N32G45X_FR_WB系列芯片IAP升级应用笔记_V1.0

简介

本文档主要介绍 N32G45X_FR_WB 系列芯片（以下简称 N32G45X）的 IAP 升级应用例程、实际应用开发时容易遇到的问题和解决办法。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

目录

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

1 IAP 简介

IAP是（In Application Programming）即在应用编程英文缩写，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。通常实现 IAP 功能时，用户程序运行中进行对自身的更新操作，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，只是通过某种通信方式(如 USB、UART)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 的不同区域中，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. 检查是否需要对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，以后需要程序更新时再通过第一部分 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，它们一般存放在 N32G45X 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。**新 APP 程序除了可以存放到 Flash 之外，还可以存放到 Sram 里面执行**，后续章节会分别举例说明。所以根据以上描述，我们要实现 2 个程序：Bootloader 和 APP。N32G45X 正常的程序运行流程如图 1.0 所示。

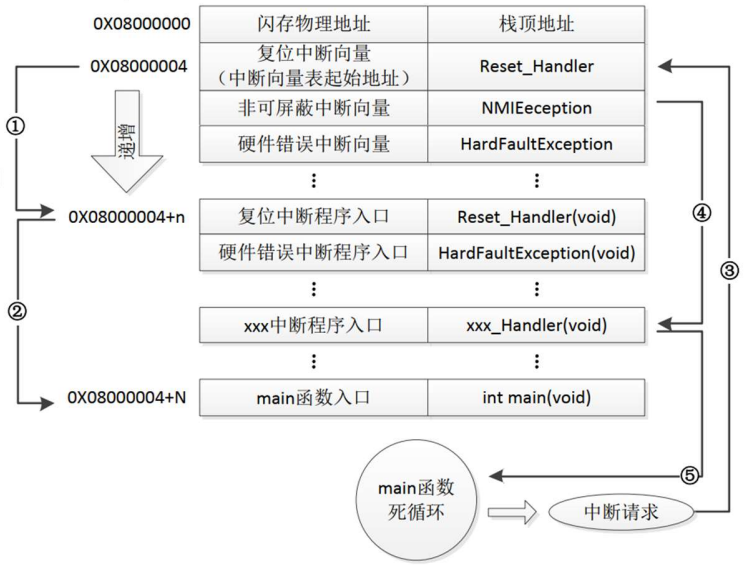

图 1.0

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

与其他友商的芯片不同，N32G45X 在上电后会先执行内置的 Bootloader（独立于 Flash 之外），在进行完一系列的配置和安全性检测后，才会执行 Flash 的程序。如上图所示，N32G45X 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 N32G45X 是基于 Cortex-M4F 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，N32G45X 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。

