从SXX32F030移植到AT32F421

前言

本应用笔记旨在帮助您分析从现有的SXX32F030器件移植到AT32F421器件所需的步骤。本文档收集了最重要的信息，并列出了需要注意的重要事项。

要将应用程序从SXX32F030系列移植到AT32F421系列，用户需要分析硬件移植、外设移植和固件移植。

支持型号列表：

支持型号	AT32F421xx

目录

表目录

1 AT32F421 与 SXX32F030 异同

AT32F421系列微控制器大部分兼容SXX32F030系列，有在对整体性价比考量下进行部分功能取舍，同时亦有强化许多新功能，故导致有些许地方存在不同，详述于本文档。

1.1 相同点概述

- 管脚定义：相同封装管脚定义相同。为扩增的外设作管脚复用定义延伸
- 编译工具：完全相同，例如Keil, IAR

1.2 差异点概述

表 1.AT32F421 与 SXX32F030 差异概述

	AT32F421	SXX32F030
内核	Cortex-M4 支持硬件DSP功能	Cortex-M0
系统时钟	主频120 MHz, AHB/APB 120 MHz	主频48 MHz, AHB/APB 48 MHz
STOP唤醒(调压器处于低功耗模式)	450 µs	最大9 µs
Standby唤醒	1250 µs	60.4 µs
Flash容量	64 KB	256 KB
SRAM容量	16 KB	32 KB
系统存储器 (System Memory)	4 KB，支持对闪存内容进行CRC校验	12 KB
闪存16-bit写入时间	40 µs	53.5 µs
闪存页擦除时间	6.4 ms	30 ms
闪存整片擦除时间	8 ms	30 ms
SPI	SPI1/SPI2都支持I²S功能	不支持I²S功能
安全库区设定	支持，详细描述请参阅AT32F421参考手册 4.3.5.1节和安全库区(sLib)应用指南.pdf	无
系统存储器区域作为主存扩展使用	支持，详细描述请参阅AT32F421参考手册 4.3.5.2节和AN0066设定系统存储器为主存扩展（AP模式）	无
选择字节	256 Bytes	16 Bytes
ADC	2 Msps (max. ADCCLK = 28MHz)	1 Msps (max. ADCCLK = 14MHz)
内核电压	1.2 V操作电流更低	1.8 V
ESD参数	HBM: 6 kV, CDM: 1000 V	HBM: 2 kV, CDM: 500 V
运行模式	7.5 mA@48 MHz	22.0 mA@48 MHz
睡眠功耗	5.7 mA@48 MHz	14.0 mA@48 MHz
停机功耗	210 uA	7.9 uA
待机功耗	3.6 uA	4.8 uA

2 硬件移植

AT32F421与SXX32F030系列的各引脚基本上相兼容，只有与SXX32F030CC极少数的引脚存在差异，转化起来极其方便，详细信息如下表。

表 2. 硬件引脚兼容性

AT32F421		SXX32F030CC	
QFP48	引脚	QFP48	引脚
35	PF6	35	V_{SS}
36	PF7	36	V_{DD}

3 外设移植

3.1 外设对比

AT32F421在外设部分和SXX32F030相对比有些外设还是存在有一定的区别，且有些相对来说算是一个全新的设计。故针对这些外设需在应用层级的程序开发中进行修改或参考新外设驱动进行全新开发。

表 3. 外设兼容性

外设	AT32F421	SXX32F030	兼容性		
			特性	引脚排列	固件驱动
SPI	有+	有	SXX32F030： 提供两个FIFO，4位到16位数据大小可供选择 AT32F421： SPI支持I²S功能，支持I²S的WS线与Data实时同步，AT32F421的SPI波特率最高支持50 MHz	相同	部分兼容
WWDG	有	有+	特性相同	N/A	完全兼容
IWDG	有	有+	增加了窗口模式	N/A	部分兼容
MCUDBG	有	有	特性相同	N/A	兼容
CRC	有	有+	增加了反转功能和初始CRC值	N/A	部分兼容
EXTI	有	有+	某些外设能够在停止模式下生成事件	相同	部分兼容
DMA	有	有	1个具有5个通道的DMA控制器	N/A	部分兼容
TMR	有	有+	增强	相同	部分兼容
PWR	有	有+	V_{DDA}可以大于V_{DD}，内核采用1.8 V模式	对于同一特性相同	部分兼容
RCC	有++	有	主频、AHB、APB频率提高，MCO增强	相同	部分兼容
USART	有	有+	独立时钟源选择，超时特性，从停止模式唤醒	相同	部分兼容
I²C	有	有	SXX32F030: 支持由硬件管理通信事件，FM+，从停止模式唤醒，数字滤波器 AT32F421: 支持1 MHz速率	相同	部分兼容
ADC	有	有+	模拟部分相同，但有新的数字接口	相同	部分兼容
RTC	有++	有	具有备份域寄存器	相同	兼容
FLASH	有++	有	选择字节扩展到256个字节，增加安全库区功能，增加系统存储器区域作为主存扩展使用功能，支持按字节、半字节和字访问	N/A	部分兼容
GPIO	有	有	-	相同	兼容
COMP	有	N/A	-	N/A	N/A
SYSCFG	有	有	-	N/A	部分兼容

