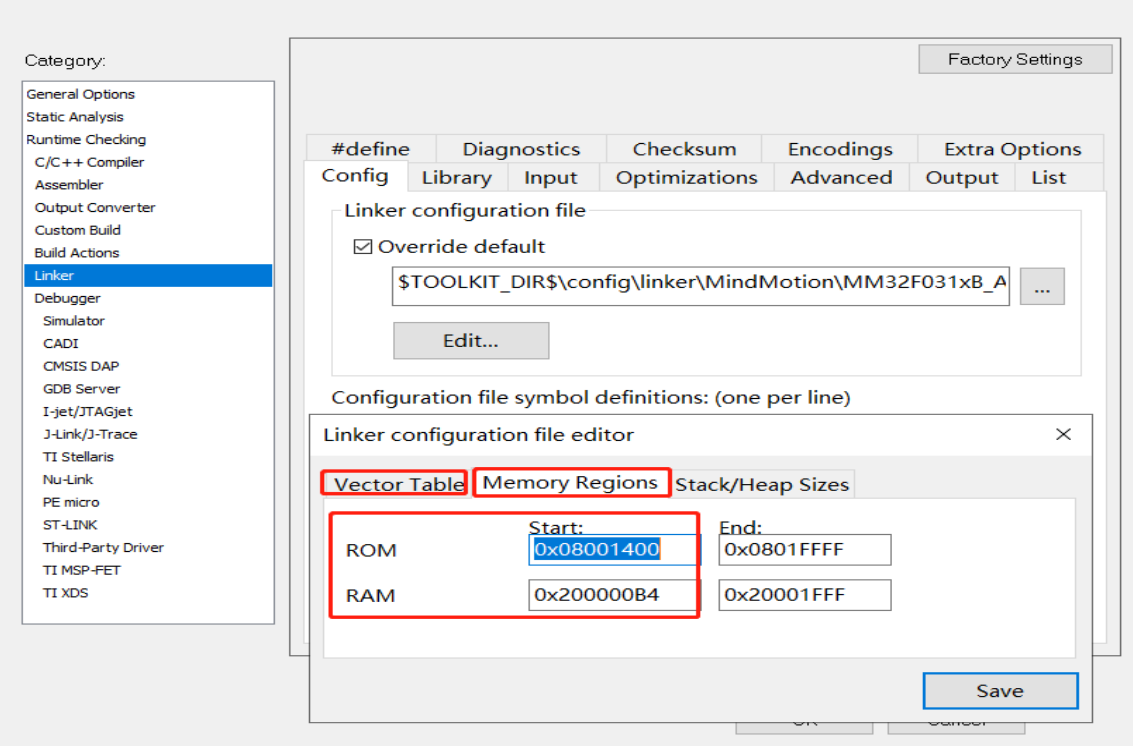

图 12 IAR 工程设置

在 APP 中业务逻辑功能主要为 1s 周期闪灯，用到 systick 中断和 timer 更新中断，还可以设置全片读保护，与读保护相关的接口函数为：

```
Flash_CheckReadProtect() ;  //判断是否存在读保护

Flash_EnableReadProtect();  //设置全片读保护功能
```

在 APP 的中断向量处理与 BootLoader 的相类似，不同之处为需要使用 startup_APP_MM32xx.s 以及无需包含 m0_isr_redefine.c 文件，其它步骤都必须包括，.s 文件修改如下：

```
ApplicationStart

            LDR     R0,=0x4002101C

            LDR     R1,=0x04000000

            STR     R1, [R0]

            LDR     R0,=0x40006818

            LDR     R1,=0x00000012

            STR     R1, [R0]

            LDR     R0,=0x4000681C

            LDR     R1,=0x00000012

            STR     R1, [R0]

            LDR     R0,=0x4002101C
```

```
        LDR     R1,=0x00000000
        STR     R1, [R0]
        LDR     R0, =SystemInit
        BLX     R0
        LDR     R0, =__main
        BX      R0
        ENDP
```

2.5 移植说明

在对本 Demo 的 BootLoader 和 APP 做移植时，除了必须的底层外设驱动之外，还需要关注上述的工程配置以及中断向量表的处理，每一项操作都是必须的，否则会导致无法正常使用。