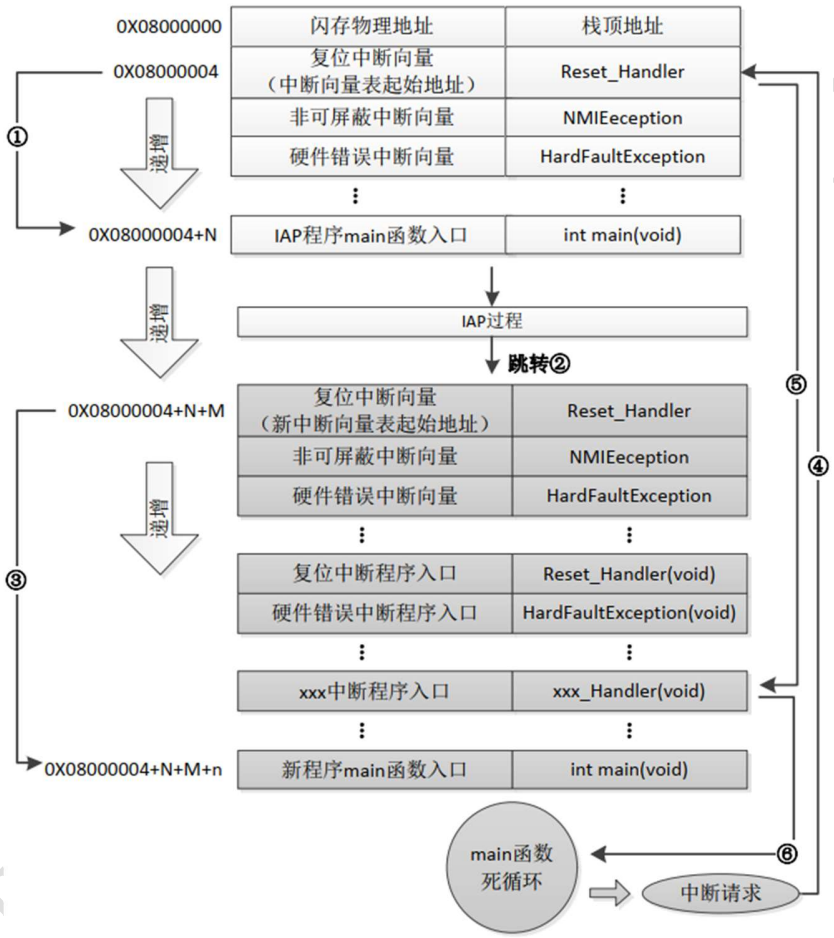

图 1.1

如图 1.1 所示，芯片在上电执行完内置的 Bootloader(独立于 Flash)之后，会从 Flash 的 0x08000004 地址提取复位中断向量的地址，并跳转到中断复位函数，执行完之后会跳转到 IAP 的 mian 函数开始执行，在 main 函数循环等待升级的过程中，用户可以通过 USB 或者 UART 等方式，发送更新文件进行 APP 程序升级。用户在升级的过程中可以边接收边更新，也接收完成整包 APP 程序后再进行更新。由于 Bootloader 预留的 Flash 和 Sram 都比较小，所以本应用笔记的例程将会以分包发送、边接收边更新的方式进行 APP 升级。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

在更新完 APP 程序之后，程序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 APP 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 N32G45X 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。新程序的复位中断向量起始地址为 0X08000004+N+M，M 为新程序的跳转偏移量，后续章节会介绍如何在工程上设置偏移量。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

2 IAP 软件实现流程

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；

2.1 配置 APP 程序起始地址

2.1.1 SRAM_APP 起始地址设置

main.c | bsp_usart.c | IAP.c | iap.h

```
#ifndef __IAP_H__
#define __IAP_H__

#include "n32g4fr.h"

#define Sram_buf_len        1024*32                         // 最大接收32K,长度为0x8000
#define Sram_buf_addr       (0x20000000 + Sram_buf_len)     // app接收缓存起始地址


typedef  void (*iapfun)(void);                              // 定义一个函数类型的参数
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
```

图 2.0

如图 2.0 所示，在 Bootloader 的工程中，定义了一个存放 APP 程序的数组 Sram_buf，其起始地址是 0x20008000，长度为 1024*32 共 32K。

