写在前面：三个周之前，我突然想写一个远程升级的程序。那个时候我只是大概知道 IAP 的意思是在应用编程，但怎么编，我还一无所知。我给自己定下一个个阶段目标，从最基础的代码一点点写起，解决一个又一个的问题。三个周之后，我用自己设计的方法实验了 50 多次，无一例升级失败。

三个周来，遇到了很多的不解、困惑，甚至是想放弃，但我现在想说的是：很多未知的困难会挡在我们面前，我们会感觉毫无头绪甚至觉得毫无出路忍不住要放弃，但多坚持一下，那些困难不但能烟消云散还能带给我们进步。

本设计是基于 LPC2114 和 Keil MDK（V4.10），但所有支持 IAP 的处理器都可借鉴本方案，重要的是思想，而不是用什么。

0 引言

在应用编程(IAP)技术为系统在线升级和远程升级提供了良好的解决方案，也为数据存储和现场固件的升级都带来了极大的灵活性。通常可利用芯片的串行口接到计算机的 RS232 口、通过现有的 Internet 或、无线网络或者其他通信方式很方便地实现在线以及远程升级和维护。

本文以 NXP 的 LPC2114 ARM 微处理器为平台，以 Keil MDK 为开发工具，阐述 IAP 的原理、Flash 的划分、分散加载机制、中断重映射以及在线升级的实现方案及其优化。本方案使用多种校验技术，最大限度的保障传输数据的正确性；使用 bootloader 机制，即使因意外事件（断电，编程 Flash 失败等）造成升级失败后，程序也能返回到升级前的状态。

1 LPC2114 的 Flash 规划

1.1 扇区描述

LPC2114 共有 128KB 片内 Flash，共分为 16 个扇区，分别为 0 扇区～15 扇区，每个扇区为 8KB 存储空间。其中第 15 扇区出厂时被固化为 Boot Block 区，控制复位后的初始化操作，并提供实现 Flash 编程的方法。所以用户可用的 Flash 空间只有 120KB。IAP 程序固化于 Boot Block 中，IAP 操作是以扇区为单位，并占用片内 RAM 的高 32 字节。下表列出 LPC2114 器件所包含的扇区数和存储器地址.

表 **1.1 LPC2114 Flash** 器件中的扇区

扇区号	存储器地址		扇区号	存储器地址	
	地址范围	规格(KB)		地址范围	规格(KB)
0	0x0000~0x1FFF	8	8	0x10000~0x11FFF	8
1	0x2000~0x3FFF	8	9	0x12000~0x13FFF	8
2	0x4000~0x5FFF	8	10	0x14000~0x15FFF	8
3	0x6000~0x7FFF	8	11	0x16000~0x17FFF	8
4	0x8000~0x9FFF	8	12	0x18000~0x19FFF	8
5	0xA000~0xBFFF	8	13	0x1A000~0x1BFFF	8
6	0xC000~0xDFFF	8	14	0x1C000~0x1DFFF	8
7	0xE000~0xFFFF	8	15	0x1E000~0x1FFFF	8

1.2 Flash 的扇区划分

本设计将 Flash 划分为四个区，扇区 0 存放跳转程序和升级引导程序（Bootloader）。分站上电后执行跳转程序，跳转到用户程序处。用户程序运行过程中，如果接收到升级指令，会从用户程序跳转到引导程序区（Bootloader），接收新程序数据包，完成 Flash 编程并跳转到新程序区执行程序。扇区 1～扇区 7 为程序存储低区；扇区 8～扇区 13 为程序存储高区；扇区 14 存放当前程序运行区域标志，如果当前程序运行在高区，该标志区的最低四个字节为 0x00010000，如果当前程序运行在低区，该标志区的最低四个字节为 0x00008000。

2 IAP 的原理与软件设计

2.1 IAP 的原理

IAP 函数是固化在微处理器内部 flash 上的一些函数代码，最终的用户程序可以直接通过调用这些函数来对内部 flash 进行擦除和编程操作。LPC2114 微处理器的内部 flash 有一个块称为 Boot Block，位于 flash 的顶端，可供调用的 IAP 函数就位于该块中。上电后 Boot Block 被映射到内部地址空间的顶端，同样 IAP 函数人口地址也被映射到地址 0x7ffffff0 处。用户可通过跳转到该地址来调用相应的 IAP 函数。

