**AN\_MM32 IAP UART\_sc**

目录

[1. IAP需求及基础介绍 1](#_Toc26743)

[1.1 不同编程方式简介 1](#_Toc6342)

[1.1.1 ISP 1](#_Toc29630)

[1.1.2 ICP 1](#_Toc29466)

[1.1.3 IAP 1](#_Toc2156)

[1.2 不同传输协议简介 1](#_Toc436)

[1.2.1 Xmodem 1](#_Toc20436)

[1.2.2 Ymodem 4](#_Toc18047)

[1.2.3 Binary 5](#_Toc21131)

[1.2.4 Ascii 5](#_Toc25771)

[1.3 IAP实现原理 6](#_Toc14386)

[2. IAP\_UART项目工程详解 7](#_Toc23002)

[2.1 项目目录结构 7](#_Toc12885)

[2.2 所用外设资源 8](#_Toc31225)

[2.2.1 Flash 8](#_Toc21374)

[2.2.2 UART 8](#_Toc2703)

[2.2.3 Timer 8](#_Toc23471)

[2.3 程序执行流程 8](#_Toc450)

[2.4 工程源码及配置 11](#_Toc3553)

[2.4.1 BootLoader工程 11](#_Toc20079)

[2.4.2 APP工程 16](#_Toc32310)

[2.5 移植说明 18](#_Toc31230)

[2.6 注意事项 18](#_Toc1792)

[3. IAP\_UART功能测试 19](#_Toc14812)

[3.1 编译BootLoader与APP代码 19](#_Toc17106)

[3.2 HEX文件合并烧录 20](#_Toc30145)

[3.3 应用程序更新 21](#_Toc7214)

图片

[图 1 . Xmodem累加和方式传输流程 3](#_Toc22589)

[图 2 . Xmodem CRC方式传输流程 3](#_Toc7047)

[图 3 . 项目目录结构 7](#_Toc30572)

[图 4 . Flash用户空间划分 8](#_Toc2397)

[图 5 . 不加BooLoader程序运行流程示意 9](#_Toc21335)

[图 6 . 加入BooLoader程序运行流程示意 10](#_Toc18555)

[图 7 . IAP\_UART Demo程序运行流程示意 11](#_Toc32538)

[图 8 . Keil工程设置 12](#_Toc15485)

[图 9 . IAR工程设置 12](#_Toc28604)

[图 10 . Keil工程Asm设置 13](#_Toc7251)

[图 11 . Keil工程设置 16](#_Toc31322)

[图 12 . IAR工程设置 17](#_Toc395)

[图 13 . Keil工程预编译选项 19](#_Toc32129)

[图 14 . HexToBin工具 20](#_Toc31950)

[图 15 . MM32-Link Program编程操作 21](#_Toc14811)

[图 16 . 开发板上电后首次打印信息 21](#_Toc21422)

[图 17 . 开发板接收升级命令 21](#_Toc7694)

[图 18 . 开发板接收升级文件 22](#_Toc19359)

[图 19 . 开发板接收成功跳转到APP 22](#_Toc2529)

# 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](http://www.hqchip.com/app/1039" \t "http://www.elecfans.com/d/_blank)轻松将应用代码下载到[MCU](http://www.elecfans.com/tags/mcu/" \t "http://www.elecfans.com/d/_blank)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。