3.2 存储器映射

因性能上的考量和优化，AT32F421在架构上进行了更深层次的调整。相对于SXX32F030而言，外设

地址和总线的排列分布有一定的区别，以下就详细的列出地址映射区别和总线所属关系。

表 4. 存储器映射关系差异

外设	SXX32F030		AT32F421		结论
	总线	基址	总线	基址	
CRC	AHB1	0x40023000	AHB	0x40023000	相同
FMC		0x40022000		0x40022000	相同
RCC		0x40021000		0x40021000	相同
DMA		0x40020000		0x40020000	相同
GPIOF	AHB2	0x48001400		0x48001400	相同
GPIOC		0x48000800		0x48000800	相同
GPIOB		0x48000400		0x48000400	相同
GPIOA		0x48000000		0x48000000	相同
MCUDBG	APB	0x40015800	CPU core	0xE00420000	不相同
TMR17		0x40014800	APB2	0x40014800	相同
TMR16		0x40014400		0x40014400	相同
TMR15		0x40014000		0x40014000	相同
USART1		0x40013800		0x40013800	相同
SPI1		0x40013000		0x40013000	相同
TMR1		0x40012C00		0x40012C00	相同
ADC		0x40012400		0x40012400	相同
USART6		0x40011400		N/A	不相同
EXTI		0x40010400		0x40010400	相同
SYSCFG		0x40010000		0x40010000	相同
COMP		N/A		0x40010000	相同
PWR		0x40007000	APB1	0x40007000	相同
I^2C2		0x40005800		0x40005800	相同
I^2C1		0x40005400		0x40005400	相同
USART5		0x40005000		N/A	不相同
USART4		0x40004C00		N/A	不相同
USART3		0x40004800		N/A	不相同
USART2		0x40004400		0x40004400	相同
SPI2		0x40003800		0x40003800	相同
IWDG		0x40003000		0x40003000	相同
WWDG		0x40002C00		0x40002C00	相同
RTC		0x40002800		0x40002800	相同
TMR14		0x40002000		0x40002000	相同
TMR7		0x40001400		N/A	不相同
TMR6		0x40001000		0x40001000	相同
TMR3		0x40000400		0x40000400	相同

3.3 功能区别

3.3.1 RCC 接口

AT32F421与SXX32F030对比在RCC部分有以下区别：

表 5. RCC 接口差异

RCC	AT32F421	SXX32F030
HSI	48 MHz RC除频6，48 MHz RC	8 MHz RC
HSE	4-25 MHz	4-32 MHz
HSI14	N/A	用于ADC
MCO	ADCCLK、SYSCLK、LSI、LSE、HSI、HSE、PLL/2、PLL/4	HSI14、SYSCLK、HSI、HSE、PLL/2、PLL、LSE、LSI、HSI48

3.3.2 RCC PLL

- 描述：

  因AT32F421RCC IP的更新，在系统时钟配置流程中，RCC PLL配置和使能之前需要根据实际所使用的PLL时钟源来对参考时钟配置表PLL_FREF参数进行配置(寄存器RCC_PLL[26:24])。

位 26：24	**PLL_FREF**：PLL 输入时钟选择，仅在 PLLCFGEN=0 时起作用 000：PLL 使用 3.9 MHz ~ 5 MHz 输入时钟； 001：PLL 使用 5.2 MHz ~ 6.25 MHz 输入时钟； 010：PLL 使用 7.8125 MHz ~ 8.33 MHz 输入时钟； 011：PLL 使用 8.33 MHz ~ 12.5 MHz 输入时钟； 100：PLL 使用 15.625 MHz ~ 20.83 MHz 输入时钟； 101：PLL 使用 20.83 MHz ~ 31.255 MHz 输入时钟； 110：保留； 111：保留。

