Application Note

IAP Upgrade

Introduction

This document mainly introduces the IAP upgrade application routine, the problems and solutions that may be encountered during application development of N32G45X_FR_WB series chips (hereinafter referred to as N32G45X).

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Contents

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

1 Overview

IAP is an abbreviation of in application programming. It is to burn in some areas of User Flash during the running of the user program. The purpose is to easily update the firmware program in the product through reserved communication ports after the product is released. To realize the IAP function, this is, it will be updated when the user program is running, two project codes need to be written when designing the firmware program. The first project code does not perform normal functional operations, but only receives programs or data through some communication methods (such as USB and UART) to update the second part of the code. The second project code is the real functional code. These two parts of project code are burned in different areas of User Flash simultaneously. When the chip is powered on, the first project code starts to run, and it performs the following operations:

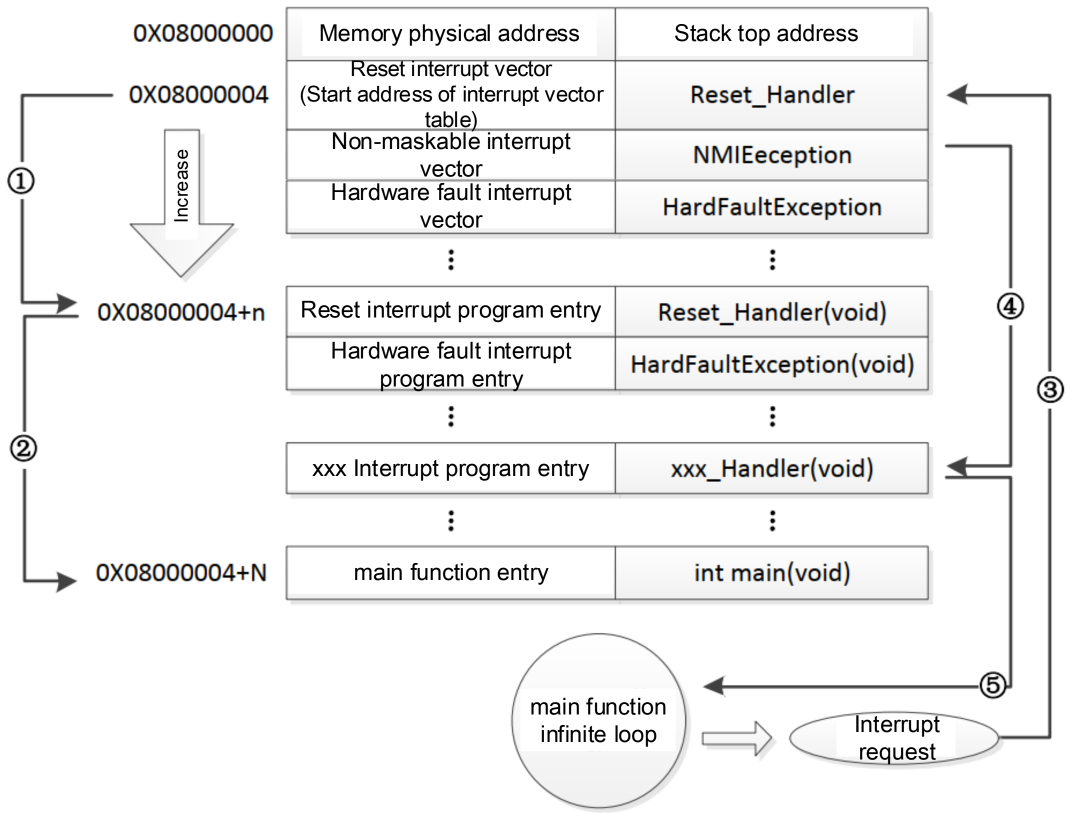
1. Check whether second project code needs to be updated;
2. If no update is required, go to step 4.
3. Perform the update operation.
4. Jump to the second project code for execution;

The first part of the code must be burned in by other methods, such as JTAG or ISP. The second part of the code can be burned in using the IAP function of the first part of the code. It also can be burned in together with the first part of the code. When a program update is needed, it can be updated through the IAP code of the first part.

The first project code is known as Bootloader program, and the second project code is known as APP program. They are generally stored in different address range of N32G45X Flash. Generally, Bootloader is stored from the lowest address area, followed by APP program. New APP programs can be stored in Flash as well as Sram for execution. Subsequent chapters will provide examples for illustration. So according to the above description, we need to implement two programs: Bootloader and APP. The normal program running process of N32G45X is shown in Figure 1-1.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