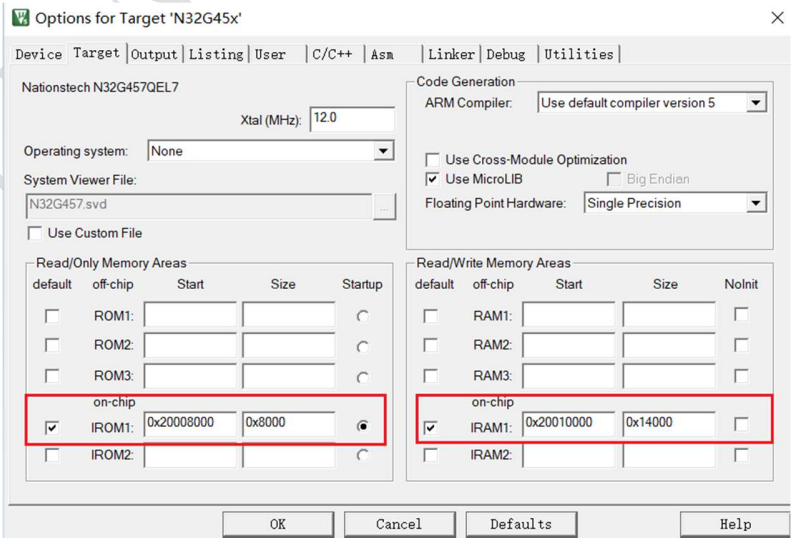

图 2.1

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

由于 N32G45X 的 SRAM 起始地址为 0x20000000，截止地址 0x20024000，整个 SRAM 的大小为 144K。所以我们在 SRAM_APP 工程中，按照图 2.1 设置偏移量：点击“魔法棒”，选择“Target”，在 IROM1 一栏的“Start”填入 0x20008000，Size 填入 0x8000；在 IRAM1 一栏的 Start 填入 0x20010000，Size 填入 0x14000。所以整个 SRAM 的资源排布情况如下：**前 32K 分给了 Bootloader 使用，紧接着的 32K 用来存储 APP 程序，后面的 80K 分配给 APP 程序调用。**

2.1.2 FLASH_APP 起始地址设置

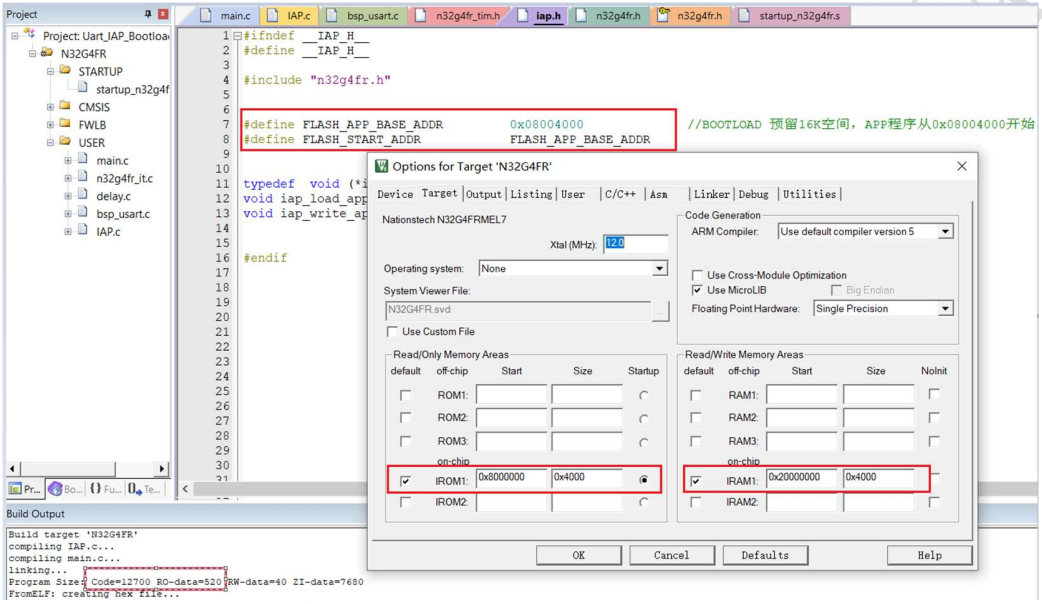

图 2.2

如图 2.1 所示，在 Flash_App 对应的 **Bootloader 工程**中，由于代码量较少大约 13K，所以我们为此预留了 16K 的 Flash 和 16K 的 Sram，并设置 Flash 跳转的地址 0x0800400。点击“魔法棒”，选择“Target”, 在 IROM1 一栏的“Start”填入 0x08000000，Size 填入 0x4000；在 IRAM1 一栏的 Start 填入 0x20000000，Size 填入 0x4000。