- 解决方法：

  示例使用8 MHz晶振配置120 MHz系统时钟的修改方法如下：

```
static void SetSysClockTo120M(void)
{
…
#if defined (AT32F421xx)
    RCC->CFG |= (uint32_t)(RCC_CFG_PLLRC_HSE | RCC_CFG_PLLMULT15);
#else
    RCC->CFG |= (uint32_t)(RCC_CFG_PLLRC_HSE | RCC_CFG_PLLMULT15 |
RCC_CFG_PLLRANGE_GT72MHZ);
#endif
…
}
```

修改为：

```
static void SetSysClockTo144M(void)
{
…
RCC->PLL |= (0x2 << 24);  // PLL使用8M参考时钟配置表
#if defined (AT32F421xx)
    RCC->CFG |= (uint32_t)(RCC_CFG_PLLRC_HSE | RCC_CFG_PLLMULT15);
#else
    RCC->CFG |= (uint32_t)(RCC_CFG_PLLRC_HSE | RCC_CFG_PLLMULT15 |
RCC_CFG_PLLRANGE_GT72MHZ);
#endif
…
}
```

3.3.3 DMA 接口

AT32F421与SXX32F030对比在DMA部分区别不大

表 6. DMA 接口差异

外设	DMA请求	AT32F421	SXX32F030	结论
TMR17	TMR17_UP TMR17_CH1	DMA_Channel1/DMA_Channel2 DMA_Channel1/DMA_Channel2	DMA_Channel1/DMA_Channel2 DMA_Channel1/DMA_Channel2	相同
TMR16	TMR16_UP TMR16_CH1	DMA_Channel3/DMA_Channel4 DMA_Channel3/DMA_Channel4	DMA_Channel3/DMA_Channel4 DMA_Channel3/DMA_Channel4	相同
TMR15	TMR15_UP TMR15_CH1 TMR15_TRIG TMR15_COM	DMA_Channel5 DMA_Channel5 DMA_Channel5 DMA_Channel5	DMA_Channel5 DMA_Channel5 DMA_Channel5 DMA_Channel5	相同
USART1	USART1_Rx USART1_Tx	DMA_Channel3/DMA_Channel5 DMA_Channel2/DMA_Channel4	DMA_Channel3/DMA_Channel5 DMA_Channel2/DMA_Channel4	相同
SPI1	SPI1_Rx SPI1_Tx	DMA_Channel2 DMA_Channel3	DMA_Channel2 DMA_Channel3	相同
TMR1	TMR1_UP TMR1_CH1 TMR1_CH2 TMR1_CH3 TMR1_CH4 TMR1_TRIG TMR1_COM	DMA_Channel5 DMA_Channel2 DMA_Channel3 DMA_Channel5 DMA_Channel4 DMA_Channel4 DMA_Channel4	DMA_Channel5 DMA_Channel2 DMA_Channel3 DMA_Channel5 DMA_Channel4 DMA_Channel4 DMA_Channel4	相同
ADC	ADC	DMA_Channel1 DMA_Channel2	DMA_Channel1 DMA_Channel2	相同
I²C2	I2C2_Rx I2C2_Tx	DMA_Channel5 DMA_Channel4	DMA_Channel5 DMA_Channel4	相同
I²C1	I2C1_Rx I2C1_Tx	DMA_Channel3 DMA_Channel2	DMA_Channel3 DMA_Channel2	相同
USART2	USART2_Rx USART2_Tx	DMA_Channel5 DMA_Channel4	DMA_Channel5 DMA_Channel4	相同
SPI2	SPI2_Rx SPI2_Tx	DMA_Channel4 DMA_Channel5	DMA_Channel4 DMA_Channel5	相同
TMR6	TIM6_UP	DMA_Channel3	DMA_Channel3	相同
TMR3	TMR3_UP TMR3_CH1 TMR3_TRIG TMR3_CH3 TMR3_CH4	DMA_Channel3 DMA_Channel4 DMA_Channel4 DMA_Channel2 DMA_Channel3	DMA_Channel3 DMA_Channel4 DMA_Channel4 DMA_Channel2 DMA_Channel3	相同
USART3	USART3_Rx USART3_Tx	N/A	DMA_Channel3 DMA_Channel2	不相同

