|  |  |
| --- | --- |
| File Name: | RT1061 bootloader程序分析分析与应用程序开发说明 |
| File No. | 待分配 |
| Version: | V1.0.0 |

**Without authorization, dissemination and duplication of the content**

**of this article are strictly prohibited. Violators must be prosecuted.**

编制与审核

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编 制： | 任湘辉 | 签 名： |  | 日 期 |  |
| 校 核： | 王 兵 | 签 名： |  | 日 期 |  |
| 审 核： | 董 敏 | 签 名： |  | 日 期 |  |
| 安 全： | / | 签 名： |  | 日 期 |  |
| 确 认： | / | 签 名： |  | 日 期 |  |
| 批 准： | 唐 伟 | 签 名： |  | 日 期 |  |

文件变更履历记录

| **版本号** | **作 者** | **日 期** | **修改说明** | **安全评估** |
| --- | --- | --- | --- | --- |
| V1.0.0 | 孙宇尧 | 2023/10/19 | File Create |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目 录

[目录 III](#_Toc148705657)

[1. 应用程序开发说明 1](#_Toc148705658)

[1.1 Flash分区与RAM区域说明 1](#_Toc148705659)

[1.1.1 Flash分区 1](#_Toc148705660)

[1.1.2 RAM区域说明 2](#_Toc148705661)

[1.2 分散加载文件说明 4](#_Toc148705662)

[1.2.1 Image header的添加（检验文件头的添加） 4](#_Toc148705663)

[2. Bootloader程序运行过程分析 12](#_Toc148705664)

[2.1 程序执行过程 12](#_Toc148705665)

[2.1.1 流程图 13](#_Toc148705666)

[2.1.2 应用程序上传、切换和bootloader程序跳转分析 13](#_Toc148705667)

应用程序开发说明

Flash分区与RAM区域说明

Flash分区



1）Bootloader code 区：放置的是bootloader代码。

2）Boot mete1/2 区：放置的是flash分区信息，如每个分区的起始地址，大小，可以通过这些信息提取和校验image的header。

3）Swap mete1/2 区：放置了primary 区和secondary 区image的大小信息，还有swap\_type，swap\_status，copy\_status等状态信息。这些状态信息在程序中被更改以后需要实时更新并再次写入swap 区，这个过程需要使用flash驱动实现。具体定义可以在程序查看，在swap mete区最重要的两个信息是swap\_type和image\_info，bootloader程序swap\_type的值来确定执行相应的操作，根据image\_info的值来确定flash擦写范围。

以下为swap\_type几个主要的值的含义：

kSwapType\_None：在bootloader中检测到swap\_type为这个值以后，表示应用程序不需要更新，不需要执行任何操作。

kSwapType\_ReadyForTest：在bootloader中检测到swap\_type为这个值以后，将primery区的程序放到scratch区，然后将secondary区的程序放到primary区。这个值表示下一步进行应用程序的切换操作。在执行完kSwapType\_ReadyForTest相对应的操作以后，swap\_type的值会被修改为kSwapType\_Test。

kSwapType\_Test：在bootloader中检测到swap\_type为这个值以后，首先将primary区的程序放到secondary区，然后将scratch区的程序放到primary区。这个值表明下一步需要进行https接收操作。在执行完kSwapType\_Test相应的操作以后，swap\_type的值会被修改为kSwapType\_None。

kSwapType\_Permanent：在bootloader中检测到swap\_type为这个值以后，表示应用程序已经更新完毕，不需要执行任何操作。

4）Primary 区：每次bootloader都会跳转到这个区域开始执行应用程序，所以这个区域放置的是正在执行的应用程序。

5）Secondary 区和scratch区：上传的bin文件都会放在secondary区。Scratch区存放的接收更新的应用程序前运行的应用程序，如果在更新完应用程序以后需要回到之前的版本，在bootloader程序启动阶段，在串口输入3即可将scratch区备份的程序放置到primary区运行。

需要注意的是，Scracth区的大小不是固定的，它取决于其他两个区的应用程序的大小（即image\_info的值），不过下flash大小足够的情况下，基本不需要关注sractch区的大小。

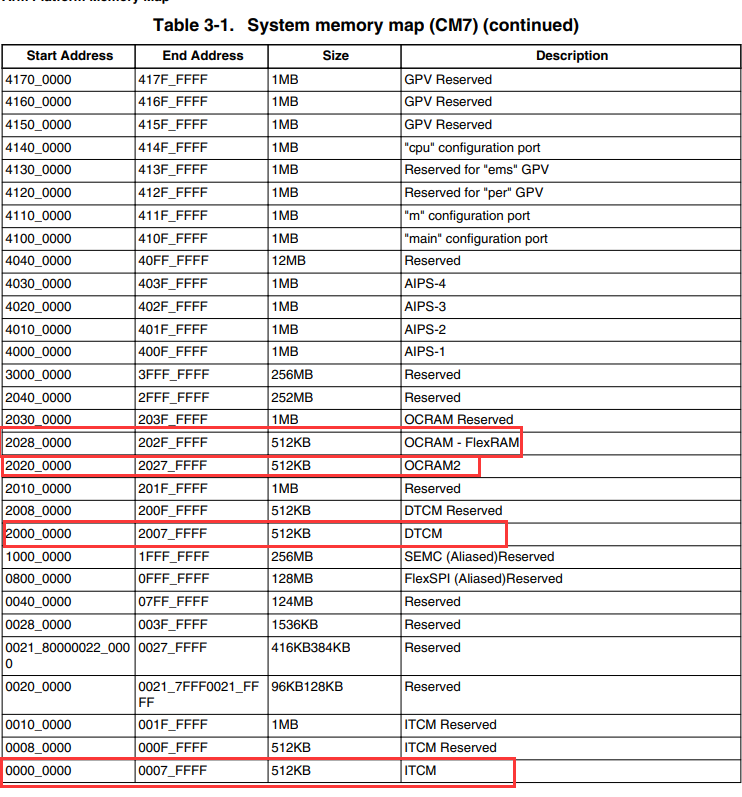
Bootloade程序的作用：bootloader程序是一个独立可运行的程序，它的主要作用是接收上传的应用程序和控制primary区、secondary区、scracth区应用程序的交换。