另外，样例中包含了 config.h 文件，里面包含了整个工程的参数配置，可以改变串口定义及波特率选择，可以使用不同的传输协议，可以选择是否开启 APP 数据校验以及初次烧录是否合并了 APP 代码等等，用户可以根据实际需要进行修改宏定义。在 Flash.h 中包含了芯片的 Flash 大小定义，可结合实际情况更改。

2.6 注意事项

用户实际使用中需要注意以下几点：

- 如果可能尽量使用带有传输管理的协议进行升级，直接传输文件的 bin 或者 hex 方式对软硬件条件要求较高，且失败风险较大。
- BootLoader 程序段如果被破坏，产品必须返厂才能重新烧写程序，这是很麻烦并且非常耗费时间和金钱的。针对这样的需求，可以将 BootLoader 程序区域进行设置写保护，防止被意外地破坏。
- Bootloader 中尽可能不使用中断，特别是系统级别的中断如 SystickHandler。
- Bootloader 和 APP 工程中针对中断向量的中转处理必须完整包括，否则在用到中断时系统可能会卡死。
- 在擦写 Flash 之前要注意将其它中断全部关闭，避免其它中断的产生会给 Flash 操作带来意外干扰致使操作失败。
- 对 Flash 的分区需要根据实际情况去做调整，本 Demo 针对的是寄存器方式的 BootLoader 划分情况。
- 在 APP 工程中可以更改 ConstData 数组的大小，从而更改 APP 程序大小。
- BootLoader 使用的是系统默认的 HSI 8M 时钟，以及总线分频系数都是为默认的不分频，用户可自行修改时钟配置。

3. IAP_UART 功能测试

以 MM32SPIN27PF（P5）这颗芯片的 Keil 工程为例，来实际操作一遍 IAP_UART Demo，以及测试 IAP 功能还需要结合一些助手工具，这里也将简单介绍。

3.1 编译 BootLoader 与 APP 代码

打开 BootLoader 寄存器版项目工程，选择好 MM32SPIN27PF Target，在.config.h 中修改全局参数，选择使用 Xmdem 传输协议，且无需校验 bin 数据，打开调试信息以及出厂时烧录合并文件选项使能，而后进行编译得到 MM32SPIN27_P_BOOT_UART.hex。