3.3.4 中断向量

AT32F421与SXX32F030对比在中断号及中断向量部分区别不大。

表 7. 中断向量差异

位置	AT32F421	SXX32F030	结论
0	WWDG	WWDG	相同
1	PVD	Reserved	相同
2	RTC	RTC	相同
3	FLASH	FLASH	相同
4	RCC	RCC	相同
5	EXTI0_1	EXTI0_1	相同
6	EXTI2_3	EXTI2_3	相同
7	EXTI4_15	EXTI4_15	相同
8	Reserved	Reserved	相同
9	DMA_CH1	DMA_CH1	相同
10	DMA_CH2_CH3	DMA_CH2_CH3	相同
11	DMA_CH4_CH5	DMA_CH4_CH5	相同
12	ADC_COMP	ADC	相同
13	TMR1_BRK_UP_TRG_COM	TIM1_BRK_UP_TRG_COM	相同
14	TMR1_CC	TIM1_CC	相同
15	Reserved	Reserved	相同
16	TMR3	TIM3	相同
17	TMR6	TIM6	相同
18	Reserved	Reserved	相同
19	TMR14	TIM14	相同
20	TMR15	TIM15	相同
21	TMR16	TIM16	相同
22	TMR17	TIM17	相同
23	I2C1_EV	I2C1	不相同
24	I2C2_EV	I2C2	不相同
25	SPI1	SPI1	相同
26	SPI2	SPI2	相同
27	USART1	USART1	相同
28	USART2	USART2	相同
29	Reserved	USART3_4_5_6	不相同
30	Reserved	Reserved	相同
31	Reserved	Reserved	相同
32	I2C1_ER	Reserved	不相同
33	Reserved	Reserved	相同
34	I2C2_ER	Reserved	不相同

3.3.5 EXTI 中断源选择

AT32F421 与 SXX32F030 EXTI 相同。

3.3.6 GPIO 接口

AT32F421 与 SXX32F030 GPIO 区别不大，如下表：

表 8. GPIO 接口差异

GPIO	AT32F421	SXX32F030
输入模式	悬空 PU PD	悬空 PU PD
输出模式	PP	PP PP+PU PP+PD
	OD	OD OD+PU OD+PD
功能复用	PP	PP PP+PU PP+PD
	OD	OD OD+PU OD+PD
20 mA吸入能力	不支持	支持

3.3.7 ADC 接口

AT32F421与SXX32F030在ADC上对比有如下区别

表 9. ADC 接口差异

ADC	AT32F421		SXX32F030
通道数	15通道+3内部通道		16通道+2内部通道
转换模式	单一/连续/间断/扫描		单一/连续/间断/扫描
分辨率	12位		12位
最大采样率	2 MSPS		1 MSPS
外部触发	规则组 TMR1 CC1 TMR1 CC2 TMR1 CC3 TMR3 TRGO TMR15 CC1 EXTI line11 SWSTR	注入组 TMR1 TRGO TMR1 CC4 TMR3 CC4 TMR15 TRGO EXTI line15 JSWSTR	外部事件 TIM1_TRGO TIM1_CC4 TIM3_TRGO TIM15_TRGO

3.3.8 SPI 接口

对比SXX32F030在SPI接口上存在部分的不同。

经性价比等全方面的考虑和分析，AT32F421剔除了SXX32F030上的部分SPI的如下功能特性。NSS的脉冲模式和TI模式配置。

1. 数据帧长度的可编程控制。
2. Tx/Rx FIFO缓冲区。

但同时，AT32F421增加了如下特性：

1. SPI支持I²S功能
2. 支持I²S的WS线与Data实时同步
3. AT32F421的SPI波特率最高支持50 MHz

3.3.9 I²C 接口

相比于SXX32F030的I²C有较大的区别。

经性价比等全方面的考虑和分析，AT32F421剔除了SXX32F030上的部分I²C的功能特性。

二者在结构、特性和编程上都不同，因此基于SXX32F030的I²C部分的代码需要重新编写后才可以在AT32F421上运行。

Demo请参考AT32F4xx_StdPeriph_Lib_Vx.x.x\Project\AT_START_F421\Examples\I2C

详细的编程发放和接口请参阅AT32F421说明手册第15章节。

3.3.10 USART 接口

AT32F421内置的USART外设相比于SXX32F030的USART有较大的区别。二者在结构、特性和编程上都不同，因此USART部分的代码需要重新编写后才可以在AT32F421上运行。

