AN0007
Application Note

AT32 IAP using the USB HID

Introduction

For most Flash-memory-based systems, an important requirement is the ability to update firmware installed in the end product, which is referred to as in-application programming (IAP).

This application note is written to provide general guidelines for creating an IAP application by USB HID.

The source codes of IAP_Programmer.exe host computer software and embedded HID IAP example are in the *utilities* folder of BSP firmware library.

Applicable products:

Part number	AT32F4xx

Note: Only the parts with USB peripherals are applicable.

Contents

List of Tables

List of Figures

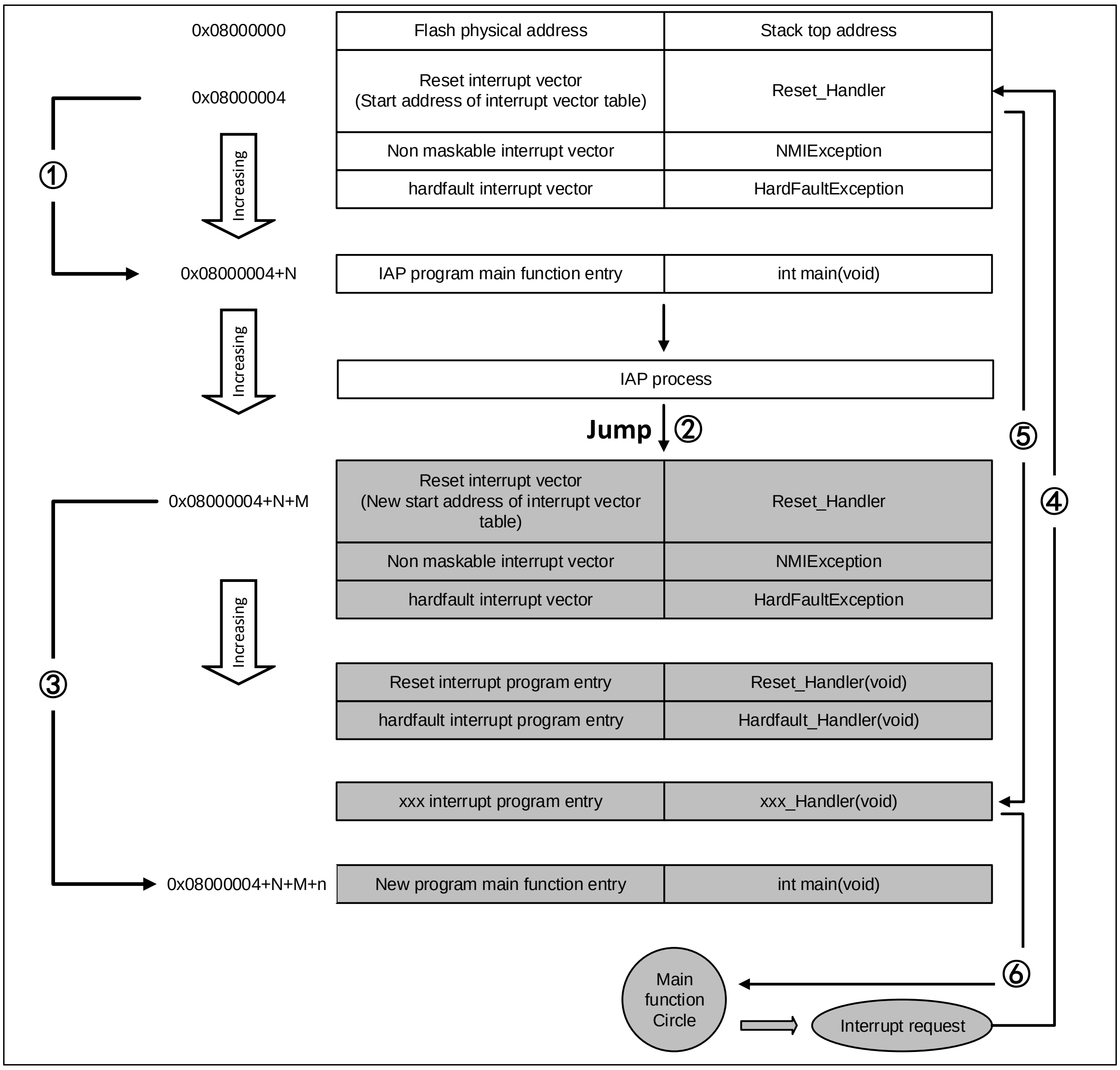
1 IAP online upgrade principle

IAP is the acronym of in-application-programming used by the user program to write on part of user Flash during the running process, with the purpose of upgrading the firmware in the product through the reserved communication interfaces after the product is released.

