

## Application note

---

# AN\_Two-level BOOT scheme based on CAN interface

---

## Introduction

During the development of embedded products, users will develop their own secondary BOOT code (indicated by IAP later) to upgrade the application program (indicated by APP later), and there are many peripheral interfaces for upgrading, such as I2C, USART, CAN, SPI, ETH, USB, etc.

This document is mainly aimed at the above-mentioned application scenarios of National Technology MCU series products, and guides users how to use IAP examples to realize the function of IAP upgrade APP through CAN peripheral interface.

This document is only suitable for National Technology MCU products with CAN peripheral interface. The currently supported product series are N32G43x series, N32L43x series, and N32L40x series products.

## Contents

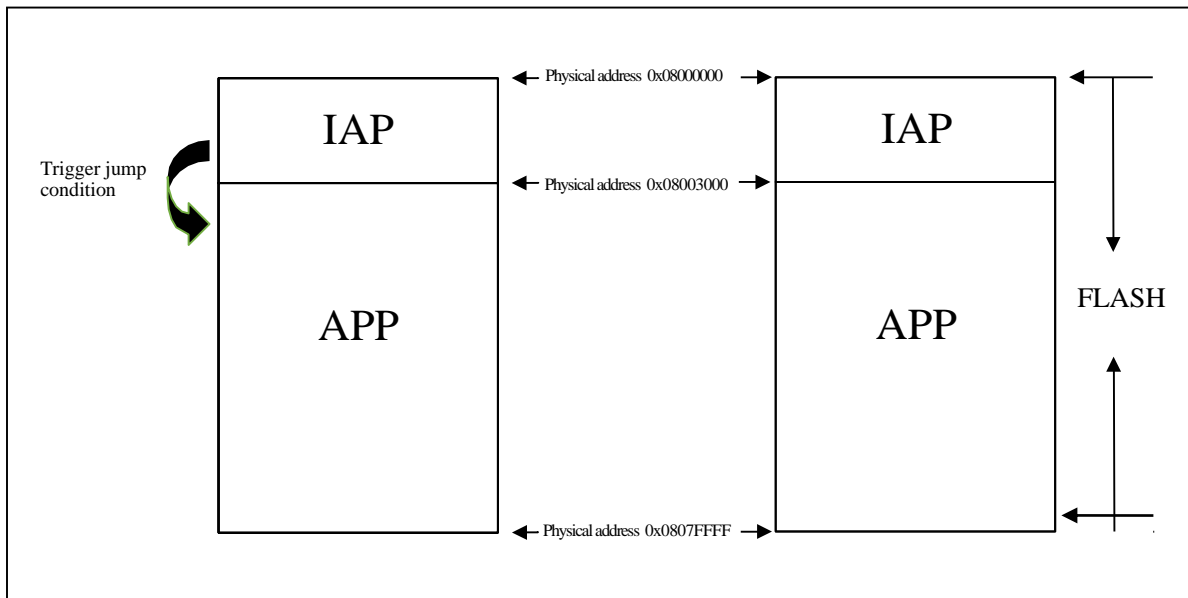
<b>1. IAP upgrade APP implementation mechanism .....</b>	<b>2</b>
<b>2. Function description of IAP example based on CAN peripheral interface.....</b>	<b>3</b>
2.1 Trigger condition judgment .....	3
2.2 IAP Upgrade Mode.....	5
<b>3. Upgrade Instructions .....</b>	<b>6</b>
3.1 Instructions and data structures.....	6
3.2 Instructions description.....	7
<b>4. Demo IAP engineering.....</b>	<b>14</b>
4.1 Hardware connection .....	14
<b>5. Conclusion.....</b>	<b>17</b>
<b>6. Historic version .....</b>	<b>18</b>
<b>7. Notice .....</b>	<b>19</b>

## 1. IAP upgrade APP implementation mechanism

When users develop some products, they need to upgrade the APP software code for the subsequent use of the products. In this case, IAP code should be pre-built in the FLASH of MCU, and the APP should be downloaded and updated through the reserved peripheral interface.

