AN0008
应用笔记

从用户代码跳转到系统bootloader

前言

在 AT32F4xx 的 memory map 里面有一块启动程序代码区，里面存放的是系统的 Bootloader。但如果要去执行系统 Bootloader，必须要通过 BOOT Pin 去配置，通常是将 BOOT0 拉高，BOOT1 拉低的方式。在实际使用中可能没有将 BOOT Pin 接出来，此时就不能够通过切换 BOOT Pin 的方式去进入系统 Bootloader。这里就提供一种直接从用户代码直接跳转到系统 Bootloader 的方法.

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x 板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	AT32 全系列

目录

表目录

图目录

1 软件实现

1.1 跳转到 Bootloader 的前提条件

从用户代码跳转到启动程序代码区去执行Bootloader之前必须执行以下操作：

1) 关闭所有外设时钟
2) 关闭PLL的使用
3) 禁用所有的中断
4) 清除所有挂起的中断标志位

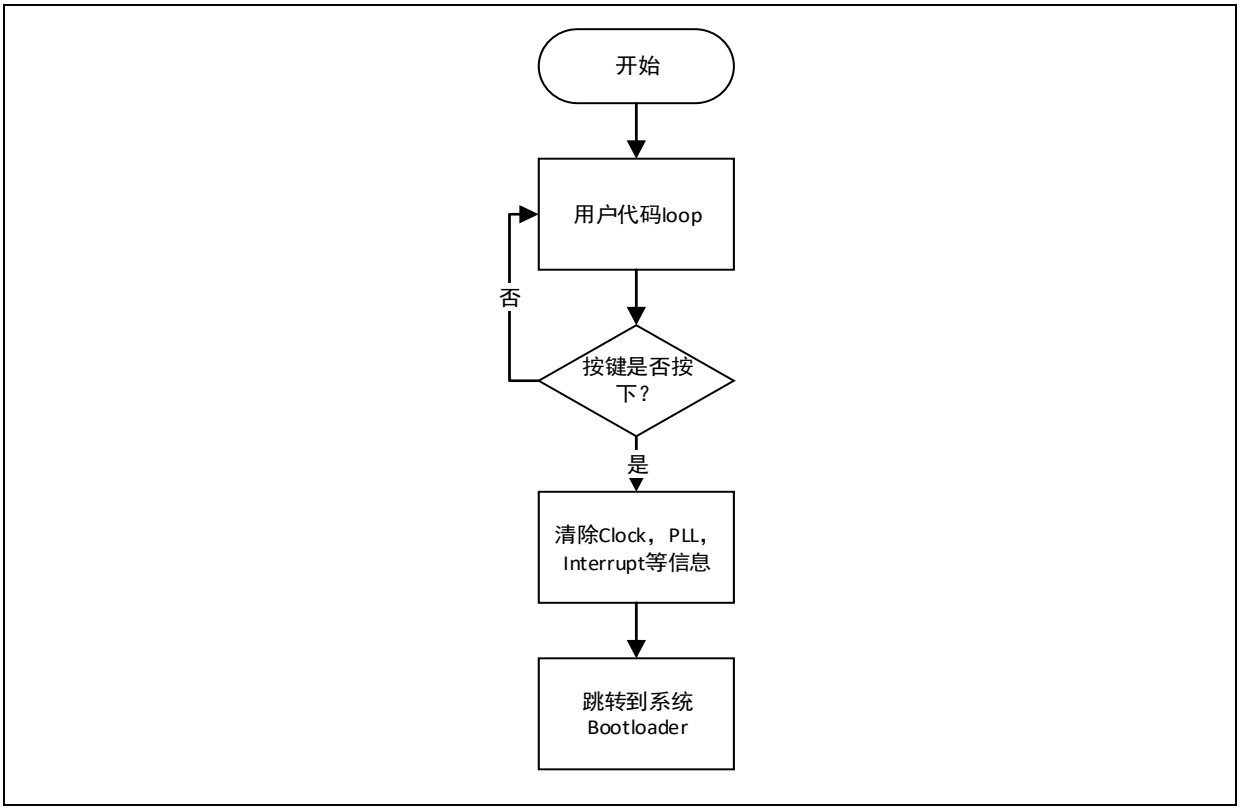
这个过程我们有两种方式去实现：

1) 方式1: 用代码清除以上4点之后直接执行跳转（这里需要根据客户代码开启的外设不同清除对应的Clock，PLL，Interrupt）
2) 方式2：用Reset的方式自动清除，Reset之后在SystemInit里面用户代码初始化之前跳转，用以保证以上4点能够满足

1.2 实现方式 1

使用AT-START开发板设计一个用户程序APPJumpToBootloaderMethod1，程序主要完成一个LED操作，另外在检测一个用户按键，当按键按下之后就直接跳转到系统Bootloader（跳转之前需清除中断等信息）,如果觉得在跳转之前要清除所有的Clock，中断等信息比较麻烦，可以参考方式2的实现，使用Reset自动清除。

图 1. 软件流程图



1.2.1 方式 1 代码实现

main函数主要是一个闪灯和检测按键，通过按键按下表示需要进入系统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())
		{
			/*清除 Clock, PLL, Interrupt*/
			app_clear_sys_status ();
			app_jump_to_bootloader ();
		}
		if(LedTimer == LedTog)
		{
			at32_led_toggle(LED4);
			LedTimer = 0;
		}
		LedTimer ++;
	}
}
```

app_jump_to_bootloader 函数负责跳转到Bootloader

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

	/*跳转之前，需要保证所有外设 Clock 关闭，PLL 关闭，关闭所有中断，清除所有的中断挂起标志*/
	SET_MSP(dwStkPtr);
	pfTarget = (void (*)(void))dwJumpAddr;
	pfTarget();
}
```

app_clear_sys_status 函数负责清除 Clock，PLL，关闭 Interrupt，和清除中断挂起标志

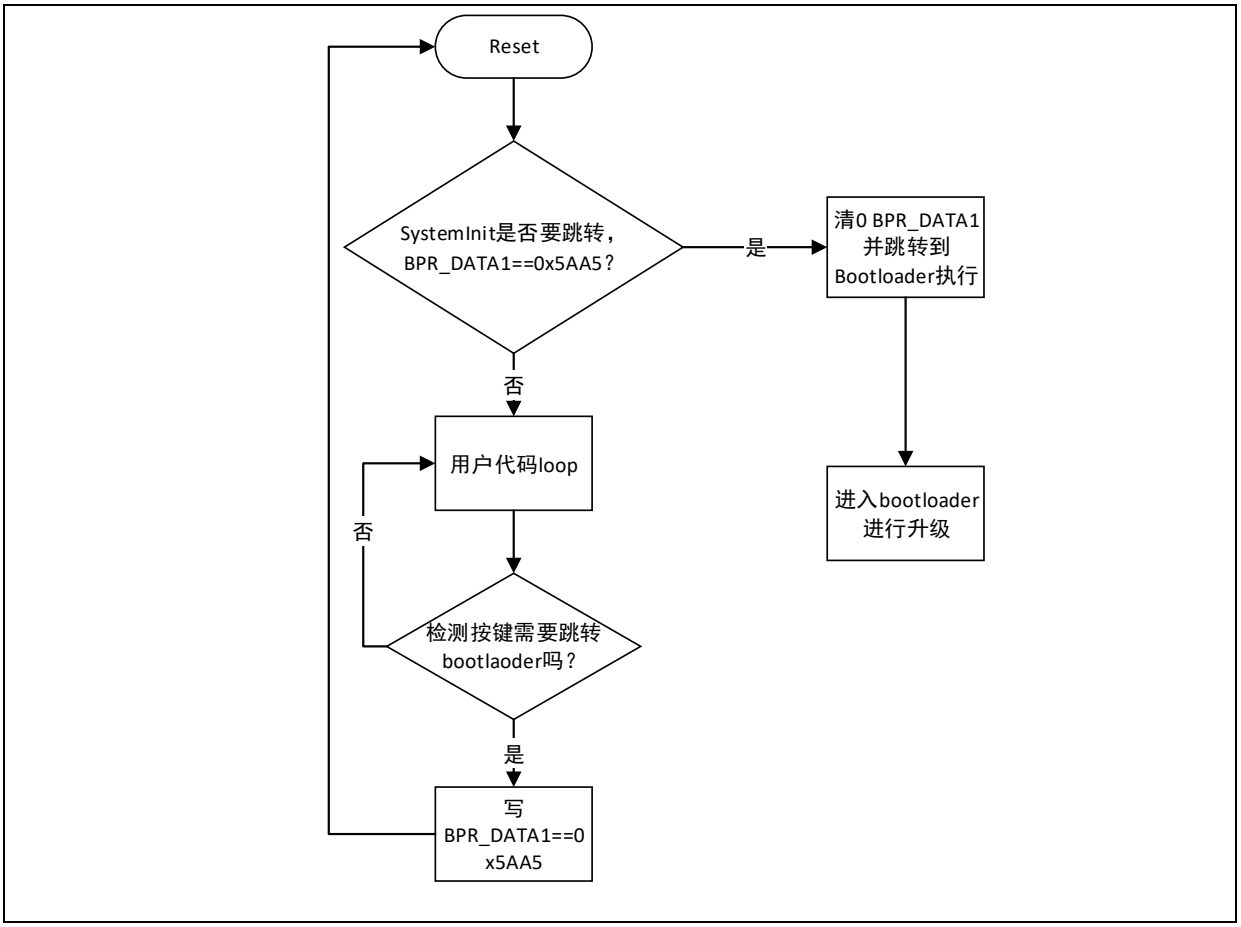
```
void app_clear_sys_status()
{
	/*Close Peripherals Clock*/
	CRM->apb2rst = 0xFFFF;
	CRM->apb2rst = 0;
	CRM->apb1rst = 0xFFFF;
	CRM->apb1rst = 0;
	CRM->apb1en = 0;
	CRM->apb2en = 0;
	/*Close PLL*/
	/* Reset SW, AHBDIV, APB1DIV, APB2DIV, ADCDIV and CLKOUT_SEL bits */
	CRM->cfg_bit.sclksel = 0;
	CRM->cfg_bit.ahbdiv = 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;
	/*Colse Systick*/
	SysTick->CTRL = 0;

	/*Disable ALL interrupt && Pending Interrupt Flag*/
	/*这里需要根据用户开启的外设进行清除中断和挂起的中断标志*/
	/*
	user add code...
	*/
}
```

1.3 实现方法 2

实现方法2，即使用reset的方式，在SystemInit初始化用户代码之前跳转。

我们使用AT-START开发板设计了一个用户程序的APPJumpToBootloaderMethod2，程序主要完成了一个LED的操作，另外通过检测一个用户按键，但按下按键的时候，表示要从用户程序进入系统Bootloader，此时会在后备寄存器BPR_DATA1中写入0x5AA5。然后产生软件复位，软件复位后在SystemInit开始的地方去判断BPR_DATA1是否为0x5AA5,并将BPR_DATA1清0， 如果是就会进入跳转到Bootloader的程序。这里在进入SystemInit开始的地方就进行跳转，是为了防止用户Code初始化之后和系统Bootloader的配置不匹配。

图 2. 软件流程图



1.3.1 方式 2 代码实现

main函数主要是一个闪灯和检测按键，通过按键按下表示需要进入系统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())
		{
			/*保存 BPR 一个状态标志，表示 APP 需要跳转到 Bootloader */
			BPR_Write_Flag();
		}
		if ( LedTimer == LedTog)
		{
			at32_led_toggle(LED4);;
			LedTimer = 0;
		}
		LedTimer ++;
	}
}
```

bpr_write_flag函数主要是写一个跳转标志到BPR_DATA1中，并进行软件复位

```
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();
}
```

bpr_check_flag函数主要是在reset之后判断是否要跳转到系统bootloader，返回1 表示要跳转到bootloader，0表示不跳转

```
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;
}
```

app_jump_to_bootloader函数负责跳转到Bootloader

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

	/*跳转之前，需要保证所有外设 Clock 关闭，PLL 关闭，关闭所有中断，清除所有的中断挂起标志*/
	SET_MSP(dwStkPtr);
	pfTarget = (void (*)(void))dwJumpAddr;
	pfTarget();

}
```

SystemInit中，在用户代码初始化之前，判断是否需要跳转到系统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 使用用户代码跳转 Bootloader 实验

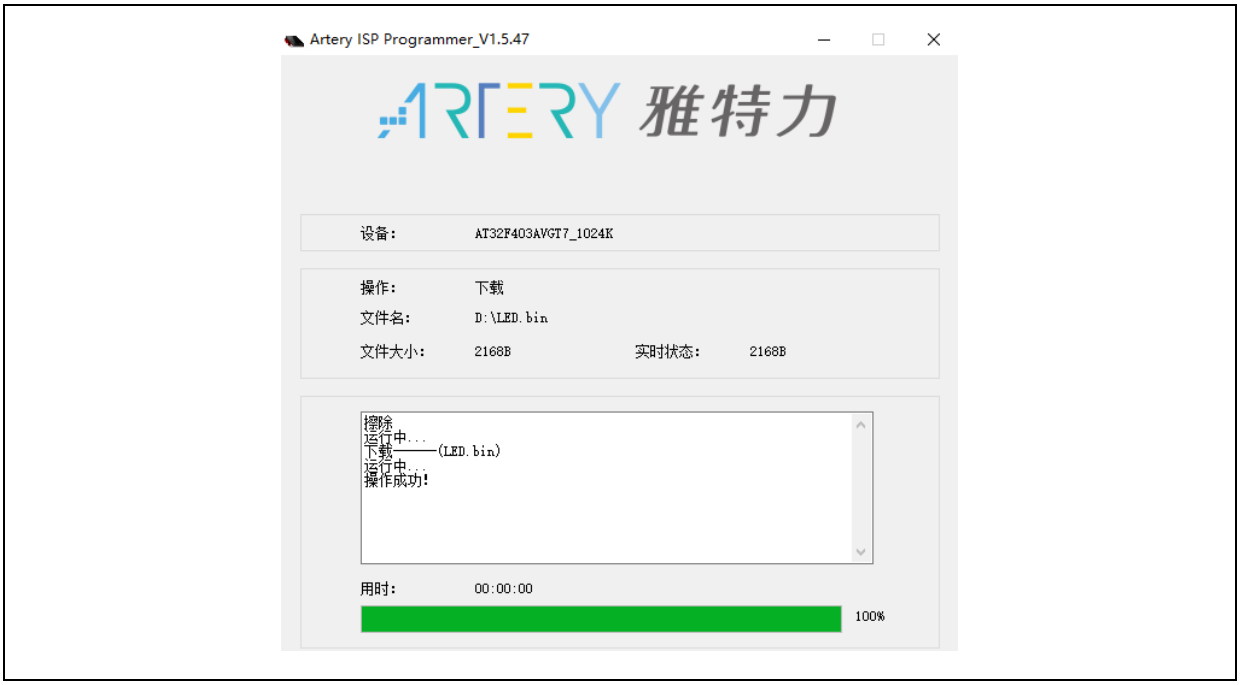
实验使用AT-STAT-F403AV1.0 开发板，使用DFU的方式进行升级（串口升级流程相同）

1. 下载APPJumpToBootloaderMethod1或者APPJumpToBootloaderMethod2程序到AT-START-F403A V1.0开发板
2. 可以看到LED3闪烁
3. 按下PB2 USER 按键，此时LED3熄灭，表示已经进入系统Bootloader
4. 插拔AT-STAT-F403A V1.0 上的USB
5. 打开Artery ISP Programmer,可以看到一个USB DFU的设备已经连接上

图 3. ISP Programmer 设备连接

6. 之后可以按照正常的ISP 升级流程进行程序升级

图 4. ISP Programmer 升级过程



3 版本历史

表 1. 文档版本历史

日期	版本	变更
2021.12.8	2.0.0	最初版本

重要通知 – 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。