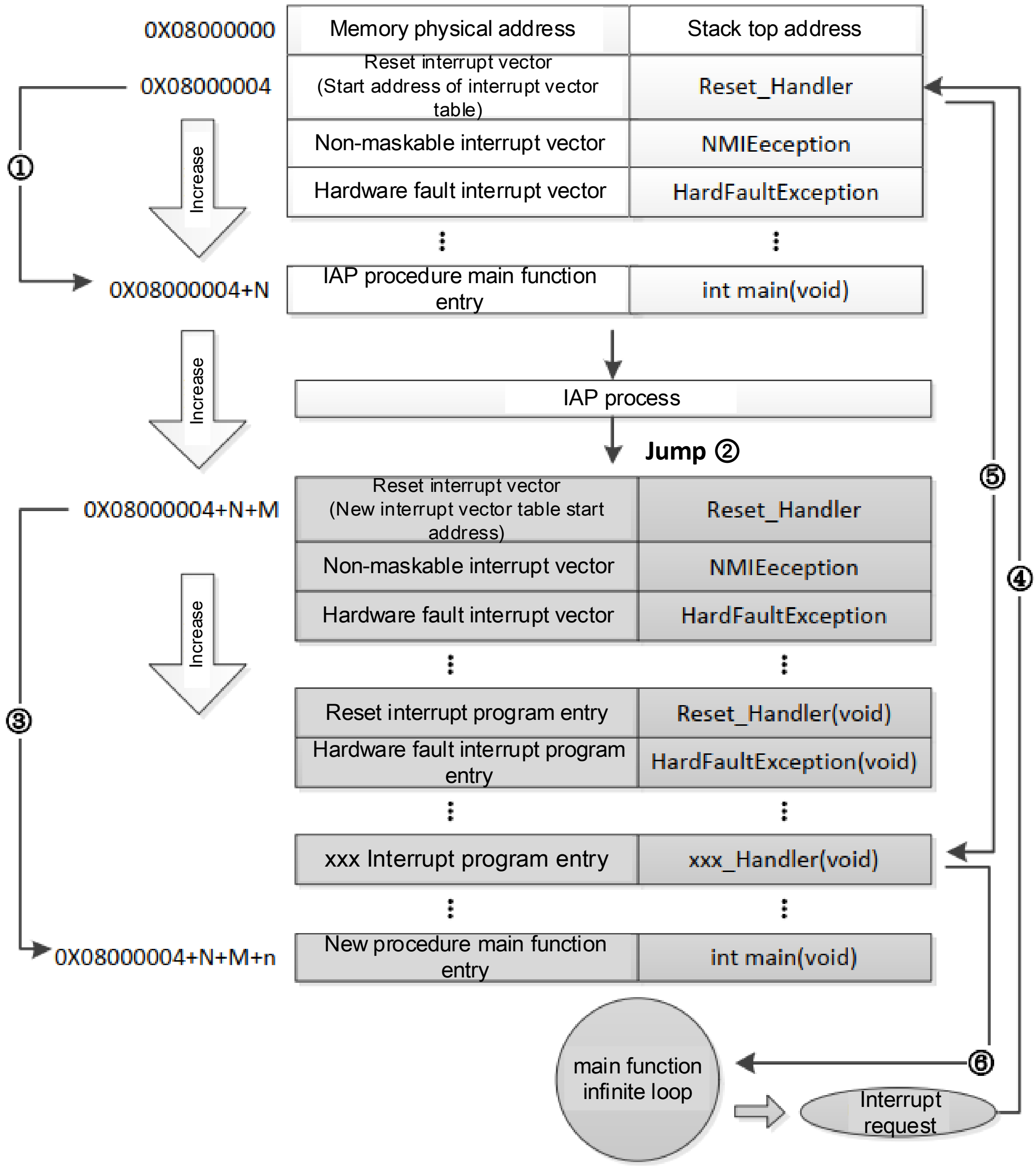
Figure 1-1 The Normal Program Running Process



As shown in the figure above, the address of N32G45X's embedded Flash starts at 0x08000000, the program files are written starting from this address. The N32G45X is a microcontroller based on the Cortex-M4F kernel, which internally responds to interrupts through an interrupt vector table. After the program is startup, it will first take out the reset interrupt vector from the interrupt vector table and execute the reset interrupt program to complete the startup. The starting address of this interrupt vector table is 0x08000004. When the interrupt comes, the internal hardware mechanism of N32G45X will automatically locate the PC pointer to the interrupt vector table. According to the interrupt source, the corresponding interrupt vector is extracted to execute the interrupt service program.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 1-2 APP Update Process



As shown in Figure 1-2, after powering on, the chip will extract the address of reset interrupt vector from the address 0x08000004 of Flash and jump to the interrupt reset function. After executing the interrupt reset function, the program will jump to the main function of IAP and start executing. When the main function is waiting for the upgrade, users can update the APP by transmitting the update file through USB or UART. During the upgrade process, users can update while receiving, or update after receiving the whole package of APP program. Since Flash and Sram reserved by Bootloader are relatively small, the routine of this application note will update APP by subcontracting transmitting and receiving while updating.

After the APP program is updated, the program pointer jumps to the reset vector table of the newly written program. The address of the reset interrupt vector are taken out from the new program, then program pointer jumps to the reset interrupt service program of the new program, and then jumps to the main function of the APP program, steps 2 and 3 as shown in Figure 1-2, Main function is an infinite loop. And it is noticed that N32G45X Flash has two interrupt vector tables in different positions.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

During the execution of main function, if the CPU gets an interrupt request, the PC pointer still forcibly jumps to address 0X08000004 instead of the interrupt vector table of the new program, step 4 as shown in Figure1-2. Then the program jumps to the new interrupt service program corresponding to the interrupt source according to the offset of the interrupt vector table set by us, step 5 as shown in Figure 1-2. After executing the interrupt service program, the program returns the main function to continue running, step 6 as shown in the Figure 1-2. The start address of the reset interrupt vector of the new program is 0X08000004+N+M, where M is the jump offset of the new program. Subsequent chapters will explain how to set the offset in project.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

2 IAP Software Implementation Process

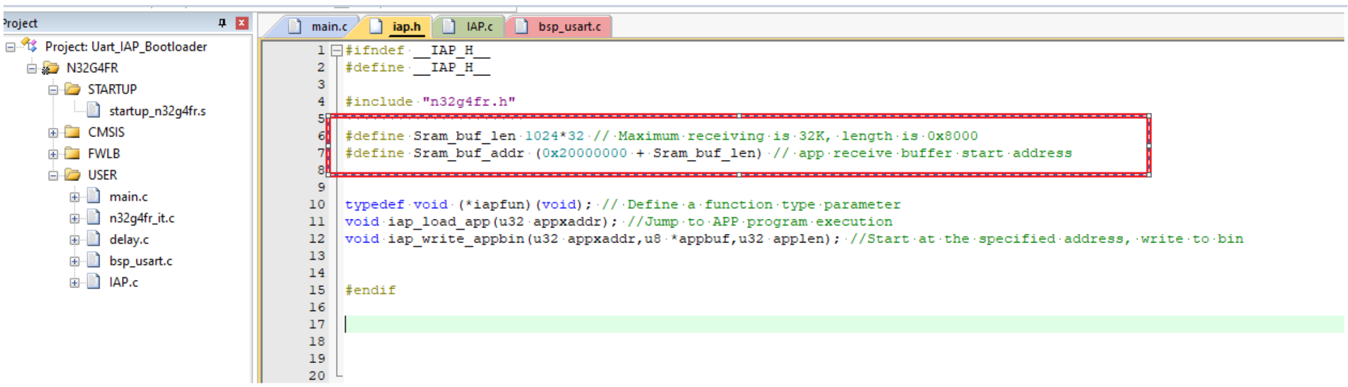
Through the analysis of the above two processes, we know that the IAP application must meet two requirements:

1. The new program must start at an address with offset X after the IAP program;
2. The interrupt vector table of the new program must be moved with an offset of X;