## 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。

### 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。

### 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)。

### 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。

## 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。

### 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：

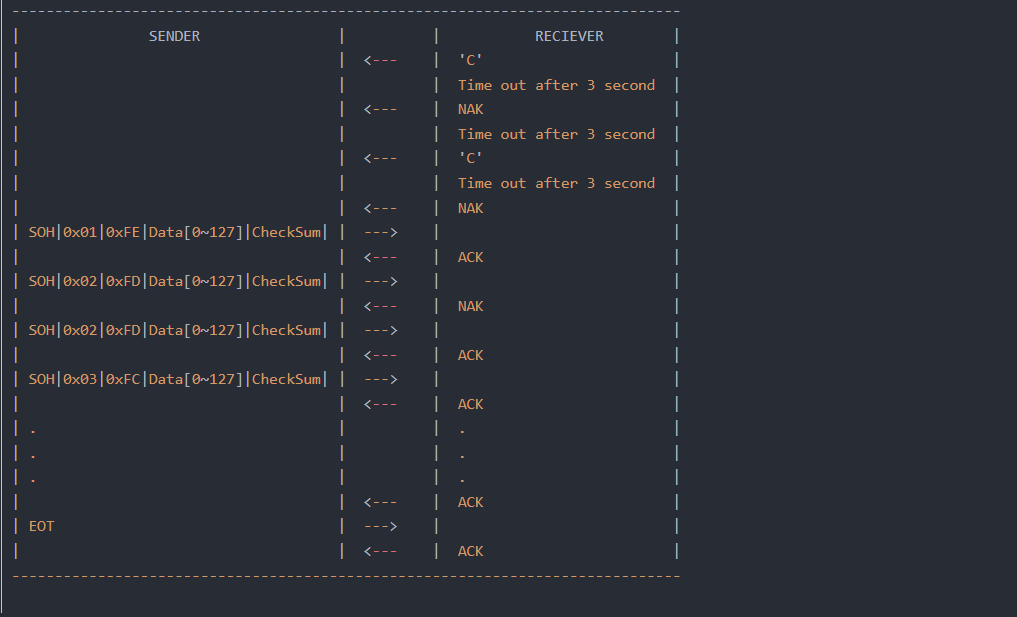


图 2. **Xmodem CRC方式传输流程**

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。

### 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

### 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。

### 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。

## 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MM32 UART IAP Demo功能的工程源码。

# 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M0产品系列**

MM32 M3 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 |
| M3 | MM32F103xxxx\_n4 / MM32L3xxxx\_n4  MM32F103xxxx\_o3 / MM32L3xxxx\_o3 |