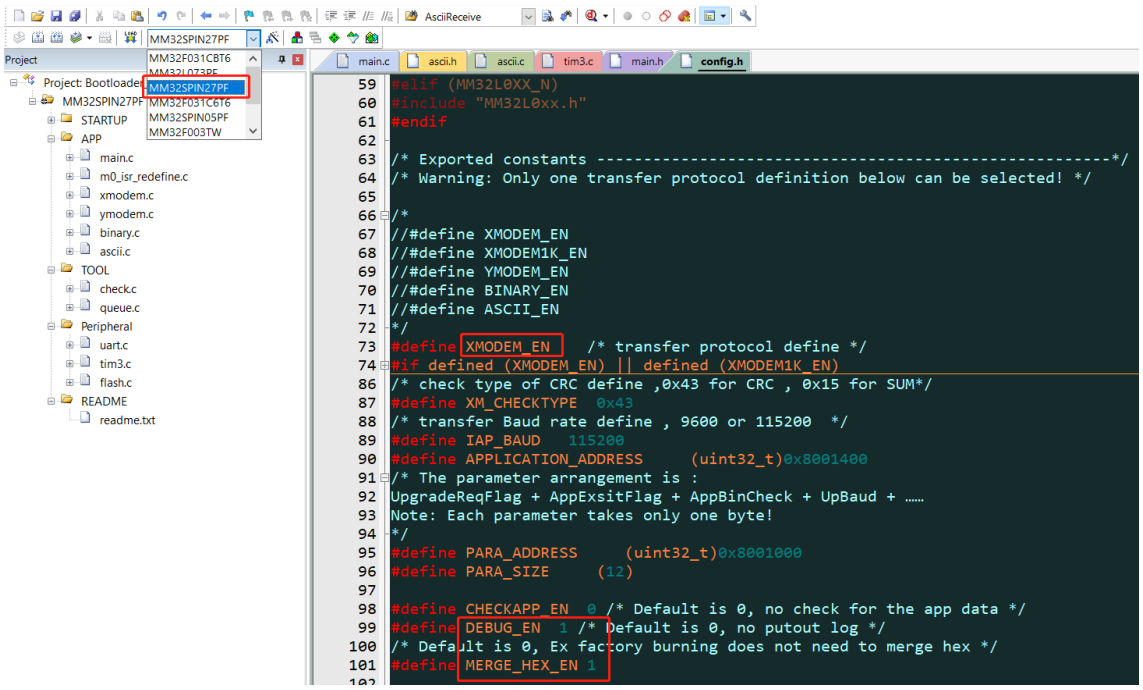

图 13 Keil 工程预编译选项

打开 APP 工程，选择好 MM32SPIN27PF Target，在工程设置中的 User 选项中，配置生成.bin 文件，而后进行编译得到 MM32SPIN27_P_APP_DEFAULT.hex 和 MM32SPIN27_P_APP_DEFAULT.bin 文件。

```
详细命令参数为:  fromelf --bin "$L@L.axf" --output "$L@L.bin"
```

也可以使用 HexToBin 文件工具将 MM32SPIN27_P_APP_DEFAULT.hex 转成 MM32SPIN27_P_APP_DEFAULT.bin 文件。

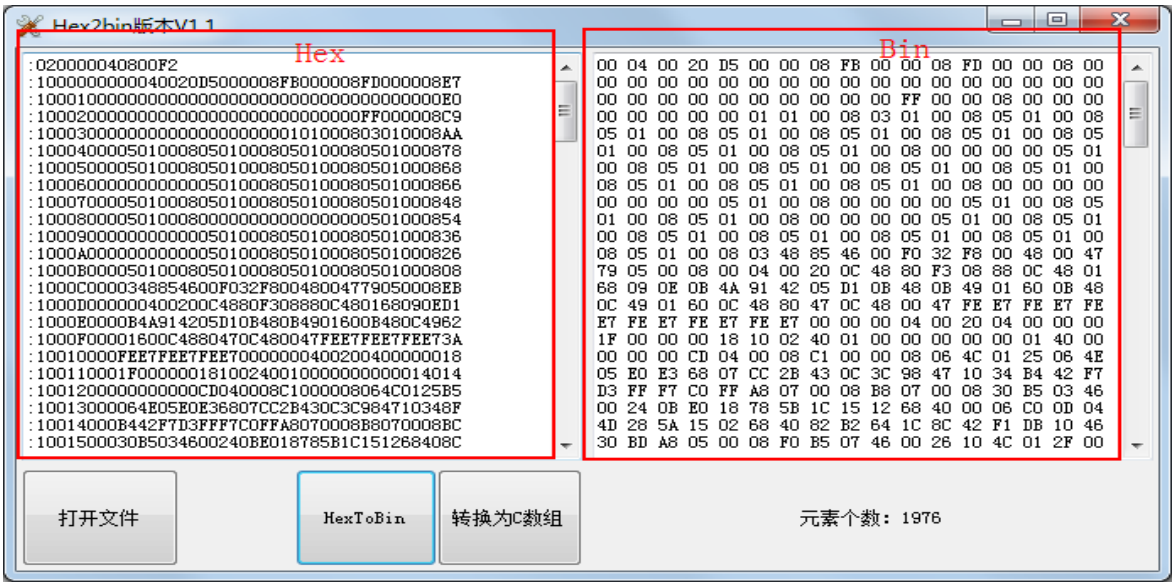

图 14 HexToBin 工具

3.2 HEX 文件合并烧录

将编译好的BootLoader与APP .hex文件用HEX合并工具进行文件合并，得到UART_IAP_Merge.hex，然后使用 MM32-Link 工具和配套的 MM32-Link Program 上位机将合并程序文件烧录到开发板中，烧录前需要将芯片擦除全片保证升级参数区是出厂时候的全 0xFF 数据。HEX 合并工具使用步骤如下：