2.2 IAP 命令

对于在应用编程来说，应当通过寄存器 r0 中的字指针指向存储器(RAM)包含的命令代码和参数来调用 IAP 程序。IAP 命令的结果返回到寄存器 r1 所指向的返回表。用户可通过传递寄存器 r0 和 r1 中的相同指针重用命令表来得到结果。参数表应当大到足够保存所有的结果以防结果的数目大于参数的数目。参数传递见图 2-1。参数和结果的数目根据 IAP 命令而有所不同。参数的最大数目为 5，由“将 RAM 内容复制到 Flash”命令传递。结果的最大

数目为 2，由“扇区查空”命令返回。命令处理程序在接收到一个未定义的命令时发送状态代码 INVALID_COMMAND。IAP 程序是 thumb 代码，位于地址 0x7FFFFFF0。

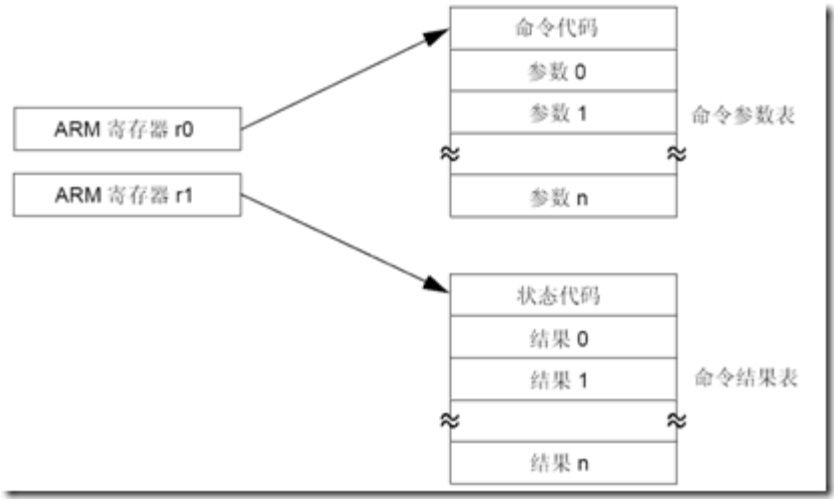


图 2-1 IAP 的参数传递

表 2-1 描述了 IAP 的命令。

表 2-1 IAP 命令汇总

IAP 命令	命令代码	描述
准备编程扇区	50	该命令必须在执行“将 RAM 内容复制到 Flash”或“擦除扇区”命令之前执行。这两个命令的成功执行会导致相关的扇区再次被保护。该命令不能用于 boot 扇区。要准备单个扇区，可将起始和结束扇区号设置为相同值。
将 RAM 内容复制到 Flash	51	该命令用于编程 Flash 存储器。受影响的扇区应当先通过调用“准备写操作的扇区”命令准备。当成功执行复制命令后，扇区将自动受到保护。该命令不能写 boot 扇区。
擦除扇区	52	该命令用于擦除片内 Flash 存储器的一个或多个扇区。boot 扇区不能由该命令擦除。要擦除单个扇区可将起始和结束扇区号设定为相同值。
扇区查空	53	该命令用于对片内 Flash 存储器的一个或多个扇区进行查空。要查空单个扇区可将起始和结束扇区号设定为相同值。
读器件 ID	54	该命令用于读取器件的 ID 号。
读 Boot 版本	55	该命令用于读取 boot 代码版本号。
IAP 比较	56	该命令用来比较两个地址单元的存储器内容。当源或目标地址包

		含从地址 **0** 开始的前 **64** 字节中的任意一个时，比较的结果不一定正确。前 **64** 字节重新映射到 **Flash boot** 扇区。

2.3 IAP 编程函数接口

IAP 功能可用下面的 C 代码来调用。

定义 IAP 程序的入口地址。由于 IAP 地址的第 0 位是 1，因此，当程序计数器转移到该地址时会引起 Thumb 指令集的变化。

#define IAP_LOCATION 0x7ffffff1

定义数据结构或指针，将 IAP 命令表和结果表传递给 IAP 函数

unsigned long command[5];

unsigned long result[2];

定义函数类型指针，函数包含 2 个参数，无返回值。注意：IAP 将函数结果和 R1 中的表格基址一同返回。

typedef void (*IAP) (unsigned int [ ] , unsigned int [ ]);

IAP iap_entry;

设置函数指针

iap_entry=(IAP) IAP_LOCATION;

使用下面的语句来调用 IAP。

iap_entry (command , result);