Demo请参考AT32F4xx_StdPeriph_Lib_Vx.x.x\Project\AT_START_F421\Examples\USART

详细的编程方法和接口请参阅AT32F421说明手册第16章节。

其中USART智能卡模式下存在接收数据异常可能：

- 描述：
  初始化USART的智能卡模式时，若先使能USART再使能智能卡模式，USART 智能卡模式使能后的第一帧数据长度时段内无法接收数据。
- 解决方法：
  使能USART之前先使能智能卡模式。

```
/********先使能智能卡模式********/
    /* Enable the NACK Transmission */
    USART_SmartCardNACKCmd(USARTx, ENABLE);
    /* Enable the Smartcard Interface */
    USART_SmartCardCmd(USARTx, ENABLE);

/******配置完成后再使能 USART ******/
```

3.3.11 PWR

- AT32F421系列采用Cortex-M4内核，PWR以WFE进入lower-power模式有所区别，需要按如下使用范例操作。
- 使用范例：

```
/* Request Wait For Event */
__SEV();
__WFE();
__WFE();
```

3.4 功能增强

3.4.1 高频 PLL

- 描述：
  - AT32F421内置的PLL可输出最高可达120 MHz时钟，设定略有不同
- 使用范例：
  - AT32F421PLL设定程序范例：
  - AT32F4xx_StdPeriph_Lib_V1.x.x\Project\AT_START_F421\Templates中system_at32f4xx.c的static void SetSysClockTo120M(void)函数
- 描述：
  - 由于主频提高至120 MHz，相关预分频器做出扩增
  - ADC预分频器扩增支持/12, /16输出
  - 请参阅AT32F421参考手册3.3.2 RCC_CFG寄存器叙述

3.4.2 安全库区保护

- 描述
  - 目前越来越多的微控器(MCU)应用需要使用到复杂的算法及中间件解决方案(middleware solution)，因此，如何保护软件方案商开发出来的核心算法等知识产权代码(IP-Code)，便成为微控制器应用中一项很重要的课题。

    为因应这一重要的需求，AT32F421系列提供了安全库区(sLib)的功能，以防止重要的IP-Code被终端用户的程序做修改或读取，进而达到保护的目的。
- 使用范例
  - 请参考*AT32F421Security Library Application Note*
  - Demo请参考AT32F4xx_StdPeriph_Lib_Vx.x.x\Utilities\AT32F421_SLIB_Demo

3.4.3 GPIO 5V 容忍引脚兼容

- 描述
  - AT32F421芯片相较GX32E230提供更多5V电压输入容忍引脚，仅有引脚PC14、PC15、PF0、PF1不具5V电压输入容忍特性，这些引脚输入电平不可超过VDD + 0.3V。
  - 其他引脚皆为5V电压输入容忍。

4 使用库进行固件移植

本节介绍了如何移植基于 SXX32F030 标准外设库的应用程序，目的是使用 AT32F421 标准外设库。AT32F421 和 SXX32F030 库的结构相同，并且均与 CMSIS 兼容；对于所有兼容的外设，它们都使用相同的驱动程序命名方式和相同的 API。

将应用程序从 SXX32F030 系列移植到 AT32F421 系列产品时，只需更新几个外设驱动程序。

注： *在本节的其余部分中，术语“SXX32F030 库”是指 SXX32F030 标准外设库，术语“AT32F421 库”是指 AT32F421 标准外设库（除非另有规定）。*

4.1 移植步骤

要更新应用程序代码，以在 AT32F421 库上运行，用户需要按照下列步骤操作：

1 下载 AT32F421 最新的 BSP 和 PACK 文件

2 安装 PACK 文件以支持各种调试编译器（IAR、Keil 等）

3. 更新工具启动文件

a) 项目文件：器件连接和 Flash 加载程序。这些文件随支持 AT32F421x 器件的最新版本工具一起提供。有关详细信息，请参见相关工具文档。

b) 链接器配置和向量表位置文件：这些文件根据 CMSIS 标准开发并且包含在 AT32F421 库安装包内，位置如下：Libraries\CMSIS\CM4\DeviceSupport。