In most cases, to perform IAP function, two project codes have to be written when designing the firmware program. The first project code does not perform normal function, but simply receives the program or data through some communication modes (like USB or USART) to update the second code, whereas the second code comes with real function. Both codes are programmed in the User Flash at the same time. After the chip is powered, it is the first code to start running, and proceeds as follows:

1. Check if it is necessary to update the second code;
2. If unnecessary, go to Step 4;
3. Execute the update operation;
4. Jump to the second code and execute.

Figure 1. IAP code execution flowchart


0x08000000
Flash physical address
Stack top address
0x08000004
Reset interrupt vector (Start address of interrupt vector table)
Reset_Handler
Non maskable interrupt vector
NMIException
hardfault interrupt vector
HardFaultException
①
Increasing
0x08000004+N
IAP program main function entry
int main(void)
Increasing
IAP process
Jump ②
⑤
④
0x08000004+N+M
Reset interrupt vector (New start address of interrupt vector table)
Reset_Handler
Non maskable interrupt vector
NMIException
hardfault interrupt vector
HardFaultException
Increasing
③
Reset interrupt program entry
Reset_Handler(void)
hardfault interrupt program entry
Hardfault_Handler(void)
xxx interrupt program entry
xxx_Handler(void)
0x08000004+N+M+n
New program main function entry
int main(void)
⑥
Main function Circle
Interrupt request

In this flowchart, after AT32 is reset, it still fetches the address of the reset interrupt vector from the 0X08000004 and jumps to the reset interrupt service program. After running the reset interrupt service program, it jumps to the main function of IAP, as shown in ①. After executing IAP (that is, write a new APP code into AT32 Flash, shown in grey. The start address of the reset interrupt vector of the new program is 0X08000004+N+M), it jumps to the reset vector table of the new program, fetches the address of the reset interrupt vector of the new program, and jumps to run the reset interrupt service program of the new program, and then jumps to the main function of the new program, as shown in ② and ③. Similarly, the main function is an endless loop, and it should be noted that AT32 Flash has two interrupt vector tables in different positions.

During the execution of main function, if CPU received an interrupt request, the PC pointer still forcibly jumps to the interrupt vector table of the address 0X08000004, instead of the interrupt vector table of the new program, as shown in ④. Then, the program jumps to the new interrupt service program corresponding to the interrupt source according to the interrupt vector table offset, as shown in ⑤. At the end of the execution of the interrupt service program, the program returns to the main function to continue running, as shown in ⑥.

Based on the above analysis, we know that two requirements must be met for IAP program:

1. The new program must start from the address offset x after the IAP program
2. The interrupt vector table of the new program must be moved accordingly at an offset x.

2 How to use AT32 USB HID IAP

2.1 Hardware requirements

This application note uses AT-START-AT32F403A demo board as an example. The IAP demo source code also includes other models of AT32, and the user only needs to compile the corresponding model project and program to AT-START demo board.

1) LED2/LED3/LED4
2) USB(PA11/PA12)
3) AT-START demo board

2.2 Software requirements

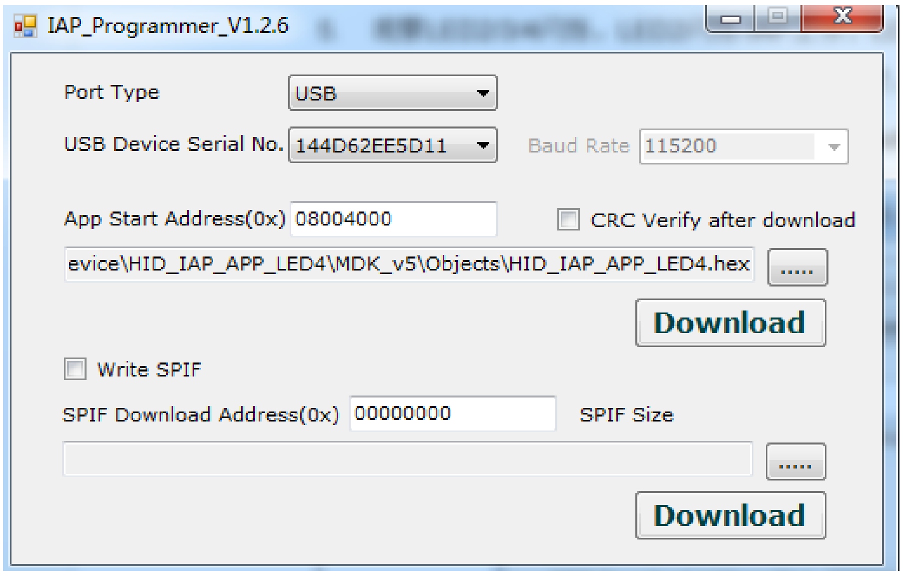
1. APP_Release
   - IAP_Programmer.exe (PC tool)
2. SourceCode
   - Bootloader (IAP source program, LE2 blinks when running)
   - app_led3_toggle (app1 source code, LED3 blinks when running)
   - app_led4_toggle (app2 source code, LED4 blinks when running)

