AN0008

Application Note

Jump to Boot Memory

Introduction

There is a boot memory in the memory map of AT32F4xx, which stores the bootloader. To execute the bootloader, the user must configure it through BOOT pin, generally pulling up BOOT0 and pulling down BOOT1. In actual application, the BOOT pin may not be connected, and it is not possible to enter bootloader by switching BOOT pin. Therefore, this application note is written to introduce how to directly jump from user code to bootloader.

Note: The corresponding code in this application note is developed on the basis of V2.x.x BSP provided by Artery. For other versions of BSP, please pay attention to the differences in usage.

Applicable products:

Part number	AT32Fxx

Contents

List of Tables

List of Figures

1 Software implementation

1.1 Preconditions for jumping to bootloader

Before jumping from user code to boot memory to execute the bootloader, the user must complete the following operations:

1) Disable all peripheral clocks;
2) Disable PLL;
3) Disable all interrupts;
4) Clear all pending interrupt flags.

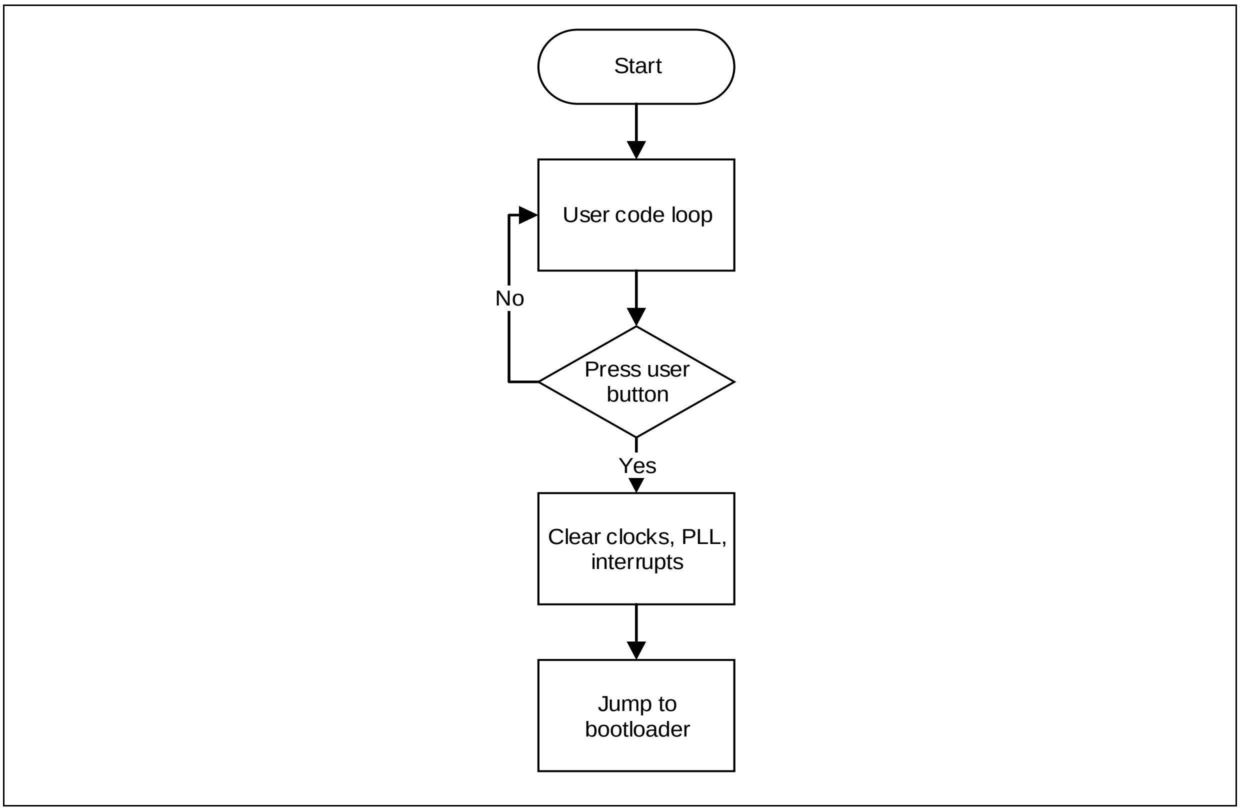
Jumping to bootloader can be implemented by the following two methods:

1) Method 1: Use code to complete the aforementioned four steps, and then jump to bootloader directly (the corresponding clocks, PLL and interrupts are cleared according to the peripherals enabled by customer’s code).
2) Method 2: Automatically clear by performing a reset. After reset, jump to bootloader before initialization of user code in SystemInit.