用户可用的Flash区域是：0x60040400-0x60240000, 0x60441000-0x60800000

RAM区域说明

1）内存分配

内存的地址映射如下图：



RT1061的RAM分为三种类型:

ITCM：存放在这个区域的代码指令，执行效率非常高

DTCM：存放在这个区域的数据提取效率非常高

OCRAM：相比ITCM和DTCM两种专用型内存，它像是通用型内存，它的指令执行效率和数据提取效率高于外部flash，但低于其他两种类型的内存。

ITCM与DTCM连接的总线的频率与内核同频，而OCRAM只有内核的四分之一。

RT1061总共有1MB的RAM空间，有512kb固定RAM空间，还有512kb可分配的RAM空间。

如上图所示：

固定的RAM空间：0x20200000-0x2027FFFF，即OCRAM2区。

可分配区：可分配的512kb可以分配在如上图地址映射表的ITCM、DTCM、OCRAM-FlexRAM三个空间。其中OCRAM-FlexRAM至少要分配64kb，因为系统自带的bootloader需要64kb的OCRAM来运行。分配方案是：ITCM:448kb，DTCM：0kb，OCRAM：64kb。

内存空间的使用情况：ITCM：0x00000000-0x00070000，这一区域用于存放应用程序main函数中RO属性的数据和指令。所以应用文件的大小一般不能超过450kb。

DTCM：未被使用。

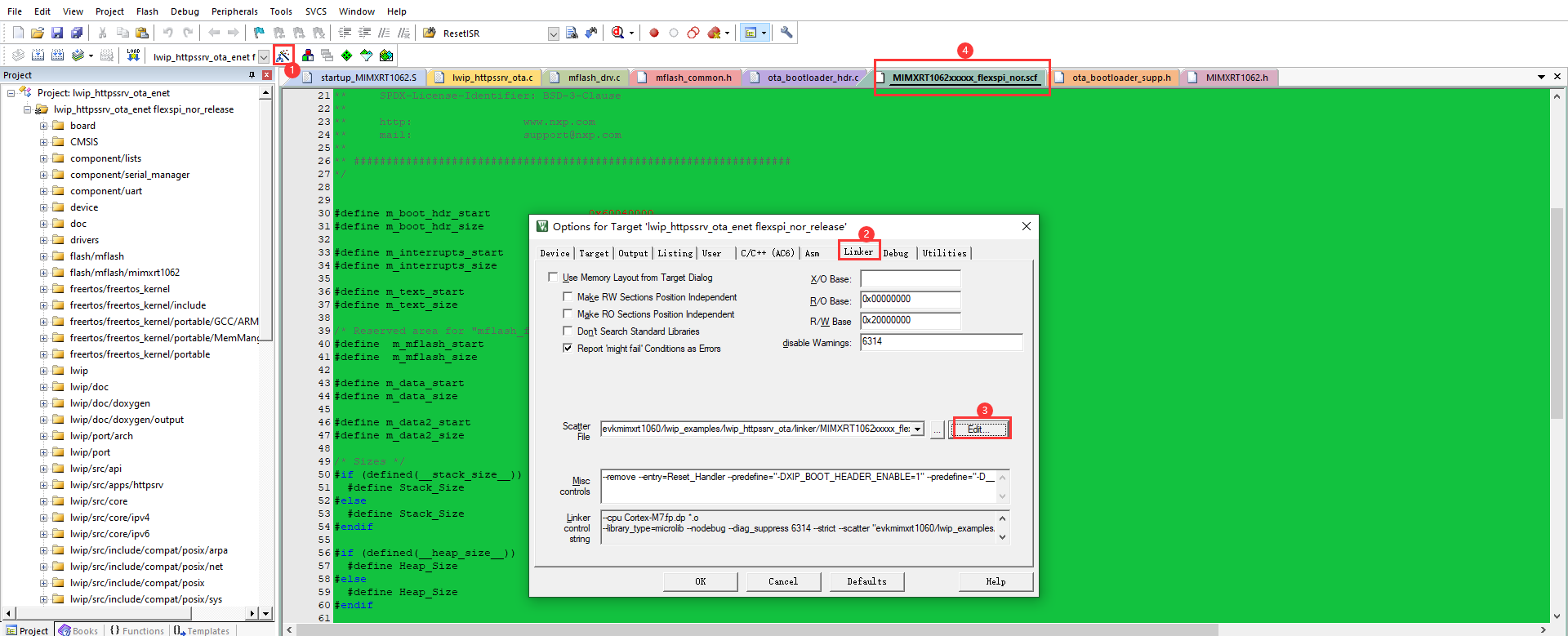
OCRAM：0x20200000-0x20290000，这区域用于存放RW属性的指令和数据，还有ZI属性的数据。堆和栈也是在此区域进行分配。

2）内存的重新分配

在以后的应用程序开发过程中，如果对内存有特定的要求需要对可分配内存空间重新分配，可以使用特定的寄存器来进行控制，而配置代码将放置在启动文件中，这样可以防止在内存配置完成之前被访问。具体的配置过程和代码官方有一个详细的文档，即本文档同级目录下的“using I.MXRT FlexRAM”文件。

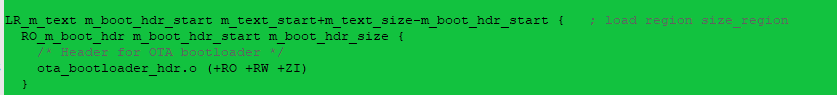
分散加载文件说明

Keil的分散加载文件相当于一个链接文件，它可以告诉内核去哪里提取需要的指令或数据。打开分散加载文件的方法如下图：



关于分散加载文件的使用和配置方法，在本文档的同级目录下的“周立功单片机：分散加载文件浅释”有详细说明。在这里只简要分析一下它的作用。

我们可以看到在分散加载文件的开头有一些地址的宏定义，这些地址有的是加载域的起始地址，有的是执行域的起始地址。加载域就是存放程序的储存空间，执行域就是内核提取数据和指令的储存空间，而分散加载文件的作用就是告诉内核将加载域中的哪些指令数据放到执行域去。然后在程序运行的时候内核需要这些指令或数据的时候，就会去相应的执行域提取。所以我们可以通过分散加载文件指定某些源文件里面的指令和数据的存放位置。我们将应用程序中用于校验的文件头放置在bin文件开头大小为0x400的位置就是使用了分散加载文件的这一功能。如下图：