IAP codes are generally stored in the first address segment of FLASH in MCU, while APP codes are stored after IAP, as shown in Figure 1-1.

Figure 1-1 Store addresses of IAP and APP and redirect diagram



Chip performance is usually divided into two modes:

1. In APP application mode, IAP code runs after the chip is powered on, and jumps to APP execution code after the jump condition is triggered.
2. IAP upgrade mode, after the chip is powered on, the IAP code is run, and the jump condition is not triggered, the IAP code is executed for APP upgrade

## **2. Function description of IAP example based on CAN peripheral interface**

Open the corresponding KEIL project under the SDK example directory (Nationstech.n32g43x\_Library.1.2.0\projects\n32g43x\_EVAL\Applications\IAP\CAN).

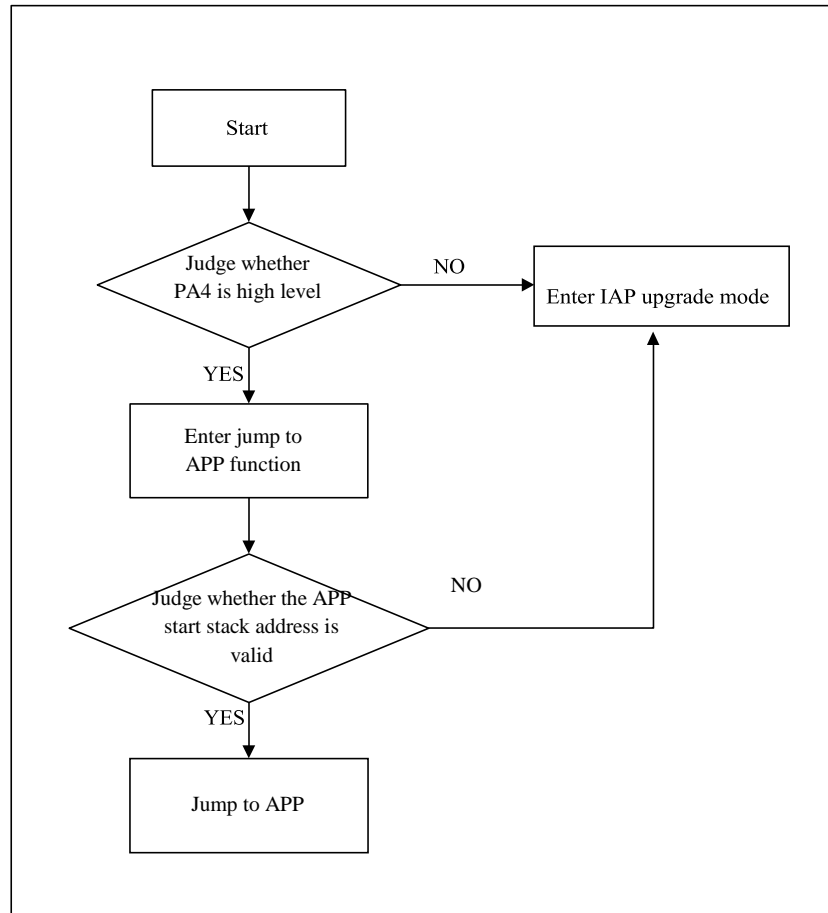
The IAP routine functionality is divided into two main parts:

1. Judge whether the trigger condition of jumping to APP is effective, and control whether the program jumps to APP according to this;
2. Enter IAP upgrade mode, receive instructions from upper computer (or other chips) through CAN peripheral interface and make corresponding reply;

### **2.1 Trigger condition judgment**

IAP routines are triggered by the level state of GPIO PA4 (this can be done by converting the PA4 of the N32G43x series minimum system development board N32G43XR-STB connect to high level or low level). Figure 2-1 below is the schematic diagram of program flow at different level states of PA4.

