Appplication Note

# AN070701

## IAP（在应用中编程）及其应用

Rev 1.0 Date:4/19/2007

产品应用笔记

文件信息

| 类别 | 内容 |
| --- | --- |
| 关键词 | IAP、在线升级 |
| 摘要 | 以 LPC2300 为例，讲述 IAP 功能在实际使用过程中的应用方法以及相关解决方案。 |

修订历史

| 版本 | 日期 | 原因 |
|---|---|---|
| Rev 1.0 | 2007/04/17 | 创建 |
| | | |
| | | |
| | | |

# 产品制造商与技术支持

公司：广州致远电子有限公司嵌入式系统事业部
地址：广州市天河区车陂路黄洲工业区三栋二楼（售后服务）
邮编：510660
电话：(020) 22644249 22644399（销售服务）
(020) 28872347 （技术支持）
传真：(020) 38601859
E–mail： lpc2103@ zlgmcu.com （技术支持）
网址：www.embedtools.com

# 销售与服务网络

广州周立功单片机发展有限公司
地址：广州市天河北路 689 号光大银行大厦 15 楼 F1 邮编：510630
电话：(020)38730916 38730917 38730976 38730977
传真：(020)38730925
网址：http://www.zlgmcu.com

广州专卖店
地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634 87569914
传真：(020)87578842

南京周立功
地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221 83613271 83603500
传真：(025)83613271

北京周立功
地址：北京市海淀区知春路 113 号银网中心 715 室（中发电子市场斜对面）
电话：(010)62536178 62536179 82628073
传真：(010)82614433

重庆周立功
地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1115 室
电话：(023)68796438 68796439
传真：(023)68796439

杭州周立功
地址：杭州市登云路 428 号浙江时代电子市场 205 号
电话：(0571)88009205 88009932 88009933
传真：(0571)88009204

成都周立功
地址：成都市一环路南一段 57 号金城大厦 612 室
电话：(028)85399320 85437446
传真：(028)85439505

深圳周立功
地址：深圳市深南中路 2070 号电子科技大厦 A 座 24 楼 2403 室
电话：(0755)83781768 83781788 83782922
传真：(0755)83793285

武汉周立功
地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497 87168297 87168397
传真：(027)87163755

上海周立功
地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452 53083453 53083496
传真：(021)53083491

西安办事处
地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296 83063000 85399492
传真：(029)87880865

# 目录

# 1. 适用范围

此应用笔记适用于 LPC2300 系列所有具有片内 Flash 的 ARM 芯片。

# 2. Boot 简介

LPC2300 系列处理器在出厂时，由厂家在片内固化了一段 Boot 代码。Boot 装载程序控制芯片复位后的初始化操作，并提供对 Flash 编程的方法。Boot 程序可以对芯片进行擦除、编程。

- **在系统编程：**在系统编程（ISP）是通过 Boot 装载程序和 UART0 对片内 Flash 存储器进行擦除/编程的方法，如图 2.1 所示。


图 2.1 ISP 编程示意图

- **在应用编程：**在应用编程（IAP）是用户的应用代码对片内 Flash 存储器进行擦除/编程的方法，如图 2.2 所示。

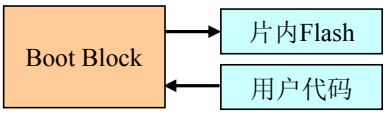


图 2.2 IAP 编程示意图

Boot 装载程序提供了 ISP 和 IAP 编程接口，可以实现对片内 Flash 存储器的编程。Boot 区位于地址 0x0007 E000 ～ 0x0007 FFFF 处。不过，芯片上电以后，会首先对 Boot 区执行一次重映射，映射到片内存储器空间的最高处，即接近 2G（0x8000 0000）的地方，如图 2.3 所示。

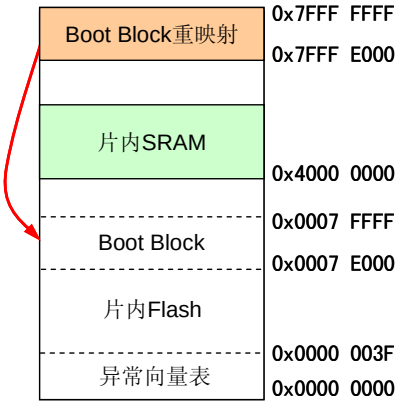


图 2.3 Boot 代码上电后的重映射

# 3. IAP 概述

IAP——在应用中编程。即用户的应用代码可以在运行过程中，自行对 Flash 存储系统进行修改。IAP 程序是 Thumb 代码，位于地址 0x7FFF FFF0。在 ARM 系统中实现状态转换的指令是“BX Addr”，目标地址 Addr 的最低位（bit0）仅来确定最终状态，实际的“目的地址= Addr & 0xFFFF FFFE”。在调用 IAP 函数时，不仅要实现跳转而且还要完成状态转换，如图 3.1 所示，具体 C 代码如程序清单 3.1 所示。

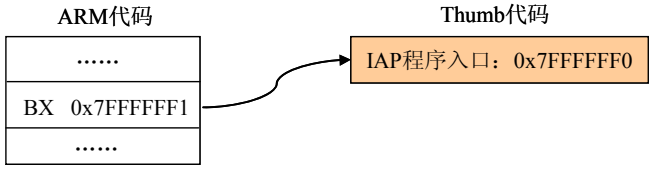


图 3.1 IAP 函数调用示意图

程序清单 3.1 IAP 函数的调用方法

```
#define     IAP_LOCATION     0x7FFFFFF1                    // IAP 程序入口          （1）
typedef     void (*IAP)  (unsigned  int [ ] , unsigned  int [ ]);    //  定义函数类型指针
 ……
IAP  iap_entry;
unsigned  long   command[5];                  // IAP 命令表
unsigned  long   result[2];                   // IAP 返回值
iap_entry  =  (IAP)  IAP_LOCATION;            //  设置函数指针
iap_entry (command , result);                 //  调用 IAP
```

程序说明：

（1）执行跳转指令时，如果“目的地址”的 bit0 = 1，表示处理器需要进行状态切换，由 ARM 状态切换到 Thumb 状态，Thumb 代码是“半字”对齐格式，即地址 & 0xFFFF FFFE。因此，虽然此处跳转目的地址 = 0x7FFFFFF1，实际上跳转到地址 0x7FFFFFF0，同时进行切换到 Thumb 状态。