表 2. **适用M3产品系列**

## 项目目录结构

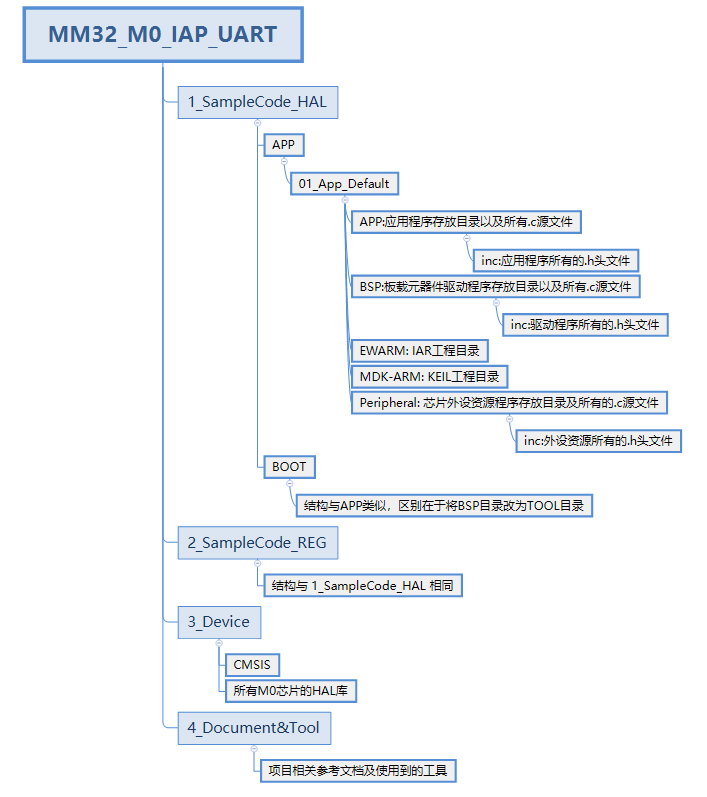
MM32 M0 IAP\_UART项目目录结构如下图所示：

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。

## 所用外设资源

在本Demo中主要涉及到的MM32 MCU外设资源为Flash、UART、Timer，以下简单介绍这几个IP功能。

### Flash

在MM32 MCU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：

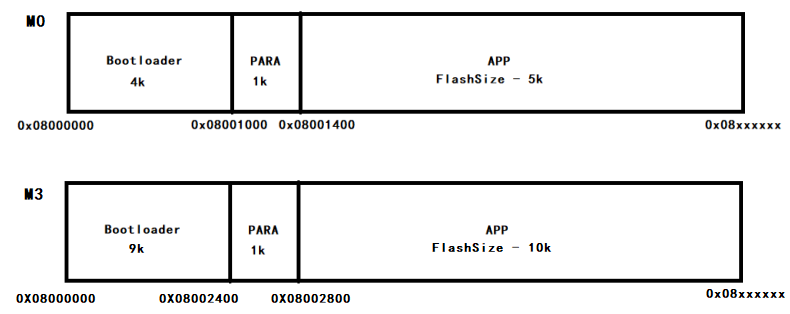


图 4. **Flash用户空间划分**

### 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。

### 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。

## 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：

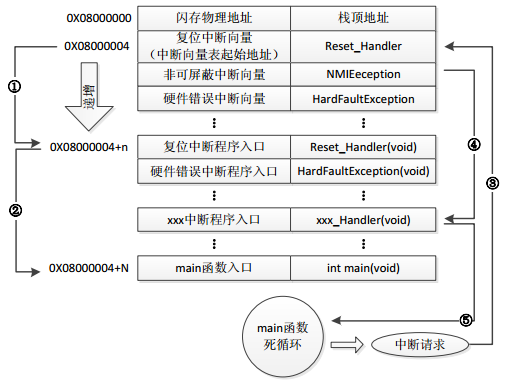


图 5. **不加BooLoader程序运行流程示意**

MM32 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：

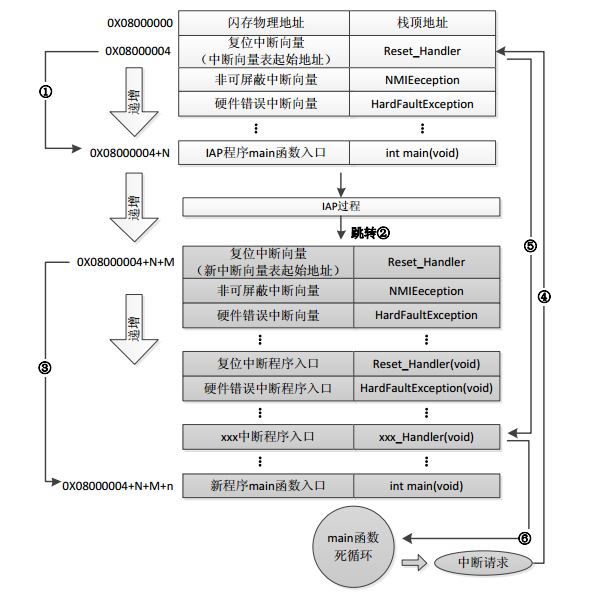


图 6. **加入BooLoader程序运行流程示意**