2.1 Set The Start Address of The APP Program

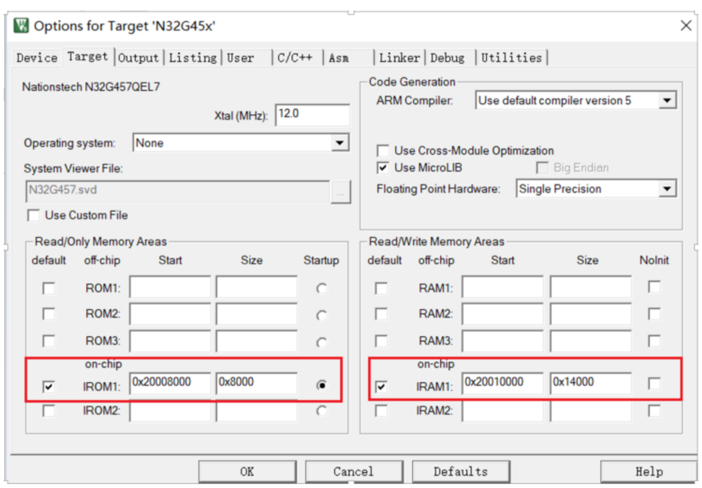
2.1.1 Set Sram_APP Start Address

Figure 2-1 Define Start Address and Length of Array Sram_buf in the Bootloader Project



As shown in Figure 2-1, in the Bootloader project, the array Sram_buf is defined to store the APP program. Its starting address is 0x20008000 and the length is 1024*32 (32K).

Figure 2-2 Division of Sram Space



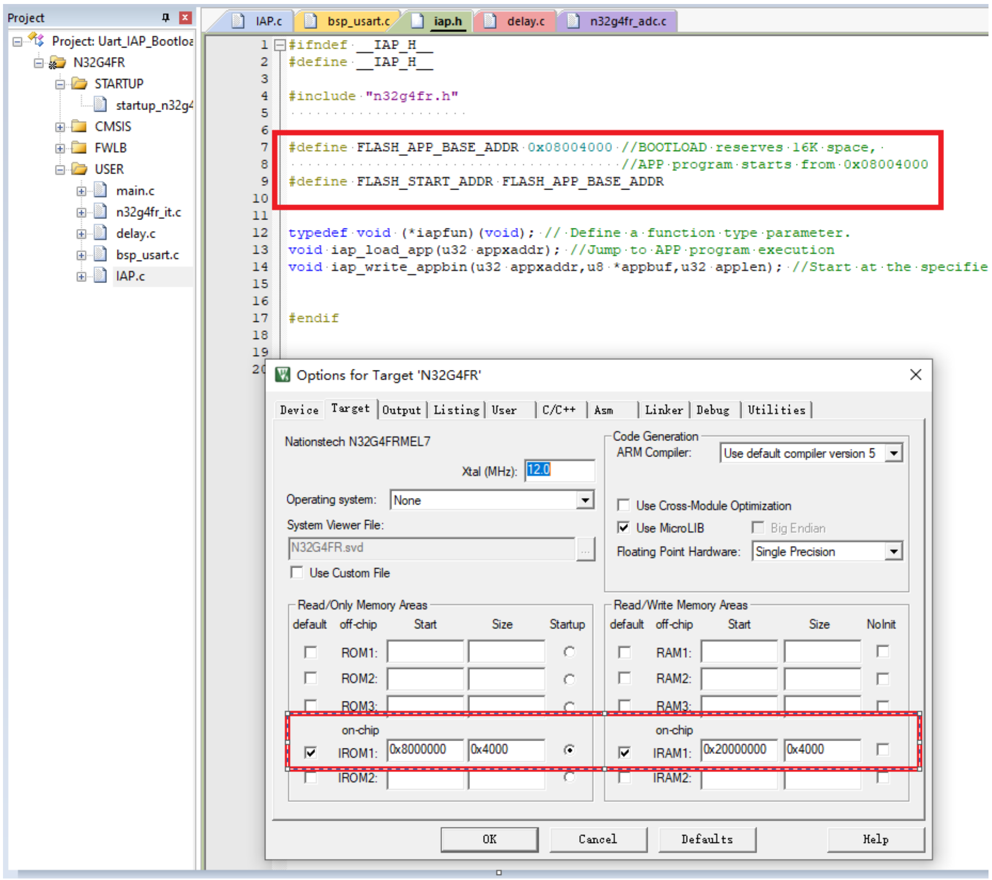
Since the Sram of N32G45X starts at 0x20000000 and ends at 0x20024000, the size of the entire Sram is 144K. So in the SRAM_APP project, setting the offset as shown in Figure 2-2: Click “magic wand”, select “Target”, type 0x20008000 for “Start” and 0x8000 for “Size” in the “IROM1” column; type 0x20010000 for “Start” and 0x14000 for “Size” in the “IRAM1”

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

column. Therefore, the distribution of the Sram resources is as follows: the first 32K is allocated to the Bootloader, the next 32K is used to store APP programs, and the remaining 80K is allocated to APP program to call.

2.1.2 Set Flash_APP Start Address

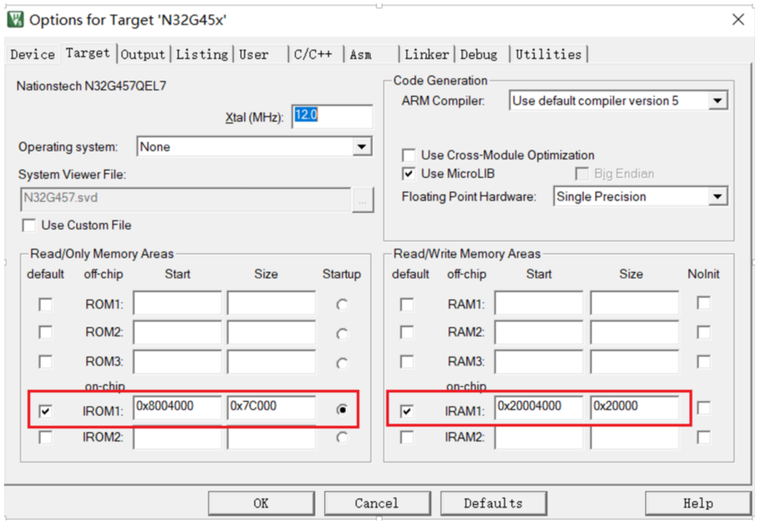
Figure 2-3 Configure FLASH_APP Start Address



As shown in Figure 2-3, in the Bootloader project corresponding to Flash_App, 16K Flash and 16K Sram are reserved for the small amount of code about 13K, and the Flash jump address 0x0800400 is set. Click “magic wand”, select “Target”, type 0x08000000 for “Start” and 0x4000 for “Size” in the “IROM1” column; type 0x20000000 for “Start” and 0x4000 for “Size” in the “IRAM1” column.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 2-4 Division of Flash Space



The Flash of N32G45X has a maximum capacity of 512K, from address 0x0
8000000 to 0x08080000. The routine uses the first 16K Flash for the Bootloader and the remaining 496K Flash for APP. As shown in Figure 2-4, in Flash_App project, click “magic wand”, select “Target”, type 0x08004000 for “Start” and 0x7C000 for “Size” in “IROM1” column; type 0x20004000 for “Start” and 0x20000 for “Size” in the “IRAM1” column,.