由于 IAP 是 Thumb 代码，因此，需要在 ADS 编译选项中，选中 **ARM/Thumb Interworking** 选型，如图 3.2 所示。

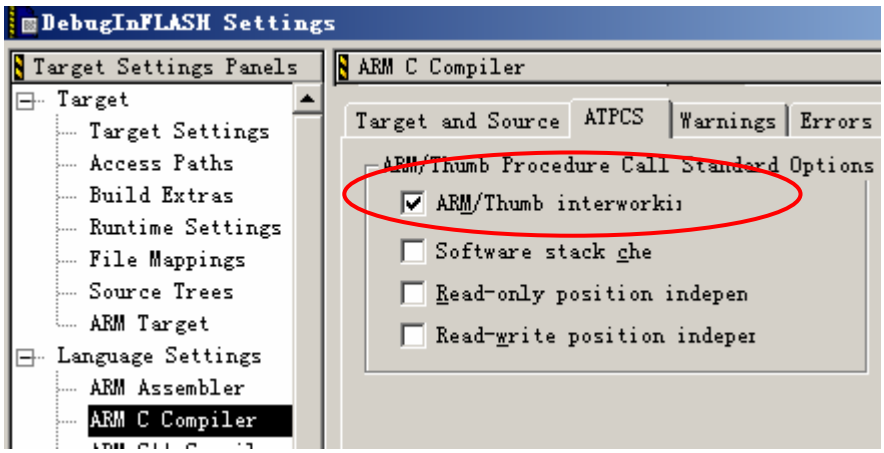


图 3.2 ARM/Thumb Interworking 选项

IAP 程序会使用片内 RAM 空间的顶部 32 个字节，如图 3.3 所示。因此，在支持 IAP 的场合，用户程序应该避免使用这部分空间，可以在启动代码 Startup.s 文件中的 InitStack 函数内调整各个模式的堆栈空间位置，CPU 的模式可以通过修改“程序状态寄存器”来切换，“程序状态寄存器”的位分配如图 3.4 所示，有关“程序状态寄存器”的描述请参考其它书籍中有关“ARM7 体系结构”方面的介绍，这里不再重述。模式堆栈的修改代码如程序清单 3.2 所示。

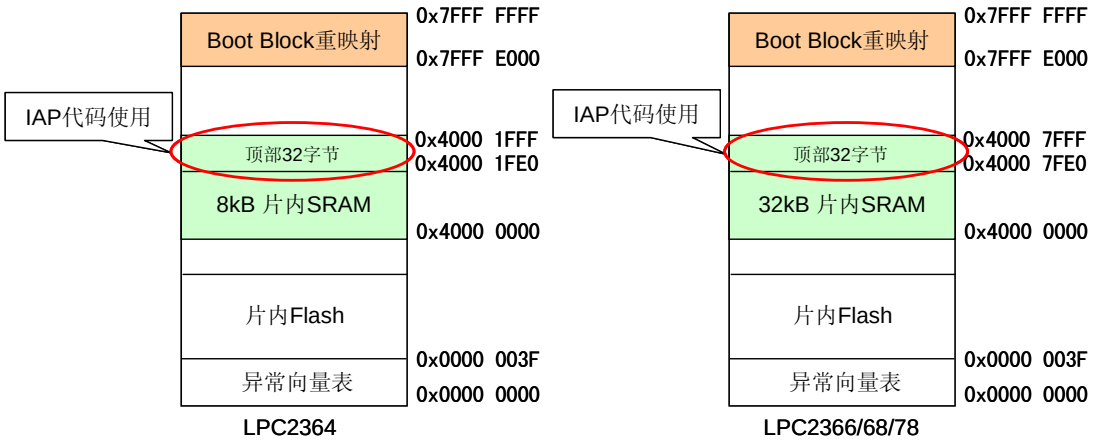


图 3.3 IAP 代码占用的 RAM 空间

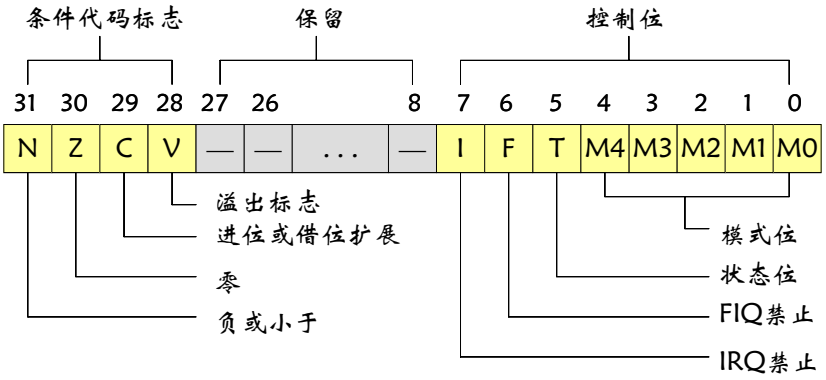


图 3.4 程序状态寄存器

程序清单 3.2 堆栈分配函数

```
InitStack
    ……
    ;设置系统模式堆栈
    MSR     CPSR_c, #0xdf
    LDR     SP, =StackUsr – 32          ;// 避免使用片内 RAM 的顶部 32 个字节
    MOV     PC, R0
```

IAP 的命令如表 3.1 所示，IAP 各命令返回代码及意义见表 3.2。

表 3.1 IAP 命令汇总

| ISP 命令 | 命令代码 |
|---|---|
| 准备编程扇区 | 50 |
| 将 RAM 内容复制到 Flash | 51 |
| 擦除扇区 | 52 |

续上表

| ISP 命令 | 命令代码 |
|---|---|
| 扇区查空 | 53 |
| 读器件 ID | 54 |
| 读 boot 代码版本 | 55 |
| 比较 | 56 |

表 3.2 IAP 状态代码汇总

| 返回代码 | 符号 | 描述 |
|---|---|---|
| 0 | CMD_SUCCESS | 命令被成功执行。 |
| 1 | INVALID_COMMAND | 无效命令。 |
| 2 | SRC_ADDR_ERROR | 源地址没有以字为边界。 |
| 3 | DST_ADDR_ERROR | 目标地址的边界错误。 |
| 4 | SRC_ADDR_NOT_MAPPED | 源地址没有位于存储器映射中。计数值必须考虑可用性。 |
| 5 | DST_ADDR_NOT_MAPPED | 目标地址没有位于到存储器映射中。计数值必须考虑到可用性。 |
| 6 | COUNT_ERROR | 字节计数值不是 4 的倍数或是一个非法值。 |
| 7 | INVALID_SECTOR | 扇区号无效。 |
| 8 | SECTOR_NOT_BLANK | 扇区非空。 |
| 9 | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | 为写操作准备扇区命令未执行。 |
| 10 | COMPARE_ERROR | 源和目标数据不相等。 |
| 11 | BUSY | Flash 编程硬件接口忙 |

# 4. LPC2300 系列处理器片内 Flash 存储系统

在利用 IAP 代码来操作片内 Flash 时，必须熟悉片内 Flash 的扇区分布。片内 Flash 的操作是以“扇区”为单位进行的，每个“扇区”的大小不定，图 4.1 列出了 LPC2300 片内 Flash 和 Boot Block 的关系，表 4.1 列出了 LPC2300 系列器件的扇区数和存储器地址之间的对应关系。由于 IAP 代码位于 Boot 区内，所以 IAP 命令不允许对 Boot 扇区执行写/擦除操作。另外，对于 LPC2368/78 来说，Boot 区位于 512kB Flash 的顶部，因此在 LPC2368/78 器件中，只有 504kB Flash 可供用户使用，如图 4.1 所示。