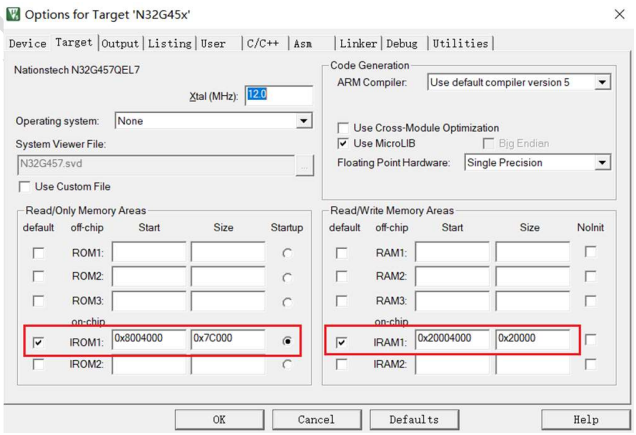

图 2.3

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

N32G45X 的 Flash 最大 512K，从地址 0x08000000 到 0x08080000 截止，例程是把前面 16K 的 Flash 作为 Bootloader 使用，后面的 496K 留给 APP 使用。如图 2.2 所示，在 **Flash_App 工程**中，点击“魔法棒”，选择“Target”，在 IROM1 一栏的“Start”填入 0x08004000，Size 填入 0x7C000；在 IRAM1 一栏的 Start 填入 0x20004000，Size 填入 0x20000。

2.2 中断向量表的偏移量设置方法

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 还完成了中断向量表的设置，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：

```
#ifdef 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表的起始地址。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 0x4000,所以我们可以在 FLASH APP 的 main 函数最开头处添加如下代码实现中断向量表的起始地址的重设：

```
SCB->VTOR = FLASH_BASE | 0x4000;
```

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+0x8000,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 | 0x8000;
```

这样，我们就完成了中断向量表偏移量的设置

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

2.3 在 APP 工程中，生成 bin 文件

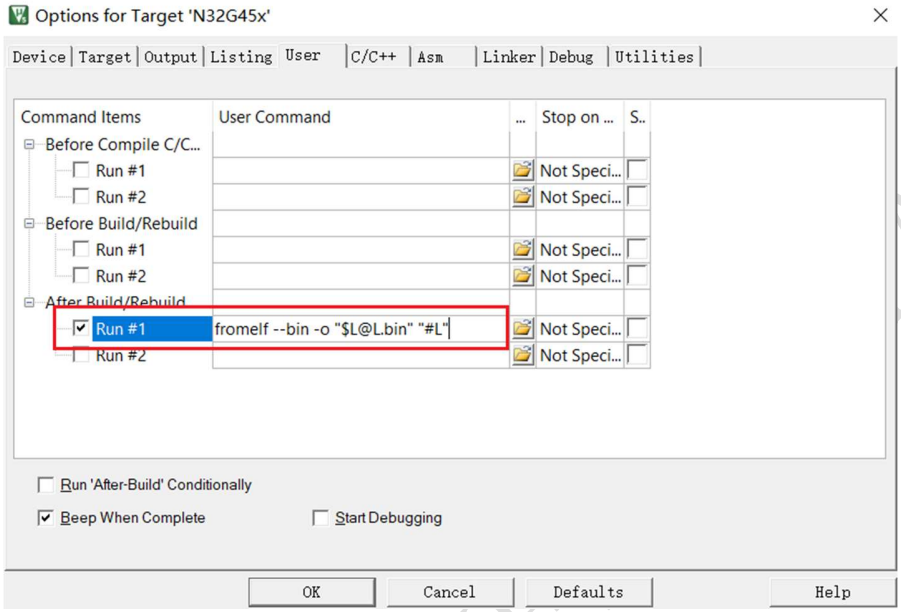

图 2.4

在 Sram_App 和 Flash_App 工程中，点击“魔法棒”，选择“USER”，在 “After Build/Rebuid”下，把“RUN #1”左边的框打勾，并在右边栏填入“**fromelf --bin -o "$L@L.bin" "#L"**”，点击 OK 后重新编译，即可生成 BIN 文件，保存在\MDK-ARM\Objects 目录下。

2.4 软件实现流程

Bootloader 的软件流程主要有三个步骤：

1) 上电初始化串口，判断等待接收 App 的 bin 文件;
2) 分包接收 bin 文件，把内容转存到明确指定地址的 Sram_buf，或者写入到指定的 Flash 地址；
3) 接收 bin 文件完成，程序跳转；