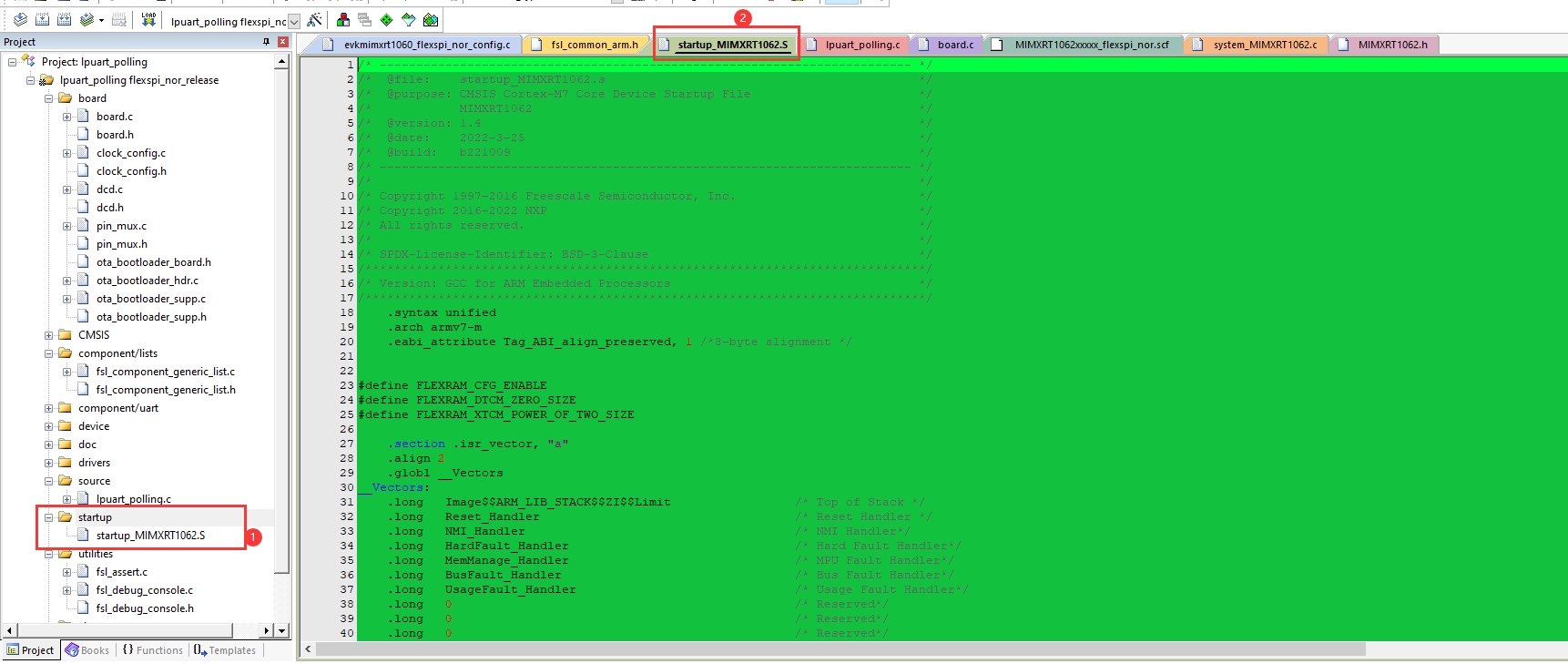


在后面的应用程序开发过程中，大部分的可以直接使用例程中的分散加载文件，有特殊要求的应用程序也可以在这个基础上进行修改。

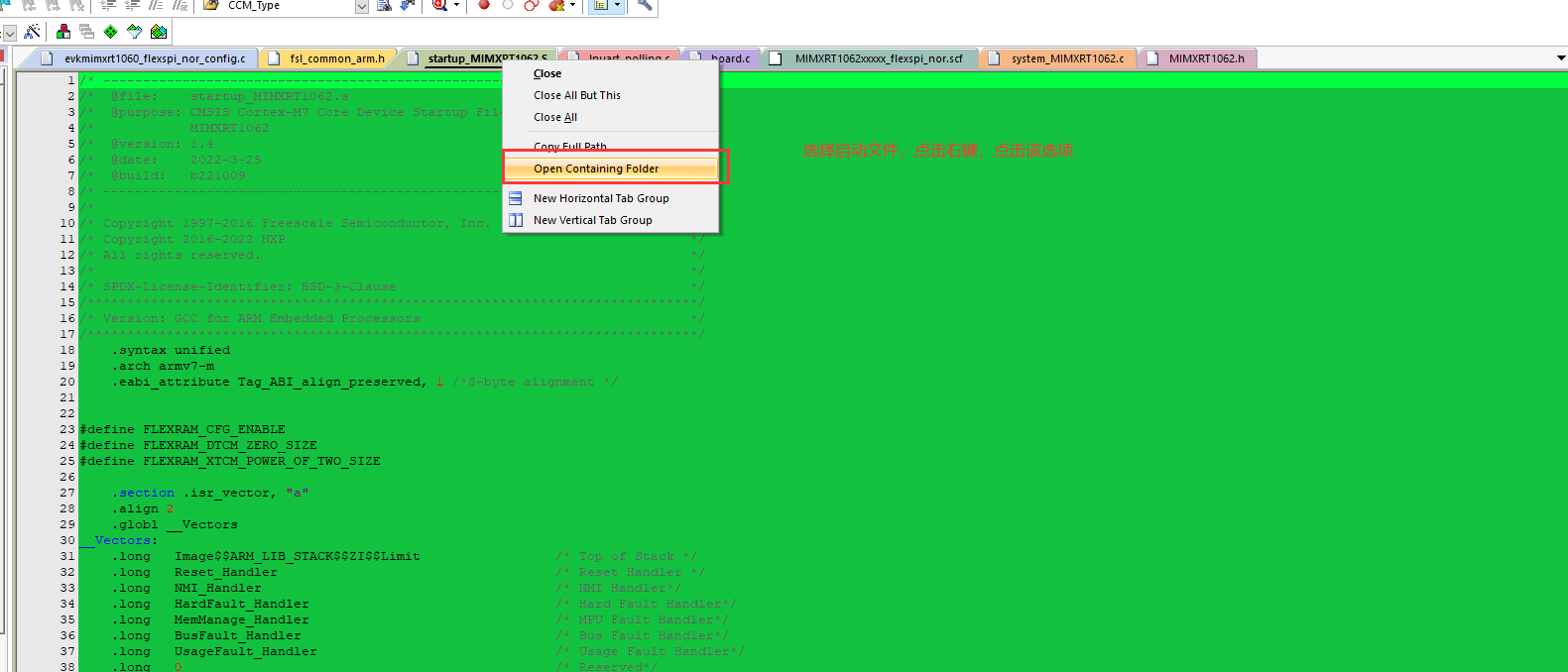
应用程序分散加载文件和启动文件的替换

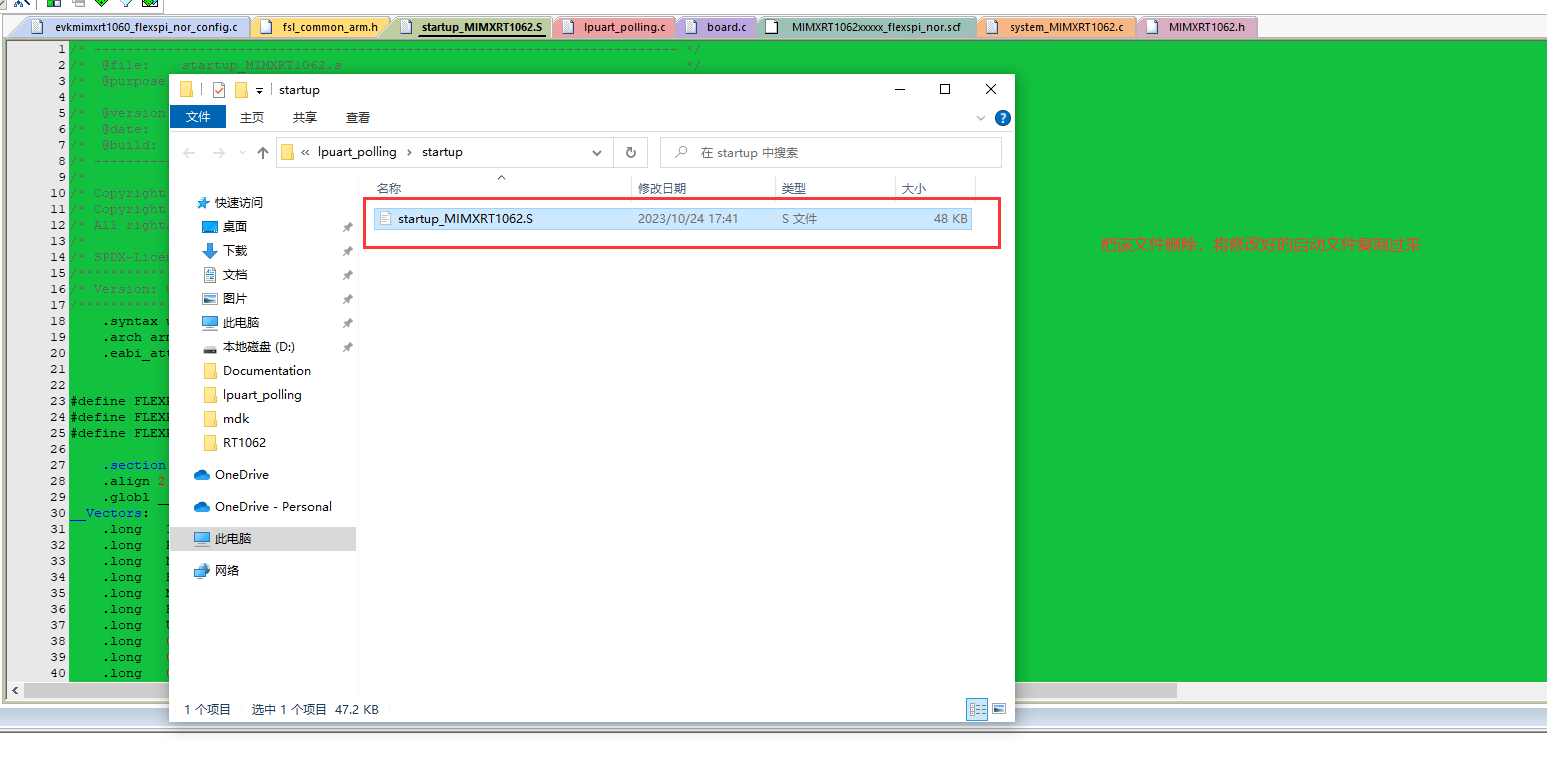
在之后的应用程序开发过程中，为了使储存空间的分区与bootlooader程序相匹配，需要使用经过修改的启动文件和分散加载文件，替换方法如下：

1）首先打开启动文件



2）找到文件所在位置，使用本文档上级目录下“startup.file+scatter.file”文件夹中的启动文件替换掉现有的启动文件



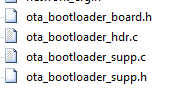


3）分散加载文件的替换也是同样的流程。

Image header的添加（检验文件头的添加）

在使用串口上传文件时会对文件进行检验，如果上传的文件没有相应的文件头将会下载失败。所以后面需要下载到开发板运行的应用都需要在文件的中添加文件头。添加方法如下;