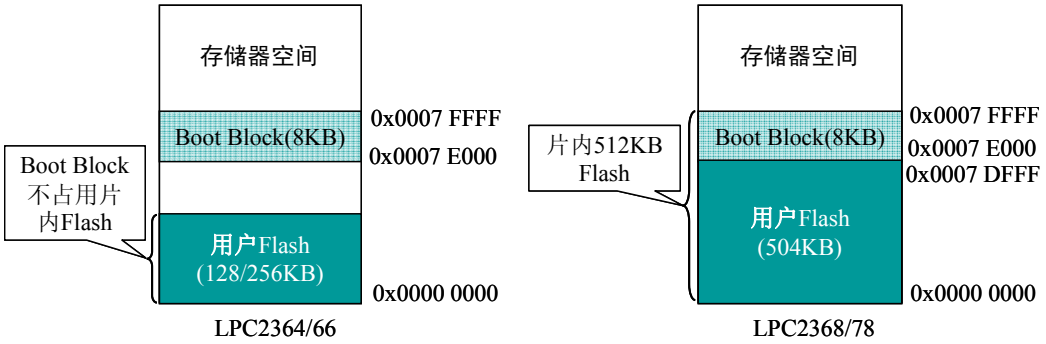


图 4.1 LPC2300 片内 Flash 与 Boot Block 的关系

表 4.1 LPC2300 器件的扇区分布

| 扇区号 | 扇区规格（KB） | 地址范围 | LPC2364（128kB ） | LPC2366（256kB ） | LPC2368/78（512kB） |
|---|---|---|---|---|---|
| 0 | 4 | 0x0000 0000 -0x0000 0FFF | √ | √ | √ |
| 1 | 4 | 0x0000 1000 -0x0000 1FFF | √ | √ | √ |
| 2 | 4 | 0x0000 2000 -0x0000 2FFF | √ | √ | √ |
| 3 | 4 | 0x0000 3000 -0x0000 3FFF | √ | √ | √ |
| 4 | 4 | 0x0000 4000 -0x0000 4FFF | √ | √ | √ |
| 5 | 4 | 0x0000 5000 -0x0000 5FFF | √ | √ | √ |
| 6 | 4 | 0x0000 6000 -0x0000 6FFF | √ | √ | √ |
| 7 | 4 | 0x0000 7000 -0x0000 7FFF | √ | √ | √ |
| 8 | 32 | 0x0000 8000 -0x0000 FFFF | √ | √ | √ |
| 9 | 32 | 0x0001 0000 -0x0001 7FFF | √ | √ | √ |
| 10（0x0A） | 32 | 0x0001 8000 -0x0001 FFFF | √ | √ | √ |
| 11（0x0B） | 32 | 0x0002 0000 -0x0002 7FFF | — | √ | √ |
| 12（0x0C） | 32 | 0x0002 8000 -0x0002 FFFF | — | √ | √ |
| 13（0x0D） | 32 | 0x0003 0000 -0x0003 7FFF | — | √ | √ |
| 14（0x0E） | 32 | 0x0003 8000 -0x0003 FFFF | — | √ | √ |
| 15（0x0F） | 32 | 0x0004 0000 -0x0004 7FFF | — | — | √ |
| 16（0x10） | 32 | 0x0004 8000 -0x0004 FFFF | — | — | √ |
| 17（0x11） | 32 | 0x0005 0000 -0x0005 7FFF | — | — | √ |
| 18（0x12） | 32 | 0x0005 8000 -0x0005 FFFF | — | — | √ |
| 19（0x13） | 32 | 0x0006 0000 -0x0006 7FFF | — | — | √ |

续上表

| 扇区号 | 扇区规格（KB） | 地址范围 | LPC2364（128kB ） | LPC2366（256kB ） | LPC2368/78（512kB） |
|---|---|---|---|---|---|
| 20（0x14） | 32 | 0x0006 8000 -0x0006 FFFF | — | — | √ |
| 21（0x15） | 32 | 0x0007 0000 -0x0007 7FFF | — | — | √ |
| 22（0x16） | 4 | 0x0007 8000 -0x0007 8FFF | — | — | √ |
| 23（0x17） | 4 | 0x0007 9000 -0x0007 9FFF | — | — | √ |
| 24（0x18） | 4 | 0x0007 A000 -0x0007 AFFF | — | — | √ |
| 25（0x19） | 4 | 0x0007 B000 -0x0007 BFFF | — | — | √ |
| 26（0x1A） | 4 | 0x0007 C000 -0x0007 CFFF | — | — | √ |
| 27（0x1B） | 4 | 0x0007 D000 -0x0007 DFFF | — | — | √ |

# 5. 数据存储解决方案

通过 IAP，用户可以使用片内 Flash 作为非易失性数据存储器，存储一些设备的配置信息。这样不仅可以节约成本，而且还可以减小线路板的面积。

利用 IAP 将 Flash 作为数据存储器时，用户需要控制自身代码量的大小及代码定位。绝对不能够出现 Flash 数据区和 Flash 代码区重叠的现象。因为在利用 IAP 向片内 Flash 存储器写入数据时，需要对数据扇区进行擦除。如果数据区和代码区重合，就有可能会破坏系统的代码空间，造成系统死机或崩溃。本方案的 Flash 分布如表 5.1 所示。

表 5.1 Flash 空间分配

| 用途 | 扇区 | 大小 |
|---|---|---|
| 代码空间 | 0～6 | 28kB |
| 数据空间 | 7 | 4kB |

需要注意一点：任何 Flash 都是有寿命的，如果对片内 Flash 的操作过于频繁，就会对其造成损坏，LPC2300 的 Flash 擦除/写入次数为 10 万次。