Note: All projects are built around *keil v5* and *IAR8.2*. If users want to use them in other compiling environments, please refer to *AT32F403A_407_Firmware_Library_V2.x.x\project\at_start_f403a\templates (such as IAR6/7/8, keil 4/5, eclipse_gcc)* for a simple change.

2.3 How to use IAP Programmer

1. Open the bootloader source program, compile and download to demo board;
2. Open IAP Programmer.exe;
3. As shown in the figure below, select **USB**. The HID device is used, so no driver is required.
4. Select APP download address (this address must match the IAP download address) and bin files. Tick the CRC verify if necessary, and then click on **Download**.
5. Check the status of LED2/3/4: LED2 blinks-bootloader working, LED3 blinks-app1 working, and LED4 blinks-app2 working.
6. Support power failure protection. It remains in IAP mode for the next start if the download is not successful.

Figure 2. IAP demo host PC



IAP_Programmer_V1.2.6

Port Type: USB

USB Device Serial No.: 144D62EE5D11

Baud Rate: 115200

App Start Address(0x): 08004000

☐ CRC Verify after download

evice\HID_IAP_APP_LED4\MDK_v5\Objects\HID_IAP_APP_LED4.hex

☐ Write SPIF

SPIF Download Address(0x): 00000000

SPIF Size

3 AT32 USB HID IAP settings

3.1 Address distribution

Table 1. Address distribution

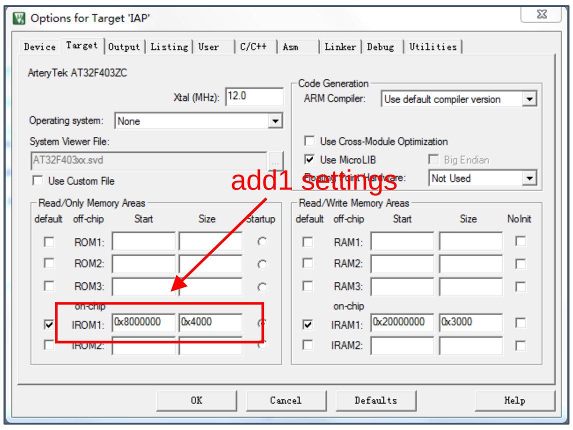
ITEM	Address and Size
app code	Add2: 0x800 4000 ~
bootloader code	Add1: 0x800 0000 size: 0x4000 (16K Byte)

Note: The last page in bootloader is used to place the flag that may be lost due to power-off. Please do not operate this address when modifying bootloader.

3.2 Bootloader project settings

1) Keil settings

Figure 3. Bootloader project add1 settings in Keil



Options for Target 'IAP'

ArteryTek AT32F403ZC

Xtal (MHz): 12.0

Operating system: None

System Viewer File: AT32F403xx.svd

☐ Use Custom File

Code Generation

ARM Compiler: Use default compiler version

☐ Use Cross-Module Optimization

☑ Use MicroLIB ☐ Big Endian

Floating Point Hardware: Not Used

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x8000000	0x4000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x20000000	0x3000	☐
☐	IRAM2:			☐

2) Modify hid_iap_user.h file in bootloader source program

Figure 4. Bootloader project add2 settings in program

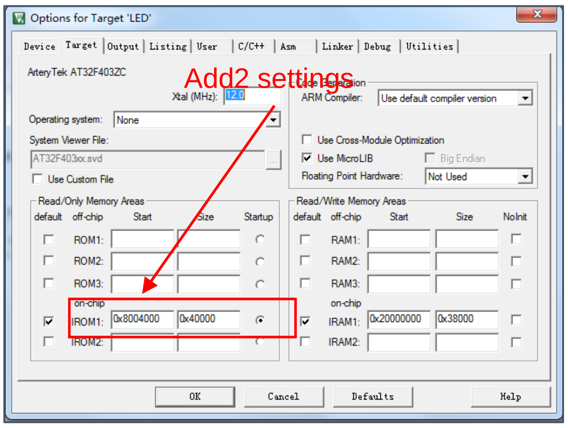
```
#define FLASH_APP_ADDRESS                    0x08004000
```