Flash 存储器在写或擦除操作过程中不可被访问。执行 Flash 写/擦除操作的 IAP 命令

使用片内 RAM 顶端的 32 个字节空间。如果应用程序中允许 IAP 编程，那么用户程序不应

使用该空间。

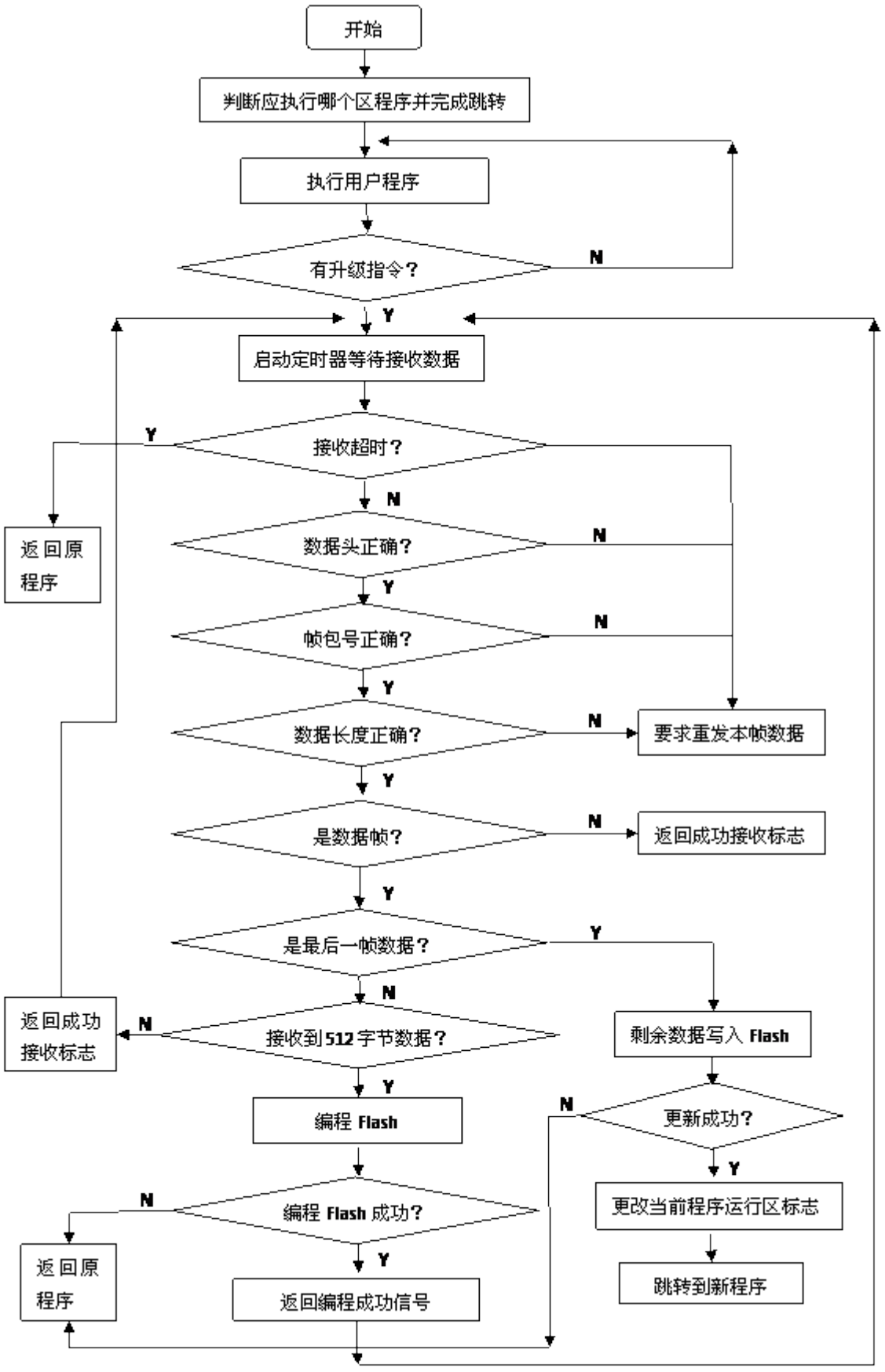
3 LPC2114 升级实现过程

由于在升级程序软件设计中，分散加载机制、中断向量的重映射、软中断等的实现还与所使用的编译器紧密相关，因此，本文结合 Keil MDK（V4.10）编译工具，来详细阐述升级程序的实现过程。

