

帐号

用户名/Email

☒ 自动登录

找回密码

密码

登录

立即注册

首页

帖子大杂烩

精华帖

最新主题

资料下载

在线视频

原子哥平台

快捷导航

请输入搜索内容

搜索

热搜：正点原子 原子哥 精英STM32 战舰STM32 探索者STM32 阿波罗STM32 ALIENTEK

视频教程免费下载...

首页 > 单片机/嵌入式 > STM32-F0/F1/F2专区 > STM32串口IAP实验（战舰STM32开发板实验）

STM32开发板

Linux开发板

FPGA/ZYNQ开发板

扫码关注
“正点原子”
微信公众号

点击进入
“正点原子”
技术交流QQ群

发帖

返回列表

1

2

3

1 / 3 页

下一页

查看: 83530 | 回复: 117

Admin

230 主题

1952 帖子

10 精华

论坛元老

积分 4562

金钱 4562

注册时间 2010-12-14

在线时间 32 小时

发消息

STM32串口IAP实验（战舰STM32开发板实验）

发表于 2013-1-1 12:20:53 | 只看该作者 | 只看大图

楼主 电梯直达

"原子哥"在线教学平台正式上线啦，请下载手机APP"原子哥"，海量视频免费观看，包含Linux/STM32/FPGA/ZYNQ/FreeRTOS/UcosIII等

第四十八章 串口IAP实验

IAP，即在应用编程。很多单片机都支持这个功能，STM32也不例外。在之前的FLASH模拟EEPROM实验里面，我们学习了STM32的FLASH自编程，本章我们将结合FLASH自编程的知识，通过STM32的串口实现一个简单的IAP功能。ffice" />

48.1 IAP简介

IAP（In Application Programming）即在应用编程，IAP是用户自己的程序在运行过程中对用户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。通常实现IAP功能时，即用户程序运行中作自身的更新操作，需要在设计固件程序时编写两个项目代码，第一个项目程序不执行正常的功能操作，而只是通过某种通信方式(如USB、USART)接收程序或数据，执行对第二部分代码的更新；第二个项目代码才是真正的功能代码。这两部分项目代码都同时烧录在User Flash中，当芯片上电后，首先是第一个项目代码开始运行，它作如下操作：
1) 检查是否需要第二部分代码进行更新
2) 如果不需要更新则转到4)
3) 执行更新操作
4) 跳转到第二部分代码执行
第一部分代码必须通过其它手段，如JTAG或ISP烧入；第二部分代码可以使用第一部分代码IAP功能烧入，也可以和第一部分代码一起烧入，以后需要程序更新是再通过第一部分IAP代码更新。
我们将第一个项目代码称之为Bootloader程序，第二个项目代码称之为APP程序，他们存放在STM32 FLASH的不同地址范围，一般从最低地址区开始存放Bootloader，紧跟其后的就是APP程序（注意，如果FLASH容量足够，是可以设计很多APP程序的，本章我们只讨论一个APP程序的情况）。这样我们就是要实现2个程序：Bootloader和APP。
STM32的APP程序不仅可以放到FLASH里面运行，也可以放到SRAM里面运行，本章，我们将制作两个APP，一个用于FLASH运行，一个用于SRAM运行。
我们先来看看STM32正常的程序运行流程，如图48.1.1所示：

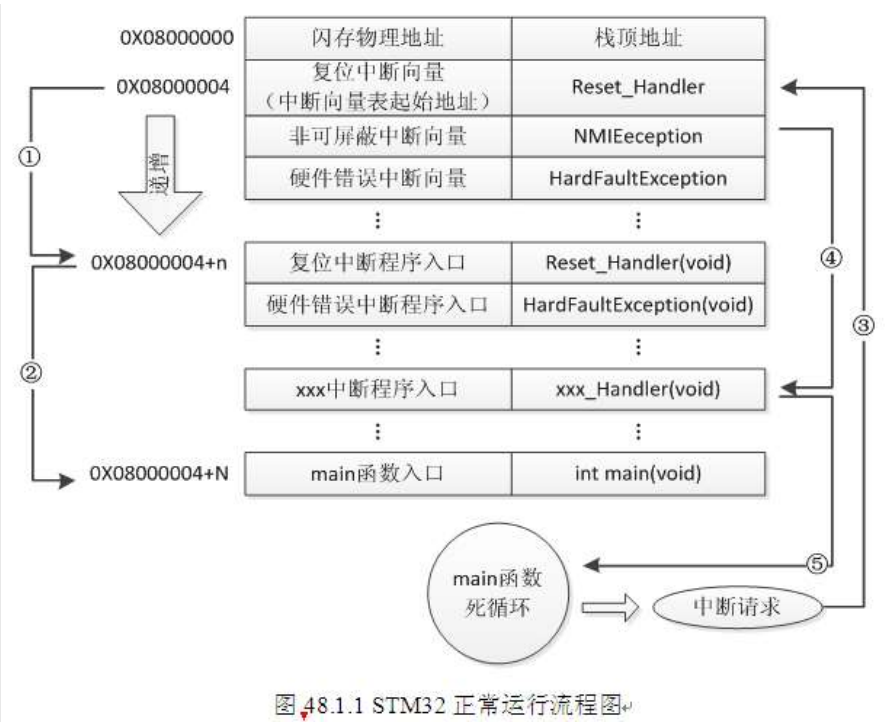


图 48.1.1 STM32 正常运行流程图

图48.1.1 STM32正常运行流程图

STM32的内部闪存（FLASH）地址起始于0x08000000，一般情况下，程序文件就从此地址开始写入。此外STM32是基于Cortex-M3内核的微控制器，其内部通过一张“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，STM32的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。