1) 双击 merge_Version.bat 运行。

2) 版本要输入数字。比如 10,20,30,31，分别代表 1.0,2.0,3.0,3.1

3) hex 文件放置在 hex 文件夹下，boot 固件地址在前，app 固件地址在后。

4) 同时生成两个文件，文件是以当前时间命名的。一个是合并后的 hex 文件，是刷机用的文件；另一个是 bin 文件是 app.hex 转成 bin 格式的文件，是用来进行 IAP 升级用的文件。

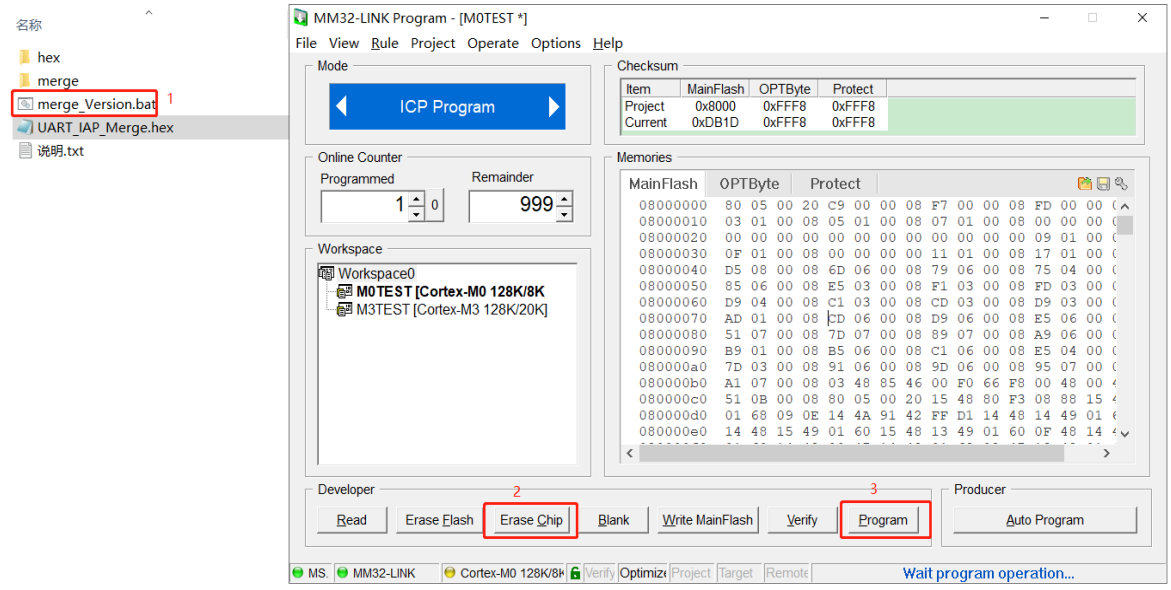

图 15 MM32-Link Program 编程操作

3.3 应用程序更新

烧录好合并的程序后进行断电复位，可以看到开发板上的指示灯每秒闪烁一次（接到 PB5 引脚上），且串口输出调试信息：

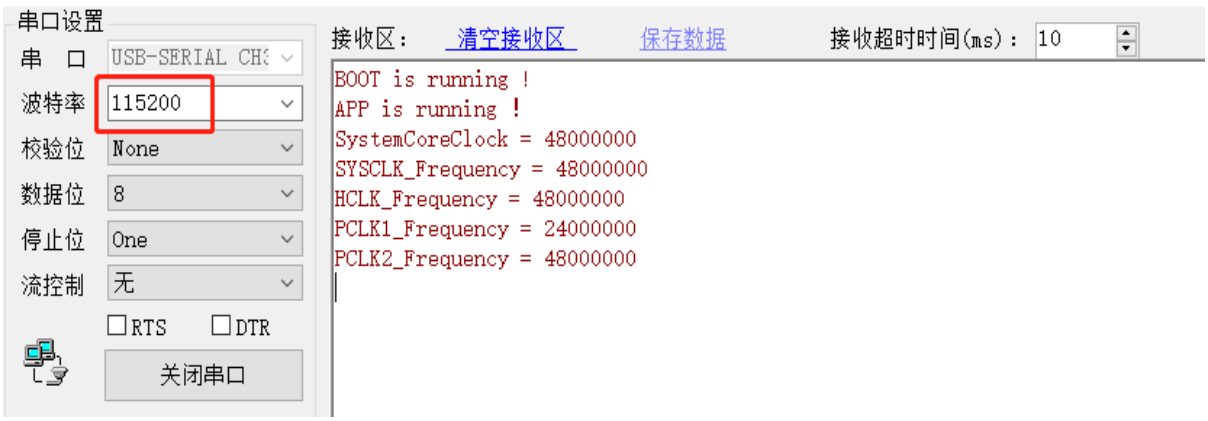

图 16 开发板上电后首次打印信息

往串口发送测试时候用到的升级命令包， 55 01 00 4B 01 88 99 AA BB CC FF ，其中 55 和 最后的 FF 为帧头和帧尾，接着 55 后面的 4 个参数为 UpgradeReqFlag + AppExsitFlag + AppBinCheck + UpBaud 。