2.2 Set The Offset of Interrupt Vector Table

When the system startup, the systemInit function is called first to initialize the clock, and the systemInit function also completes the setup of the interrupt vector table. At the end of the systemInit function code, there are following lines of code:

```
#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
```

It can be understood from the code that the VTOR register stores the start address of the interrupt vector table. VECT_TAB_SRAM is not defined by default, so perform SCB - > VTOR = FLASH_BASE | VECT_TAB_OFFSET; For Flash APP, we set it to FLASH_BASE+ offset 0x4000, so we can add the following code before jumping to the main function of FLASH APP to reset the start address of the interrupt vector table:

SCB->VTOR = FLASH_BASE | 0x4000;

The above is the case of Flash APP. When using Sram APP, we set the start address as: SRAM_bASE+0x8000. Using the same method, before jumping to the main function of Sram APP, we add the following code:

SCB->VTOR = SRAM_BASE | 0x8000;

This completes the setting of the interrupt vector table offset

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

2.3 Generate BIN File in APP project

Figure 2-5 Configure to Generation the "BIN" File

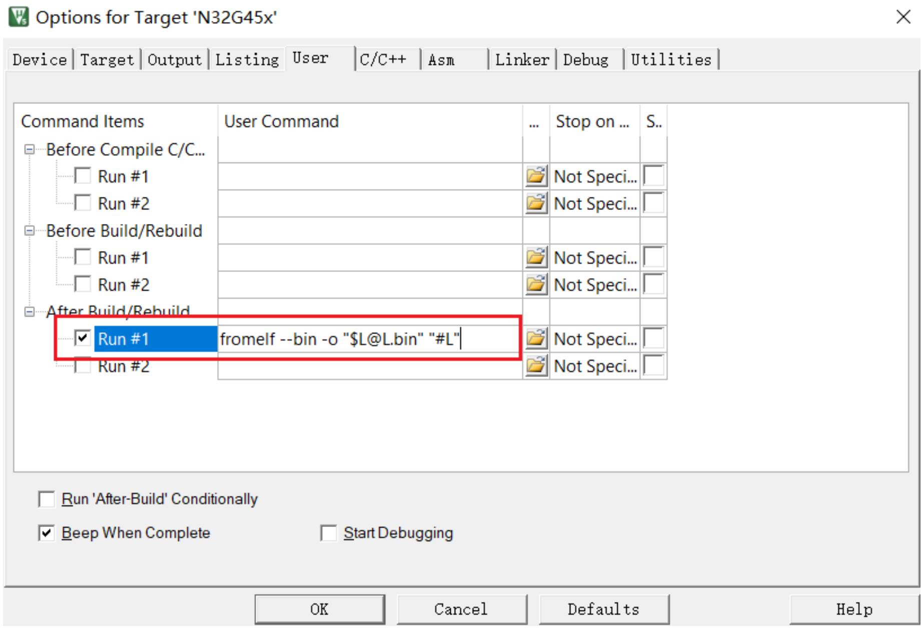


Figure 2.4

In the Sram App and Flash App projects, click "Magic Wand" and select "USER". Under "After Build/Rebuid", tick the box to the left of "RUN #1", and fill "fromelf --bin -o "$L@L.bin" "#L"" in the right column. After clicking OK, recompile the program and the BIN file can be generated. The BIN file is saved in the "\MDK-ARM\Objects" directory.