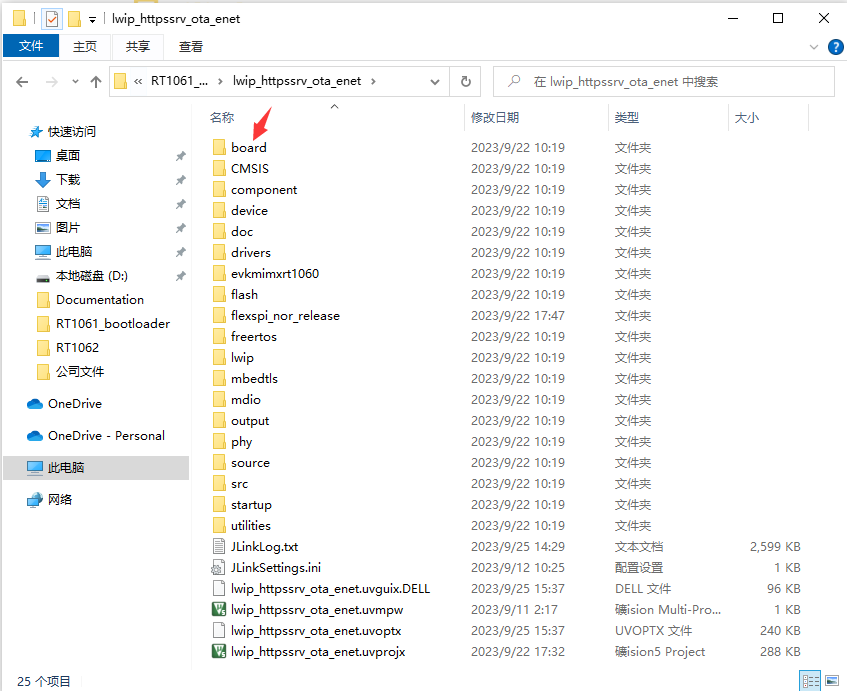
1）添加生成header的四个文件到应用程序工程中，如下图：



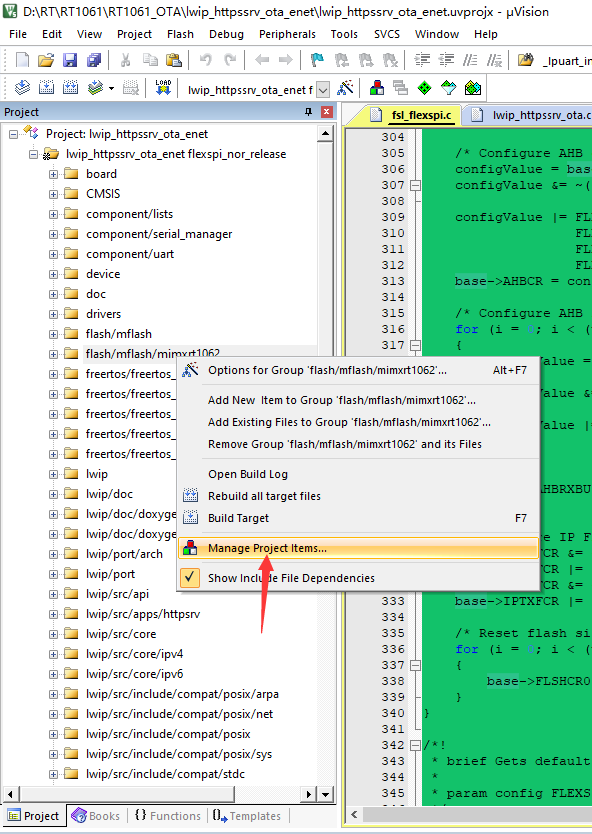
这四个文件放在“image\_header\_implementation\_file“这个文件夹中。

Keil工程中添加文件的方法：

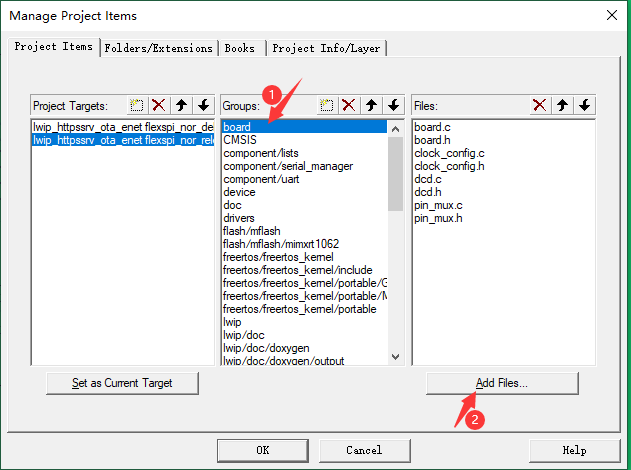
（1）首先将这四个文件复制到keil项目里面的board文件夹下（每个SDK例程都有这个文件夹）。