**使用 IAP 将 SRAM 中的数据编程到 Flash 时，源数据区只能够使用片内局部总线上的 SRAM，不能使用通用 USB SRAM 和以太网 SRAM，如**图 5.1 **所示。**

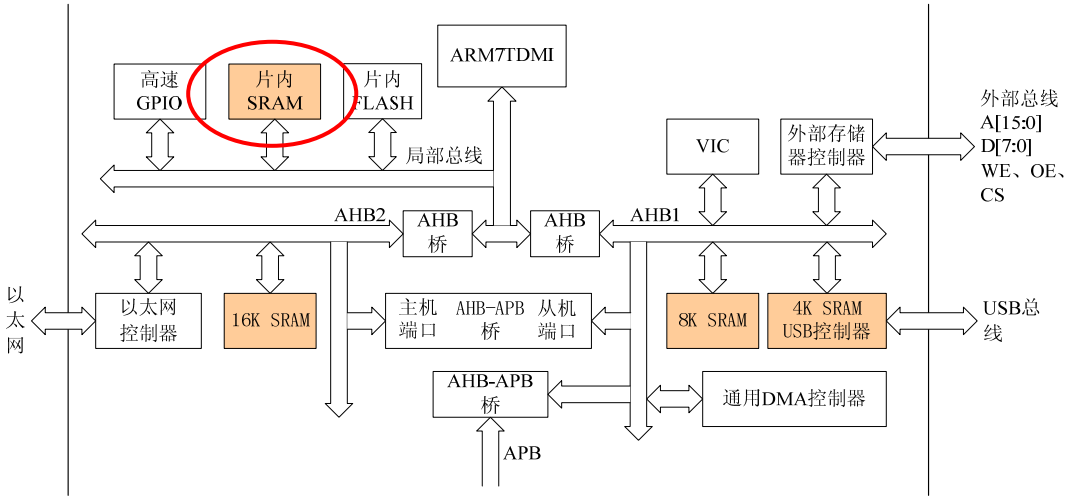


图 5.1 LPC2300 片内存储器分布图

## 5.1 系统概述

通过 IAP 向片内 Flash 的扇区 7 写入 512 个字节的数据。

## 5.2 编程片内 Flash 的步骤

使用 IAP 函数对片内 Flash 执行编程操作时，需要按步骤进行操作，如图 5.2 所示，在对某一个扇区进行擦除/编程操作之前，必须选择扇区，然后才能正常操作。

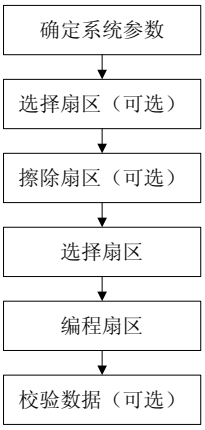


图 5.2 片内 Flash 的编程步骤

**（1）确定参数**

在使用 IAP 代码之前，需要定义一些常量，如系统时钟、IAP 函数入口、IAP 入口缓冲区和出口缓冲区等，如程序清单 5.1 所示。

程序清单 5.1 IAP 代码全局变量定义

```
/*定义 CCLK 值大小，单位为 KHz*/
#define     IAP_FCCLK           48000
#define     IAP_ENTER_ADR       0x7FFFFFF1      // IAP 入口地址定义
uint32      paramin[8];                         // IAP 入口参数缓冲区
uint32      paramout[8];                        // IAP 出口参数缓冲区
```

**（2）选择扇区**

对某一个扇区执行擦除、写入等操作之前，必须先选择该扇区。但也可以一次选择多个扇区，程序清单 5.2 为扇区选择函数。

程序清单 5.2 扇区选择函数

```
/*********************************************************************************************
** 函数名称：  SelSector
** 函数功能：  IAP 操作缓冲区选择，代码为 50。
** 入口参数：  sec1        起始扇区
**             sec2        终止扇区
** 出口参数：  IAP         操作状态码
**             IAP         返回值（paramout 缓冲区）
*********************************************************************************************/
uint32  SelSector(uint8  sec1,uint8  sec2)
{
    paramin[0] = IAP_SELECTOR;          // 设置命令字
    paramin[1] = sec1;                  // 设置参数
    paramin[2] = sec2;
    (*(void(*)())IAP_ENTER_ADR)(paramin,paramout);      // 调用 IAP 服务程序
    return(paramout[0]);                                // 返回状态码
}
```

**（3）擦除扇区**

同其它的 Flash 芯片一样，LPC2300 的片内 Flash 在写入数据前也需要执行擦除操作。不过这一步是可选的。如果目标区域已经被擦除了，那么就不必重复擦除，直接写入数据即

可。擦除操作一次可以擦除多个扇区，如程序清单 5.3 所示。

程序清单 5.3 扇区擦除函数

```
/*********************************************************************************************
** 函数名称：  EraseSector
** 函数功能：  擦除扇区，命令代码 52。
** 入口参数：  sec1        起始扇区
**             sec2        终止扇区
** 出口参数：  IAP         操作状态码
**             IAP         返回值（paramout 缓冲区）
*********************************************************************************************/
uint32  EraseSector(uint32  sec1,uint32  sec2)
{
    paramin[0] = IAP_ERASESECTOR;                    // 设置命令字
    paramin[1] = sec1;                               // 设置参数
    paramin[2] = sec2;
    paramin[3] = IAP_FCCLK;
    (*(void(*)())IAP_ENTER_ADR)(paramin,paramout);   // 调用 IAP 服务程序
    return(paramout[0]);                             // 返回状态码
}
```

**（4）编程 Flash**

以上几步执行完毕后，就可以编程 Flash 了。执行编程扇区的操作时，IAP 函数会将 RAM 中的数据拷贝到 Flash 中，此时有几个限制：

- Flash 的目标地址必须是 256 字节对齐，即目标地址[7：0]为 0，如图 5.3 所示；
- RAM 数据区的源地址必须字对齐，即起始地址[1：0]为 0，如图 5.3 所示；
- 源数据区必须是局部总线上的 SRAM，不能使用通用 USB SRAM 和以太网 SRAM；
- 一次写入的字节数固定：256、512、1024 或者 4096。

小知识：如果一个数据是从偶地址开始的连续存储，那么它就是半字对齐，否则就是非半字对齐；如果一个数据是以能被 4 整除的地址开始的连续存储，那么它就是字对齐，否则就是非字对齐。如图 5.3 所示。

| 方式 | 半字对齐 | 字对齐 |
|---|---|---|
| 地址 | ……<br>0x4002<br>0x4004<br>…… | ……<br>0x4004<br>0x4008<br>…… |
| 特征 | Bit0=0<br>其他位为任意值 | Bit1=0,Bit0=0<br>其他位为任意值 |

图 5.3 半字、字对齐示意图

程序清单 5.4 所示为编程 Flash 的函数代码。

程序清单 5.4 编程 Flash 函数

```
/*********************************************************************************************
** 函数名称：  RamToFlash
** 函数功能：  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 256 字节为分界
**             src         源地址，即 RAM 地址，地址必须字对其
```

```
**                no          复制字节个数，为 256/512/1024/4096
** 出口参数：  IAP   操作状态码
**                IAP   返回值（paramout 缓冲区）
*********************************************************************************************/
uint32   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] = IAP_FCCLK;
      (*(void(*)())IAP_ENTER_ADR)(paramin,paramout);     //调用 IAP 服务程序
      return(paramout[0]);                //返回状态码
}
```

**（5）校验数据**

IAP 代码还为用户提供了一个数据校验的手段，这样用户就可以不必自己动手来校验写入 Flash 中的数据是否正确。用户需要提供目标地址、源地址和比较字节的个数。注意：源地址、目的地址和比较字节的个数都必须是字对齐。如程序清单 5.5 所示。