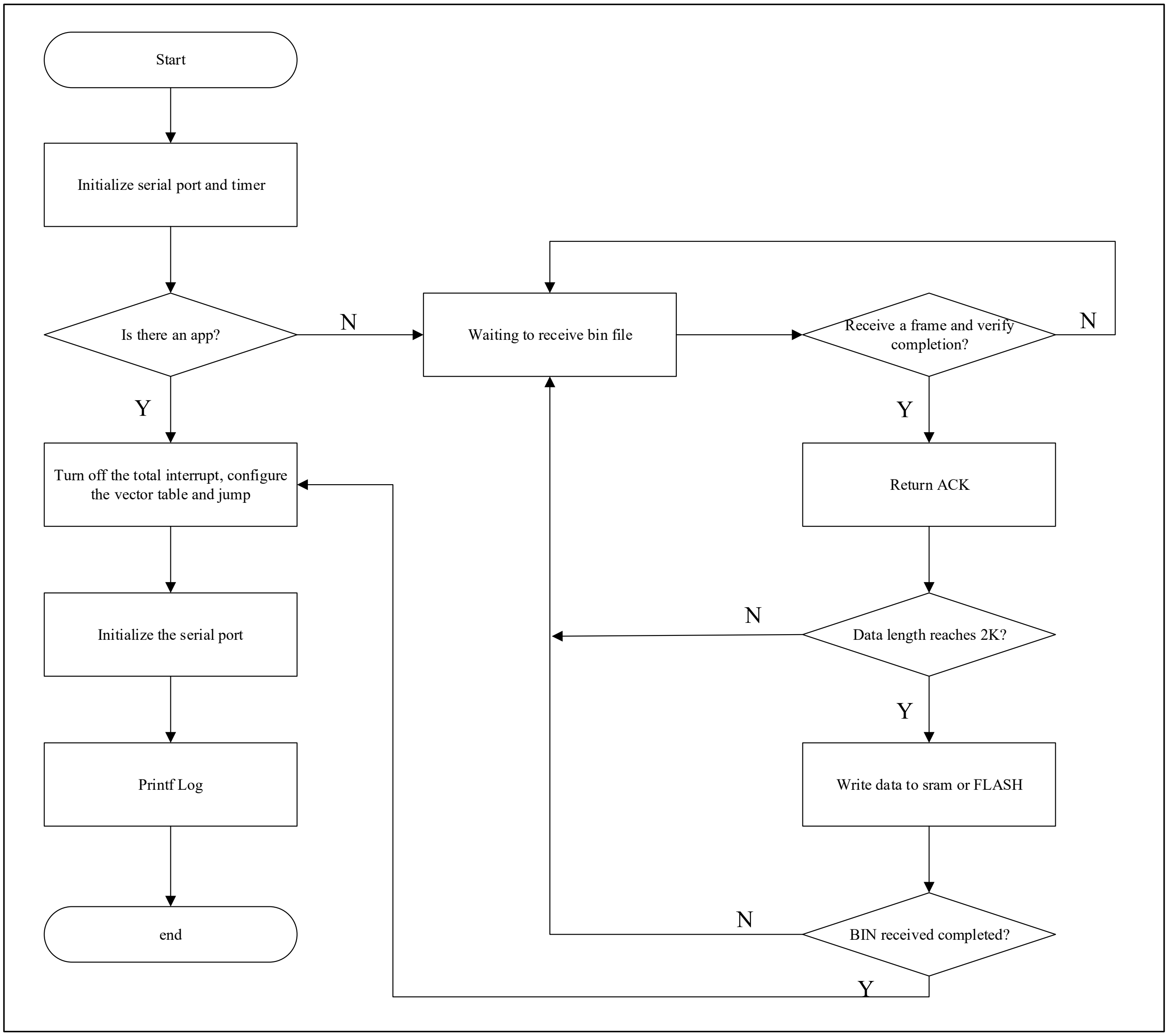
2.4 Software Implementation Process

The software process of Bootloader mainly consists of three steps:

1. Power on and initialize the serial port to determine whether the BIN file of the App is waiting to receive.
2. Subcontracting receives the BIN file, then store the contents to Sram_buf at the specified address, or writes to the specified Flash address;
3. After receiving BIN file is complete, the program jumps;

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 2-5 Upgrade Flow



NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

2.4.1 Bootloader Process of Sram_APP

Open Uart_IAP_Bootloader project, we can see that the program is mainly in "main.c", "IAP.c", and "bsp_usart.c" files. The code for the three steps will be detailed below.

Figure 2-6 Code of the main Function When APP Programs are Stored in Sram

```c
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) //No APP program, waiting to receive updates
        {
            if(f_final_frame == 1)
            {
                receive_app_done = 1; //After receiving the BIN upgrade file
                m_delay_ms(500);
                break;
            }
        }
        if(receive_app_done) //App has been updated
        {
            receive_app_done = 0;
            TIM_Enable(TIM3, DISABLE); //Turn off timer interrupt
            //
            printf("APP address:%x\r\n",(Sram_buf_addr));
            printf("Start to execute SRAM user code!!\r\n");

            SCB->VTOR = SRAM_BASE | 0x8000;   //Set up interrupt vector table before jump
            iap_load_app(Sram_buf_addr); //Jump to the start address of the APP, during which it cannot be interrupted by other interrupts, otherwise the jump will fail
        }
    }
}
```

1. Waiting to receive the bin file

3. After the reception is completed, the program jumps

Figure 2-7 Code of the USART1_IRQHandler Function When APP Programs are Stored in Sram

```c
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; //Calculate the data length of the current pack
            if((RX_buf[0] == 0x01)&&(RX_buf[1] == 0x01)&& (receive_cnt== current_pack_length)) //Frame header is fixed to 0x01, 0x01.
            {//pack length is fixed to uart_rx_buf[3] + 5 bytes,
                receive_cnt = 0; //The maximum length is 128+5 bytes
                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]; //Calculate SUM check
                }
                sum_check = ~sum_check + 1;
                if(sum_check == buf_temp[current_pack_length-1]) //Compare SUM, if it is different, discard the current packet and wait for the host computer to resend
                {
                    send_ack(); //Respond to the host computer
                    memcpy(&Sram_buf[rx_nunber*128],&RX_buf[4],current_pack_length-5); //Data transfer to Sram_buf
                    rx_nunber ++;
                    if((current_pack_length==5)&&(RX_buf[3]==0)) //After sending the last packet of bin content, the host computer will send a 5-byte frame end
                    {
                        rx_nunber = 0;
                        f_final_frame = 1; //After receiving the last frame of data
                    }
                    current_pack_length = 0;
                }
                memset(RX_buf,0x00,sizeof(RX_buf));
            }
        }
    }
}
```

Subcontract to receive bin file and verify

2. Cache the bin file to Sram and return ACK

As shown in Figure 2-6 and Figure 2-7, in main function, after initialization, there are two while(1) loops, one for waiting to receive and the other for jumping to Sram execution program respectively. BIN upgrade file is received in the serial port interrupt USART1_IRQHandler(void) function. In order to minimize the use of Bootloader resources, be compatible with receiving large BIN files and ensure the integrity of BIN files, we split BIN into small packages for transmitting, transmitting 128 bytes each time. Therefore, after receiving a packet of data, it will be verified according to the transmission protocol. If

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

the verification fails, the current packet will be discarded and the host computer retransmits the current packet. Transport protocols will be described in detail in subsequent section.

Figure 2-8 Code of the iap_load_app Function When APP Programs are Stored in Sram

```
__asm void MSR_MSP(u32 addr)
{
    MSR MSP, r0 //set Main Stack value
    BX r14
}
/**============================================= =================
        APP jump
        appxaddr: The starting address of the user code.
============================================== ==============*/
void iap_load_app(u32 appxaddr)
{
    if(((*(vu32*)appxaddr)&0x0FFFFFFF) <(1024*144)) // Check whether the top address of the stack is legal.
    {
        jump2app = (iapfun)*(vu32*)(appxaddr+4);
        MSR_MSP(*(vu32*)appxaddr); // initialize the stack pointer
        jump2app(); // Jump to APP.
    }
}
/**============================================= =================
============================================== ==============*/
```

set offset and jump

As shown in Figure 2-8, after receiving the complete BIN file, program jumps to iap_load_app (Sram_buf_addr) in Figure 2-5; Sram_buf_addr is the starting address 0x20008000 of Sram_buf that we set in Figure 2-1.