Figure 2-1 Schematic diagram of the program flow in different levels of PA4

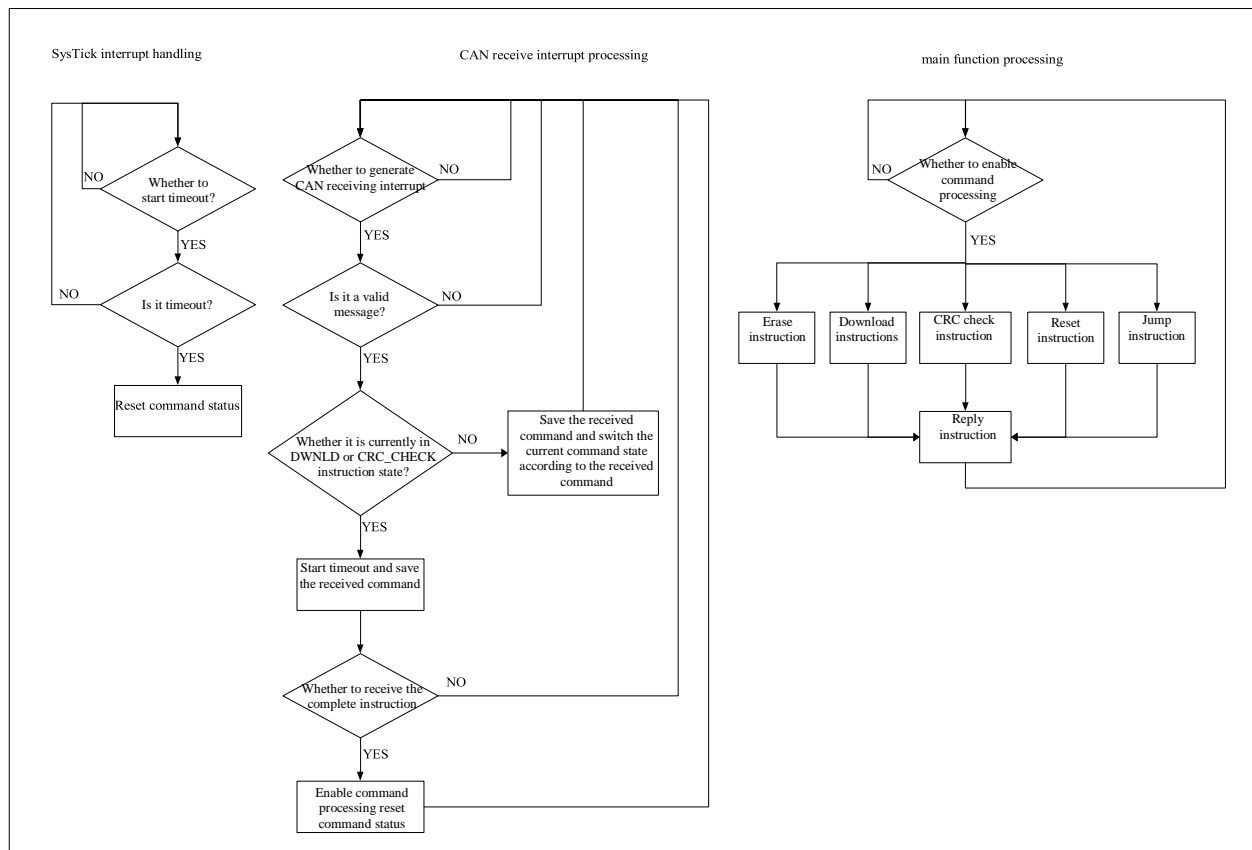


## 2.2 IAP Upgrade Mode

After entering the IAP upgrade mode, the upgrade instructions and data are received through the CAN peripheral interface, for details of the upgrade instructions, please refer to the upgrade instructions in Chapter 3. Since one CAN message contains up to 8 bytes of valid data, some commands need to receive multiple CAN messages to form a complete command.