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

2.4.1 Sram_App 的 Bootloader 程序流程

打开 Uart_IAP_Bootloader 工程，我们可以看到，程序主要在 main.c、IAP.c、bsp_usart.c 三个文件里面。下面将会详细介绍三个步骤的代码。

```
int main(void)
{
    tim3_init(99, 71);                                  //72MH/(71+1)=1M Hz;1M Hz/(99+1)=100us
    USART_Config();
    printf("NZ3601_init success! \r\n");
    while(1)
    {
        while(receive_app_done == 0)                    //无APP程序，等待接收更新        1.等待接收bin文件
        {
            if(f_final_frame == 1)
            {
                receive_app_done = 1;                   //接收完BIN升级文件
                m_delay_ms(500);
                break;
            }
        }
        if(receive_app_done)                            //已有更新APP
        {
            receive_app_done = 0;
            TIM_Enable(TIM3, DISABLE);                  //关闭定时器中断
            //
            printf("APP address: %x\r\n",(Sram_buf_addr));          3.接收完成后，程序跳转
            printf("开始执行SRAM用户代码!!\r\n");
            iap_load_app(Sram_buf_addr);                //跳转到APP起始地址，期间不能被其他中断打断，否则会跳转失败
        }
    }
}
```

图 2.5

```
void USART1_IRQHandler(void)
{
    uint8_t i = 0;
    uint8_t buf_temp[256] = {0};
    uint8_t sum_check = 0;
    //
    if(USART_GetFlagStatus(DEBUG_USARTx, USART_INT_RXDNE) != RESET)
    {
        USART_ClrIntPendingBit(DEBUG_USARTx, USART_INT_RXDNE);
        if(receive_cnt <= 134)
        {
            RX_buf[receive_cnt++] = USART_ReceiveData(DEBUG_USARTx);
            current_pack_length = RX_buf[3]+5;                          //计算当前pack的数据长度
            if((RX_buf[0] == 0x01)&&(RX_buf[1] == 0x01)&& (receive_cnt== current_pack_length))  //帧头 固定为0x01 ,0x01。
            {                                                           //pack长度固定为 uart_rx_buf[3] + 5 个字节，
                receive_cnt = 0;                                        //长度最大为128+5个字节
                f_receive_frame = 1;
                memcpy(buf_temp,RX_buf,256);
                for(i = 0; i < current_pack_length -1 ; i++)
                {
                    sum_check = sum_check + buf_temp[i];    分包接收bin文件的传输协议和校验     //计算SUM 校验
                }
                sum_check = ~sum_check + 1;
                if(sum_check == buf_temp[current_pack_length-1])        //比较SUM，如果不同，则丢弃当前包，等待上位机重发
                {
                    send_ack();                                         //应答上位机
                    memcpy(&Sram_buf[rx_nunber*128],&RX_buf[4],current_pack_length-5);  //数据转存到Sram_buf
                    rx_nunber ++;
                    if((current_pack_length==5)&&(RX_buf[3]==0))        //发完bin内容的最后一包后，上位机会发5字节的帧尾
                    {
                        rx_nunber = 0;                  3. 接收并缓存BIN数据
                        f_final_frame = 1;                              //接收完最后一帧数据
                    }
                    current_pack_length = 0;
                }
                memset(RX_buf,0x00,sizeof(RX_buf));
            }
        }
    }
}
```

图 2.6

如图 2.5、图 2.6 所示，在 main 函数中，初始化完之后，有两个 while(1)循环，分别是等待接收和跳转到 Sram 执行程序；而接收 BIN 升级文件，则是在串口中断 USART1_IRQHandler(void)函数里面进行。为了尽量少占用 Bootloader 的资源、兼容接收大体积的 BIN 文件并且保证 BIN 文件的完整性，我们采用了把 BIN 拆分成小包的方式