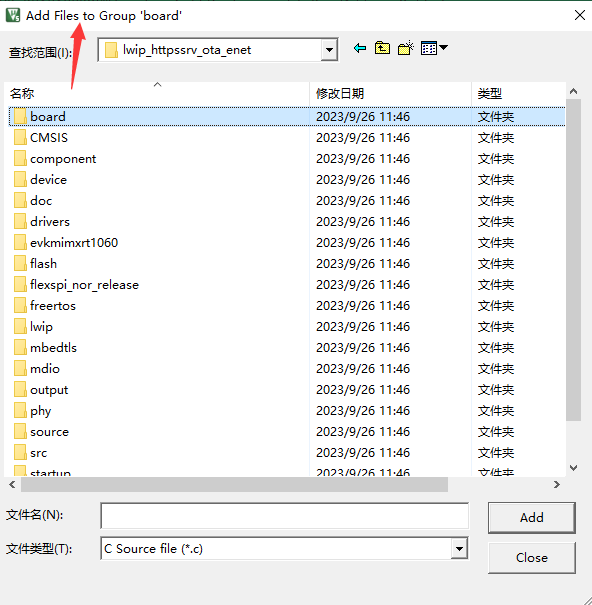


然后进入到需要添加文件的keil工程，在project栏随机选择一个文件组，点击右键选择项目管理选项。

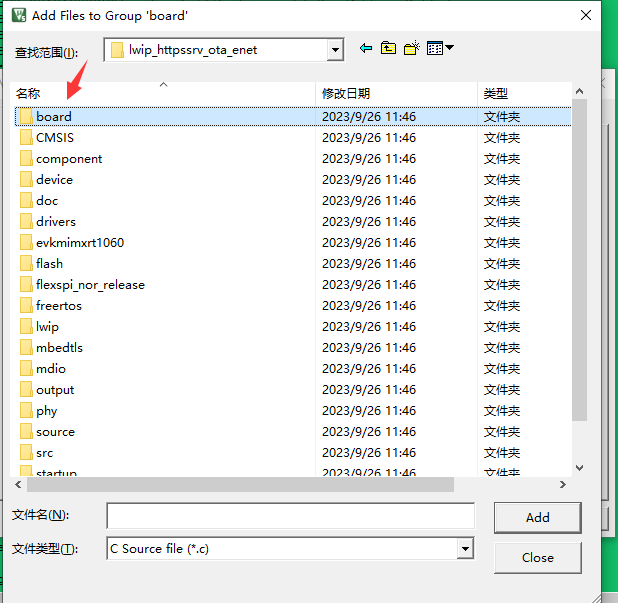


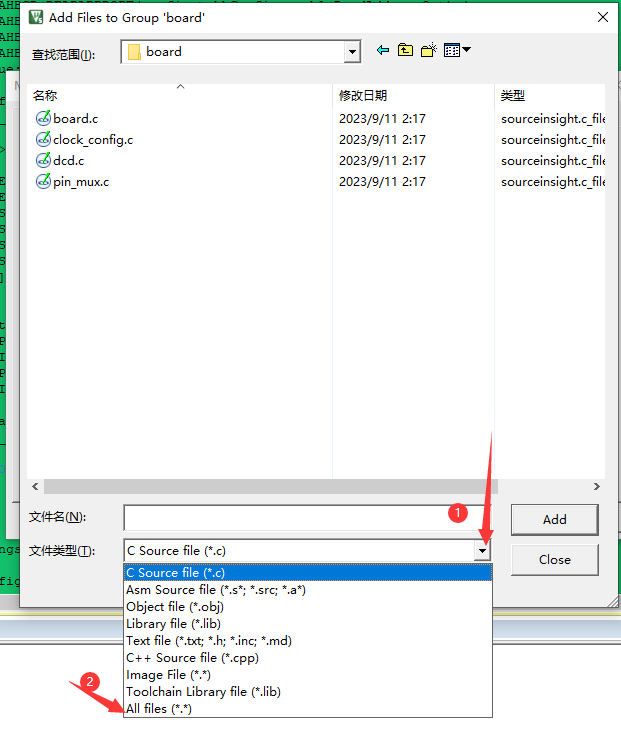
（2）选择board将需要新增的四个文件添加到该组，点击ADD files。

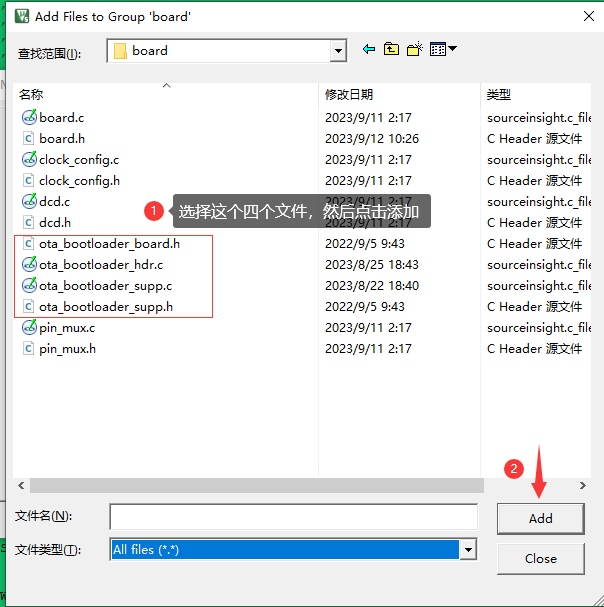




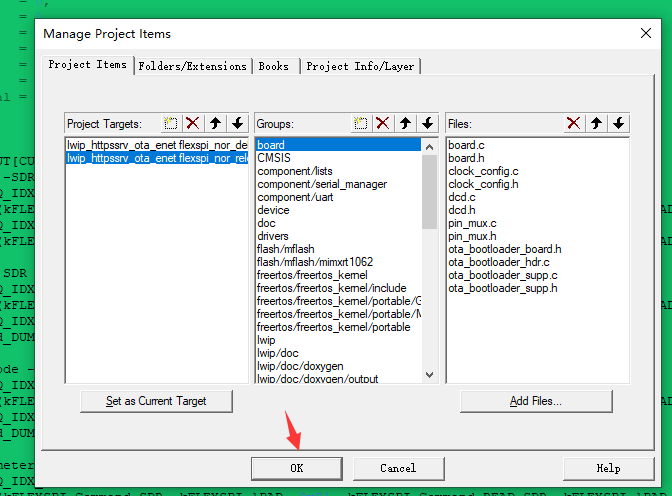
（3）找到之前放入那四个文件的目录（board），选择这个四个文件，然后点击添加。



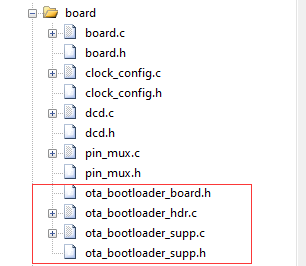




添加完毕后点击“Close”。然后在Manage Project Items界面点击”OK”,



可以到board文件组中的出现了这四个文件



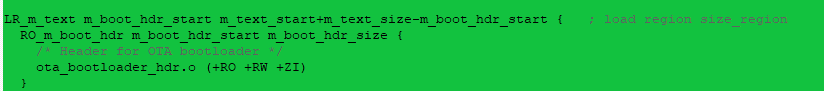
2）后面所有更新的应用程序中，ota\_bootloader\_hdr.c文件需要在开头添加\_\_attribute\_\_((used))语句，防止被链接器优化。在“image\_header\_implementation\_file“这个文件夹的文件已经添加，具体实现可以去文件中查看。

3）在keil的分散加载文件中，指定ota\_bootloader\_hdr.c文件的链接地址。

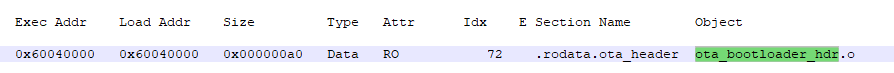
首先在起始地址分配一个 大小为0x400的空间给image header存放，如下图：