在图48.1.1中，STM32在复位后，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，会跳转到我们的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STM32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。

当加入IAP程序之后，程序运行流程如图48.1.2所示：

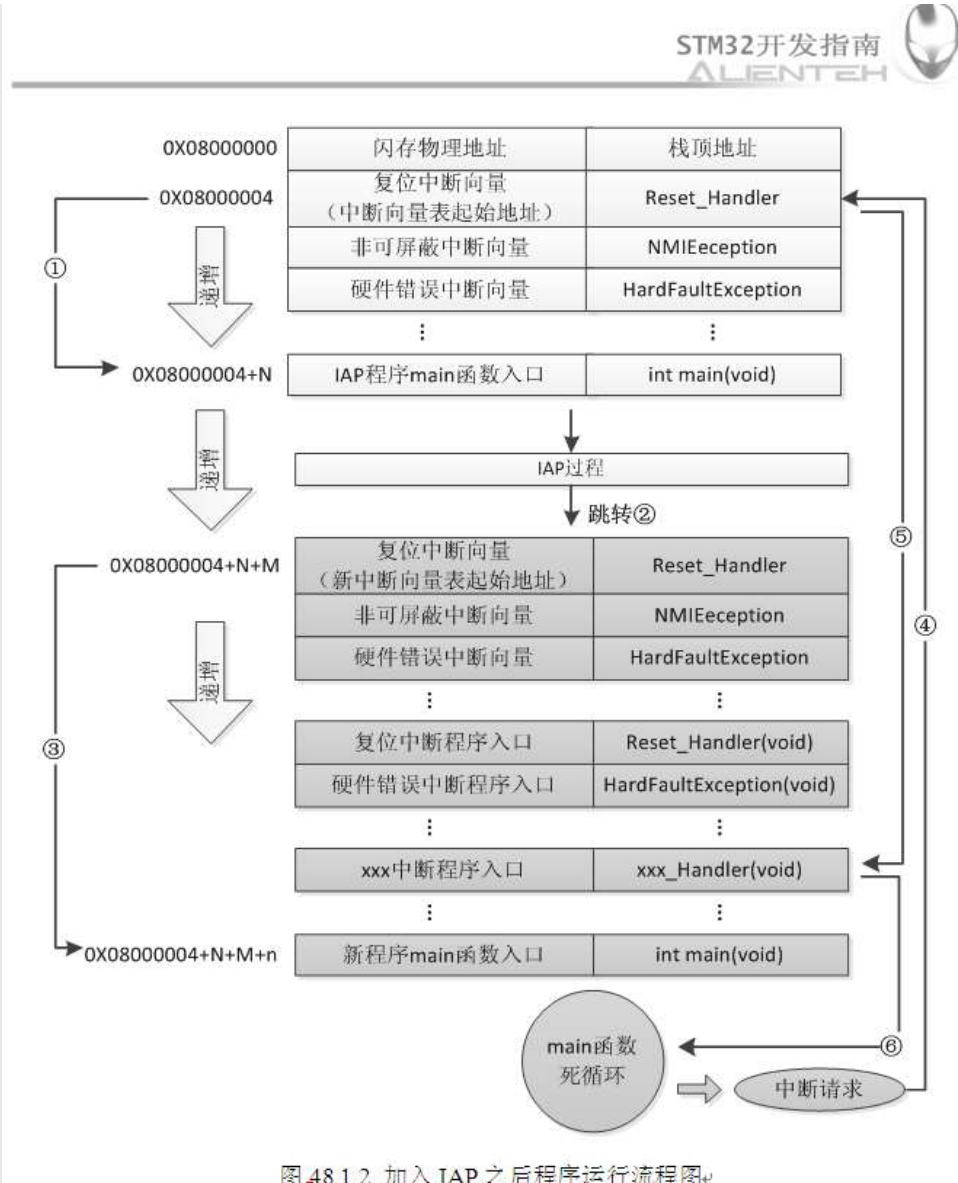


图 48.1.2 加入 IAP 之后程序运行流程图

在图48.1.2所示流程中，STM32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IAP的main函数，如图标号①所示，此部分同图48.1.1一样；在执行完IAP以后（即将新的APP代码写入STM32的FLASH，灰底部分。新程序的复位中断向量起始地址为0X08000004+N+M），跳转至新写入程序的复位向量表，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STM32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。