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：

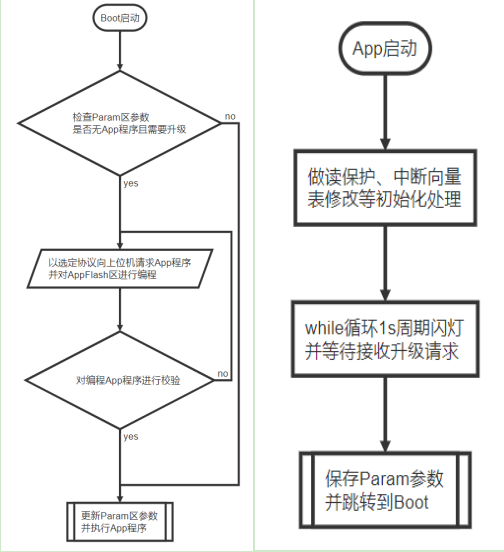


图 7. **IAP\_UART Demo程序运行流程示意**

## 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。

### 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 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：

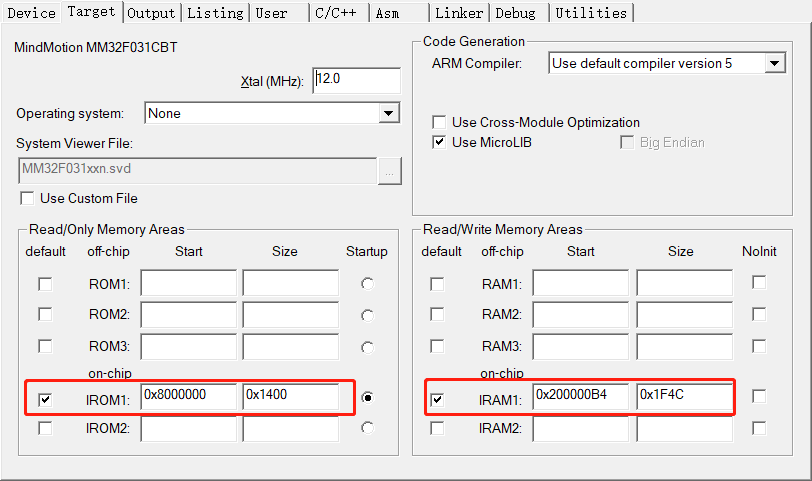


图 8. **Keil工程设置**

IAR工程中的.icf文件需要做如下修改设置：

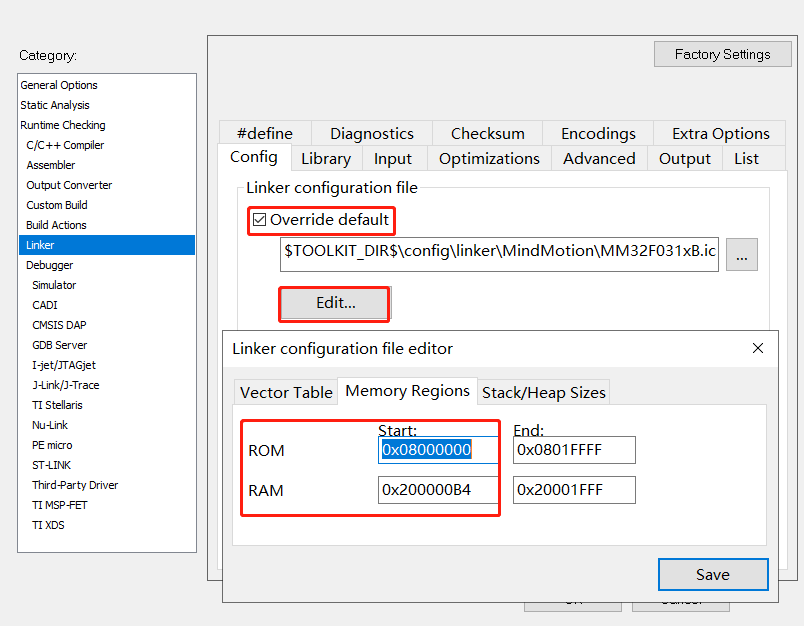


图 9. **IAR工程设置**

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，M0内核的Demo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:

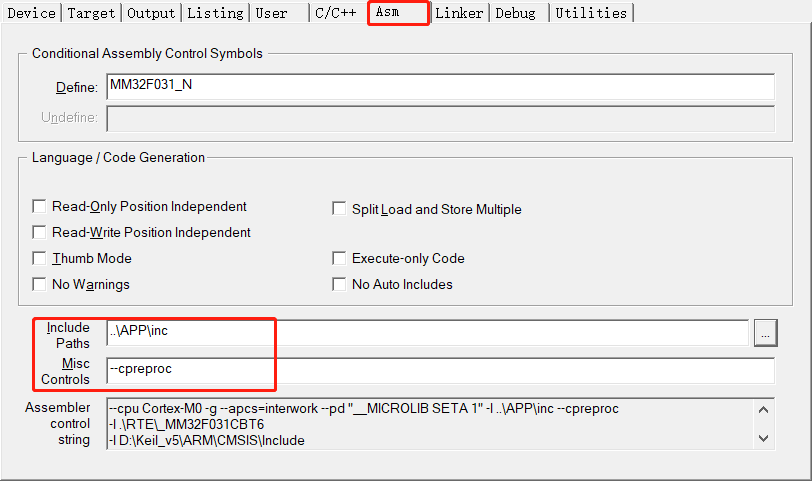


图 10. **Keil工程Asm设置**

.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

1. 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"