2.4.2 Bootloader Process of Flash_App

Figure 2-9 Code of the main Function When APP Programs are Stored in Flash

```
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) //Whether the power-on detection needs to jump directly
        {
            receive_app_done = 1;
        }
        while(receive_app_done == 0) //No APP program, waiting to receive updates
        {
            if(f_IAP_flashing == 1)
            {
                TIM_Enable(TIM3, DISABLE);
                USART_Enable(DEBUG_USARTx, DISABLE);
                //
                IAP_UPDATE_APP(); //Update the received pack package
                f_IAP_flashing = 0;
                f_receive_frame = 0; //Clear the receive frame flag
                if(f_final_frame == 1)
                {
                    f_final_frame = 0;
                    receive_app_done = 1; //Update is complete
                    app_flag_write(0x12345678 ,app_update_flag_addr);//Write IAP upgrade flag
                }
                TIM_Enable(TIM3, ENABLE);
                USART_Enable(DEBUG_USARTx, ENABLE);
            }
        }
        if(receive_app_done) //App has been updated
        {
            receive_app_done = 0;
            TIM_Enable(TIM3, DISABLE); //Turn off timer interrupt
            //
            printf("APP address:%x\r\n",(FLASH_START_ADDR));
            printf("Start to execute Flash user code!!\r\n");
            iap_load_app(FLASH_START_ADDR); //Jump to the start address of the APP, during which it cannot be interrupted by other interrupts, otherwise the jump will fail
        }
    }
}
```

1. Check whether you need to jump directly

2. The Flash writes the received BIN file

3.BIN update completed, jump

As shown in Figure 2-9, in the main function, the program will determine whether it needs to jump directly after initialization, because the program in Flash will not be lost in power down and can be maintained all the time after updating, but the data in Sram will be lost after power down, so there is no such judgment. If the program has not been updated outside

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

the Bootloader area, it will wait to receive the BIN file through the serial port for updating. Since one Flash page of N32G45X is 2K, to avoid too much address judgment, the routine is to write the Flash once after receiving the packet of 2K size. It can avoid occupying too much Sram resources. After writing the last frame of the BIN data packet, a flag will be written into the Flash, and the next power-on will directly jump to the APP program.

Figure 2-10 Code of the USART1_IRQHandler Function When APP Programs are Stored in Flash

```
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;                                   //Calculate the data length of the current pack
            if((RX_buf[0] == 0x01)&&(RX_buf[1] == 0x01)&& (receive_cnt== current_pack_length))   //Frame header is fixed to 0x01, 0x01
            {                                                                    //pack length is fixed to uart_rx_buf[3] + 5 bytes
                receive_cnt = 0;                                                 //Maximum 128+5 bytes
                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];                         //Calculate SUM check
                }
                sum_check = ~sum_check + 1;
                if((sum_check == buf_temp[current_pack_length-1])&&(f_IAP_flashing==0))   //Compare SUM, if flash is being written, discard the current packet, and wait for the host computer to resend
                {
                    send_ack();                                                  //Respond to the host computer
                    memcpy(&flash_buf[rx_nunber*128],&RX_buf[4],current_pack_length-5);   //Dump data to flash_buf
                    rx_nunber ++;
                    if(rx_nunber >= 16)                                          //After receiving 16 times for a total of 2K, write a flash
                    {
                        rx_nunber = 0;
                        f_IAP_flashing = 1;
                        f_IAP_start = 1;
                    }
                    else if((current_pack_length==5)&&(RX_buf[3]==0))            //After sending the last packet of bin content, the host computer will send a 5-byte frame end
                    {
                        rx_nunber = 0;
                        f_IAP_flashing = 1;
                        f_final_frame = 1;
                    }
                    current_pack_length = 0;
                }
                memset(RX_buf,0x00,sizeof(RX_buf));
            }
```

Transmission protocol and checksum of subcontracted receiving bin files

After receiving 2K data, write flash once

Receive the last end of frame

As shown in Figure 2-10, Flash reception is slightly different from Sram reception in that the Flash Bootloader defines a 2K cache buffer that will be written to Flash after 2K reception.

Figure 2-11 Code of the IAP_UPDATE_APP Function When APP Programs are Stored in Flash

```
/**================================================= =================
        //Upgrade 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)); //IAP upgrade 2K each time
    //
    memset(flash_buf,0x00,2048);
    pages_number++;

}
```

0x08004000

As shown in Figure 2-11, after receiving 2K data or the last frame of the data packet, the IAP_UPDATE_APP(void) is called for upgrading. The starting address FLASH_APP_BASE_ADDR is 0x08004000.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

3 Download Verification

3.1 Host Computer Transmission Protocol

The host computer tool used for verification is XCOM V2.6, its transmission protocol has a frame header of 2 bytes and can be flexibly configured. It supports ACK response and subcontracting to send the BIN file. The maximum length of each packet is 255 bytes. It has SUM, CRC16 and other verification methods.

Table 3-1 Protocol Format

Protocol Format	Frame Header 1	Frame Header 2	Frame Number	Length of the Frame	Data	Data	Data	Data	Checksum
	0x01	0x01	n	length	Data 0	Data 1	Data 2	Data n	SUM

The protocol consists of the first 4 bytes, which are 2 bytes frame header, the current frame number and the length of the frame. The frame header can be set at will. When the frame number exceeds 255, it will continue to increase from 0. The frame length is arbitrarily set by the user. The frame header of the routine is 0x01, and the frame length is 0x80. The SUM mode is verified and selected. After the frame number is increased to 255, the next frame will be counted from 0.