3.1 总体思路

分站上电后，首先运行位于 Flash 0x000～0x3FF 中的跳转程序。跳转程序会读取位于 14 扇区的当前程序运行标志，如果该扇区的最低四个字节为 0x00010000，表示当前程序运行在高区，跳转程序会跳转到 Flash 的 0x00010000 处执行用户程序；如果该标志区的最低四个字节为 0x00008000，表示当前程序运行在低区，跳转程序会跳转到 Flash 的 0x00002000 处执行用户程序。用户程序正常执行后，会按照设计进行正常的程序采集、数据处理传送。当接收到升级命令后，用户程序会跳转到 Flash 的 0x00000400 处的 Bootloader 处进行升级的一些操作。当升级成功后，Bootloader 程序更新当前程序运行区标志，程序跳转到新程序处运行，如果升级不成功，返回升级前的程序。

流程图如下所示：


开始
判断应执行哪个区程序并完成跳转
执行用户程序
有升级指令？
N
Y
启动定时器等待接收数据
接收超时？
Y
返回原程序
N
数据头正确？
N
Y
帧包号正确？
N
Y
数据长度正确？
N
要求重发本帧数据
Y
是数据帧？
N
返回成功接收标志
Y
是最后一帧数据？
Y
N
返回成功接收标志
N
接收到512字节数据？
剩余数据写入 Flash
Y
编程 Flash
更新成功？
N
Y
N
编程 Flash 成功？
更改当前程序运行区标志
返回原程序
Y
跳转到新程序
返回编程成功信号

3.2 跳转程序的设计

跳转程序是分站上电后最先运行的程序，根据当前程序运行区标志，跳转到相应的用户程序区执行。本段程序占用 Flash 的最低 1K 字节空间，与 Bootloader 同在第 0 扇区。

跳转程序的启动代码仅初始化堆栈，不使用 PLL 和存储加速功能。代码 1 描述了跳转程序的主要启动代码。

```
; Enter User Mode and set its Stack Pointer

MSR CPSR_c, #Mode_USR

MOV SP, R0

SUB SL, SP, #USR_Stack_Size

; Enter the C code

IMPORT __main

LDR R0, =__main

BX R0
```

代码 1：跳转程序启动代码

当跳转程序确定要跳转到高区用户程序或者低区用户程序后，使用函数指针跳转到 0x00010000 处（高区用户函数入口地址）或 0x00002000 处（低区用户函数入口地址）。

定义函数指针：

```
void (*UserProgram)() ;
```

指定入口地址：

```
UserProgram = (void (*)()) (0x00010000)；

UserProgram = (void (*)()) (0x00002000)；
```

实现跳转：

(*UserProgram)() ；

要将用户代码精确定位到 Flash 的 0x00010000 处（高区用户函数入口地址）或 0x00002000 处（低区用户函数入口地址），需要使用编译器的分散加载机制，将在 Bootloader 中详细描述实现过程。

另外，跳转程序还在烧录代码的同时初始化当前程序运行区标志，即对 Flash 的 0x0001C000 地址处写入 0x00008000，表示当前用户程序在低区。主要使用了编译器的__**at** 关键字：精确定位变量。需要注意的是，使用该关键字必须包含头文件 absacc.h。

const uint32 x __at(0x0001C000)=0x00008000; //初始化用户程序标志区,默认运行低区

3.3 升级程序 Bootloader 的设计

升级程序的好坏，在很大程度上取决于 Bootloader 设计的好坏。

一个优秀的 IAP 升级 Bootloader，必须做好升级中出现故障等异常的处理。保证系统不会崩溃，即使升级失败，也能返回升级前的程序。

Ø 有升级指令，进行初始化工作（串口、定时器、看门狗）

Ø 接收升级数据包，检测帧头、长度、帧号、数据区校验，最大程度的保证升级数据的完整性、正确性。

Ø 实时检测接收状态，10 S 内没有接收到数据或接收到的数据包都是错的，则退出升级，返回原程序。

Ø 接收的数据按照 512 字节一组写入 Flash，写入后再读出与原数据进行对比校验，校验成功后，本次编程 Flash 成功。允许连续 3 次编程 Flash，三次都不成功，退出升级程序，执行原程序。