通过以上两个过程的分析，我们知道IAP程序必须满足两个要求：

- 1) 新程序必须在IAP程序之后的某个偏移量为x的地址开始；
- 2) 必须将新程序的中断向量表相应的移动，移动的偏移量为x；

本章，我们有2个APP程序，一个为FLASH的APP，程序在FLASH中运行，另外一个位SRAM的APP，程序运行在SRAM中，图48.1.2虽然是针对FLASH APP来说的，但是在

SRAM里面运行的过程和FLASH基本一致，只是需要设置向量表的地址为SRAM的地址。

1.APP程序起始地址设置方法

随便打开一个之前的实例工程，点击Options for Target→Target选项卡，如图48.1.3所示：

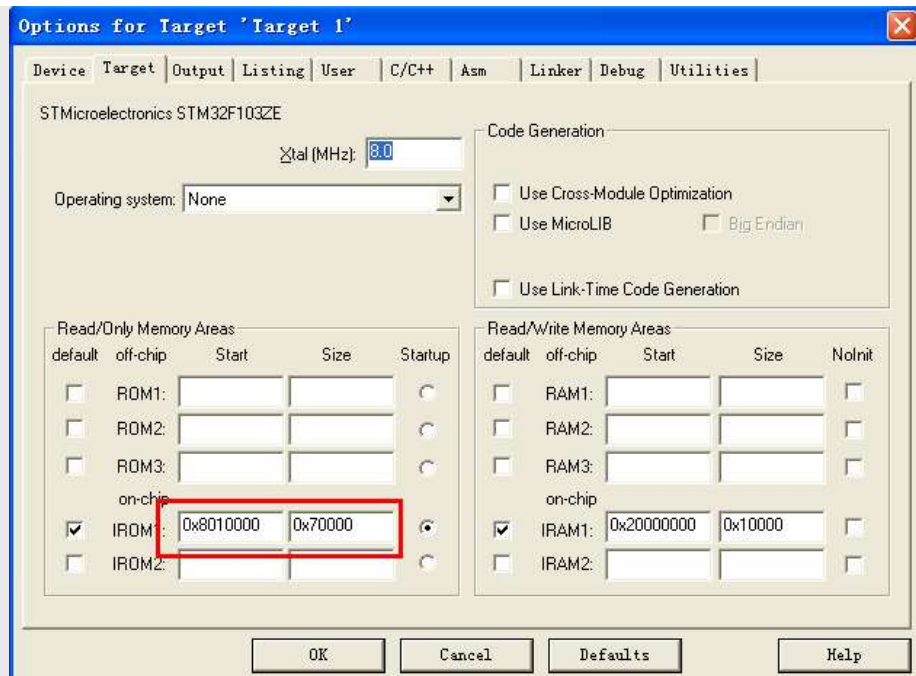


图48.1.3 FLASH APP Target选项卡设置

默认的条件下，图中IROM1的起始地址（Start）一般为0X08000000，大小（Size）为0X80000，即从0X08000000开始的512K空间为我们的程序存储（因为我们的STM32F103ZET6的FLASH大小是512K）。而图中，我们设置起始地址（Start）为0X08010000，即偏移量为0X10000（64K字节），因而，留给APP用的FLASH空间（Size）只有0X80000-0X10000=0X70000（448K字节）大小了。设置好Start和Size，就完成APP程序的起始地址设置。

这里的64K字节，需要大家根据Bootloader程序大小进行选择，比如我们本章的Bootloader程序为22K左右，理论上我们只需要确保APP起始地址在Bootloader之后，并且偏移量为0X200的倍数即可（相关知识，请参考：<http://www.openedv.com/posts/list/392.htm>）。这里我们选择64K（0X10000）字节，留了一些余量，方便Bootloader以后的升级修改。