**程序清单 5.5 校验数据函数**

```
/*********************************************************************************************
** 函数名称：  Compare
** 函数功能：  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 RAM/RAM 地址，地址必须字对齐
**                no          比较字节个数，必须能被 4 整除
** 出口参数：  IAP         操作状态码
**                IAP         返回值（paramout 缓冲区）
*********************************************************************************************/
uint32   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;
      (*(void(*)())IAP_ENTER_ADR)(paramin,paramout);     // 调用 IAP 服务程序
      return(paramout[0]);                                      // 返回状态码
}
```

## 5.3 程序主体

在这个程序中，我们向扇区 7 写入 512 个字节的数据，然后通过 AXD 的内存窗口来观察 Flash 中的数据。

如程序清单 5.6 所示，扇区 7 就可以作为一个 $E^2$PROM 来使用，将系统的一些配置信息存储在这里。

程序清单 5.6 主程序代码

```
#define     DestAddr   0x00007000              // 扇区 7 的起始地址
/*********************************************************************************************
** 函数名称：main
** 函数功能：数据存储解决方案。
*********************************************************************************************/
int main (void)
{
    __align(4)   uint8   SendData[512];              // 定义变量区
    uint32   i;
    for(i = 0; i < 512; i++)                         // 初始化变量区数据
    {
        SendData[i] = i;
    }
    SelSector(7, 7);                                 // 选择扇区
    EraseSector(7,7);                                // 擦除扇区
    SelSector(7, 7);                                 // 选择扇区
    RamToFlash(DestAddr, (uint32)SendData, 512);     // 写数据到 FLASH
    while(1);
    return 0;
}
```

从 AXD 内存变量窗口中观察到的数据如图 5.4 所示。

| Address | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | ASCII |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00007000 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | ..... |
| 0x00007010 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F | ..... |
| 0x00007020 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F | !"#$%&'()*+,-./ |
| 0x00007030 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F | 0123456789:;<=>? |
| 0x00007040 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F | @ABCDEFGHIJKLMNO |
| 0x00007050 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F | PQRSTUVWXYZ[\]^_ |
| 0x00007060 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F | `abcdefghijklmno |
| 0x00007070 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | pqrstuvwxyz{\|}~ |
| 0x00007080 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | ..... |
| 0x00007090 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | ..... |
| 0x000070A0 | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | ..... |
| 0x000070B0 | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | ..... |
| 0x000070C0 | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | ..... |

图 5.4 Flash 扇区 7 中的数据

# 6. 在线升级解决方案

“在线升级”实际上是 Flash 数据存储的一个应用特例。系统开发完毕后，在应用过程中，如果需要增加部分功能，那么为了避免重新拆装设备，可以借助“在线升级”方式。

目前，在线升级是很多系统都必需的一个功能，对于 LPC2300 来说，利用 IAP 函数即可实现在线升级。用户程序接收新的代码，然后调用 IAP 函数将新的代码编程到 Flash 扇区中，实现在线升级。

## 6.1 系统概述

系统通过串口接收升级代码，然后调用 IAP 函数实现在线升级。为了实现在线升级，本系统将芯片的片内 Flash 重新分区，如图 6.1 所示。

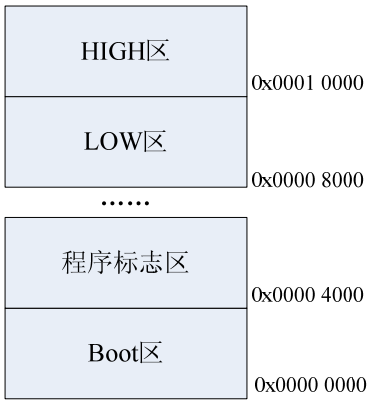


图 6.1 片内 Flash 分布图

- **Boot 代码区：**又叫固件区，存放系统的 BootLoader，可以完成代码升级。Boot 区的首地址位于 0x0000 0000；
- **LOW 区和 HIGH 区：**用户代码分为两个区，LOW 区和 HIGH 区，当程序位于 LOW 区时，可以对 HIGH 区进行升级。反之，如果程序位于 HIGH 区，可以对 LOW 区进行升级。LOW 区的首地址为：0x0000 8000，HIGH 区的首地址为：0x0001 0000，每个用户代码区的容量为 32kB；
- **程序标志区：**标记当前用户程序运行的区，存储用户程序区的首地址。程序标志区的首地址为：0x0000 4000。对于这个区间，仅仅使用了前 4 个字节，用来保存当前用户代码区的首地址。
  - 如果程序标志 =0x0000 8000，则当前程序运行在 LOW 区；
  - 如果程序标志 =0x0001 0000，则当前程序运行在 HIGH 区；
  - 如果程序标志为其它值，则当前程序运行在固件区域。

## 6.2 软件设计

系统复位后，会首先运行 BootLoader，BootLoader 判断引脚 P0.6 的电平，如果该引脚为低电平，那么会通过串口接收新的用户代码，并调用 IAP 函数将代码编程到 LOW 区或者 HIGH 区。接下来，运行新的用户代码。BootLoader 的软件流程如图 6.2 所示。

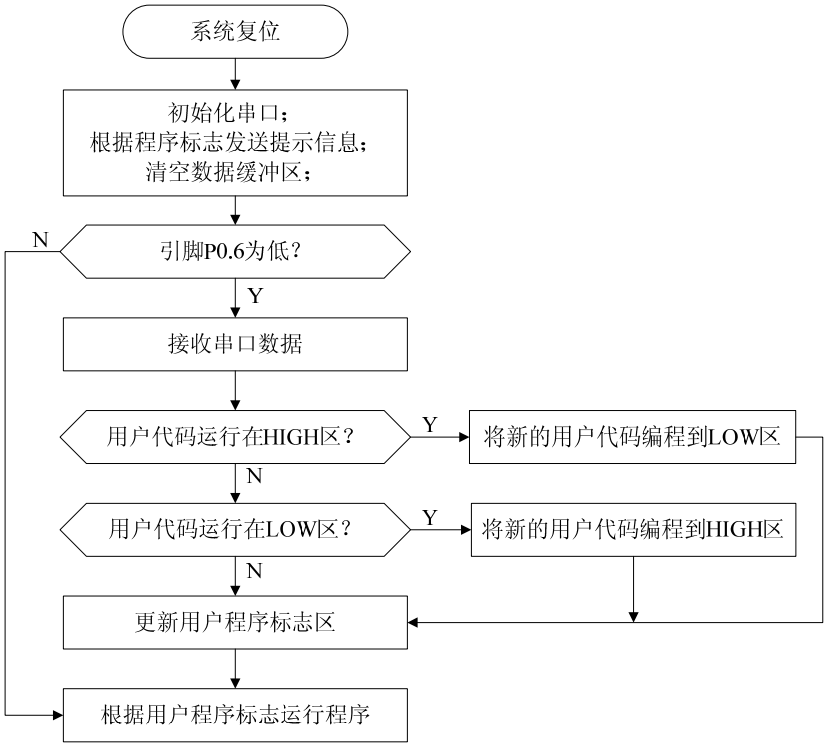


图 6.2 BootLoader 流程图

**（1）内存分布**

在实现在线升级时，接收数据缓冲区使用 16kB 以太网缓冲区，编程 Flash 时，使用片内 SRAM（局部总线上的 SRAM），如程序清单 6.1 所示。

程序清单 6.1 系统内存分布

```
#define     Ethernet_RAM   0x7FE00000          // 以太网 RAM 起始地址
uint8   *RcvData = (uint8 *)Ethernet_RAM;      // 接收数据缓冲区
__align(4)  uint8   IAP_Tmp[4096];             // 定义 4k 空间，编程 Flash 时使用
```