Figure 2-2 describes the upgrade instruction processing process in IAP upgrade mode:

Figure 2-2 Schematic diagram of upgrade instruction processing flow



## 3. Upgrade Instructions

All instruction data structures and meanings are described in detail below.

### 3.1 Instructions and data structures

#### 3.1.1 Instruction list

Table 3-1 Command list

Instruction name	Level 1 instruction field	Level 2 instruction field	Instruction description
CMD_FLASH_ERASE	0x10	0x00	Erase the specified range FLASH
CMD_FLASH_DWNLD	0x11	0x00	Download APP to FLASH
CMD_DATA_CRC_CHECK	0x12	0x00	CRC check the downloaded APP
CMD_SYS_RESET	0x13	0x00	Reset system
CMD_APP_GO	0x14	0x00	Jump to APP run

#### 3.1.2 Instruction data structure

Here are some conventions in the following description, where "<>" represents a field that must be included, and "()" represents a data field included according to different instructions.

1. Receiving instruction data structure:

<CMD\_H + CMD\_L + LEN + Par[0] + Par[1] + Par[2] + Par[3]> + (DAT).

CMD\_H represents the first-level instruction field, CMD\_L represents the second-level instruction field; LEN represents the received data length (that is, the byte length of DAT); Par[0]~ Par[3] represents 4 bytes of instruction parameters; DAT represents the specific received data;

2. Reply instruction data structure:

<CMD\_H + CMD\_L + LEN + Par[0] + Par[1] + Par[2] + Par[3]> + (DAT).

CMD\_H represents the first-level instruction field, CMD\_L represents the second-level instruction field, and the

response first and second-level instruction fields are the same as the corresponding received ones; LEN represents the length of the sent data (that is, the byte length of DAT); DAT represents the specific data of the response instruction; The two bytes of Par[0]~ Par[1] represent the execution result of the instruction returned to the upper layer, and the two bytes of Par[2]~ Par[3] remain as reserved 0; if the received first level instruction, second level The instruction field does not belong to any instruction in the instruction list, reply Par[0] =0xBB, Par[1] = 0xCC.

## 3.2 Instructions description

### 3.2.1 CMD\_FLASH\_ERASE

The function of erasing FLASH in units of pages. The erasure page address number and page number are provided by the user. The erased FLASH space cannot exceed the entire FLASH space, and at least one page is erased.

#### Receiving instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 instruction field:0x10							
1(CMD_L)	Level 2 instruction field: reserve							
2~3(LEN)	Send data length: reserved							
4~7(Par)	Page address number 2 bytes: 0~255 Number of pages 2 bytes: 1~256							
(DAT)	nothing							

- LEN Send data length: 0x00(LEN[0]), 0x00(LEN[1]),  $LEN = LEN[0] + (LEN[1] \ll 8)$ .
- The erase address and range consist of four bytes in the Par field:

Par[0~1] : Page Address number 2 bytes (0~255) Page address number =  $Par[0] + Par[1] \ll 8$ ;

Par[2~3] : page number 2 bytes (1~256) Page number =  $Par[2] + Par[3] \ll 8$ ;

The first address of page 0 is 0x0800\_3000. The number of subsequent pages is increased by 1, and the first address is added to 0x800. Such as:

The address at the beginning of page 1 is  $0x0800\_3000 + 1 * 0x800 = 0x0800\_3800$

The address at the top of page 2 is  $0x0800\_3000 + 2 * 0x800 = 0x0800\_4000$



The entire address range erased

For example, the page address is 0x01 and the number of pages is 0x02

Erasing address range:

$(0x0800\_3000 + 1*0x800) \sim (0x0800\_3000 + 1*0x800 + 2*0x800)$

That is, (first address of the page number) to (First address of the page number + number of pages x page size)

- Reserved value: 0x00;

#### Reply instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x10							
1(CMD_L)	Second-level command field: Reserved							
2~3(LEN)	Length of sent data: Reserved							
4~7(Par)	Par[0] : status byte 1 Par[1] : status byte 2 Par[2] ~ Par[3] : reserved							
(DAT)	nothing							