这是针对FLASH APP的起始地址设置，如果是SRAM APP，那么起始地址设置如图48.1.4所示：

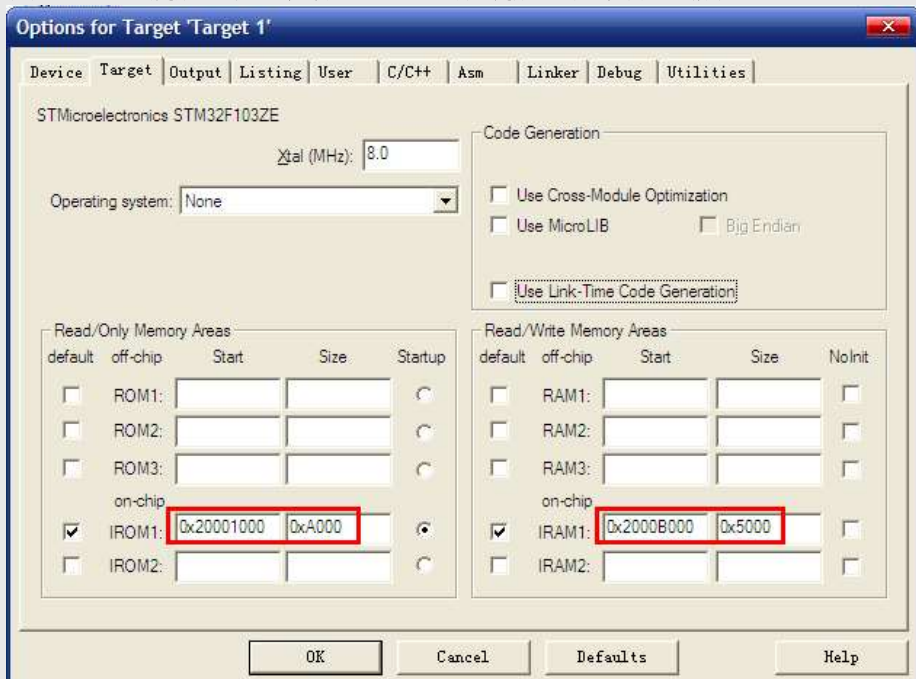


图48.1.4 SRAM APP Target选项卡设置

这里我们将IROM1的起始地址（Start）定义为：0X20001000，大小为0XA000（40K字节），即从地址0X20000000偏移0X1000开始，存放APP代码。因为整个STM32F103ZET6的SRAM大小为64K字节，所以IRAM1（SRAM）的起始地址变为0X2000B000（0x20001000+0xA000=0X2000B000），大小只有0X5000（20K字节）。这样，整个STM32F103ZET6的SRAM分配情况为：最开始的4K给Bootloader程序使用，随后的40K存放APP程序，最后20K，用作APP程序的内存。这个分配关系大家可以根据自己的实际情况修改，不一定和我们这里的设置一模一样，不过也需要注意，保证偏移量为0X200的倍数（我们这里为0X1000）。

2.中断向量的偏移量设置方法

之前我们讲解过，在系统启动的时候，会首先调用systemInit函数初始化时钟系统，同时systemInit还完成了中断向量的设置，我们可以打开systemInit函数，看看函数体的结尾处有这样几行代码：

```
#ifndef VECT_TAB_SRAM
SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET;
/* Vector Table Relocation in Internal SRAM. */
```

```
#else
```

```
SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET;
```

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

```
#endif
```

从代码可以理解，VTOR寄存器存放的是中断向量的起始地址。默认的情况VECT_TAB_SRAM是没有定义，所以执行SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET;

对于FLASH APP，我们设置为FLASH_BASE+偏移量0x10000，所以我们可以FLASH APP的main函数最开头处添加如下代码实现中断向量的起始地址的重设：

```
SCB->VTOR = FLASH_BASE | 0x10000;
```

以上是FLASH APP的情况，当使用SRAM APP的时候，我们设置起始地址为：SRAM_base+0x1000，同样的方法，我们在SRAM APP的main函数最开始处，添加下面代码：

```
SCB->VTOR = SRAM_BASE | 0x1000;
```

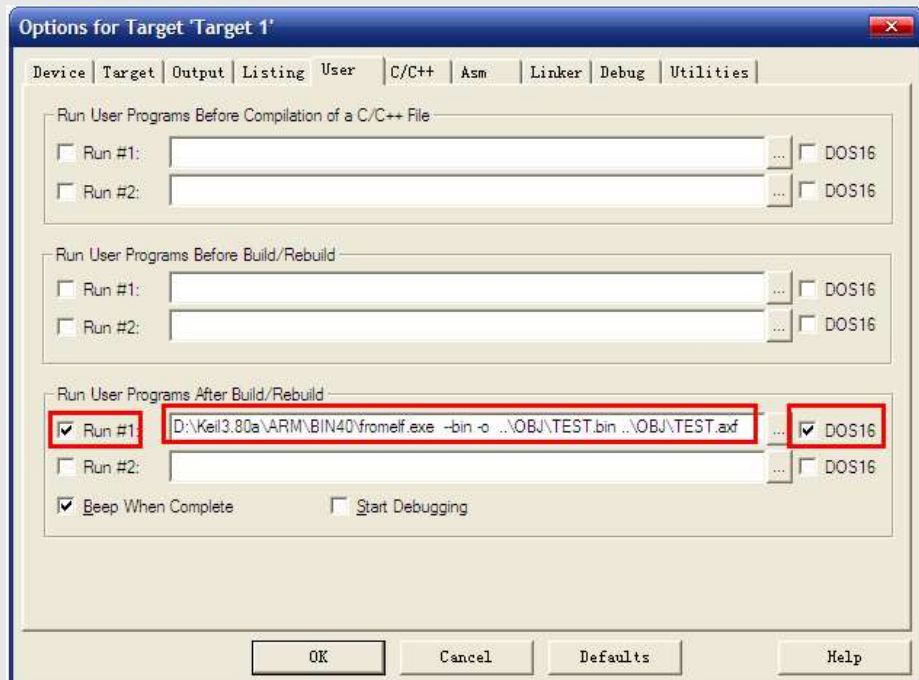
这样，我们就完成了中断向量偏移量的设置。

通过以上两个步骤的设置，我们就可以生成APP程序了，只要APP程序的FLASH和SRAM大小不超过我们的设置即可。不过MDK默认生成的文件是.hex文件，并不方便我们用作IAP更新，我们希望生成的文件是.bin文件，这样可以方便进行IAP升级（至于为什么，请大家自行百度HEX和BIN文件的区别！）。这里我们通过MDK自带的格式转换