Ø 升级成功后，更新当前程序运行区标志，跳转到新程序，同时原程序保存。

本设计的 Bootload 位于 Flash 的 0x400 开始的扇区 0 存储区内，使用分散加载机制，将程序的入口地址定位到 0x00000400 处。当用户程序接收到升级指令后，就会使用函数指针跳转到这个入口处。

3.3.1 使用 IAP

图 3-1 描述了使用 IAP 编程 Flash 所必须的步骤。

图 3-1 Flash 编程步骤

3.3.1.1 定义系统参数

在使用 IAP 前，需要定义一些系统参数，比如系统时钟、IAP 中断入口、输入输出缓存。

```
#define IAP_CLK 11059200UL

#define IAP_LOCATION 0x7FFFFFF1

typedef void(*IAP)(uint32 [],uint32 []); //定义函数类型指针

IAP iap_entry=(IAP)IAP_LOCATION; //设置函数指针

unsigned long command[5] = {0,0,0,0,0};

unsigned long result[2]= {0,0};
```

代码 3-1：定义系统参数

3.3.1.2 选择扇区

在任何擦除和编程 Flash 之前，必须选中扇区，可以选中一个或多个。

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

* 名称：SelSector()

* 功能：IAP 操作扇区选择，命令代码 50。

* 入口参数：sec1 起始扇区

* sec2 终止扇区

* 出口参数：IAP 返回值(paramout 缓冲区) CMD_SUCCESS,BUSY,INVALID_SECTOR

*****************************************************************/

void SelSector(uint8 sec1, uint8 sec2)

{

paramin[0] = IAP_SELSECTOR; // 设置命令字

paramin[1] = sec1; // 设置参数

paramin[2] = sec2;

iap_entry(paramin, paramout); // 调用 IAP 服务程序

}
```

代码 3-2 选择扇区

3.3.1.3 擦除扇区

在编程 Flash 前必须执行擦除操作，如果某个扇区已经擦除，就不需要再次擦除。可以一次擦除一个或多个扇区。

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

* 名称：EraseSector()
```

```
* 功能：扇区擦除，命令代码 52。

* 入口参数：sec1 起始扇区

* sec2 终止扇区

* 出口参数：IAP 返回值(paramout 缓冲区) CMD_SUCCESS,BUSY,INVALID_SECTOR
*********************************************************************/

void EraseSector(uint8 sec1, uint8 sec2)

{ paramin[0] = IAP_ERASESECTOR; // 设置命令字

paramin[1] = sec1; // 设置参数

paramin[2] = sec2;

paramin[3] = Fosc/1000; // 当不使用 PLL 功能时，Fcclk=Fosc

iap_entry(paramin, paramout); // 调用 IAP 服务程序
```

代码 3-3 擦除扇区

3.3.1.4 编程扇区

通过这个过程，数据可以从 RAM 中编程到片内 Flash 中。

注：

1. 数据只能从片内 SRAM 编程到片内 Flash。

2. 片内 Flash 的地址必须 512 字节对齐。

3. 片内 RAM 应位于局部总线，即 USB 或以太网的 SRAM 不可以使用。

4. 每一次编程字节应该是 512、1024、4096、8192 中的一个。

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

* 名称：RamToFlash()
```

```
* 功能：复制 RAM 的数据到 FLASH，命令代码 51。

* 入口参数：dst 目标地址，即 FLASH 起始地址。以 512 字节为分界

* src 源地址，即 RAM 地址。地址必须字对齐

* no 复制字节个数，为 512/1024/4096/8192

* 出口参数：IAP 返回值(paramout 缓冲区)
CMD_SUCCESS,SRC_ADDR_ERROR,DST_ADDR_ERROR,

SRC_ADDR_NOT_MAPPED,DST_ADDR_NOT_MAPPED,COUNT_ERROR,BUSY,未选择扇区

*********************************************************************/

void RamToFlash(uint32 dst, uint32 src, uint32 no)