需要注意一点：在使用以太网缓冲区时，必须为以太网模块提供时钟，如程序清单 6.2 所示。

程序清单 6.2 为以太网模块提供时钟

```
PCONP |= (1 << 30);                    // 为以太网模块提供时钟
```

**（2）UART0 接口初始化**

由于系统使用串口接收数据，所以必须合理初始化 UART，UART 的初始化函数如程序清单 6.3 所示。

程序清单 6.3 UART0 初始化函数

```
/*********************************************************************************************
** 函数名称：UART0_Init
** 功能描述：对 UART0 进行初始化
** 入口参数：baud   串口通信波特率
** 出口参数：无
*********************************************************************************************/
void   UART0_Init(uint32   baud)
{
    PCONP |= (1 << 3);
    PINSEL0 = (PINSEL0 & (~(0x0f << 4))) | (1 << 4) | (1 << 6);          // 初始化 UART0 引脚
```

```
    /* 设置串口波特率 */
    U0LCR = 0x80;
    U0DLM = ((Fpclk/16)/baud)/256;
    U0DLL = ((Fpclk/16)/baud)%256;
    /* 设置串口模式：8 位数据位、1 位停止位、无奇偶校验位 */
    U0LCR = 0x03;
    U0FCR = 0x81;                                   // FIFO 触发点为 8 个字节
    U0IER = 0x01;                                   // 接收中断使能
    IRQ_Init(UART0_INT, 0, (uint32)UART0_ISR);    // 初始化 UART IRQ 中断信息
}
```

**（3）UART0 中断服务函数**

这里将 UART0 的通道优先级设置为 0，服务程序为 UART0_ISR，完成串口接收任务。如程序清单 6.4 所示。有关 UART 的操作，请参考前面介绍 UART 的相关内容，这里不再重述。串口接收到的数据全部保存到以太网缓冲区——RcvData 中。

程序清单 6.4 UART0 中断服务函数

```
/*********************************************************************************************
** 函数名称：UART0_ISR
** 函数功能：串口中断服务函数。
*********************************************************************************************/
__irq  void  UART0_ISR(void)
{
    volatile uint8 i;
    T0TCR |= 0x02;                  // 定时器复位
    T0TCR = 0x01;
    do
    {
        switch (U0IIR & 0x0e)
        {
            case  0x04:             // 接收中断,FIFO 满中断
                for(i = 0; i < RxHardFIFO_Size - 1; i++)
                {
                    RcvData[RcvCount++] = U0RBR;
                }
                break;
            case  0x0c:             // 接收超时中断
                while((U0LSR & 0x01) != 0)
                {
                    RcvData[RcvCount++] = U0RBR;
                }
                RcvOver = 1;    // 接收完成
                break;
            case  0x02:             // 发送中断
                break;
            case  0x06:
```

```
                    i = U0LSR;
                    break;
                default:
                    break;
            }
        }while((U0IIR & 0x01) == 0);
        VICVectAddr = 0x00;
}
```

**（4）用户代码区编程**

用户代码区编程函数（ProgramUserData）会将接收到的串口数据编程到 LOW 区或者 HIGH 区。串口数据保存在 RcvData 中，接收字节数保存在 RcvCount 中。具体的代码如程序清单 6.5 所示。

**程序清单 6.5 编程用户代码函数**

```
/*********************************************************************************************
** 函数名称: ProgramUserData
** 功能描述: 编程用户代码区。
** 入口参数: 无
** 出口参数: 无
*********************************************************************************************/
void   ProgramUserData(void)
{
    uint32   Addr;                      // Addr:字节偏移量
    uint32   ProgramCount;              // ProgramCount:编程到 Flash 扇区的字节数
    if(*FlagPoint == 0x10000)           // 当前程序运行在 HIGH 区，对 LOW 区进行升级      （1）
    {
        SelSector   (8, 8);             // 选择 LOW 区
        EraseSector(8, 8);              // 擦除 LOW 区
    }
    else                                // 当前程序运行在 LOW 区，对 HIGH 区进行升级      （2）
    {
        SelSector   (9, 9);             // 选择 HIGH 区
        EraseSector(9, 9);              // 擦除 HIGH 区
    }
    Addr = 0;                           // 字节偏移量清 0
    while(RcvCount != 0)                                                              （3）
    {
        if(RcvCount > (1024 * 4))       // 一次最多写入 4K 代码量                        （4）
        {
            memcpy(IAP_Tmp , RcvData + Addr, 1024 * 4);
            RcvCount -= (1024 * 4);
            ProgramCount = 1024 * 4;
        }
        else                                                                          （5）
        {
```

```
                memcpy(IAP_Tmp , RcvData + Addr, RcvCount);
                ProgramCount = RcvCount;
                RcvCount = 0;
                if( (ProgramCount == 256) || (ProgramCount == 512)
                  || (ProgramCount == 1024) || (ProgramCount == 4096) )
                {
                    goto ProgramFlash;
                }
                /* 满足编程字节数的要求,256、512、1024 等 */
                if(ProgramCount < 256)                                          （6）
                {
                    ProgramCount = 256;
                    goto ProgramFlash;
                }
                if(ProgramCount < 512)                                          （7）
                {
                    ProgramCount = 512;
                    goto ProgramFlash;
                }
                if(ProgramCount < 1024)                                         （8）
                {
                    ProgramCount = 1024;
                    goto ProgramFlash;
                }
                if(ProgramCount < 4096)                                         （9）
                {
                    ProgramCount = 4096;
                    goto ProgramFlash;
                }
            }
ProgramFlash:
            /* 升级用户程序空间 */
            if(*FlagPoint == 0x10000)  // 当前程序运行在 HIGH 区，需要对 LOW 区进行升级      （10）
            {
                SelSector(8, 8);        // 选择 LOW 区
                RamToFlash(User_LOW + Addr, (uint32)IAP_Tmp, ProgramCount);     // 编程 LOW 区
                Addr += ProgramCount;
            }
            else                        // 当前程序运行在 LOW 区，需要对 HIGH 区进行升级      （11）
            {
                SelSector(9, 9);        // 选择 HIGH 区
                RamToFlash(User_HIGH + Addr, (uint32)IAP_Tmp, ProgramCount);    // 编程 HIGH 区
                Addr += ProgramCount;
            }
```

```
    }
}
```