Table 3-2 ACK Format

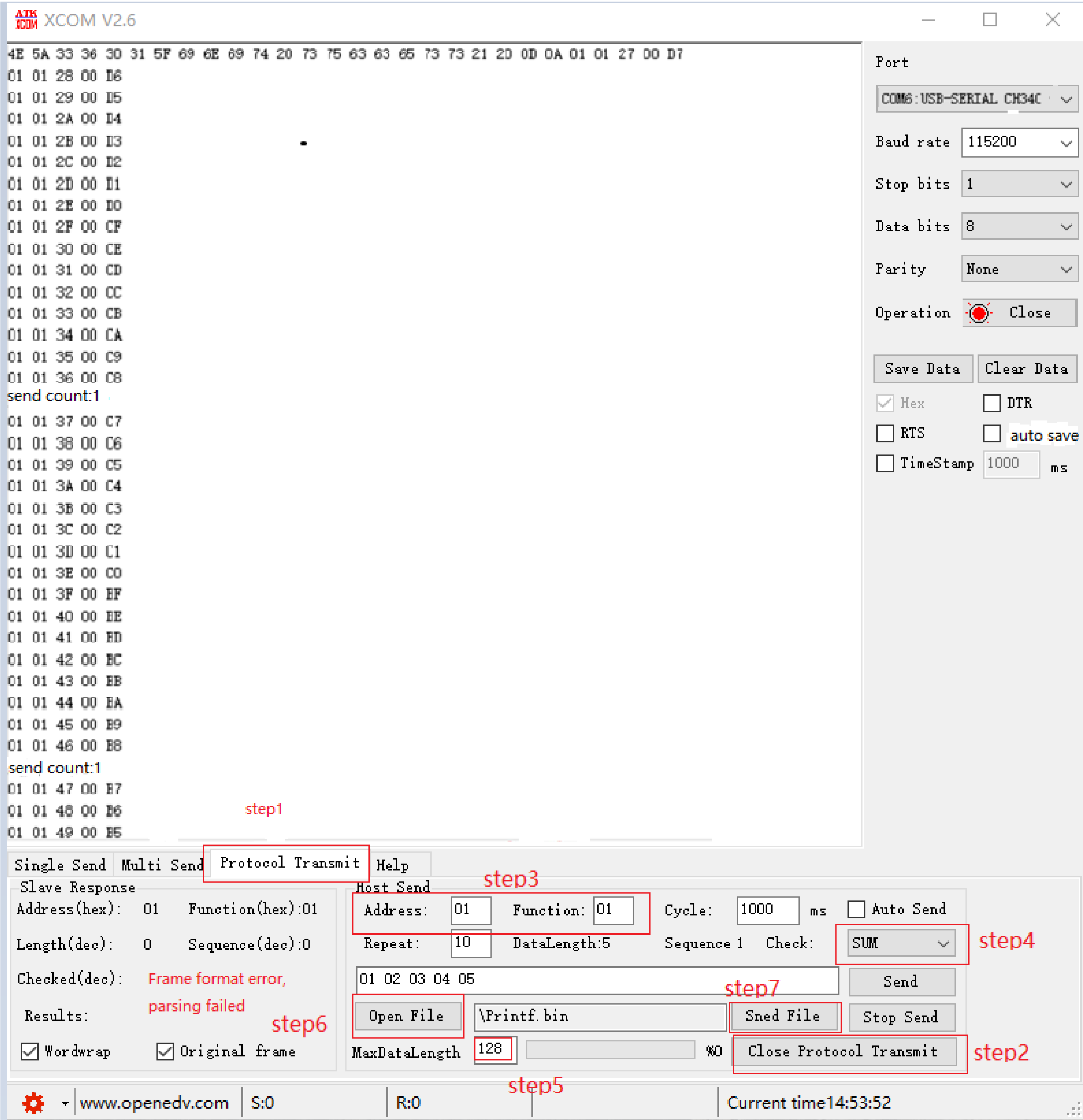
ACK Format	Frame Header 1	Frame Header 2	Frame Number	Length of The Frame	Checksum
	0x01	0x01	n	0	SUM

After receiving a complete packet, the chip will respond to the host computer with an ACK signal. If no ACK is received, the host computer will send the packet of the current frame repeatedly.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

3.2 Process for Downloading the BIN File

Figure 3-17 Procedure for Downloading the "BIN" File from the Host Computer



As shown in Figure 3-1, there are 7 steps for downloading the BIN file from the host computer:

Step1: open XCOM V2.6 and select "Protocol Transmit";

Step2: click "Open Protocol Transmit";

Step3: configure a 2-byte frame header and fill in 0x01;

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Step4: Select “SUM” as the validation method;

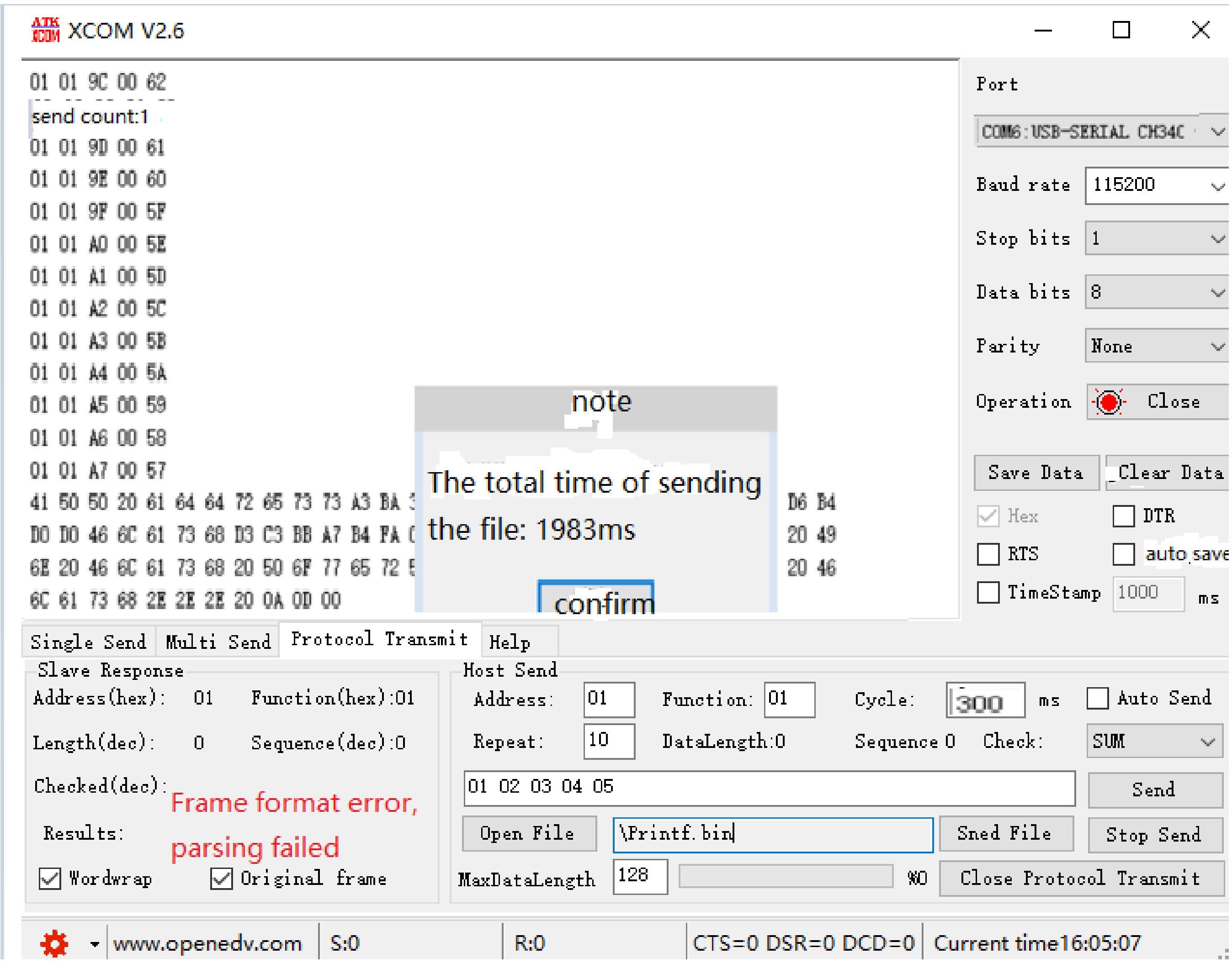
Step5: Set the frame length to 128;