1. 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

1. 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();**

**}**

1. 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

### 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：

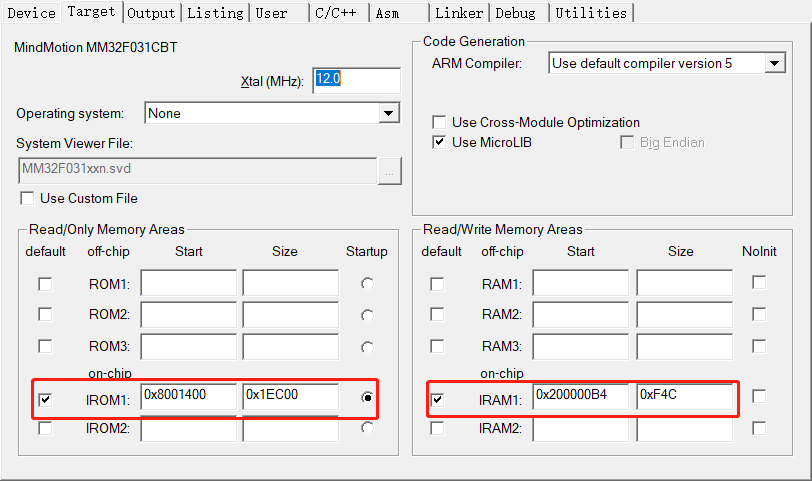


图 11. **Keil工程设置**

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：

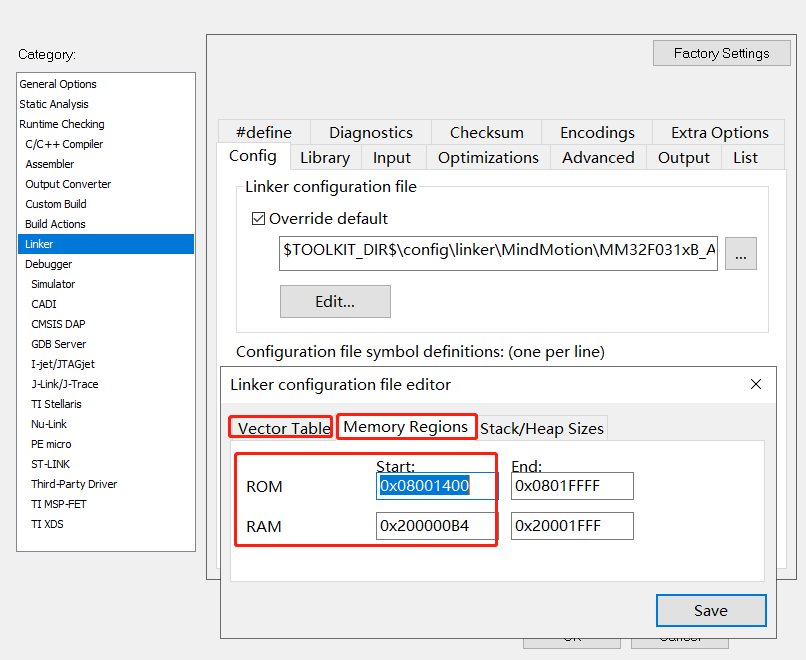


图 12. **IAR工程设置**

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

M0 Demo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

M3 Demo的中断向量表处理无需那么复杂，只需要在APP的main函数初始化时设置SCB寄存器：

/\* Vector Table Relocation in Internal FLASH.\*/

SCB->VTOR = FLASH\_BASE | FLASH\_VTOR\_OFFSET;

## 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。

## 注意事项

用户实际使用中需要注意以下几点：

* M0和M3 Demo中对于中断向量表的处理方式不一样，由于M0内核没有SCB寄存器，所以操作稍复杂。
* 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。
* M0 Demo中的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。