工具fromelf.exe，来实现.axf文件到.bin文件的转换。该工具在MDK的安装目录\ARM\BIN40文件夹里面。

fromelf.exe转换工具的语法格式为：fromelf [options] input_file。其中options有很多选项可以设置，详细使用请参考光盘《mdk如何生成bin文件.pdf》。

本章，我们通过在MDK点击Options for Target→User选项卡，在Run User Programs After Build/Rebuild 栏，勾选Run#1和DOS16，并写入：D:\Keil3.80a\ARM\BIN40\fromelf.exe --bin -o ..\OBJ\TEST.bin ..\OBJ\TEST.axf，如图48.1.6所示：



通过这一步设置，我们就可以在MDK编译成功之后，调用fromelf.exe（注意，我的MDK是安装在D:\Keil3.80A文件夹下，如果你是安装在其他目录，请根据你

自己的目录修改fromelf.exe的路径），根据当前工程的TEST.axf（如果是其他名字，请记住修改，这个文件存放在OBJ目录下，格式为xxx.axf），生成一个

TEST.bin的文件。并存放在axf文件相同的目录下，即工程的OBJ文件夹里面。在得到.bin文件之后，我们只需要将这个bin文件传送给单片机，即可执行IAP升级。

最后再来APP程序的生成步骤：

1) 设置APP程序的起始地址和存储空间大小

对于在FLASH里面运行的APP程序，我们可以按照图48.1.3的设置。对于SRAM里面运行的APP程序，我们可以参考图48.1.4的设置。

2) 设置中断向量偏移量

这一步按照上面讲解，重新设置SCB->VTOR的值即可。

3) 设置编译后运行fromelf.exe，生成.bin文件。

通过在User选项卡，设置编译后调用fromelf.exe，根据.axf文件生成.bin文件，用于IAP更新。

以上3个步骤，我们就可以得到一个.bin的APP程序，通过Bootlader程序即可实现更新。

大家可以打开我们光盘的两个APP工程，熟悉这些设置。

48.2 硬件设计

本章实验（Bootloader部分）功能简介：开机的时候先显示提示信息，然后等待串口输入接收APP程序（无校验，一次性接收），在串口接收到APP程序

之后，即可执行IAP。如果是SRAM APP，通过按下KEY0即可执行这个收到的SRAM APP程序。如果是FLASH APP，则需要先按下WK_UP按键，将串口接

收到的APP程序存放到STM32的FLASH，之后再按KEY2既可以执行这个FLASH APP程序。通过KEY1按键，可以手动清除串口接收到的APP程序。DS0用于指示程序运行状态。

本实验用到的资源如下：

- 1) 指示灯DS0
- 2) 四个按键（KEY0/KEY1/KEY2/WK_UP）

3) 串口

4) TFTLCD模块

这些用到的硬件，我们在之前都已经介绍过，这里就不再介绍了。

48.3 软件设计

本章，我们总共需要3个程序：1，Bootloader；2，FLASH APP；3）SRAM APP；其中，我们选择之前做过的RTC实验（在第二十章介绍）来做为FLASH APP程序

（起始地址为0X08010000），选择触摸屏实验（在第三十一章介绍）来做SRAM APP程序（起始地址为0X20001000）。Bootloader则是通过TFTLCD显示实验（在第

十八章介绍）修改得来。本章，关于SRAM APP和FLASH APP的生成比较简单，我们就不细说，请大家结合光盘源码，以及48.1节的介绍，自行理解。本章软件设计仅针对Bootloader程序。

打开本实验工程，可以看到我们增加了IAP组，在组下面添加了iap.c文件以及其头文件isp.h。打开iap.c，代码如下：

```
#include "sys.h"
#include "delay.h"
#include "usart.h"
#include "stmflash.h"
#include "iap.h"
iapfun jump2app;
u16 iapbuf[1024];
//appaddr:应用程序的起始地址
//appbuf:应用程序CODE.
//appsize:应用程序大小(字节).
void iap_write_appbin(u32 appxaddr,u8 *appbuf,u32 appsize)
{
    u16 t;
    u16 i=0;
    u16 temp;
    u32 fwaddr=appxaddr;//当前写入的地址
    u8 *dfu=appbuf;
    for(t=0;t<appsize;t+=2)
    {
        temp=(u16)dfu[1]<<8;
        temp+=(u16)dfu[0];
        dfu+=2;//偏移2个字节
        iapbuf[i++]=temp;
        if(i==1024)
        {
            i=0;
            STMFLASH_Write(fwaddr,iapbuf,1024);
            fwaddr+=2048;//偏移2048 16=2*8.所以要乘以2.
        }
        if(i)STMFLASH_Write(fwaddr,iapbuf,i);//将最后的一些内容字节写进去.
    }
    //跳转到应用程序段
    //appaddr:用户代码起始地址.
    void iap_load_app(u32 appxaddr)
    {
        if(((*(vu32*)appxaddr)&0x2FFE0000)==0x20000000) //检查栈顶地址是否合法.
        {
            jump2app=(iapfun)*(vu32*)(appxaddr+4);
            //用户代码区第二个字为程序开始地址(复位地址)
            MSR_MSP(*(vu32*)appxaddr);
            //初始化APP堆栈指针(用户代码区的第一个字用于存放栈顶地址)
            jump2app(); //跳转到APP.
        }
    }
```