{ paramin[0] = IAP_RAMTOFLASH; // 设置命令字

paramin[1] = dst; // 设置参数

paramin[2] = src;

paramin[3] = no;

paramin[4] = Fosc/1000; // 当不使用 PLL 功能时，Fcclk=Fosc

iap_entry(paramin, paramout); // 调用 IAP 服务程序

}
```

代码 3-4 编程扇区

3.3.1.5 比较数据

通过这个函数，可以检查写入 Flash 中的数据和 RAM 中的是否相同。

注意源地址、目标地址和字节数必须是 4 的倍数。可使用 Keil MDK 提供的关键字__align(n)来指定 n 字节对齐。

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

* 名称：Compare()

* 功能：校验数据，命令代码 56。

* 入口参数：dst 目标地址，即 RAM/FLASH 起始地址。地址必须字对齐

* src 源地址，即 FLASH/RAM 地址。地址必须字对齐

* no 复制字节个数，必须能被 4 整除

* 出口参数：IAP 返回值(paramout 缓冲区)
CMD_SUCCESS,COMPARE_ERROR,ADDR_ERROR

*****************************************************************/

void Compare(uint32 dst, uint32 src, uint32 no)

{ paramin[0] = IAP_COMPARE; // 设置命令字

paramin[1] = dst; // 设置参数

paramin[2] = src;

paramin[3] = no;

iap_entry(paramin, paramout); // 调用 IAP 服务程序
```

代码 3-5 比较数据

3.3.2 IAP 编程期间的中断管理

LPC2114 片上 Flash 在擦除/编程期间绝不可被中断打断。但 Bootloader 中定时和串口接收又使用了中断，因此必须在擦除/编程之前禁止总中断，待操作完成后再使能总中断。Bootloader 运行在用户模式下，不具有禁止/使能中断的权力，所以在本设计中使用软中断禁止/使能总中断。Keil MDK 提供了关键字__svc 来触发软中断。

软中断函数声明：

__svc(0x00) void EnableIrq(void); //使能中断,软中断 0

__svc(0x01) void DisableIrq(void); //禁止中断,软中断 1

软中断函数代码：

```
/*
**********************************************************************
* 功 能:禁止中断
* 描 述:利用软中断实现在用户模式下调用函数关中断
**********************************************************************/
void DisableIrqFunc(void)
{
int temp;
__asm
{
MRS temp,SPSR
ORR temp,temp,#0x80
MSR SPSR_c,temp
}
}
/*
**********************************************************************
```

```
* 功 能:使能中断

* 描 述:利用软中断实现在用户模式下调用函数开中断

*********************************************************************

*/

void EnableIrqFunc(void)

{

int temp;

__asm

{

MRS temp,SPSR

BIC temp,temp,#0x80

MSR SPSR_c,temp

}

}
```

代码 3-6 禁止/使能总中断

更改启动代码，挂接软中断入口：

```
;软中断入口

EXPORT SWI_Handler

extern EnableIrq1

extern DisableIrq1
```

```
SWI_Handler

STMFD SP!, {R0,R12,LR} ； 入栈

LDR R0, [LR,#-4] ； 取软中断指令，软中断号就包含其中

BIC R0,R0,#0xFF000000

CMP R0,#0 ； 判断是否软中断 0

BLEQ EnableIrqFunc

BLNE DisableIrqFunc

LDMFD SP!,{R0,R12,PC}^
```

代码 3-7 挂接软中断入口

在程序中,如果想禁止中断,只需使用 DisableIrq()；若是能中断，只需使用 EnableIrq()。

3.3.3 使用分散加载机制精确定位入口地址

应用程序接收到升级指令后，会跳转到 0x00000400 处执行 Bootloader 升级程序。因此 Bootloader 程序的入口地址必须精确定位到 0x00000400 处。这可以使用 Keil MDK 提供的分散加载机制来完成。

分散加载代码见代码 3-8.

```
; *************************************************************

; *** Scatter-Loading Description File generated by uVision ***

; *************************************************************

LR_IROM1 0x00000400 0x00001C00 { ; load region size_region

ER_IROM1 0x00000400 0x00001C00 { ; load address = execution address

*.o (RESET, +First)
```

```
*(InRoot$$Sections)

.ANY (+RO)

}

RW_IRAM1 0x40000040 0x00003FA0 { ; RW data

.ANY (+RW +ZI)

}

}
```

代码 3-8 分散加载代码

这段代码显示出 Bootloader 程序从 0x00000400 处开始执行，最多占用 0x1C00 字节的 Flash 空间。另外，该程序的 RAM 从 0x40000040 开始，长度为 0x3FA0 个字节。这样 RAM 的低 64 字节保留给中断向量映射使用，高 32 字节保留给 IAP 编程使用。

3.3.4 中断向量的重映射

Bootloader 的起始地址位于 0x00000400，中断向量也从这一地址开始存储。但默认情况下 ARM 发生异常时，会跳转到 0x00000000 处的 64 字节中断向量区域执行相应操作，所以为了使 Bootloader 能相应中断，必须将位于 0x00000400 开始的 64 字节中断向量表重映射到 RAM 的低区。LPC2114 使用向寄存器 MEMMAP 写入 0x02 来完成这一过程。