3.3 APP settings

This demo offers two APP programs for testing, both with add2 (0x800 4000) as the start address. LED3 blinks for App1, and LED4 blinks for App2. Taking App LED3 as an example, proceed as follows:

1. Keil settings

Figure 5. App project add2 settings in Keil


Options for Target 'LED'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Add2 settings
OK | Cancel | Defaults | Help

ArteryTek AT32F403ZC

Xtal (MHz): 12.0

Operating system: None

System Viewer File: AT32F403xx.svd

☐ Use Custom File

Code Generation — ARM Compiler: Use default compiler version

☐ Use Cross-Module Optimization

☑ Use MicroLIB ☐ Big Endian

Floating Point Hardware: Not Used

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x8004000	0x40000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x20000000	0x38000	☐
☐	IRAM2:			☐

2. App source code settings

Modify the interrupt vector offset in main.c

Figure 6. App settings

```
/* config vector table offset */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x4000);
```

Offset setting of interrupt vector table

3. Compile and generate bin files

Call the fromelf.exe after compiling through User option, and generate .bin files according to .axf file for IAP update.

Going through the above 3 steps, we can get an APP program in the .bin format, and perform update through bootloader.

4. Enable debug app code function

If you need to debug the app project separately when designing the app code, please proceed as follows:

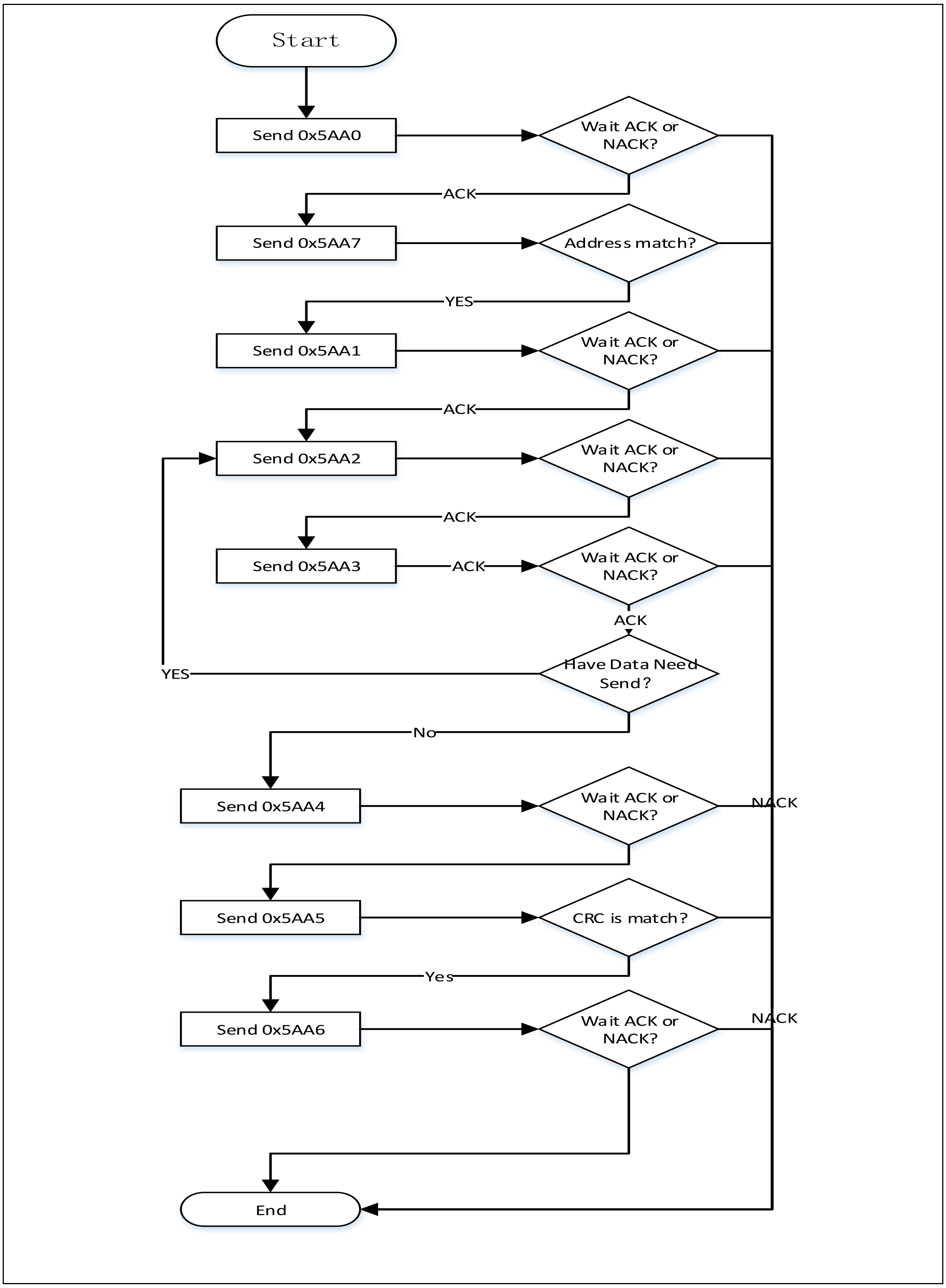
- Download bootloader project;
- The first use of debug function requires the IAP Programmer .exe to download the APP successfully once (FLAG will be written after successful download, indicating that it starts from APP nest time. The FLAG starts from bootloader, by default);