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

进行发送，每次发送 128 个字节，所以我们在接收完一包数据后，会根据传输协议进行校验，如果校验不过则丢弃当前数据包，等待上位机重新发送当前数据包。传输协议在后续章节会详细说明。

```
__asm void MSR_MSP(u32 addr)
{
    MSR MSP, r0             //set Main Stack value
    BX r14
}
/**=================================================================
        APP 跳转
        appxaddr:用户代码起始地址.
=================================================================*/
void iap_load_app(u32 appxaddr)
{
    if(((*(vu32*)appxaddr)&0x0FFFFFFF) < 1024*144)       // 检查栈顶地址是否合法.
    {
        jump2app = (iapfun)*(vu32*)(appxaddr+4);
        MSR_MSP(*(vu32*)appxaddr);                       // 初始化堆栈指针
        jump2app();                    设置偏移量并跳转   // 跳转到APP.
    }
}
/**=================================================================
```

图 2.7

如图 2.7 所示，在完整接收到 BIN 文件后，我们会进行跳转，在图 2.5 里面调用 iap_load_app(Sram_buf_addr); Sram_buf_addr 就是我们在图 2.0 所设置的 Sram_buf 的起始地址 0x20008000。

2.4.2 Flash_App 的 Bootloader 程序流程

```
int main(void)
{
    tim3_init(99, 71);                              //72MH/(71+1)=1M Hz;1M Hz/(99+1)=100us
    USART_Config();
    printf("NZ3601_init success! \r\n");
    while(1)
    {
        if(FLASH_ReadWord(app_update_flag_addr) == 0x12345678)      //上电检测是否需要直接跳转
        {
            receive_app_done = 1;      1.判断是否需要直接跳转
        }
        while(receive_app_done == 0)                //无APP程序，等待接收更新
        {
            if(f_IAP_flashing == 1)
            {
                TIM_Enable(TIM3, DISABLE);
                USART_Enable(DEBUG_USARTx, DISABLE);
                //
                IAP_UPDATE_APP();                   //更新接收到的pack包
                f_IAP_flashing = 0;
                f_receive_frame = 0;                //清除接收帧标志
                if(f_final_frame == 1)
                {
                    f_final_frame = 0;
                    receive_app_done = 1;           //更新完毕
                    app_flag_write(0x12345678 ,app_update_flag_addr);//写IAP升级标志
                }
                TIM_Enable(TIM3, ENABLE);
                USART_Enable(DEBUG_USARTx, ENABLE);  2.Flash写入接收的BIN文件
            }
        }
        if(receive_app_done)                        //已有更新APP
        {
            receive_app_done = 0;
            TIM_Enable(TIM3, DISABLE);              //关闭定时器中断
            //
            printf("APP address: %x\r\n",(FLASH_START_ADDR));
            printf("开始执行Flash用户代码!!\r\n");
            iap_load_app(FLASH_START_ADDR);         //跳转到APP起始地址，期间不能被其他中断打断，否则会跳转失败
        }                                           3.BIN更新完成，程序跳转
    }
}
```