# 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。

## 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。

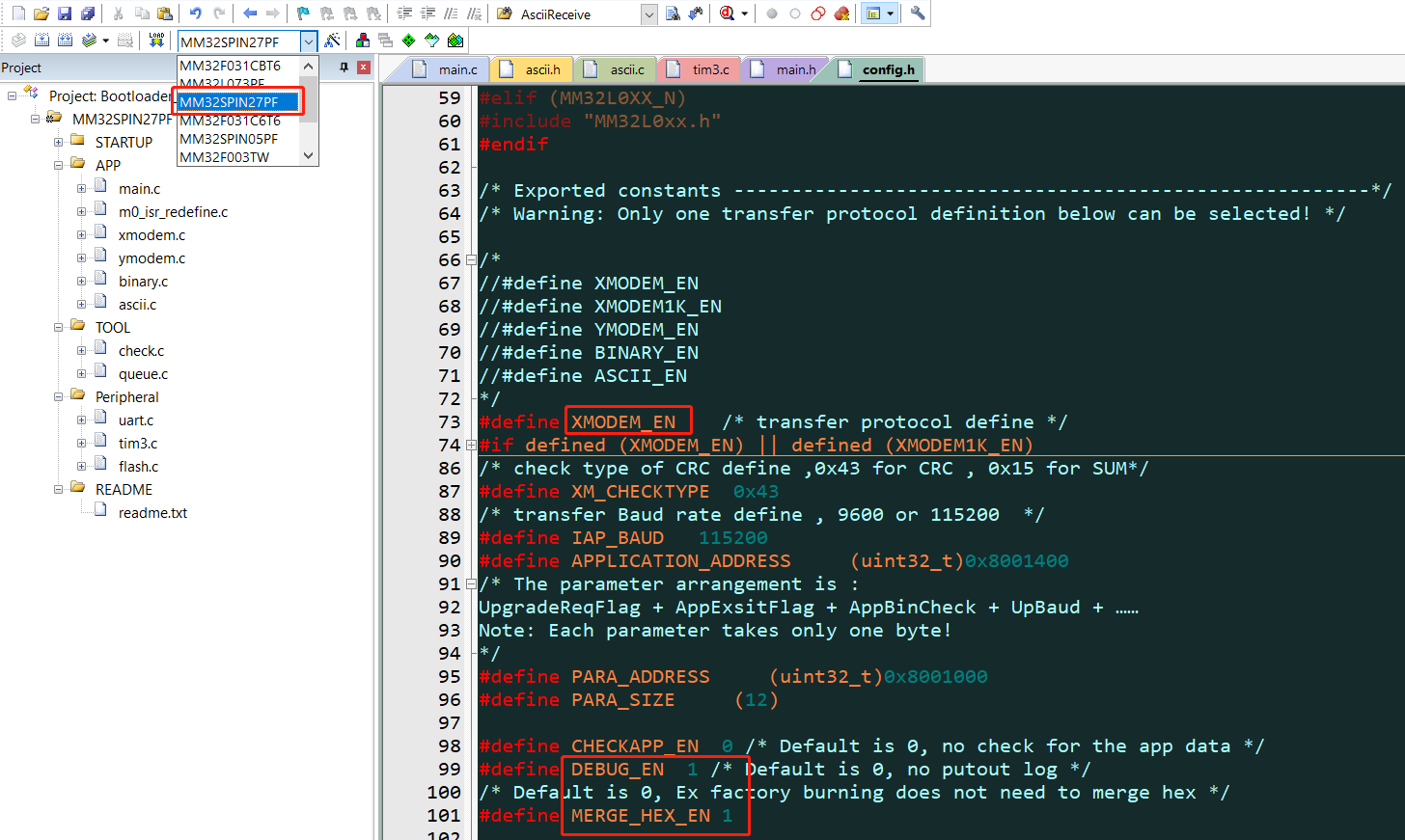


图 13. **Keil工程预编译选项**

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

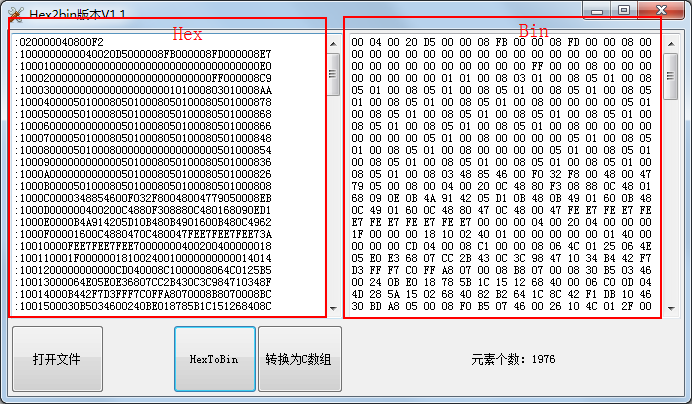


图 14. **HexToBin工具**

## 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：

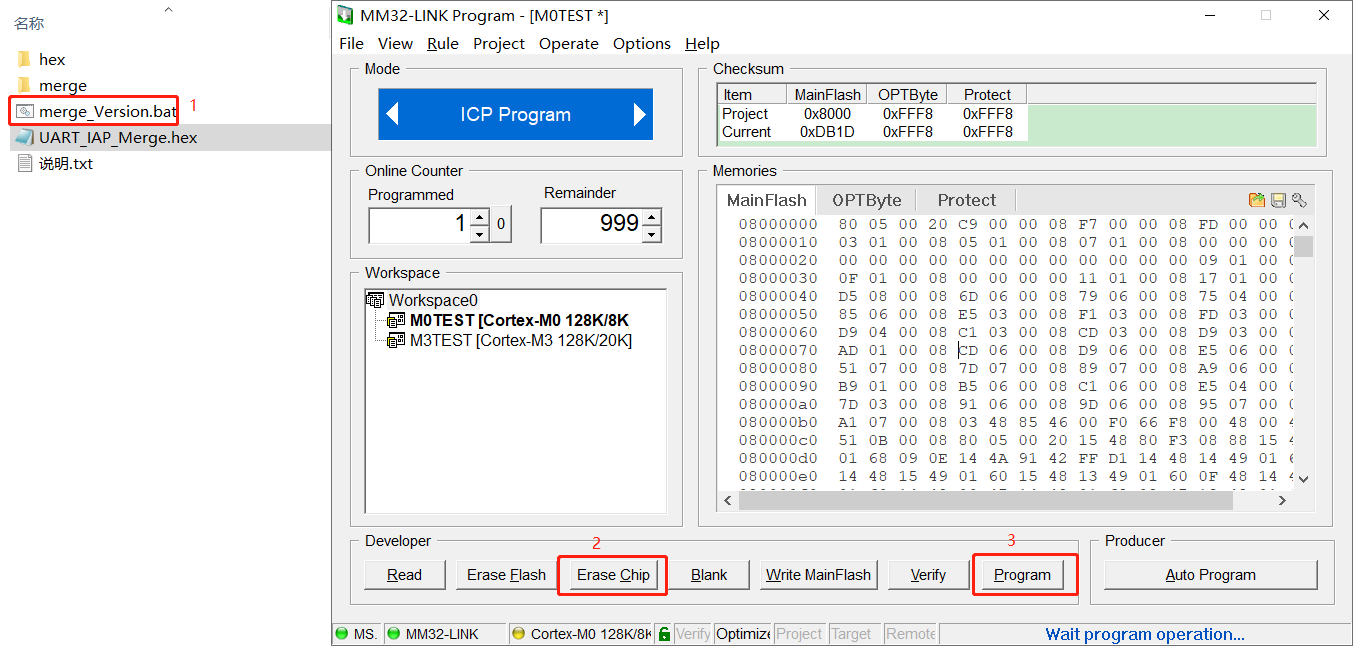