然后指定ota\_bootloader\_hdr.c加载域和执行域的位置，如下图：



这样ota\_bootloader\_hdr.c文件里面的内容就会被放到应用程序在flash中起始位置，可以从map文件中看出是否分配成功了，如下图：



到这里image的header就添加成功了。

Bootloader程序运行过程分析

程序执行过程

流程图



在开发板上电以后首先运行的是bootloader程序。Bootloader程序的执行过程：

1）首先进行系统硬件的初始化，如初始化串口，PIT定时器。

2）初始化bootlaoder的属性与功能，如flash驱动，RAM接口驱动，各类接口的驱动。

3）进入串口命令输入阶段，输入命令执行相关的操作（具体操作的内容将在后面详细说明），如果不需要输入命令程序将在2-3秒以后跳过这个阶段继续执行后面的代码。

4）在串口命令阶段的操作执行完成以后，程序就会进入到准备跳转阶段，通过重新设置SP和PC指针完成程序的跳转。

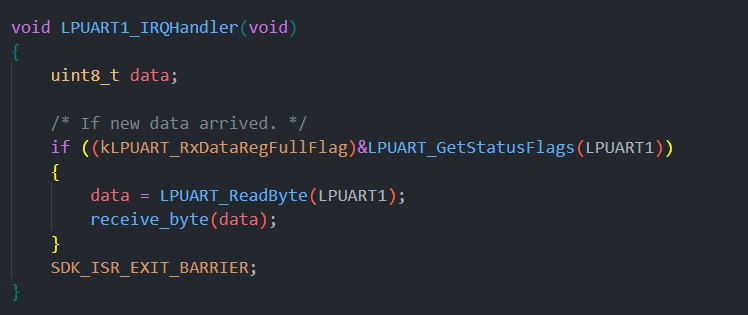
应用程序上传、切换和bootloader程序跳转分析

1）应用程序的上传

在串口命令输入阶段，输入5将会进入到串口文件传输模式，文件使用的是Xmodem协议进行传输。传输过程如下：

1. 在传输开始以后，使用串口中断接收上位机传输过来的数据包一共133字节，将这133字节缓存在一个全局数组中，此时上位机等待接收端的响应。

串口中断服务函数：



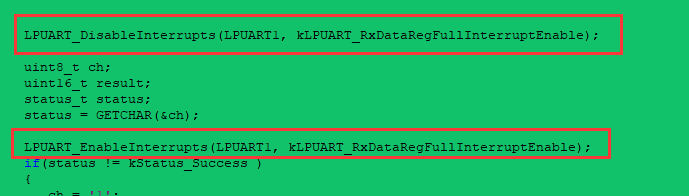
（2）在一个数据包被完整接收以后，程序对数据包进行逐字节的提取分析。首先提取第一个字节确定数据包的类型，然后提取后面两个字节对数据包的帧号和帧号的补码进行校验，校验通过将帧号和帧号的补码保存到数据包结构体中。

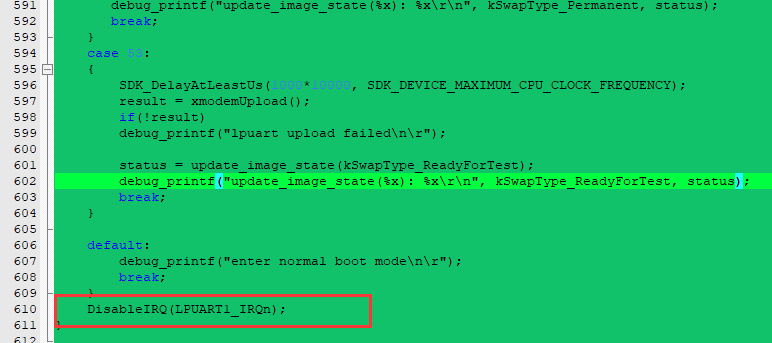
然后提取后面的130字节，使用最后两个字节的crc检验码对数据包中的数据帧进行校验。数据帧校验通过以后，将128字节的数据帧保存到数据包结构体中，然后再将数据包结构体中的数据帧缓冲到一个256字节的缓存区，我们可以通过帧号的奇偶性来确定数据帧的缓冲位置，如奇数放置前128字节，偶数放置到后128字节。在将数据帧放置到缓冲区以后，接收端就会发送响应信号给发送端，发送端就会发送下一个数据包。

当缓冲区填满了以后，首先确定写入的区域是否已经擦除，然后使用flash写函数将256字节的数据写入到相应的位置（因为flash的写入单位是256字节，所以缓冲区的大小就是256字节，即每接收到256字节的数据以后就写入flash中。每次flash的擦除大小为0x1000，即4kb)。Flash写入的首地址即为secondary区的开始地址0x60240000.

（3）不过需要注意的是，第一次接收的256字节数据不能立即写入flash，需要先放置到另外一个缓冲区，在接收完所有数据以后对这个区域的数据进行处理然后再写入到flash。因为前256字节的数据包含文件头信息，而此时文件头的信息还是不完善的，比如没有文件大小信息，而在后面的应用程序切换的过程中需要使用文件头中的这个信息，所以必须要对文件头进行处理以后才能写入到flash中。

（4） 还有需要注意的地方是，串口的接收开启了串口接收中断，为了防止与bootloader串口命令接收阶段发生冲突导致接收不到命令，需要在接收阶段关闭串口接收中断，命令接收阶段结束以后再开启串口接收中断，在最后再关闭串口中断，这样就不会影响到之后的串口的接收。





2）应用程序的切换

从上面的flash分区图，可以了解到flash主要分为三个大区：bootloader程序区，primary区，secondary区，其中secondary区包括了scratch区。

Bootloader程序区放置的是bootloader代码，上电以后首先执行bootloader代码，然后跳转到primary区。

Primary区放置的上电以后运行的应用程序，bootloader程序执行完成以后就会跳转到primary区执行这个区的应用程序。

Secondary区放置的是串口上传的应用程序。Scractch区放置的原来运行的应用程序，在我们把secondary区上传的应用程序复制到primary区去运行之前，bootloader程序会先将primary区的代码复制到scratch区。

1）切换过程分析

当我们在串口传输阶段输入2时，表明bootloader程序需要切换priamry区与secondary区的应用程序。

（1）输入命令2以后，首先会使用“update\_image\_state”函数将swap\_type的值更新为“kSwapType\_ReadyForTest”。关于swap\_type各个值得含义在flash分区已经说明了。