- LEN Send data length: 0x00(LEN[0]), 0x00(LEN[1]),  $LEN = LEN[0] + (LEN[1] \ll 8)$ .
- Status bytes (Par[0], Par[1]) are divided according to command execution:
  1. Return success: status flag bits (0xA0, 0xB0).
  2. Return failure: status flag bits (Par[0], Par[1])
    - (1) (0xE0, 0x10) : Failed to erase the FLASH.
    - (2) (0xE0, 0x11) : Erases a FLASH address that exceeds the entire FLASH size.

### 3.2.2 CMD\_FLASH\_DWNLD

This command allows the user to download code into the specified FLASH. The received data length must be 4 bytes aligned.

#### Receiving instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x11							
1(CMD_L)	Second-level command field: Reserved							
2~3(LEN)	Length of received data: N							
4~7(Par)	Start address for downloading the FLASH							
8~8+(N-1)(DAT)	DAT[0 to N-1] : specific data to be downloaded							

- LEN receive data length : 0xxx (LEN[0]), 0xxx(LEN[1]),  $LEN = LEN[0] + (LEN[1] \ll 8)$
- Par [0 ~3] : download the starting Address of the FLASH, synthetic rules: Address = Par [0] | (Par [1] < 8) | (Par [2] < 16) | (Par [3] < 24).
- DAT[0 to N-1] : Indicates the specific data to be downloaded. The maximum value is 256 bytes,  $4 \leq N \leq 256$ , the value of N must be a multiple of 4.
- Reserved value:0x00;

#### Reply instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x11							
1(CMD_L)	Second-level command field: Reserved							
2~3(LEN)	Length of sent data: Reserved							
4~7(Par)	Par[0] : status byte 1 Par[1] : status bytes 2 Par[2]~ Par[3] : reserved							
(DAT)	nothing							

- LEN Send data length: 0x00(LEN[0]), 0x00(LEN[1]),  $LEN = LEN[0] + (LEN[1] \ll 8)$
- Status bytes (Par[0], Par[1]) are divided according to command execution:
  1. Download success: status flag bits (0xA0, 0xB0).
  2. Download failed: status flag bits (Par[0], Par[1])
    - (1) (0xE0, 0x10) : Failed to download the FLASH.
    - (2) (0xE0, 0x11) : The downloaded FLASH address exceeds the size of the entire FLASH.

(3) (0xE0, 0x12) : The start address of the downloaded FLASH is not 4-byte aligned;

(4) (0xE0, 0x13) : The downloaded FLASH data length is not a multiple of 4 or greater than 256 bytes.

### 3.2.3 CMD\_DATA\_CRC\_CHECK

This command is used to check whether the downloaded data is correct. After the downloaded data is complete, the CRC32 check is performed. The receiving command must contain the CRC for downloaded data value and checksum start address and checksum length.

#### Receiving instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x12							
1(CMD_L)	Second-level command field: Reserved							
2~3(LEN)	Length of sent data: 8							
4~7(Par)	Check start address							
8~15(DAT)	DAT[0:3] : indicates the 32-bit CRC check value DAT[4:7] : Parity length (in bytes, must be a multiple of 4)							

- LEN Send data length: 0x08(LEN[0]), 0x00(LEN[1]),  $LEN = LEN[0] + (LEN[1] \ll 8)$ ;
- Par [0 ~ 3] : check the starting address, synthetic rules:  $Address = Par[0] \mid (Par[1] \ll 8) \mid (Par[2] \ll 16) \mid (Par[3] \ll 24)$ , the Address is only within the scope of the FLASH;
- DAT [0 ~ 3] : 32 bit CRC checksum value, synthetic rules for CRC32 =  $DAT[0] \mid (DAT[1] \ll 8) \mid (DAT[2] \ll 16) \mid (DAT[3] \ll 24)$ ;
- DAT (4 ~ 7) : check length, its synthesis rules for CRC\_LEN =  $DAT[4] \mid (DAT[5] \ll 8) \mid (DAT[6] \ll 16) \mid (DAT[7] \ll 24)$ , CRC\_LEN can only be in the valid range and must be a multiple of 4.
- Reserved value: 0x00;