代码 3-9 描述了中断向量重映射的过程。

```
; Copy Exception Vectors to Internal RAM ------------------------------------

ADR R8, Vectors ; 源地址

LDR R9, =RAM_BASE ; 目标地址，这里是 0x40000000

LDMIA R8!, {R0-R7} ; 装载向量表

STMIA R9!, {R0-R7} ; 存储向量表
```

```
LDMIA R8!, {R0-R7} ; 装载处理程序地址

STMIA R9!, {R0-R7} ; 存储处理程序地址

; Memory Mapping (when Interrupt Vectors are in RAM)

MEMMAP EQU 0xE01FC040 ; Memory Mapping Control

IF :DEF:REMAP

LDR R0, =MEMMAP

IF :DEF:EXTMEM_MODE

MOV R1, #3

ELIF :DEF:RAM_MODE

MOV R1, #2

ELSE

MOV R1, #1

ENDIF

STR R1, [R0]

ENDIF
```

代码 3-9 中断向量重映射

由于 Keil MDK 提供的启动代码中使用条件编译指令，所以，要想正确的执行中断向量重映射，还需要在 Keil MDK 编译器工程设置 Options for target“你的工程目标名”下的 Asm 标签中找到 Define 编辑框，在编辑框中键入“REMAP RAM_MODE”。如图 3-2 所示

图 3-2

注意：在擦除/编程 Flash 的时候还应该禁止 PLL、存储器加速模块。

3.4 用户程序的设计

用户程序运行在高区（扇区 8～13）或者低区（扇区 1～7），用于实现数据的采集、处理和上传等等，用户程序除本身功能的要求外，还需要注意：

Ø 使用分散加载机制，将程序入口精确定位到 0x00010000（高区）或 0x00008000（低区）。

Ø 进行中断向量重映射，映射到 RAM 最底处。

4 通讯协议与上位机软件

4.1 Intel 的 hex 格式

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。一个数据记录以一个回车和一个换行结束。

一个 Intel HEX 文件可以包含任意多的十六进制记录,每条记录有五个域,下面是一个记录的格式.

: LL AAAA TT [DD...] CC

每一组字母是独立的一域,每一个字母是一个十六进制数字,每一域至少由两个十六进制数字组成,下面是字节的描述.

:冒号 是每一条 Intel HEX 记录的开始

LL 是这条记录的长度域,他表示数据(dd)的字节数目.

AAAA 是地址域,他表示数据的起始地址

TT 这个域表示这条 HEX 记录的类型,他有可能是下面这几种类型

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

DD 是数据域,表示一个字节的数据,一个记录可能有多个数据字节,字节数目可以查看 LL 域的说明。

CC 是效验和域,表示记录的效验和,计算方法是将本条记录冒号开始的所有字母（包括校验字节）相加之后等于 0x00。

一个 Intel HEX 文件必须有一个文件结束记录,这个记录的类型域必须是 01,

一个 EOF 记录总是这样:

:00000001FF

00 是记录中数据字节的数目

0000 这个地址对于 EOF 记录来说无任何意义

01 记录类型是 01(文件结束记录标示)

4.2 对上位机软件的要求

Ø 上位机具备解析重组 Intel HEX 文件的能力.

Ø 上位机软件应能识别分站发来的应答信号并做出正确的响应。

Ø 上位机应能够检验代码的完整性。

Ø 上位机能根据分站发出的程序所在高区或低区标志，自动判别当前升级程序是否和升级区域相对应。

5 实验数据

为验证升级程序的稳定性，对分站进行重上电、复位、远程升级等一些列实验，实验记录及如下。

1. 测试程序跳转功能.程序在上电或复位之后，应顺利跳转到用户程序。

测试条件	用户程序所在区域	实验次数	失败次数
重上电	高区（0x00010000～0x0001BFFF）	30	0
	低区（0x00000200～0x0000FFFF）	30	0
复位	高区（0x00010000～0x0001BFFF）	30	0
	低区（0x00000200～0x0000FFFF）	30	0

2. 测试 Bootloader（一）。上位机发送升级命令但不发送升级数据包，程序应能进入 Bootloader 并发送当前程序所在的区域（高区或者低区代号），10S 后程序应跳转到用户程序。