- Debug app project.

3.4 IAP/APP and host communication flowchart

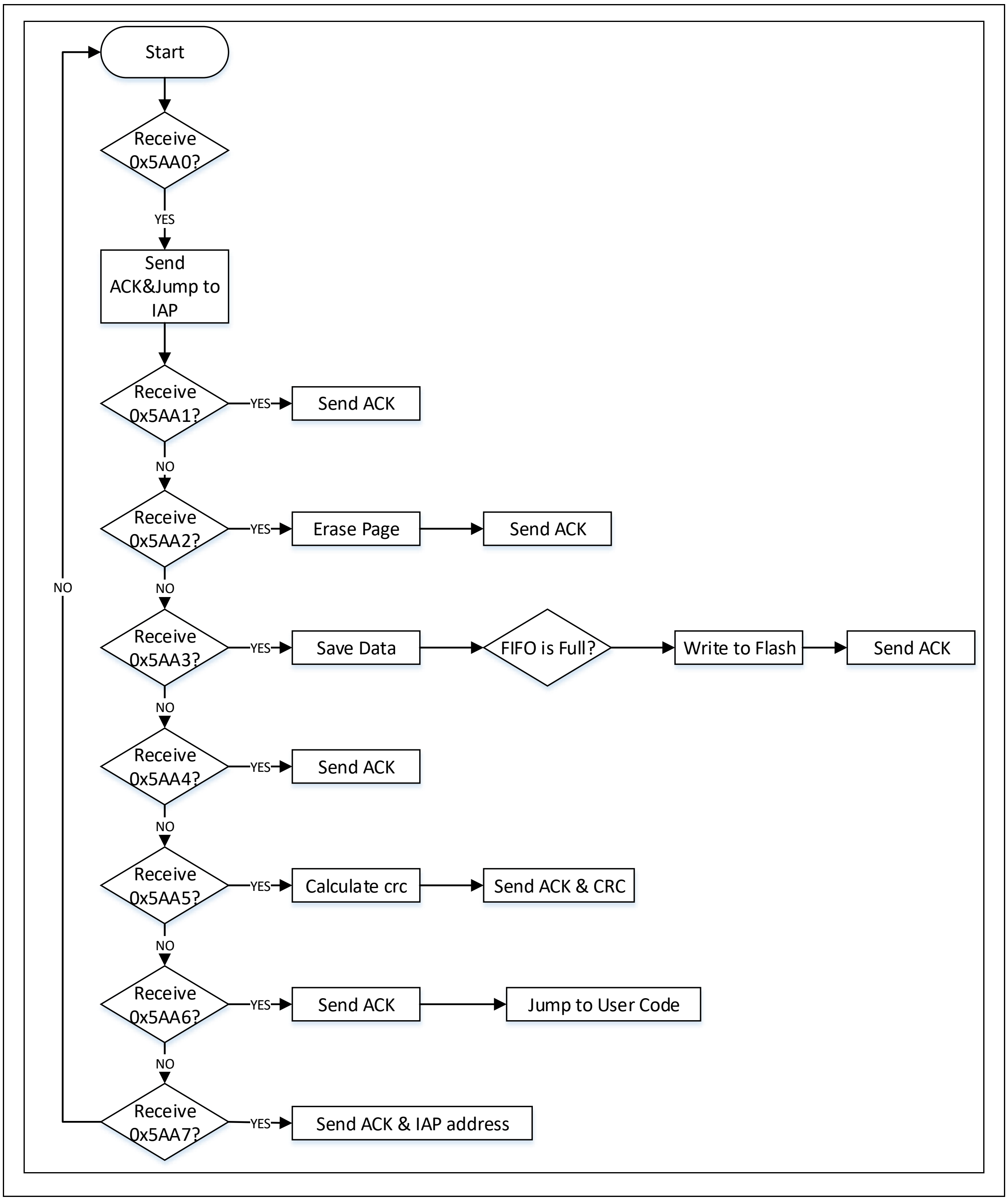
1. Host computer communication flowchart