该文件总共只有2个函数，其中，iap_write_appbin函数用于将存放在串口接收buf里面的APP程序写入到FLASH。iap_load_app函数，则用于跳转到APP程序运行，其参数appxaddr为APP程序的起始地址，程序先判断栈顶地址是否合法，在得到合法的栈顶地址后，通过MSR_MSP函数（该函数在sys.c文件）设置栈顶地址，最后通过一个虚拟的函数（jump2app）跳转到APP程序执行代码，实现IAP→APP的跳转。

打开iap.h代码如下：

```
#ifndef __IAP_H__
#define __IAP_H__
#include "sys.h"
typedef void (*iapfun)(void); //定义一个函数类型的参数.
#define FLASH_APP1_ADDR 0x08001000
//第一个应用程序起始地址(存放在FLASH)
//保留0X08000000~0X0800FFFF的空间为Bootloader使用
void iap_load_app(u32 appxaddr); //跳转到APP程序执行
void iap_write_appbin(u32 appxaddr,u8 *appbuf,u32 applen); //在指定地址开始,写入bin
#endif
```

这部分代码比较简单，。本章，我们是通过串口接收APP程序的，我们将usart.c和usart.h做了稍微修改，在usart.h中，我们定义USART_REC_LEN为55K字节，

也就是串口最大一次可以接收55K字节的数据，这也是本Bootloader程序所能接收的最大APP程序大小。然后新增一个USART_RX_CNT的变量，用于记录接收到

的文件大小，而USART_RX_STA不再使用。打开usart.c，可以看到我们修改USART1_IRQHandler部分代码如下：

```
//串口1中断服务程序
//注意,读取USARTx->SR能避免莫名其妙的错误
u8 USART_RX_BUF[USART_REC_LEN] __attribute__((at(0X20001000)));
//接收缓冲,最大USART_REC_LEN个字节,起始地址为0X20001000.
//接收状态
//bit15, 接收完成标志
//bit14, 接收到0x0d
//bit13~0, 接收到的有效字节数目
u16 USART_RX_STA=0; //接收状态标记
u16 USART_RX_CNT=0; //接收的字节数
void USART1_IRQHandler(void)
{
    u8 res;
#ifdef OS_CRITICAL_METHOD
//如果OS_CRITICAL_METHOD定义了,说明使用ucosII了.
    OSIntEnter();
#endif
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)//接收到数据
    {
        res=USART_ReceiveData(USART1);
        if(USART_RX_CNT<USART_REC_LEN)
        {
            USART_RX_BUF[USART_RX_CNT]=res;
            USART_RX_CNT++;
        }
    }
#ifdef OS_CRITICAL_METHOD
//如果OS_CRITICAL_METHOD定义了,说明使用ucosII了.
    OSIntExit();
#endif
}
```

这里，我们指定USART_RX_BUF的地址是从0X20001000开始，该地址也就是SRAM APP程序的起始地址！然后在USART1_IRQHandler函数里面，将串口发