1. 双击merge\_Version.bat运行
2. 版本要输入数字。比如10,20,30,31，分别代表1.0,2.0,3.0,3.1
3. hex文件放置在hex文件夹下，boot固件地址在前，app固件地址在后。
4. 同时生成两个文件，文件是以当前时间命名的。一个是合并后的hex文件，是刷机用的文件；另一个是bin文件是app.hex转成bin格式的文件，是用来进行IAP升级用的文件。

图 15. **MM32-Link Program编程操作**

## 应用程序更新

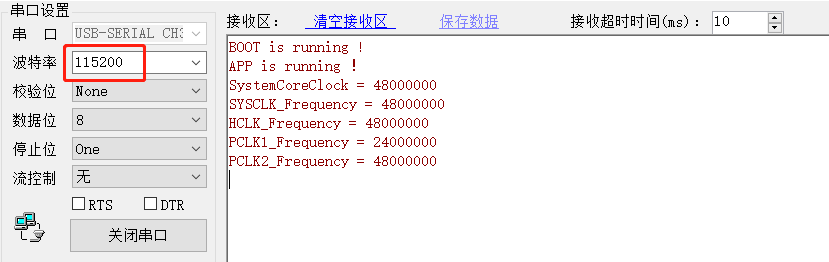
烧录好合并的程序后进行断电复位，可以看到开发板上的指示灯每秒闪烁一次（接到PB5引脚上），且串口输出调试信息：