收到升级命令包后，会跳转到 BootLoader 运行升级程序，并打印出相关调试信息：

图 17 开发板接收升级命令

在 MM32 MCU IAP TOOL 中选择文件发送模式，且选择使用 Xmodem 协议，将 MM32SPIN27_P_APP_DEFAULT.bin 文件开始传输给开发板进行升级，2 秒钟不到的时间完成后升级成功，串口会打印调试信息，开发板指示灯每秒钟闪烁一次运行：

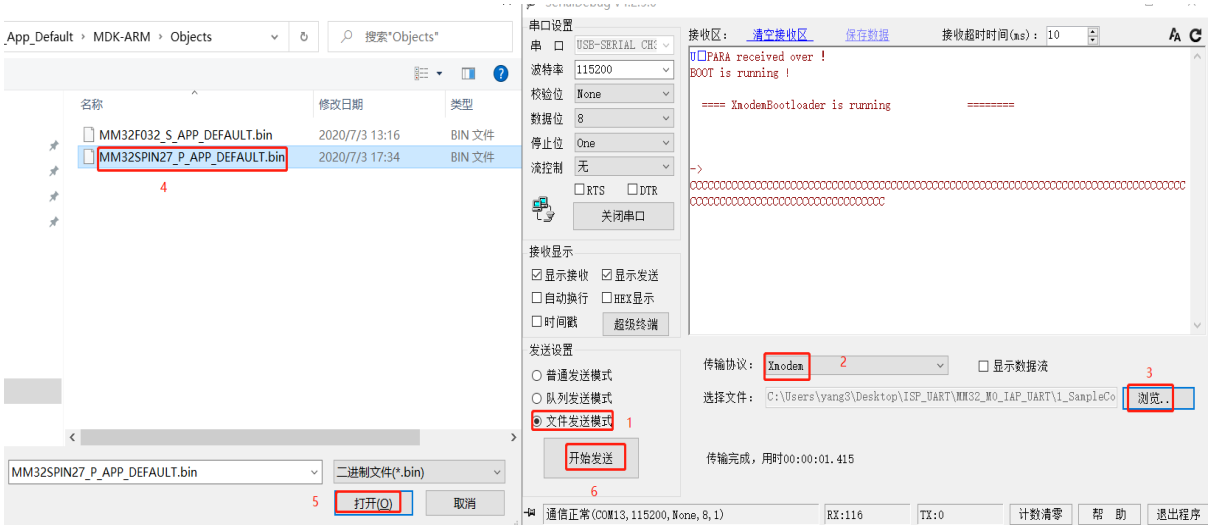

图 18 开发板接收升级文件

见到以下调试信息输出，整个升级过程成功结束：

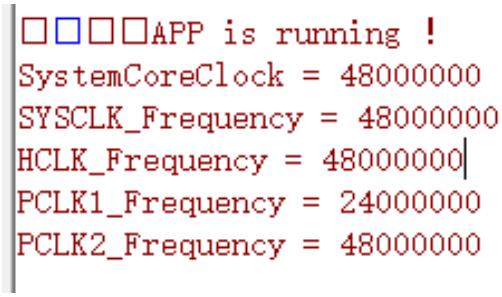

图 19 开发板接收成功跳转到 APP

4.Revision Record 修订记录

Revision Document	Author	Date	Approved by	Modification
V1.0	Willow	2020/07/06	Peter	release