送过来的数据，全部接收到USART_RX_BUF，并通过USART_RX_CNT计数。代码比较简单，我们就不多说了。

最后我们看看main函数如下：

```
int main(void)
{
    u8 t,key;
    u16 oldcount=0; //老的串口接收数据值
    u16 applenth=0; //接收到的app代码长度
    u8 clearflag=0;
    uart_init(256000); //串口初始化为256000
    delay_init(); //延时初始化
    LCD_Init(); //液晶初始化
    LED_Init(); //初始化与LED连接的硬件接口
    KEY_Init(); //按键初始化
    POINT_COLOR=RED;//设置字体为红色
    LCD_ShowString(60,50,200,16,16,"Warship STM32");
    LCD_ShowString(60,70,200,16,16,"IAP TEST");
    LCD_ShowString(60,90,200,16,16,"ATOM@ALIENTEK");
    LCD_ShowString(60,110,200,16,16,"2012/9/24");
    LCD_ShowString(60,130,200,16,16,"WK_UP:Copy APP2FLASH");
    LCD_ShowString(60,150,200,16,16,"KEY1:Erase SRAM APP");
    LCD_ShowString(60,170,200,16,16,"KEY0:Run SRAM APP");
    LCD_ShowString(60,190,200,16,16,"KEY2:Run FLASH APP");
    POINT_COLOR=BLUE;
    //显示提示信息
    POINT_COLOR=BLUE;//设置字体为蓝色
    while(1)
    {
        if(USART_RX_CNT)
        {
            if(oldcount==USART_RX_CNT)
                //新周期内,没有收到任何数据,认为本次数据接收完成.
            {
                applenth=USART_RX_CNT;
                oldcount=0;
                USART_RX_CNT=0;
                printf("用户程序接收完成!\r\n");
                printf("代码长度:%dBytes\r\n",applenth);
            }else oldcount=USART_RX_CNT;
        }
        t++; delay_ms(10);
    }
```

```
if(t==30)
{
    LED0=!LED0; t=0;
    if(clearflag)
    {
        clearflag--;
        if(clearflag==0)LCD_Fill(60,210,240,210+16,WHITE);//清除显示
    }
}
key=KEY_Scan(0);
if(key==KEY_UP)
{
    if(applenth)
    {
        printf("开始更新固件...\r\n");
        LCD_ShowString(60,210,200,16,16,"Copying APP2FLASH...");
        if(((vu32*)(0X20001000+4))&0xFF000000)==0x08000000)
            //判断是否为0X08XXXXXX.
        {
            iap_write_appbin(FLASH_APP1_ADDR,USART_RX_BUF,
                applenth); //更新FLASH代码
            LCD_ShowString(60,210,200,16,16,"Copy APP Succeeded!!");
            printf("固件更新完成!\r\n");
        }else
        {
            LCD_ShowString(60,210,200,16,16,"Illegal FLASH APP! ");
            printf("非FLASH应用程序!\r\n");
        }
    }else
    {
        printf("没有可以更新的固件!\r\n");
        LCD_ShowString(60,210,200,16,16,"No APP!");
    }
    clearflag=7;//标志更新了显示,并且设置7*300ms后清除显示
}
if(key==KEY_DOWN)
{
    if(applenth)
    {
        printf("固件清除完成!\r\n");
        LCD_ShowString(60,210,200,16,16,"APP Erase Succeeded!");
        applenth=0;
    }else
    {
        printf("没有可以清除的固件!\r\n");
        LCD_ShowString(60,210,200,16,16,"No APP!");
    }
    clearflag=7;//标志更新了显示,并且设置7*300ms后清除显示
}
if(key==KEY_LEFT)
{
    printf("开始执行FLASH用户代码!\r\n");
    if(((vu32*)(FLASH_APP1_ADDR+4))&0xFF000000)==0x08000000)
        //判断是否为0X08XXXXXX.
    {
        iap_load_app(FLASH_APP1_ADDR);//执行FLASH APP代码
    }else
    {
        printf("非FLASH应用程序,无法执行!\r\n");
        LCD_ShowString(60,210,200,16,16,"Illegal FLASH APP!");
    }
    clearflag=7;//标志更新了显示,并且设置7*300ms后清除显示
}
if(key==KEY_RIGHT)
{
    printf("开始执行SRAM用户代码!\r\n");
    if(((vu32*)(0X20001000+4))&0xFF000000)==0x20000000)
        //判断是否为0X20XXXXXX.
    {
        iap_load_app(0X20001000);//SRAM地址
    }else
    {
        printf("非SRAM应用程序,无法执行!\r\n");
        LCD_ShowString(60,210,200,16,16,"Illegal SRAM APP!");
    }
    clearflag=7;//标志更新了显示,并且设置7*300ms后清除显示
}
```