1.2 Implementation method 1

Use the AT-START evaluation board to design a user program “APPJumpToBootloaderMethod1”, which is mainly used to complete a LED operation. Then, detect a user button; when the button is pressed, jump to the bootloader directly (interrupts need to be cleared before jumping). If it is complex to clear all clocks and interrupts before jumping, the user can select method 2 to clear by performing a reset.

Figure 1. Software flowchart



1.2.1 Method 1 code implementation

The main function is mainly a LED and a detection button. Press the button if you need to enter the bootloader.

```
int main(void)
{
        uint32_t LedTimer = 0, LedTog = 0;
        system_clock_config();
        at32_board_init();
        LedTog = system_core_clock/80;
        while(1)
        {
                if(USER_BUTTON == at32_button_press())
                {
                        /*Clear Clock, PLL, Interrupt*/
                        app_clear_sys_status ();
                        app_jump_to_bootloader ();
                }
                if(LedTimer == LedTog)
                {
                        at32_led_toggle(LED4);
                        LedTimer = 0;
                }
                LedTimer ++;
        }
}
```

The *app_jump_to_bootloader* function is used for jumping to the bootloader.

```
void app_jump_to_bootloader(void)
{
       uint32_t dwStkPtr, dwJumpAddr;
       dwStkPtr = *(uint32_t *)BOOTLOADER_ADDRESS;
       dwJumpAddr = *(uint32_t *)(BOOTLOADER_ADDRESS + sizeof(uint32_t));

       /* Before jumping to the bootloader, make sure that all peripheral clocks, PLL and
interrupts are disabled, and all interrupt pending bits are cleared. */
       SET_MSP(dwStkPtr);
       pfTarget = (void (*)(void))dwJumpAddr;
       pfTarget();
}
```

The *app_clear_sys_status* function is used to disable clocks, PLL, interrupts and clear pending interrupt flags.

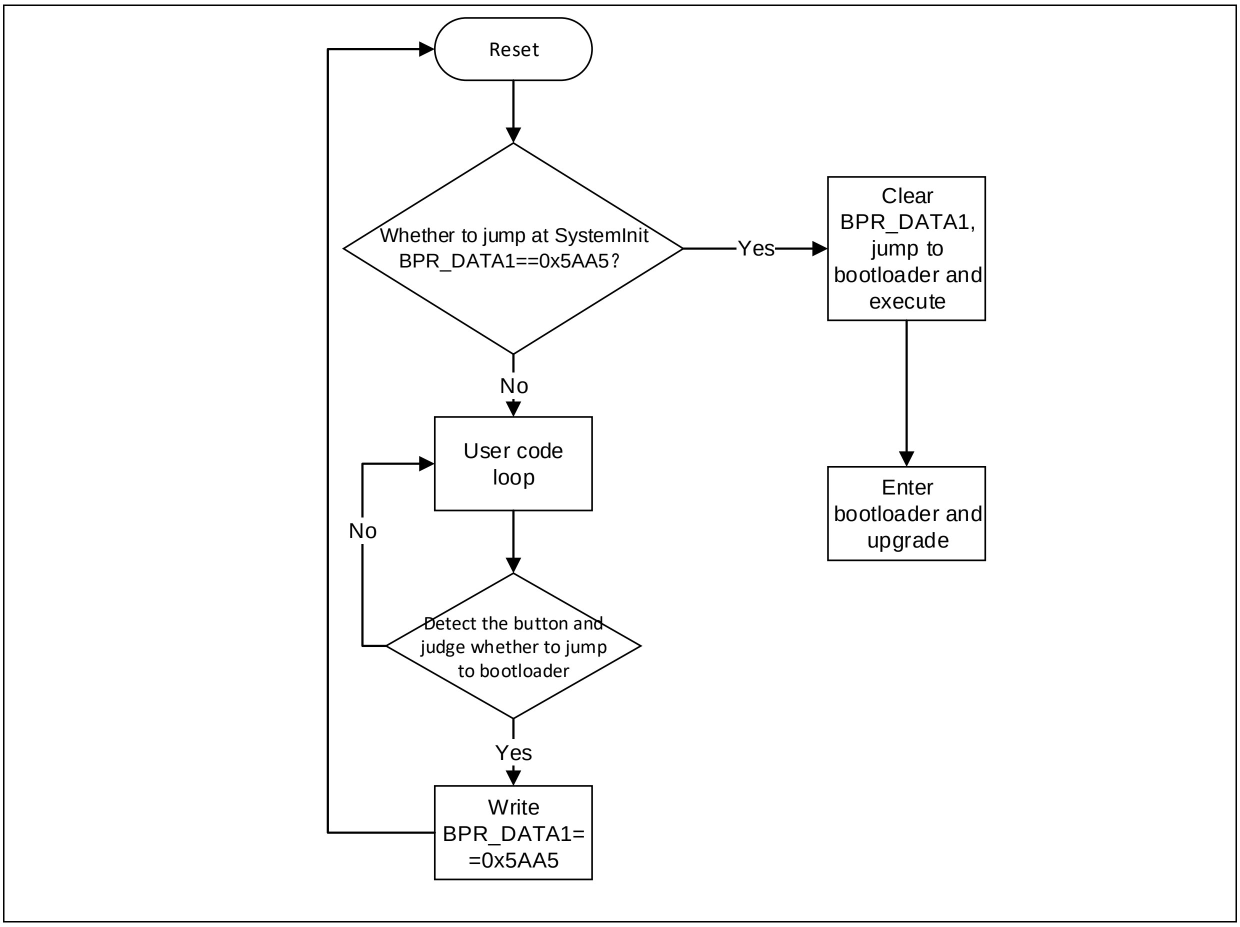
```
void app_clear_sys_status()
{
        /*Close Peripherals Clock*/
        CRM->apb2rst = 0xFFFF;
        CRM->apb2rst = 0;
        CRM->apb1rst = 0xFFFF;
        CRM->apb1rst = 0;
        CRM->apb1en = 0;
        CRM->apb2en = 0;
        /*Close PLL*/
        /* Reset SW, AHBDIV, APB1DIV, APB2DIV, ADCDIV and CLKOUT_SEL bits */
        CRM->cfg_bit.sclksel = 0;
        CRM->cfg_bit.ahbdiv = 0;
        CRM->cfg_bit.apb1div = 0;
        CRM->cfg_bit.apb2div = 0;
        CRM->cfg_bit.adcdiv_l = 0;
        CRM->cfg_bit.adcdiv_h = 0;
        CRM->cfg_bit.clkout_sel = 0;
        CRM->ctrl_bit.hexten = 0;
        CRM->ctrl_bit.cfden = 0;
        CRM->ctrl_bit.pllen = 0;
        CRM->cfg_bit.pllrcs = 0;
        CRM->cfg_bit.pllhextdiv = 0;
        CRM->cfg_bit.pllmult_l = 0;
        CRM->cfg_bit.pllmult_h = 0;
        CRM->cfg_bit.usbdiv_l = 0;
        CRM->cfg_bit.usbdiv_h = 0;
        CRM->cfg_bit.pllrange = 0;
        /* Disable all interrupts and clear pending bits    */
        CRM->clkint_bit.lickstblfc = 0;
        CRM->clkint_bit.lextstblfc = 0;
        CRM->clkint_bit.hickstblfc = 0;
        CRM->clkint_bit.hextstblfc = 0;
        CRM->clkint_bit.pllstblfc = 0;
        CRM->clkint_bit.cfdfc = 0;
        /*Close Systick*/
        SysTick->CTRL = 0;

        /*Disable ALL interrupt && Pending Interrupt Flag*/
        /*Clear interrupts and pending interrupt flags according to peripherals enabled by users*/
        /*
        user add code...
        */
}
```

1.3 Implementation method 2

Method 2 performs a reset to jump to the bootloader before initialization of user code in SystemInit.

We use the AT-START evaluation board to design a user program "APPJumpToBootloaderMethod2", which mainly completes a LED operation. Then, detect a user button; when the button is pressed, jump from the user program to the bootloader. At this point, write 0x5AA5 to the BPR_DATA1 register, and then generate a software reset. After software reset, judge whether the 0x5AA5 is written to BPR_DATA1 at the beginning of SystemInit; then, clear the BPR_DATA1. If BPR_DATA1=0x5AA5, jump to the bootloader (jump at the beginning of SystemInit, so as to prevent the user code from not matching the bootloader settings after initialization).

Figure 2. Software flowchart


Reset
Whether to jump at SystemInit
BPR_DATA1==0x5AA5?
Yes
Clear BPR_DATA1, jump to bootloader and execute
No
User code loop
No
Enter bootloader and upgrade
Detect the button and judge whether to jump to bootloader
Yes
Write BPR_DATA1= =0x5AA5

1.3.1 Method 2 code implementation

The main function is mainly a LED and a detection button. Press the button if you need to enter the bootloader.

```
int main(void)
{
        uint32_t LedTimer = 0, LedTog = 0;
        system_clock_config();
        at32_board_init();
        LedTog = system_core_clock/80;
        while(1)
        {
                if(USER_BUTTON == at32_button_press())
                {
                        /* Save a BPR status flag, indicating that APP needs to jump to bootloader */
                        BPR_Write_Flag();
                }
                if ( LedTimer == LedTog)
                {
                        at32_led_toggle(LED4);;
                        LedTimer = 0;
                }
                LedTimer ++;
        }
}
```

The *bpr_write_flag* function is mainly used to write a jump flag to BPR_DATA1 and perform software reset.

```
void bpr_write_flag (void)
{
        /* enable pwc and bpr clock */
        crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
        crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);

        /* enable write access to bpr domain */
        pwc_battery_powered_domain_access(TRUE);

        /* clear tamper pin event pending flag */
        bpr_flag_clear(BPR_TAMPER_EVENT_FLAG);

        bpr_data_write(BPR_DATA1, BKP_JUMP_FLAG);

        pwc_battery_powered_domain_access(FALSE);

        /*System Reset*/
        NVIC_SystemReset();
}
```

The *bpr_check_flag* function is mainly used to judge whether to jump to the bootloader after reset. Return 1: jump to bootloader; return 0: not to jump.

```
uint8_t bpr_check_flag (void)
{
        uint8_t ret_val = 0;
        /* enable pwc and bpr clock */
        crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
        crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);

        /* enable write access to bpr domain */
        pwc_battery_powered_domain_access(TRUE);

        /* clear tamper pin event pending flag */
        bpr_flag_clear(BPR_TAMPER_EVENT_FLAG);

        if(bpr_data_read(BPR_DATA1) == BPR_JUMP_FLAG)
        {
                bpr_data_write(BPR_DATA1, 0x00); //write 00 to bkp
                ret_val = 1;
        }

        pwc_battery_powered_domain_access(FALSE);
        crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, FALSE);
        crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, FALSE);
        return ret_val;
}
```

The *app_jump_to_bootloader* is used for jumping to the bootloader.

```
void app_jump_to_bootloader (void)
{
        uint32_t dwStkPtr, dwJumpAddr;
        dwStkPtr = *(uint32_t *)BOOTLOADER_ADDRESS;
        dwJumpAddr = *(uint32_t *)(BOOTLOADER_ADDRESS + sizeof(uint32_t));

        /* Before jumping to the bootloader, make sure that all peripheral clocks, PLL and
interrupts are disabled, and all interrupt pending bits are cleared. */
        SET_MSP(dwStkPtr);
        pfTarget = (void (*)(void))dwJumpAddr;
        pfTarget();

}
```

Before initialization of user code in SystemInit, judge whether to jump to the bootloader.

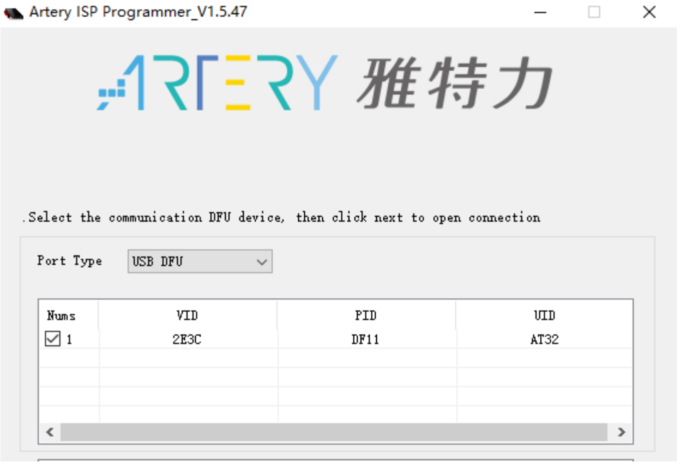
```
void SystemInit (void)
{
#if defined (__FPU_USED) && (__FPU_USED == 1U)
    SCB->CPACR |= ((3U << 10U * 2U) |                 /* set cp10 full access */
                              (3U << 11U * 2U)    );             /* set cp11 full access */
#endif

         /*check if need to go into bootloader*/
         if(bpr_check_flag () == 1)
         {
                  app_jump_to_bootloader ();
         }

    /* reset the crm clock configuration to the default reset state(for debug purpose) */
    /* set hicken bit */
    CRM->ctrl_bit.hicken = TRUE;…
}
```

2 Test of jumping to bootloader through user code

The AT-STAT-F403AV1.0 evaluation board is used for the test, and upgrade by the means of DFU (serial port upgrade process is the same).

1. Download APPJumpToBootloaderMethod1 or APPJumpToBootloaderMethod2 to the AT-START-F403A V1.0 evaluation board;
2. LED3 blinks;
3. Press PB2 USER button, and the LED3 will be OFF, indicating entering bootloader successfully;
4. Plug and unplug the USB on the AT-STAT-F403A V1.0 board;
5. Open Artery ISP Programmer, and you can find that an USB DFU device is connected.

Figure 3. ISP Programmer device connection



Artery ISP Programmer_V1.5.47

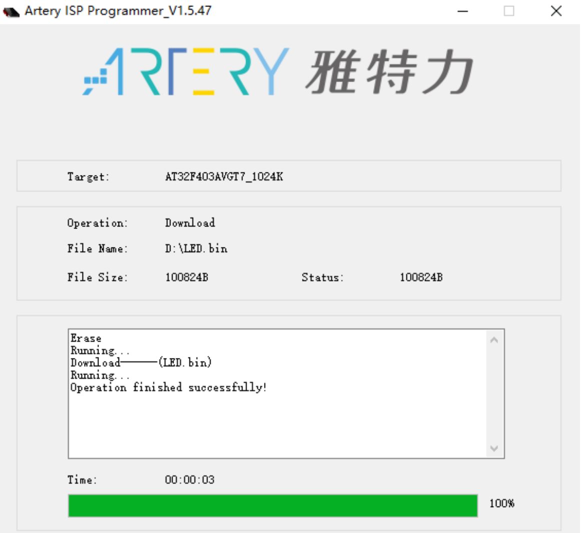
.Select the communication DFU device, then click next to open connection

Port Type: USB DFU

Nums	VID	PID	UID
☑ 1	2E3C	DF11	AT32

6. Upgrade the program according to ISP upgrade process.

Figure 4. ISP Programmer upgrade process



Artery ISP Programmer_V1.5.47

Target: AT32F403AVGT7_1024K

Operation: Download

File Name: D:\LED.bin

File Size: 100824B Status: 100824B

```
Erase
Running...
Download———(LED.bin)
Running...
Operation finished successfully!
```

Time: 00:00:03

100%

3 Revision history

Table 1. Document revision history

Date	Version	Revision note
2021.12.8	2.0.0	Initial release

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.