测试条件	用户程序所在区域	实验次数	失败次数
发送升级命令但无数据包	高区（0x00010000～0x0001BFFF）	30	0
	低区（0x00000200～0x0000FFFF）	30	0

3.测试 Bootloader（二）。上位机发送升级命令，发送升级数据包，但发送到一半时停止发送。程序在 10S 后应能跳转到用户程序区。

测试条件	用户程序所在区域	实验次数	失败次数
发送升级命令中途停发数据包	高区（0x00010000～0x0001BFFF）	20	0
	低区（0x00000200～0x0000FFFF）	20	0

4.测试 Bootloader（三）。上位机发送升级命令，发送升级数据包，但发送中途给分站断电，重新上电后，应还能执行原来的程序。

测试条件	用户程序所在区域	实验次数	失败次数
发送升级命令和数据,中途断电	高区（0x00010000～0x0001BFFF）	20	0
	低区（0x00000200～0x0000FFFF）	20	0

5.测试 Bootloader（四）。上位机发送升级命令，发送完成升级数据包。程序应能接收升级数据包并编程 Flash，完成用户程序的更新，更新用户程序后，跳转到新的用户程序。

测试条件	用户程序所在区域	实验次数	失败次数
完整测试升级过程	高区（0x00010000～0x0001BFFF）	30	0
	低区（0x00000200～0x0000FFFF）	30	0

6.总结

本次升级方案虽然是以 LPC2114 为基础的，但任何具有 IAP 功能的单片机、ARM 都可使用本设计方案。

设计的重点在于如何保证升级的安全性，分站采取了一些列校验、超时处理以及看门狗等措施，一是保障升级数据包的正确传送，二是即使升级失败也能退回原升级程序。上位机的校验措施需相关部门配合。从实验数据来看，进行了几十次的远程升级，未有一例失败，安全性能可以得到保证。

7.参考文献：

1. 周立功等 ARM 微控制器基础与实战（第二版） 北京航空航天大学出版社 2005

2. LPC2114/2124/2212/2214 使用指南.Pdf 广州周立功单片机发展有限公司

3. 韦文祥 朱志杰 车琳娜 郭宝泉 基于 LPC21 24 的一个远程系统软件升级方案 单片机与嵌入式系统应用 2006 第三期

4. 许文杰 丁志冈 张 泉基于 ARM 处理器的 IAP 设计及应用 计算机应用与软件 2009 第 3 期

5. 姜晓梅 李祥和 任朝荣 姚明基于 ARM 的 IAP 在线及远程升级技术 计算机应用 2008 第二期

6. RealView 编译工具-编译器参考指南.pdf ARM Limited 2009.3

7. RealView Compilation Tools（连接器用户指南）.pdf ARM Limited 2009.3

8. RealView 编译工具-编译器用户指南.pdf ARM Limited 2009.1

9. **RealView** 编译工具-链接器参考指南.pdf ARM Limited 2008.9

10. Intel HEX 文件格式

11. LPC2000 secondary bootloader for code update using IAP NXP Semiconductors 2009.5.26

后记：分散加载文件，软中断，中断向量表重映射，变量对齐，精确定位变量等等这些东西的详细讲解在我的参考资料上都能找的到，发现问题并能解决它，是件很美妙的事情，所以我没打算也没时间详细写这些东西的用法。

需要说的是，我在设计的时候走了一个弯路，现在想想还觉得挺可笑。我以为上面讲的东西要在一个工程里面实现才好，这样才能生成一个.hex 可烧录文件，可以一次性的将用户程序、Bootloader 程序烧写进处理器，我想弯了。正确的做法是建四个工程：跳转程序、Bootloader、用户低区程序、用户高区程序。如果你懂了.hex 文件的格式，就完全可以将跳转程序、Bootloader 和用户低区程序（或者跳转程序、Bootloader 和用户高区程序）这三个工程生成的.hex 文件合成一个。灵活多变的处理问题，这是我最大的收获。

后记的后记: 很多同学看完后都希望得到源码,这种心情我是理解的,最初的时候我也希望有一套别人的源码的,毕竟这样可以进行的快点.所以我将一个远程升级的例子放在下面的链接里,大家想看看的就去下载吧,这个不是我产品中用到的,因为我的代码毕竟是含有公司的一些信息.如果我最近有时间,会把其中的英文文档翻译一下的.

链接:http://download.csdn.net/detail/zhzht19861011/3618966