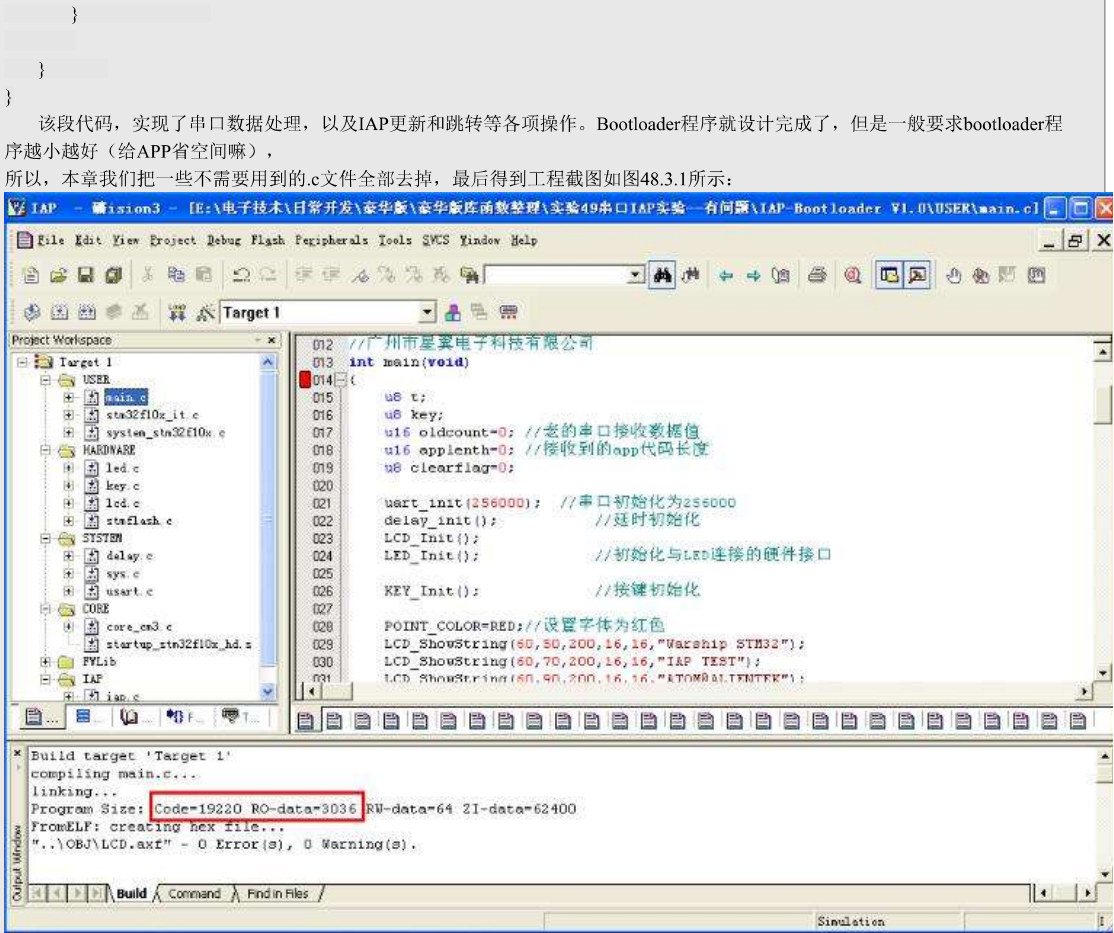



图48.3.1 Bootloader 工程截图

从上图可以看出，虽然去掉了一些不用的.c文件，但是Bootloader大小还是有22K左右，比较大，主要原因是液晶驱动和printf占用了比较大的flash，如果大家想进一步删减，可以去掉LCD显示和printf等，不过我们本章为了演示效果，所以保留了这些代码。

至此，本实验的软件设计部分结束。

FLASH APP和SRAM APP两部分代码，我们在实验目录下提供了两个实验供大家参考，不过要提醒大家，根据我们的设置，FLASH APP的起始地址必须是0X08010000，而SRAM APP的起始地址必须是0X20001000。

48.4 下载验证

在代码编译成功之后，我们下载代码到ALIENTEK战舰STM32开发板上，得到，如图48.4.1所示：



图48.4.1 IAP程序界面

此时，我们可以通过串口，发送FLASH APP或者SRAM APP到战舰STM32开发板，如图48.4.2所示：



图48.4.2 串口发送APP程序界面

先用串口调试助手的打开文件按钮（如图标号1所示），找到APP程序生成的.bin文件，然后设置波特率为256000（为了提高速度，Bootloader程序将波特率被设置为256000了），最后点击发送文件（图中标号3所示），将.bin文件发送给战舰STM32开发板。

在收到APP程序之后，我们就可以通过KEY0/KEY2运行这个APP程序了（如果是FLASH APP，则先需要通过WK_UP将其存入对应FLASH区域）。

 [实验48 串口IAP实验-战舰STM32开发板.zip](#)
1.75 MB, 下载次数: 19338

★ 收藏 40


📄 淘帖

👍 支持 4

👎 反对

SIGNATURE -----

我是开源电子网?网站管理员，对网站有任何问题，请与我联系! QQ:389063473 Email:389063473@qq.com

 **原子哥在线教学平台上线啦！** [点击进入](#)

yuanzige.com 一千讲视频免费学习，数十位名师与您相约



🗨️ 回复

👤 举报

woaimeishu



1 主题

5 帖子


0 精华

新手上路

积分 39
金钱 39

注册时间 2015-3-17
在线时间 3 小时

📧 发消息

 发表于 2015-3-17 13:43:45 | 只看该作者

推荐

原子哥，这个程序也可以在stm32 mini板上运行吧，我将这个项目程序移植到mini stm32中时出现些问题。（也改了些配置LED，key,lcd库都是用的mini板中的库），烧录到mini板上后程序启动不起来，在线调试了，发现进入这个循环中，出不来了


```
void HardFault_Handler(void)
{
    /* Go to infinite loop when Hard Fault exception occurs */
    while (1)
    {
    }
}
```

请问这是什么情况，怎么解决呢

🗨️ 回复 👍 支持 0 👎 反对 1

👤 举报

正点原子

 发表于 2013-3-15 11:21:36 | 只看该作者

推荐