#### Reply instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x12							
1(CMD_L)	Second-level command field: Reserved							

2~3(LEN)	Length of sent data: Reserved
4~7(Par)	Par[0] : status byte 1 Par[1] : status bytes 2 Par[2]~ Par[3] : reserved
(DAT)	nothing

- LEN Send data length: 0x00(LEN[0]), 0x00(LEN[1]),  $LEN = LEN[0] + (LEN[1] \ll 8)$ ;
- Status bytes (Par[0], Par[1]) are divided according to command execution:
  1. Verification succeeded: status flag bits (0xA0, 0xB0);
  2. Verification failed: status flag bits (Par[0], Par[1])
    - (1) (0xE0, 0x10) : CRC verification failed.
    - (2) (0xE0, 0x11) : The CRC check address exceeds the size of the entire FLASH.
    - (3) (0xE0, 0x13) : The CRC check length is not a multiple of 4.

### 3.2.4 CMD\_SYS\_RESET

This directive is used by software to reset IAP programs.

#### Receiving instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x13							
1(CMD_L)	Second-level command field: Reserved							
2~3(LEN)	Length of sent data: Reserved							
4~7(Par)	keep							
(DAT)	nothing							

- Reserved value: 0x00;

#### Reply instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x13							
1(CMD_L)	Second-level command field: Reserved							

2~3(LEN)	Length of sent data: Reserved
4~7(Par)	Par[0] : status byte 1 Par[1] : status byte 2
(DAT)	nothing

- Status bytes (Par[0], Par[1]) are divided according to command execution:

1. Return success: status flag bits (0xA0, 0xB0)
2. Return failed: status flag bit (0xE0, 0x10).

### 3.2.5 CMD\_APP\_GO

After downloading the application to the FLASH, jump to the APP program for execution.

#### Receiving instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x14							
1(CMD_L)	Second-level command field: Reserved							
2~3(LEN)	Length of sent data: Reserved							
4~7(Par)	Reserved							
(DAT)	nothing							

- Reserved value: 0x00;

#### Reply instruction:

byte \ bit	b7	b6	b5	b4	b3	b2	b1	b0
0(CMD_H)	Level 1 command field: 0x14							
1(CMD_L)	Second-level command field: Reserved							
2~3 (LEN)	Length of sent data: Reserved							
4~7(Par)	Par[0] : status byte 1 Par[1] : status byte 2							
(DAT)	nothing							

- Status bytes (Par[0], Par[1]) are divided according to command execution:

1. Return success: status flag bits (0xA0, 0xB0);

2. Return failed: status flag bit (0xE0, 0x10).

## 4. Demo IAP engineering

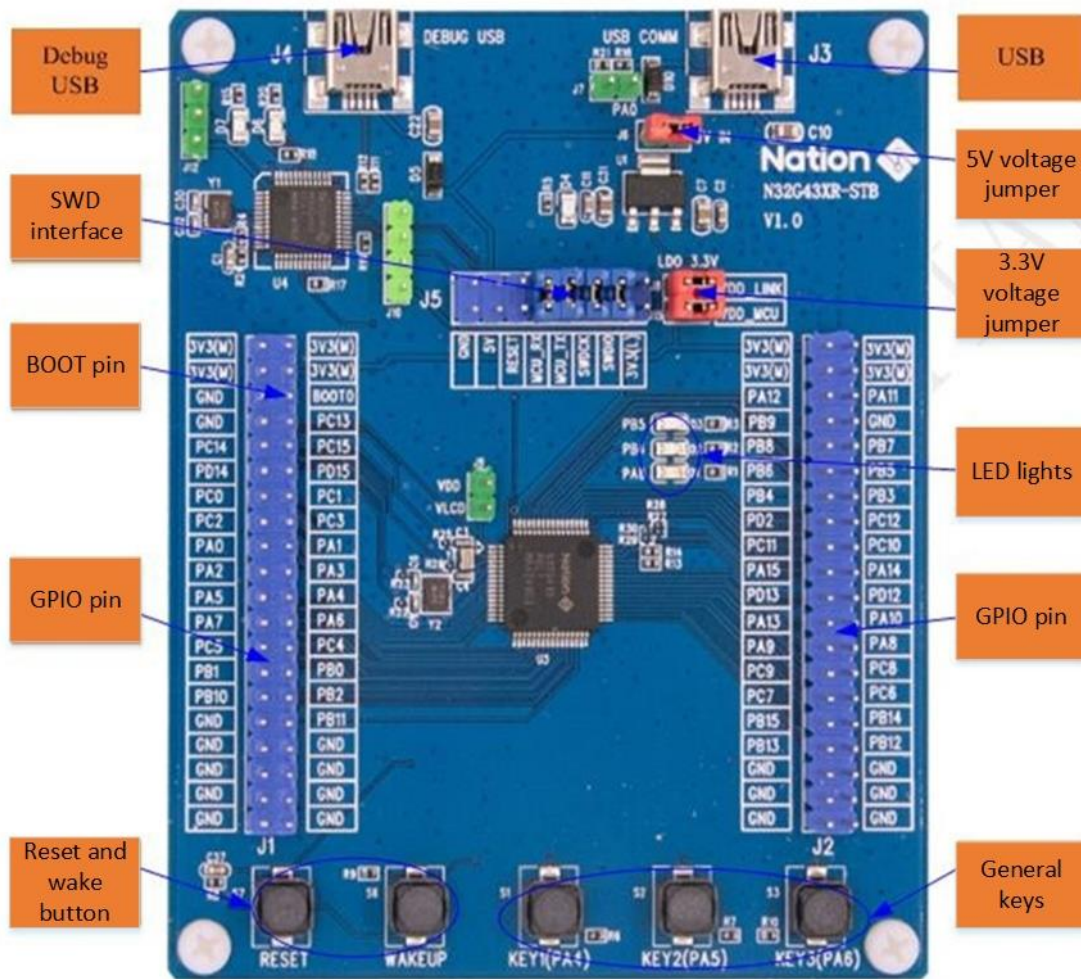
The following sections focus on how to use IAP engineering.

### 4.1 Hardware connection

#### 4.1.1 Development board

Choose N32G43x series minimum system development board N32G43XR-STB V1.0, as shown in Figure 4-1 N32G43XR-STB V1.0 minimum system development board. Please refer to the document "UG\_N32G43XR-STB Development Board Hardware Usage Guide V1.0" for the usage guide of the development board.

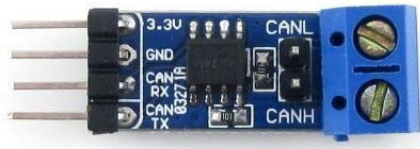
Figure 4-1 N32G43XR-STB V1.0 minimum system development board



CAN peripheral interfaces are PD0 and PD1 corresponding to CAN1, and baud rate is 500Kbps.

SN65HVD230 is selected as the CAN bus sending receiver, as shown in the development board of CAN bus sending receiver in Figure 4-2 below.

Figure 4-2 CAN Bus Transmitter and Receiver Development Board



Use CAN protocol analyzer CANNalyst-II to send CAN commands, as shown in Figure 4-3 CAN protocol analyzer.

Figure 4-3 CAN protocol analyzer



## 4.1.2 Software testing

Download IAP program to N32G43XR-STB V1.0 minimum system development board, reset and run. Open the host computer software of the CAN protocol analyzer CANNalyst-II and configure the baud rate to 500Kbps. Send the command and wait for a reply, as shown in Figure 4-4.