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：

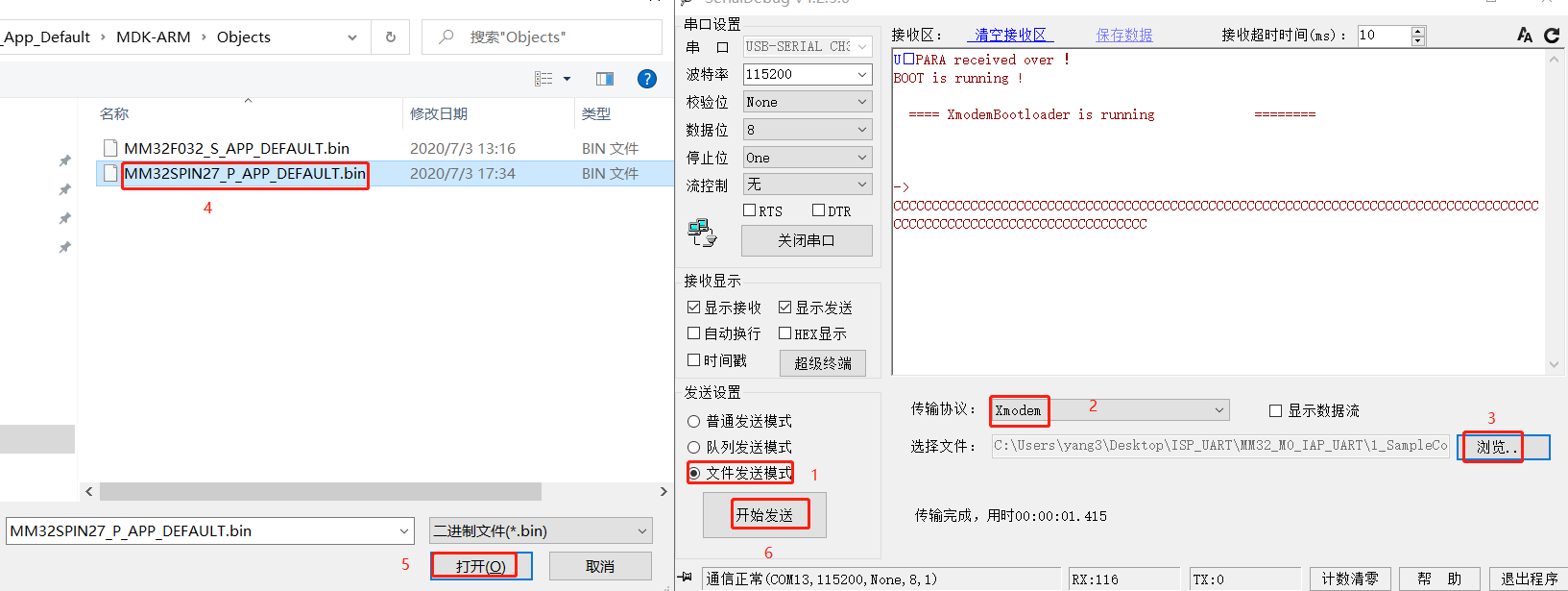


图 18. **开发板接收升级文件**

见到以下调试信息输出，整个升级过程成功结束：

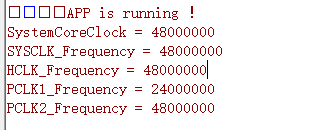


图 19. **开发板接收成功跳转到APP**

**Revision Record 修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision**  **Document** | **Author** | **Date** | **Approved**  **by** | **Modification** |
| 0.0.3 | Willow | 2020/07/06 | Peter | [formatting](http://www.baidu.com/link?url=6agLEQR8MjDWslsxjQjHWd0q7M4wjoEpgFaDWiBCHHEbqeL0eHNPmBWVmh_3FHFfjZoINY0ujZmOMC624Hy9cOekYxC4ul7_fzQkevhOCZW" \t "https://www.baidu.com/_blank) |
| 0.0.4 | Willow | 2020/07/16 | Peter | Add M3 Demo |
|  |  |  |  |  |