程序说明：

（1）、（2）首先需要擦除另外一个用户代码区，不能擦除当前代码区；

（3）如果接收数据不为 0，那么就需要向 Flash 中写入数据；

（4）调用 IAP 编程 Flash 时，一次最多只能够写入 4096 个字节的数据；

（5）～（9）调用 IAP 时，一次写入的字节数有明确规定：256、512、1024 和 4096。如果编程字节数不是这些数字时，那么就需要进行调整；

（10）、（11）不能升级当前用户代码区，只能够升级另外一个代码区。

**（5）更新程序标志区**

升级完用户程序区后，需要更新用户程序标志，函数 UpdateUserFlag 负责完成用户程序标志的更新。如程序清单 6.6 所示。用户程序标志位于地址 0x0000 4000——扇区 4 的起始地址，因此需要对扇区 4 进行编程。这段代码比较简单，用户参考注释部分可以很容易理解。

程序清单 6.6 更新用户程序区标志

```
/*********************************************************************************************
**  函数名称: UpdateUserFlag
**  功能描述: 更新用户程序标志区。
**  入口参数: 无
**  出口参数: 无
*********************************************************************************************/
void UpdateUserFlag(void)
{
    uint32   *Data32Point;
    /*  更新用户程序标志空间 0x4000 */
    memset(IAP_Tmp, 0xff, 256);                     // 临时缓冲区清空
    Data32Point = (uint32 *)IAP_Tmp;
    if(*FlagPoint == 0x10000)                       // 当前程序位于 HIGH 区，新程序位于 LOW 区
    {
        *Data32Point = 0x8000;
    }
    else                                            // 当前程序位于 LOW 区，新程序位于 HIGH 区
    {
        *Data32Point = 0x10000;
    }
    SelSector   (4, 4);                             // 选择扇区 4
    EraseSector(4, 4);                              // 擦除扇区 4
    SelSector   (4, 4);                             // 选择扇区 4
    RamToFlash(User_Flag, (uint32)IAP_Tmp, 256);    // 编程 FLASH 扇区 4
}
```

**（6）发送提示信息**

系统复位后，会通知用户当前程序的运行区域，以方便用户升级。方法：读取用户程序标志区，然后根据这个标志来进行判断，如程序清单 6.7 所示。

- **标志为 0x1 0000：**发送信息“当前程序运行在 HIGH 区，只能对 LOW 区进行升级”；
- **标志为 0x0 8000：**发送信息“当前程序运行在 LOW 区，只能对 HIGH 区进行升级”；
- **标志为其它值：**发送信息 “当前程序运行在固件区，可以升级任何区域”。

程序清单 6.7 发送提示信息

```
/*********************************************************************************************
** 函数名称： SendMessage
** 功能描述： 发送提示信息。
** 入口参数： 无
** 出口参数： 无
*********************************************************************************************/
void SendMessage(void)
{
    volatile uint8 tmp;                    // 用来清除 UART 中断标志
    if(*FlagPoint == 0x10000)              // 当前程序运行在 HIGH 区，需要对 LOW 区进行升级
    {
        UART0_SendData( (uint8 *)("当前程序运行在 HIGH 区，只能对 LOW 区进行升级 "),
                        strlen("当前程序运行在 HIGH 区，只能对 LOW 区进行升级 "));
    }
    else
    {
        if(*FlagPoint == 0x8000)           // 当前程序运行在 HIGH 区，需要对 LOW 区进行升级
        {
            UART0_SendData( (uint8 *)("当前程序运行在 LOW 区，只能对 HIGH 区进行升级 "),
                            strlen("当前程序运行在 LOW 区，只能对 HIGH 区进行升级 "));
        }
        else                               // 当前程序运行在 HIGH 区，需要对 LOW 区进行升级
        {
            UART0_SendData( (uint8 *)("当前程序运行在固件区，可以升级任何区域 "),
                            strlen("当前程序运行在固件区，可以升级任何区域 "));
        }
    }
    tmp = U0IIR;            // 清除发送中断标志
}
```

**（7）启动“在线升级”流程**

复位后，如果系统 UserISP 引脚（P0.16）为低电平，则表示进入“在线升级”流程。这里我们引入了超时机制，若在 30 秒内没有接收到用户代码，那么就会退出在线升级，而直接运行原用户程序，如程序清单 6.8 所示。

程序清单 6.8 main 函数部分代码——超时机制

```
/*********************************************************************************************
** 函数名称：main
** 函数功能：在线升级函数 Boot 代码。
*********************************************************************************************/
int  main (void)
{
```

```
    ……
    PCONP |= (1 << 30);                          // 使用以太网 SRAM，必须为以太网模块提供时钟
    UART0_Init(9600);                            // 初始化 UART0
    SendMessage();                               // 发送提示信息
    memset(IAP_Tmp, 0, 4096);                    // 缓冲区清零
    memset((char *)RcvData, 0, 1024 * 8);
    if((Read_P0() & UserISP) == 0)                                          （1）
    {/* 进入升级阶段 */
        T0MAT_Init(Fpclk * 30, 0, 1, 0, 0, 1);   // 30 秒钟定时             （2）
        RcvOver   = 0;                                                      （3）
        RcvCount  = 0;                                                      （4）
        while (RcvOver == 0)                                                （5）
        {
            if((T0IR & 0x01) != 0)                                          （6）
            {
                T0IR   = 0x01;                   // 清除中断标志            （7）
                RcvCount = 0;                                               （8）
                break;                           // 时间到，退出接收程序
            }
        }
        if(RcvCount != 0)                        // 升级用户代码            （9）
        {
            IRQDisable();
            ProgramUserData();                   // 编程用户代码区
            UpdateUserFlag();                    // 更新用户程序标志区
            IRQEnable();
        }
    }
    /* 运行用户程序 */
    ……
}
```

程序说明：

（1）复位后，UserISP（P0.16）为低电平，会进入“在线升级”流程；

（2）利用定时器的匹配功能实现 30 秒超时，有关定时器的设置方法，这里不再重述；

（3）RcvOver：接收完成标志，如果接收完成后，RcvOver 为 1，否则一直为 0。
该变量在 UART0 中断服务函数中进行操作；

（4）RcvCount：接收字节数。该变量在 UART0 中断服务函数中进行操作；

（5）等待接收完毕；

（6）、（7）、（8）如果接收超时，就会强行退出等待，并将接收字节数清零；

（9）只有接收字节数大于 0，才能够启动升级流程。

**（8）启动用户程序**

如果用户程序区（LOW 区或者 HIGH 区）存在有效代码，那么就可以使用函数指针的方式运行用户程序。如果不存在有效代码，那么就运行 Boot 区的程序。用户代码是否有效