图 2.8

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

如图 2.8 所示，在 main()函数里面，程序在初始化完成之后，会判断是否需要直接跳转，因为 Flash 的程序掉电不丢失，更新完成之后可以一直保持，而 Sram 掉电后数据会丢失，所以没有此判断。如果 Bootloader 以外区域还没更新过程序，则会一直等待串口接收 BIN 文件进行更新，由于 N32G45X 的 Flash 一页是 2K，为了避免过多的地址判断，例程里面是每接收完 2K 大小的数据包，就进行一次 Flash 写入，能避免过多占用 Sram 资源。在写入最后一帧的 BIN 数据包后，会往 Flash 里面写一个标志，下次上电会直接跳转到 APP 程序运行。

```
void USART1_IRQHandler(void)
{
    uint8_t i = 0;
    uint8_t buf_temp[256] = {0};
    uint8_t sum_check = 0;
    //
    if(USART_GetFlagStatus(DEBUG_USARTx, USART_INT_RXDNE) != RESET)
    {
        USART_ClrIntPendingBit(DEBUG_USARTx, USART_INT_RXDNE);
        slot_timer = 0;
        if(receive_cnt <= 134)
        {
            RX_buf[receive_cnt++] = USART_ReceiveData(DEBUG_USARTx);
            current_pack_length = RX_buf[3]+5;
            if((RX_buf[0] == 0x01)&&(RX_buf[1] == 0x01)&& (receive_cnt== current_pack_length))
            {
                receive_cnt = 0;
                f_receive_frame = 1;
                memcpy(buf_temp,RX_buf,256);
                for(i = 0; i < current_pack_length -1 ; i++)
                {
                    sum_check = sum_check + buf_temp[i];
                }
                sum_check = ~sum_check + 1;
                if((sum_check == buf_temp[current_pack_length-1])&&(f_IAP_flashing==0))
                {
                    send_ack();
                    memcpy(&flash_buf[rx_nunber*128],&RX_buf[4],current_pack_length-5);
                    rx_nunber ++;
                    if(rx_nunber >= 16)
                    {
                        rx_nunber = 0;
                        f_IAP_flashing = 1;
                        f_IAP_start= 1;
                    }
                    else if((current_pack_length==5)&&(RX_buf[3]==0))
                    {
                        rx_nunber = 0;
                        f_IAP_flashing = 1;
                        f_final_frame = 1;
                    }
                    current_pack_length = 0;
                }
                memset(RX_buf,0x00,sizeof(RX_buf));
            }
```

```
//计算当前pack的数据长度
//帧头 固定为0x01 ,0x01
//pack长度固定为 uart_rx_buf[3] + 5 个字节
//最大128+5个字节
分包接收BIN文件的传输协议和校验
//计算SUM 校验
//比较SUM，如果正在写入flash，丢弃当前包，等待上位机重发
//应答上位机
//数据转存到flash_buf
//每接收完16次共2K后，写一次flash
每接收完2K数据，写一次Flash
//发完bin内容的最后一包后，上位机会发5字节的帧尾
接收最后的帧尾
```

图 2.9

如图 2.9 所示，Flash 的接收跟 Sram 的接收略微不同，Flash 的 Bootloader 定义了一个 2K 的缓存 buf，每次接收完 2K 后会写入一次 Flash。

```
/**==================================================================
        //升级APP
==================================================================*/
void IAP_UPDATE_APP(void)
{
    ready_write_addr = FLASH_APP_BASE_ADDR + pages_number*2048;
    //
    while(app_flash_write((uint32_t *)flash_buf , ready_write_addr));
    //
    memset(flash_buf,0x00,2048);
    pages_number++;

}
```

```
0x08004000
//IAP每次升级2K
```

图 2.10

如图 2.10 所示，程序每次接收完 2K 或者最后一帧数据包后，调用一次 IAP_UPDATE_APP(void)进行升级，起始地址 FLASH_APP_BASE_ADDR 为 0x08004000。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

3 下载验证

3.1 上位机传输协议

验证用到的上位机工具是 XCOM V2.6，其传输协议有 2 字节的帧头，可以灵活配置，支持 ACK 应答，支持分包发送 BIN 文件，每包长度最长 255 字节，并有 SUM、CRC16 等多种校验方式。

协议格式	帧头 1	帧头 2	帧序号	帧长度	数据	数据	数据	检验和
	0x01	0x01	n	length	Data0	Data1	Data2	SUM

协议是前 4 个字节分别为 2 字节的帧头、当前的帧序号、帧 长度，帧头可以随意设定，当帧序号超过 255 后会继续从 0 增加，帧长度用户随意设定。例程的协议帧头都是 0x01，帧长度为 0x80，检验和选择了 SUM 模式。

ACK 格式	帧头 1	帧头 2	帧序号	帧长度	检验和
	0x01	0x01	n	0	SUM

芯片在收到一帧完整的数据包后，会回复上位机一个 ACK 信号，如果不回复 ACK，上位机会重复发送当前帧的数据包。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