Figure 4-4 CAN bus transmit receiver development board

	Index	System Time	Time Stamp	Channel	Direction	Frame ID	Type	Format	DLC	Data
Unsupported instruction	00000	10:34:15.168	-	chl	Send	0x0000	Data	Standard	0x08	x  00 00 00 00 00 00 00 00
	00001	10:34:15.173	0x1230093	chl	Receive	0x0400	Data	Standard	0x08	x  00 00 08 00 BB CC 00 00
Erase instruction	00002	10:34:32.924	-	chl	Send	0x0400	Data	Standard	0x08	x  10 00 00 00 00 00 FA 00
	00003	10:34:32.994	0x125B71E	chl	Receive	0x0400	Data	Standard	0x08	x  10 00 08 00 A0 B0 00 00
Download instructions	00004	10:35:19.632	-	chl	Send	0x0400	Data	Standard	0x08	x  11 00 00 01 00 30 00 08
	00005	10:35:31.554	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00006	10:35:31.554	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00007	10:35:31.565	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00008	10:35:31.575	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00009	10:35:31.585	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00010	10:35:31.596	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00011	10:35:31.607	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00012	10:35:31.616	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00013	10:35:31.625	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00014	10:35:31.635	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00015	10:35:31.645	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00016	10:35:31.655	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00017	10:35:31.665	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00018	10:35:31.675	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00019	10:35:31.686	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00020	10:35:31.695	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00021	10:35:31.706	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00022	10:35:31.717	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00023	10:35:31.725	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00024	10:35:31.734	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00025	10:35:31.745	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00026	10:35:31.754	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00027	10:35:31.765	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00028	10:35:31.777	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00029	10:35:31.787	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00030	10:35:31.795	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00031	10:35:31.805	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00032	10:35:31.817	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00033	10:35:31.826	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00034	10:35:31.835	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00035	10:35:31.845	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00036	10:35:31.856	-	chl	Send	0x0400	Data	Standard	0x08	x  11 22 33 44 55 66 77 88
	00037	10:35:31.884	0x12EB0B7	chl	Receive	0x0400	Data	Standard	0x08	x  11 00 08 00 A0 B0 00 00
CRC check instruction	00038	10:35:49.412	-	chl	Send	0x0400	Data	Standard	0x08	x  12 00 08 00 00 30 00 08
	00039	10:35:58.636	-	chl	Send	0x0400	Data	Standard	0x08	x  14 01 8D 18 00 01 00 00
	00040	10:35:58.644	0x132C543	chl	Receive	0x0400	Data	Standard	0x08	x  12 00 08 00 A0 B0 00 00
Reset command	00041	10:36:11.377	-	chl	Send	0x0400	Data	Standard	0x08	x  13 00 00 00 00 00 00 00
	00042	10:36:11.394	0x134B658	chl	Receive	0x0400	Data	Standard	0x08	x  13 00 08 00 A0 B0 00 00
Jump instruction	00043	10:36:25.513	-	chl	Send	0x0400	Data	Standard	0x08	x  14 00 00 00 00 00 00 00
	00044	10:36:25.524	0x136DDE1	chl	Receive	0x0400	Data	Standard	0x08	x  14 00 08 00 A0 B0 00 00

## 5. Conclusion

The IAP routine implements a simple upgrade of the APP's secondary BOOT based on the CAN peripheral interface. Users can add instructions or data encryption and decryption functions on this basis.

In addition to the CAN peripheral-based interface, there are IAP sample codes based on USART, USB, I2C, SPI and other peripheral-based interfaces, using the same instruction set.

## 6. Historic version

Version	Date	Remark
V1.0	2020.10.10	New document

## 7. Notice

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 NSING Technologies Inc. and NSING 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 NSING has attempted to provide accurate and reliable information, NSING 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 NSING 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. NSING 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 NSING and hold NSING 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 NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.