Step6: Open the selection BIN file;

Step7: Click “Send File”;

3.3 Verification

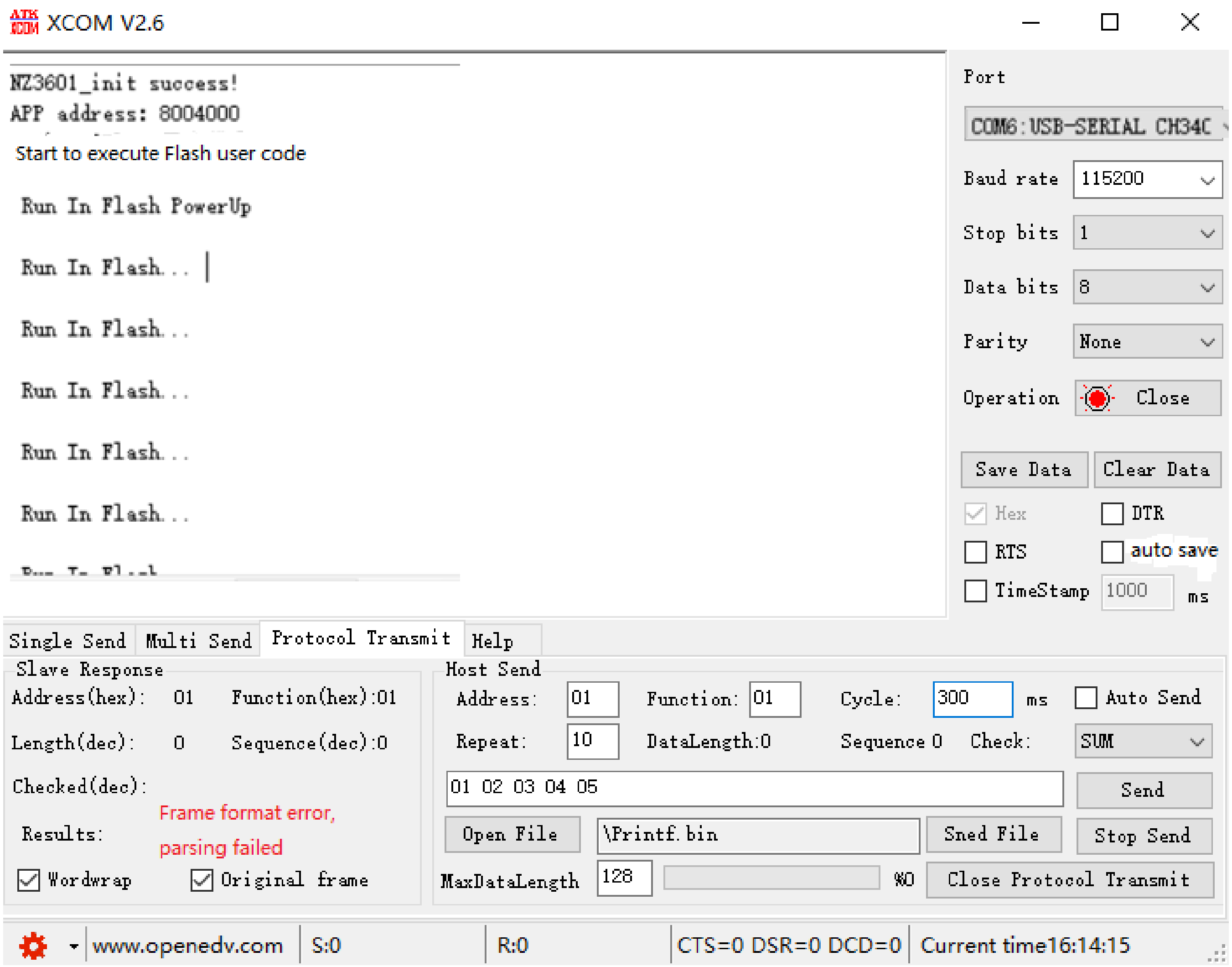
Figure 3-2 The Message Displayed after the Successful Sending



As shown in Figure 3-2, after the successful sending, the message "The total time of sending the file: XXXX ms" will be displayed.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 3-3 The Program Jumps to APP_address to Execute the Code in Flash After Initialization



As shown in Figure 3.3, after initialization, the program jumps to APP_address: 0x08004000 to start executing the program in Flash.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 3-4 The Program Jumps to APP_address to Execute the Code in Sram After Initialization

As shown in Figure 3-4, after receiving the BIN file, the program successfully jumped to APP_address: 0x20008000 to execute the code in Sram.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

4 Q&A

1. Q: The BIN file cannot be received, and the verification fails.

   A: Check whether the baud rate is consistent and whether SUM is selected as the verification method.

2. Q: The APP program fails to jump;

   A: Check whether the address set in the project matches the address the program will jump to; disable all interrupts before jumping.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

5 Version History

Version	Date	Changes
V1.0	2020.9.2	The initial release
V1.1	2021.7.1	Added IAP software flowchart

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

6 Disclaimer

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD.(Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to Nations Technologies Inc. and Nations Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections. enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NATIONS has attempted to provide accurate and reliable information, NATIONS assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NATIONS be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product.

NATIONS Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, Insecure Usage'. Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NATIONS and hold NATIONS harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NATIONS, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg