RX32x6xx 应用笔记

Bootloader 应用指导

文档编号：AN00022

版本：V1.0

目录

表目录

图目录

1 简介

在单片机系统中，bootloader 是设备上电/复位后最先执行的一段程序，类似于计算机开机时的 BIOS/UEFI。它的核心作用是：

- 硬件初始化：为后续程序运行准备最基础环境（如配置时钟、初始化串口/USB 等通信外设等）。
- 程序管理：决定是直接运行已有程序（APP），还是通过 IAP（In-Application Programming 在应用编程）机制更新 APP。
- 中断向量表适配：保障 APP 运行时中断能够正确响应，解决因程序存储地址偏移导致的中断向量表匹配问题。

本文基于 Cortex-M0 内核，RX32S610（64KB Flash / 8KB SRAM）系列芯片为例，简单介绍 bootloader 从启动准备到跳转到 APP 的工作原理，以便用户参考。

2 结合流程图的 Bootloader 原理介绍

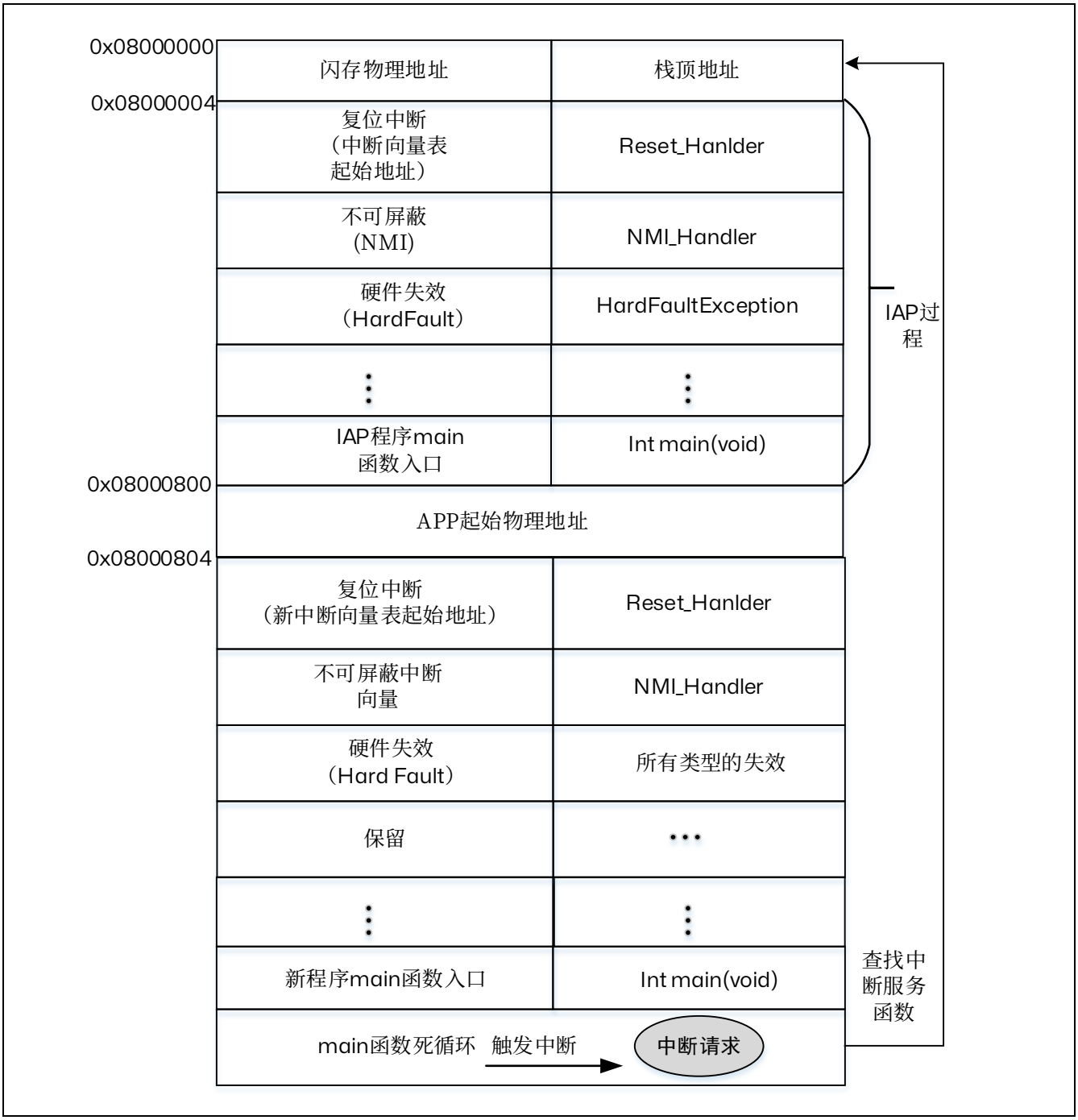

图 2.1 Boot loader 原理拆解图

(一) 阶段 1：Boot loader 自身启动（地址 0x08000000 区）。

1. 物理地址与向量表基础

芯片复位 CPU 会从固定 Flash 地址 0x08000000 取第一条指令。该地址前 4 字节是栈顶地址（为程序运行准备栈空间），紧接着 0x08000004 是复位中断向量，指向 Reset_Handler 函数（Bootloader 的启动入口）。这一区域还包括其他中断向量（如 NMI 不可屏蔽中断、HardFault 硬件错误中断等）构成 Bootloader 的中断向量表。CPU 响应中断时，会默认到

中断向量表找对应的中断服务函数。

2. Bootloader 的 main 执行逻辑

从 Reset_Handler 开始，Bootloader 执行自身 int main(void)，初始化必要的外设（如串口等）。检查“升级标志”（可通过 Fslah 特定地址存储的标志位、串口指令等判断）；若需要升级，进入 IAP 流程；若无需升级，直接准备跳转到已有的 APP。

(二) 阶段 2：IAP 流程与程序跳转（关键的控制权移交）

1. IAP 核心任务：若检测到升级需求，Bootloader 会通过通信外设（如串口）接收新 APP 固件数据
2. 擦除旧 APP 区：擦除 APP 即将存储的 FLASH 区域（避免数据冲突）。
3. 写入新固件：将接收的数据通过 FLASH_Program_DoubleDoubleWord 按 16 字节写入 FLASH，确保新 APP 完整存储。
4. APP 程序需要进行中断向量偏移：跳转之前操作，中断向量偏移，APP 程序的存储地址不是 0x08000000（根据 boot 程序大小做预留，比如设置 APP 起始地址是 0x08000800 等自定义地址），这意味着 APP 自身的中断向量表也存储在其起始地址（0x08000804 是 APP 复位中断向量，其他起始地址以此类推）。如果直接跳转 APP，CPU 仍会找到 0x08000004(Bootloader 的中断向量表)找中断函数，导致 APP 中断响应没有被执行。
5. 对于 ARM Cortex-M0 内核和 Cortex-M3/4 内核有不同的处理方法。
    1) Cortex-M3/4 内核：在跳转 APP 之前，必须设置 SCB->VTOR 寄存器（向量表偏移寄存器）将其值改为 APP 的起始地址（0x08000800）。这样，CPU 响应中断时，会到 APP 的中断向量表找中断服务函数，保障中断正确被响应。
    2) Cortex-M0 内核：可以在 Boot loader 和 APP 的工程代码中同时定义一个 Bootloader_APP_Location 全局变量，在中断服务函数中判断标志位 Bootloader_APP_Location，Bootloader_APP_Location=1 执行 Bootloader 中断服务函数，Bootloader_APP_Location=0 执行 APP 中断服务函数。
6. 跳转实现：完成向量表偏移后还需要：设置栈指针，从 APP 起始地址前 4 字节（栈顶地址）初始化栈空间。复位函数：定义函数指针 func_p = (void (*)(void))(*(uint32_t *)(sysRunningAppAddress + 4))，定义 sysRunningAppAddress 是 APP 的起始地址 0x08002000，跳转到 APP 的 Reset_Handle，正式移交程序控制权。

(三) 阶段 3：APP 程序执行（地址 0x08000800 区）

1. Cortex-M3/4 内核：APP 中断向量表接管，因 VTOR 已设置 APP 起始地址，CPU 响应中断时，会到 APP 地址区域找中断向量表：复位中断向量表 0x08000804 指向 APP 的 Reset_Handle()，其他中断也有对应中断服务函数。APP 从 Reset_Handle 开始初始化（如配置复杂的外设、加载业务参数），最终进入 int main(void)运行业务逻辑。当程序运行中触发中断时，CPU 根据 VTOR 指向的 APP 向量表，调用对应 xxx_Handle 函数，实现“中断触发->服务函数执行->回到主逻辑”的完整流程。

2. Cortex-M0 内核：与 Cortex-M3/4 内核不同的是，M0 内核没有 VTOR 寄存器（向量表偏移寄存器），无法直接配置 APP 的起始地址。通过在 Boot loader 中断服务函数中设置标志位 Bootloader_APP_Location 用来区分是 APP 或 Boot loader 所触发的中断。

3 中断向量偏移的必要性

(一) 根本矛盾程序地址与向量表的“绑定关系”。中断向量表与程序起始地址强关联：向量表起始地址=程序起始地址。若 Boot loader 程序在 0x08000000，其向量表就固定在 0x08000004 开始的区域；而 APP 程序因存储地址不同（如 0x08002000），向量表自然跟随到 0x08002004。若不做地址偏移处理，CPU 始终认为向量表在 0x08000004，触发中断时错误调用 Boot loader 区的中断函数（甚至因为 Boot loader 已跳转，函数可能覆盖/失效），导致系统崩溃。

(二) Cortex-M0 内核和和 Cortex-M3/4 内核有不同的处理方法，在上文已介绍。

4 Boot loader 代码配置

Boot loader 对应代码配置。

```
//在 0x20000100 定义一个全局变量
uint32_t   Bootloader_APP_Location   __attribute__ ((section(".ARM. __at_0x20000100")))   = 1;
//判断是否需要更新 APP 数据，1：更新 0：不更新直接跳转
if((UpgradeFALG) == sysUpgradeFlag)
{
  //解锁 FLASH
   FLASH_Unlock();
  //擦除 FLASH，可以根据代码大小擦除扇区
   FLASH_Erase_Page(i);
  //FLASH 编程
   FLASH_Program_DoubleDoubleWord(sysRunningAppAddress + i*16, buff);
  //FLASH 上锁
   FLASH_Lock();
  //擦除升级标志位所在那一页扇区，需要先解锁
   FLASH_Lock();
  //擦除升级标志位那一页扇区
   FLASH_Erase_Page(i);
  //FLASH 上锁
   FLASH_Lock();
  //软件复位
   NVIC_SystemReset();
}
else
{
  //判断如果不更新，直接跳转 APP，实际工程中延时 4 秒再跳转 APP
   vRunAPP();
}
```

4.1 中断服务函数说明

不论是 APP 或 Boot loader 工程中所触发的中断都需要在 Boot loader 的中断服务函数中进行处理，APP 的中断服务函数需要在 Boot loader 的对应的中断服务函数下使用函数指针进行跳转。并使用标志 Bootloader_APP_Location 来区分是 Boot loader 或 APP 所触发的中断，Bootloader_APP_Location=1 表示进入 bootloader 的中断。Bootloader_APP_Location = 0 表示进入 APP 的中断。

在工程代码中使用 SysTick_Handler 函数进行举例，在 Boot loader 工程启用 SysTick 定时 1 毫秒触发中断，在中断函数中翻转 PC5 端口电平，形成方波信号。在 APP 工程也启用 SysTick 定时 1 毫秒触发中断，在中断函数中翻转 PC7 端口电平，形成方波信号。

Bootloader 中断服务函数代码如下所示。

```
void SysTick_Handler(void)
{
  msTicks++;
//执行 Boot loader 中断
  if(Bootloader_APP_Location ==1)
  {
    GPIO_Toggle_Pin(GPIOC,GPIO_PIN_5);
  }
  else
    {

//执行 APP 中断
      uint32_t JumpToApp_Address;
      JumpToApp_Address= sysRunningAppAddress +0x3C; //0x3C 是 SysTick 在中断向量表偏移值
      void(*Jump)(void);
      Jump = (void(*)(void))(*(uint32_t *)(JumpToApp_Address));
      Jump();
    }
}
```

	-1	固定	硬件失效(HardFault)	所有类型的失效	0x0000_000C
	-	-	-	保留	0x0000_001C ~ 0x0000_002B
	3	可设置	SVCall	通过 SWI 指令的系统服务调用	0x0000_002C
	5	可设置	PendSV	可挂起的系统服务	0x0000_0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000_003C
0	7	可设置			0x0000_0040

参考 RX32S610 第 10 章节，嵌套向量中断控制器（NVIC），可以查看中断优先级以及地址。

需要注意 SysTick_Handler 的存储位置，在 Boot loader 工程中不需要修改，在 APP 工程中需要据 APP 代码存储的位置，手动更新 SysTick_Handler 在 APP 工程中的中断向量表的位置。本例程中 APP 起始地址是 sysRunningAppAddress（sysRunningAppAddress= 0x08000800），加上 0x3C 偏移值即可得到 SysTick_Handler 在 APP 工程中的中断向量表的位置，其他中断服务函数按照（地址）进行偏移就可。

表 10.1 RX32S610 系列向量表

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000_0000
	-3	固定	Reset	复位	0x0000_0004
	-2	固定	NMI	不可屏蔽中断 RCC 时钟安全系统（CSS） 联接到 NMI 向量	0x0000_0008
	-1	固定	硬件失效(HardFault)	所有类型的失效	0x0000_000C
	-	-	-	保留	0x0000_001C ~ 0x0000_002B
	3	可设置	SVCall	通过 SWI 指令的系统服务调用	0x0000_002C
	5	可设置	PendSV	可挂起的系统服务	0x0000_0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000_003C
0	7	可设置			0x0000_0040
1	8	可设置	PVD	PVD 中断和 EXTI 线 16	0x0000_0044
2	9	可设置	RTC	RTC 中断和 EXTI 线 17	0x0000_0048
3	10	可设置	FLASH	闪存全局中断	0x0000_004C
4	11	可设置	RCC	复位和时钟控制（RCC）中断	0x0000_0050
5	12	可设置	EXTI 0_3	EXTI 线[3:0]中断	0x0000_0054

图 4.1 RX32S610 系列向量表

5 APP 代码配置

在 APP 工程同样定义 Bootloader_APP_Location，同样也是在 0x20000100 这个位置。在 SysTick_Handler 中翻转 PC 7 端口电平，形成方波信号。

```
//定义变量
uint32_t   Bootloader_APP_Location   __attribute__ ((section(".ARM. __at_0x20000100"))) ;
//配置时钟
SetSysClockToHSI_80M();
//配置 SysTick 定时操作
SYSTICK_INIT()
GPIO_Toggle();
//APP 工程中中断服务函数
void SysTick_Handler(void)
{
  GPIO_Toggle_Pin(GPIOC,GPIO_PIN_7);
}
```

6 实验结果与结论

按照 4.1 小节的描述，通过在 Boot loader 的中断服务函数中判断标志位 Bootloader_APP_Location

的值，分别执行 APP 和 boot loader 中的 SysTick_Handler 函数，在每个中断中都执行 IO 反转的操作。本例程代码只举例 SysTick_Handler 函数，其他中断服务函数按照相同的方法配置软件代码即可。

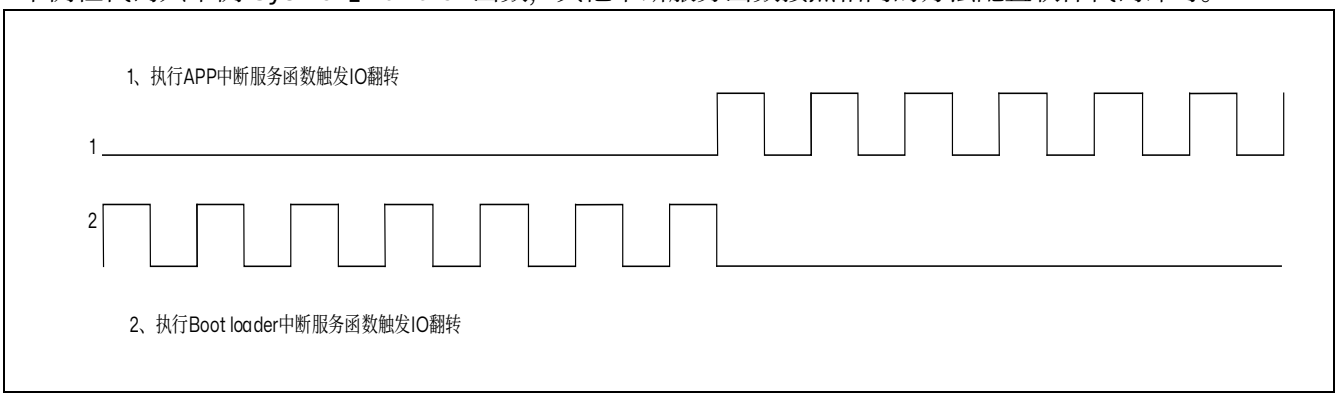

图 5.1 实验结果

7 版本历史

表 4.1 版本历史

日期	版本	更改内容
2025 年 11 月 12 日	V1.0	初版