（2）bootloader程序会在“run\_swap\_routine”函数中检测swap\_type的值，然后执行相应的操作。在这里需要执行的操作的是将primary区的应用程序与secondary的应用程序进行交换。如下图：



具体实现的代码可以去查看相关的函数。

（3）这里还有一个问题是，如何确定scratch区的大小？这就需要用到swap mete区的信息了，在swap mete区中存放了一个名为“image\_info”的数组，它保存的是primary区和secondary区应用程序的大小，它是在“update\_image\_state”函数进行赋值的，函数通过提取应用程序文件头中的信息获取应用程序的大小。

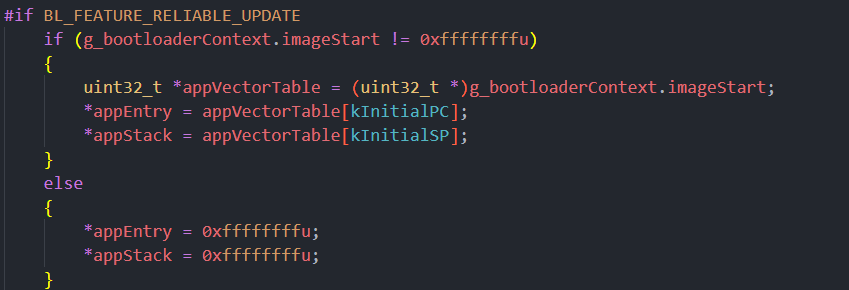
3）bootloader程序的跳转

Bootloader程序跳转的原理主要是，重新设置SP和PC指针，并且重新设置中断向量表。

（1）重设SP和PC指针

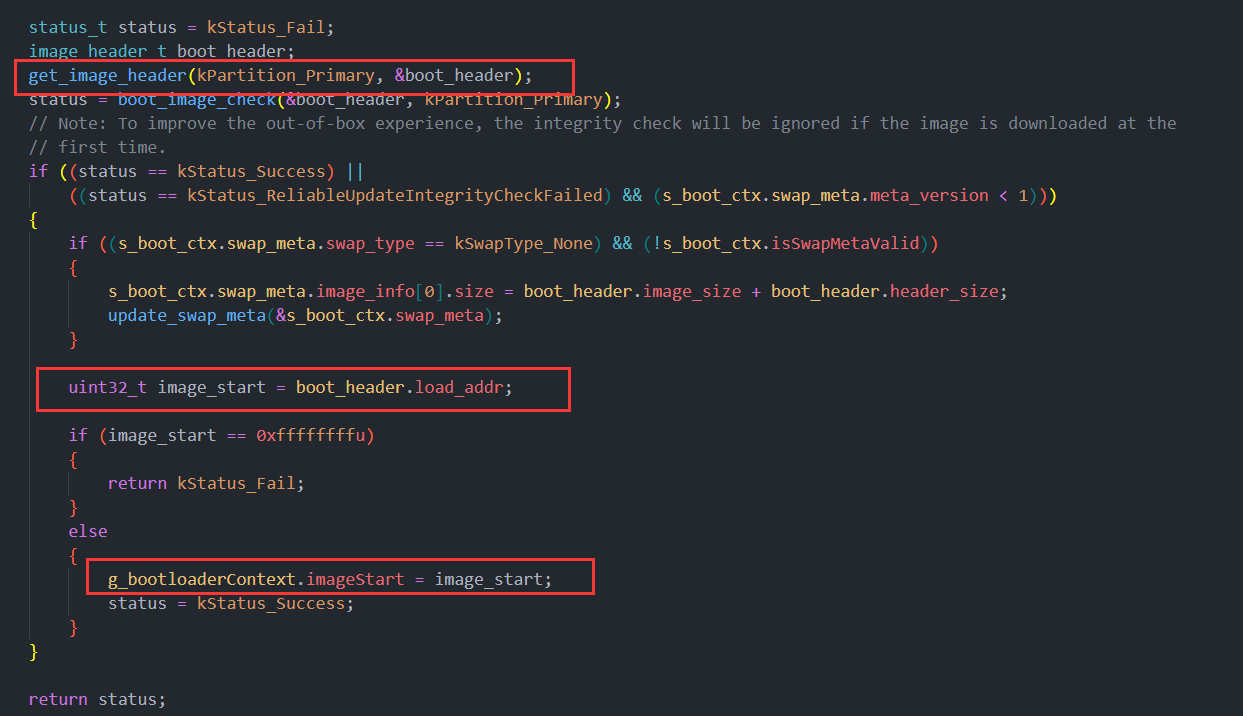
bootloader程序在“get\_user\_application\_entry”函数中获取新的SP和PC指针。

关键代码如下图：

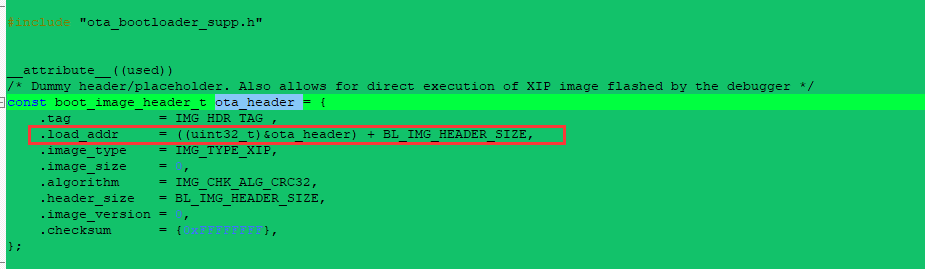


从上面的代码可以看出，参数“g\_bootloaderContext.imageStart”就是SP指针的值，SP指针偏移四个字节就是PC指针的值。

而参数“g\_bootloaderContext.imageStart”的值在“boot\_go”函数中确定。如下图：



从以上代码可以看出，只要primary区存在合法应用程序，函数即可通过获取primary区应用程序文件头的信息，来确定参数“g\_bootloaderContext.imageStart”的值，文件头中“load\_addr”的值就是参数值。“load\_addr”的定义如下：



由此可以确定参数“g\_bootloaderContext.imageStart”的值为0x60040400，然后就可以确定新的SP和PC指针的值。

（2）开始跳转

在确定新的SP和PC指针的值以后使用“jump\_to\_application”函数跳转到primary区的应用程序。具体代码实现可以查看相关函数。