是通过“程序标志区”的内容进行判断的，如程序清单 6.9 所示。这部分代码比较简单，用户可以参考注释部分进行理解。

程序清单 6.9 main 函数部分代码——运行用户程序

```
int   main (void)
{
      void (*UserProgram)();                          // 函数指针
      uint32   dly;
      ……
      /* 运行用户程序 */
      if(*FlagPoint == HIGH)
      {
            UserProgram = (void (*)()) (HIGH);        // 运行 HIGH 区代码
      }
      else
      {
            if(*FlagPoint == LOW)
            {
                  UserProgram = (void (*)()) (LOW);   // 运行 LOW 区代码
            }
            else
            {
                  while(1)                            // 在 Boot 区运行
                  {
                        ……
                  }
            }
      }
      (*UserProgram)();                               // 启动用户区程序
     return 0;
}
```

**（9）“在线升级”模板**

为了实现在线升级，我们专门设计了一套新的模板，如图 6.3 所示。该模板具有两个编译选项：HIGH 和 LOW。

- 当需要升级 HIGH 区时，就使用 HIGH 选项进行编译，用户代码位于 HIGH 区；
- 当需要升级 LOW 区时，就使用 LOW 选项进行编译，用户代码位于 LOW 区。

具体使用哪一个选项进行编译，需要根据 BootLoader 中的提示信息来决定。

main.c 文件是系统的 Boot 代码区，用户不能随意更改。用户程序代码在 User.c 文件中编写，也可以增加新的文件。**需要说明一点：使用该模板时，用户代码量不能超过 16kB。**

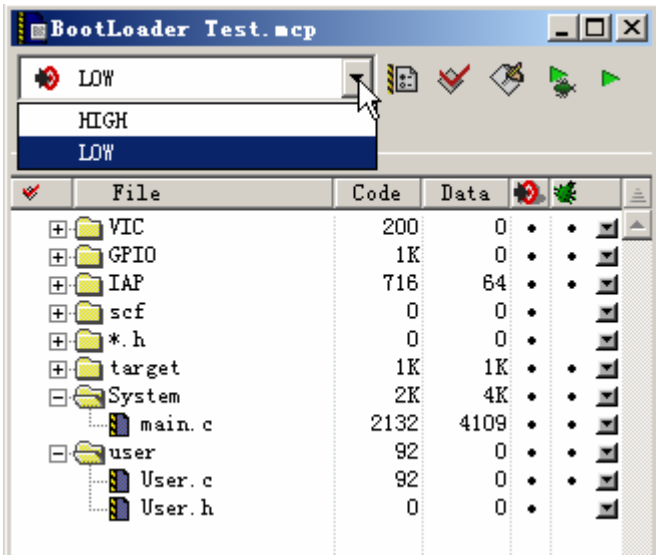


图 6.3 在线升级工程模板

## 6.3 升级方法

使用“在线升级模板”建立工程，编译以后的文件结构如图 6.4 所示。编译后，LOW 和 HIGH 选项都会生成两个二进制文件，SYSTEM 和 USER。

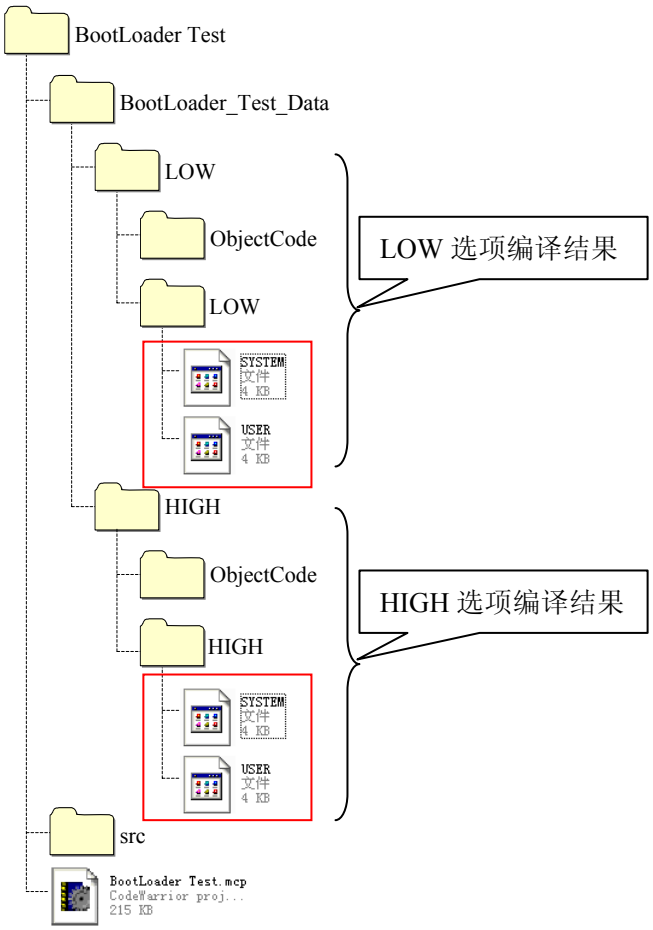

图 6.4 在线升级工程文件结构

- 如果当前程序运行在 LOW 区，那么只能升级 HIGH 区。系统复位后，将 P0.6 接地，使用串口终端软件将 HIGH 选项编译生成的 USER 文件发送给系统即可；
- 如果当前程序运行在 HIGH 区，那么只能升级 LOW 区。系统复位后，将 P0.6 接地，使用串口终端软件将 LOW 选项编译生成的 USER 文件发送给系统即可。

## 6.4 总结

本文介绍的“在线升级”解决方案并不是唯一的，本方案使用 UART0 进行升级，当然也可以使用其它的手段进行升级，而且片内 Flash 的分区方式也不是唯一的，因此本方案并不是唯一的解决方法。

在 ADS 编译器中，通过“分散加载文件”可以实现代码定位，有关“分散加载文件”和“ADS 编译器”方面的介绍请参考其它相关书籍，这里不再进行描述。

# 7. 参考资料


- NXP。LPC2364/64/68/78 User manual，2007。

# 8. 版权声明

广州致远电子有限公司随附提供的软件或文档资料旨在提供给您(本公司的客户)使用，仅限于且只能在本公司制造或销售的产品上使用。

该软件或文档资料为本公司和/或其供应商所有，并受适用的版权法保护。版权所有。如有违反，将面临相关适用法律的刑事制裁，并承担违背此许可的条款和条件的民事责任。

本公司保留在不通知读者的情况下，修改文档或软件相关内容的权利，对于使用中所出现的任何效果，本公司不承担任何责任。

该软件或文档资料“按现状”提供。不提供保证，无论是明示的、暗示的还是法定的保证。这些保证包括（但不限于）对出于某一特定目的应用此软件的适销性和适用性默示的保证。在任何情况下，公司不会对任何原因造成的特别的、偶然的或间接的损害负责。

公　　司：广州致远电子有限公司 嵌入式系统事业部
地　　址：广州市天河区车陂路黄洲工业区二栋四楼（研发部）
邮　　编：510660
网　　址：www.embedtools.com
销售电话：+86 (020) 2264-4249
技术支持：+86 (020) 2887-2347
传　　真：+86 (020) 3860-1859
E–mail：lpc2103@ zlgmcu.com（技术支持）