3.2 下载 BIN 文件步骤

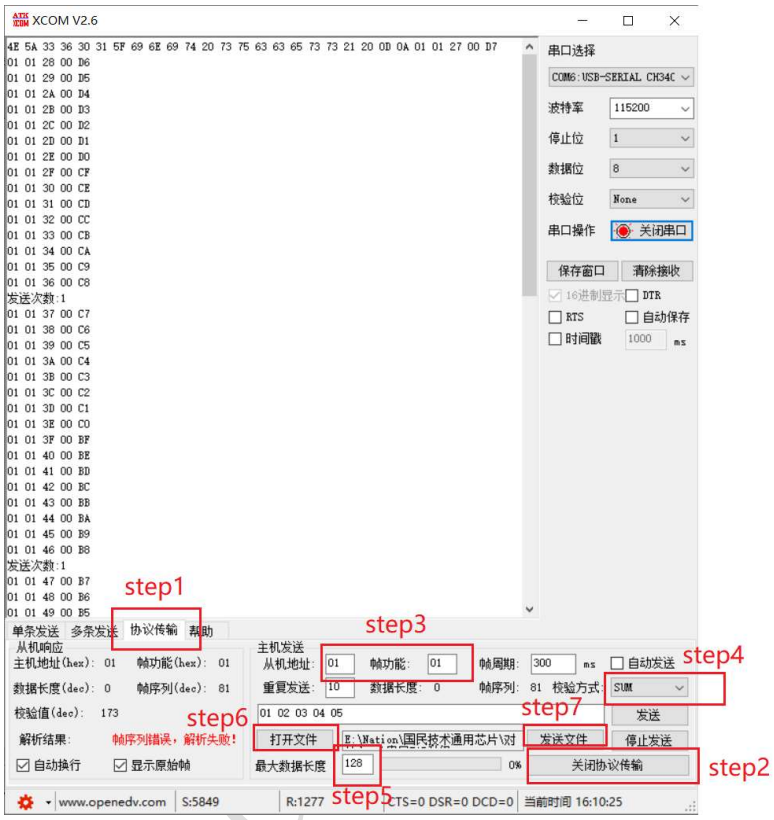

图 3.0

图 3.0 所示，通过上位机下发 BIN 文件，有 7 个步骤：

Step1：打开 XCOM V2.6 工具，选择“协议传输”；

Step2：点击“打开协议传输”；

Step3：配置 2 字节的的帧头，填入 0x01；

Step4：选择检验方式为 SUM；

Step5：配置帧长度为 128；

Step6：打开选择 BIN 文件；

Step7：点击发送文件；

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

3.3 验证

图 3.1

如图 3.1 所示，成功发送完成后，会提示“文件发送完毕，总共耗时:xxxx ms”。

图 3.2

如图 3.2 所示，程序在初始化完成后，跳转到 APP_address：0x08004000 开始执行 FLASH 的程序。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

图 3.3

如图 3.3 所示，在接收完 BIN 文件后，程序成功跳转到 APP_address：0x20008000 执行 SRAM 里面的代码。

4 常见问题

1. Q：无法接收 BIN 文件，校验出错；

   A：检查波特率是否一致，校验方式是否选择了 SUM；

2. Q：APP 程序跳转失败；

   A：检查工程设置的地址和程序上面跳转的地址是否一致；同时在跳转前，关闭所有中断；

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

5 历史版本

版本	日期	备注
V1.0	2020.9.2	新建文档

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

6 声明

国民技术股份有限公司（以下简称国民技术）保有在不事先通知而修改这份文档的权利。国民技术认为提供的信息是准确可信的。尽管这样，国民技术对文档中可能出现的错误不承担任何责任。在购买前请联系国民技术获取该器件说明的最新版本。对于使用该器件引起的专利纠纷及第三方侵权国民技术不承担任何责任。另外，国民技术的产品不建议应用于生命相关的设备和系统，在使用该器件中因为设备或系统运转失灵而导致的损失国民技术不承担任何责任。国民技术对本手册拥有版权等知识产权，受法律保护。未经国民技术许可，任何单位及个人不得以任何方式或理由对本手册进行使用、复制、修改、抄录、传播等。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057