Figure 7. Host computer communication flowchart



2. IAP slave computer communication flowchart

Figure 8. IAP slave computer communication flowchart



Note: For detailed information on protocol, please refer to AT32_HID_IAP_Protocol.pdf.

4 USB HID IAP Protocol

This section introduces USB HID upgrade protocol and how to use this protocol to communicate with the host computer to upgrade firmware.

4.1 List of commands

Value	Description
0x5AA0	Enter IAP mode
0x5AA1	Start download
0x5AA2	Set address (aligned by 1 K)
0x5AA3	Download data command
0x5AA4	Download completed
0x5AA5	CRC verify
0x5AA6	Jump command (jump to user code)
0x5AA7	Get User Code address in IAP settings

Note 1: HID MaxPacket = 64 byte;
Note 2: The first two bytes of each packet are fixed as commands;
Note 3: Commands are sent in MSB, LSB order.
ACK: 0XFF00, NACK: 0x00FF

4.2 Command descriptions

1. 0x5AA0: Enter IAP mode

It is a specific command. When the user APP receives this command, it enters IAP mode. This is implemented by clearing the flag and performing a reset after this command is received.

Host computer: [0x5A, 0xA0]

IAP device response: [0x5A, 0XA0, ACK/NACK]

2. 0x5AA1: Start download

Host computer: [0x5A,0xA1]

IAP device response: [0x5A,0xA1,ACK/NACK]

3. 0x5AA2: Set download address

Set download address, aligned by 1 KB. You need to reset the download address every time 1 KB data is downloaded.

Host computer (command+address): [0x5A, 0xA2, 0x08, 0x00, 0x40, 0x00]

IAP host response: [0x5A,0xA2, ACK/NACK]

4. 0x5AA3: Download data command (aligned by 1 KB, send multiple packets)

The download data command is sent in the format of “command+length+data”, and the maximum data size of each packet is 60 bytes (64-command-length). When the data sent reaches 1 KB, the host computer needs to wait for the ACK response of the device. At this point, the device needs to write 1 KB data to the FLASH.

Host computer (command (2 Byte) + length (2 Byte)+data (n byte)): [0x5A,0xA3,LEN1, LEN0,DATA0….DATAn]

IAP device response after receiving 1 KB data: [0x5A, 0XA3, ACK/NACK]

5. 0x5AA4: Download completed

   Host computer: [0x5A, 0xA4]

   IAP device response: [0x5A, 0xA4, ACK/NACK]

6. 0x5AA5: Firmware CRC verify

   The host computer transmits the firmware start address and firmware size/1 KB (firmware size aligned by 1KB; add 0xFF in case of insufficiency); then IAP calculates CRC and then sent back to the host computer.

   Host computer: [0x5A,0xA5, 0x08, 0x00, 0x40, 0x00, LEN1, LEN0]

   IAP device response: [0x5A, 0xA5, ACK/NACK, CRC3, CRC2, CRC1, CRC0]

7. 0x5AA6: Jump command

   Jump to user code to run.

   Host computer: [0x5A,0xA6, 0x08, 0x00, 0x40, 0x00]

   IAP device response: [0x5A,0xA6,ACK/NACK]

8. 0x5AA7: Get app address in IAP settings

   Return to the app address in IAP settings.

   Host computer: [0x5A, 0xA7]

   IAP device response: [0x5A, 0xA7, ACK/NACK, 0x08, 0x00, 0x40, 0x00]

5 Revision history

Table 2. Document revision history

Date	Version	Revision note
2021.09.02	2.0.0	Initial release