4. 将 AT32F421 库源文件添加到应用程序源

a) 用 AT32F421 库中提供的 at32f4xx_conf.h 文件替换应用程序的 sxx32f0xx_conf.h 文件。

b) 用 AT32F421 库中提供的 at32f4xx_it.c/ at32f4xx_it_it.h 文件替换应用程序的现有 sxx32f0xx_it.c/sxx32f0xx_it.h 文件。

c) 用 AT32F421 库中提供的 system_at32f4xx.c/system_at32f4xx.h 文件替换应用程序的现有 system_sxx32f0xx.c/system_sxx32f0xx.h 文件。

5. 更新使用 RCC、PWR、GPIO、FLASH、ADC 和 RTC 驱动程序的部分应用程序代码。相关详细信息，请参见下一节。

注： *AT32F421 库随附丰富的示例（总共 100 多个），具体演示了使用不同外设的方法（位于 AT32F4xx_StdPeriph_Lib_V1.x.x\Project\AT_START_F421\Examples 下）。*

4.2 RCC 驱动程序

1. 系统时钟配置：

如 3.3.1: RCC 接口一节所述，AT32F421 和 SXX32 F030 系列的时钟源配置过程相同。但是，二者产品的电压范围、PLL 配置、最大频率和 Flash 等待周期配置有所不同。利用 CMSIS 层，可从应用程序代码中隐藏这些区别；用户只需使用 system_at32f4xx.c 文件替换 system_sxx32f0xx.c 文件。此文件提供了 SystemInit()函数一种实现方法，利用该函数，可在启动时和转到 main() 程序之前配置微控制器系统。

2. 外设访问配置：

对于复位和时钟控制器 (RCC) 接口，用户需要调用不同的函数来 [使能/禁止] 外设 [时钟] 或使外设 [进入/退出] [复位模式]。AT32F421 和 SXX32 F030 系列的外设访问配置是相同的。

3. 外设时钟配置

某些 AT32F421 外设支持双时钟特性。下表概述了 AT32F421 外设与 SXX32F10xx 外设的 IP 时钟源比较。

表 10. AT32F421 与 SXX32F030 时钟源驱动程序 API 对应关系

外设	AT32F421时钟源	SXX32F030时钟源
ADC	带预分频器的 APB2 时钟	-HSI14：默认 - APB2 时钟 2 分频 - APB2 时钟 4 分频
I²C	APB1 时钟	1. I²C1 可由以下时钟源提供时钟： - 系统时钟 - HSI 2. I²C2 只能由以下时钟源提供时钟： - APB1
SPI/I²S	1. SPI1/I²S1 时钟源为APB2 2. SPI2/I²S2 时钟源为APB1	1. SPI1时钟源为APB2 2. SPI2 时钟源为APB1
USART	1. USART1 时钟源为APB2 2. USART2 时钟源为APB1	1. USART1 可由以下时钟源提供时钟： -系统时钟 - LSE 时钟 - HSI 时钟 - APB2 时钟 (PCLK) 2. USART2 只能由以下时钟源提供时钟： - APB1 时钟

注意： *AT32F421 的 APB1 和 APB2 的最高速度皆为 120 MHz，而 SXX32F030 的 APB1 和 APB2 的最高速度皆为 48 MHz*

4.3 Flash 驱动程序

下表介绍了 AT32F421 与 SXX32F030 库之间的 Flash 驱动程序 API 的对应关系。利用 AT32F421 库中的函数替换 SXX32F030 中的对应函数后，便可轻松更新应用程序代码。

表 11. AT32F421 与 SXX32F030 Flash 驱动程序 API 对应关系

	AT32F421Flash 驱动程序 API	SXX32F030 Flash 驱动程序 API
接口配置	N/A	FLASH_SetLatency(uint32_t FLASH_Latency);
	N/A	FLASH_PrefetchBufferCmd(FunctionalState NewState);
	N/A	N/A
	FLASH_INTConfig(uint32_t FLASH_INT, FunctionalState NewState);	FLASH_ITConfig(uint32_t FLASH_IT, FunctionalState NewState);
存储器编程	FLASH_Unlock(void);	FLASH_Unlock(void);
	FLASH_Lock(void);	FLASH_Lock(void);
	FLASH_ErasePage(uint32_t Page_Address);	FLASH_ErasePage(uint32_t Page_Address);
	FLASH_EraseAllPages(void);	FLASH_EraseAllPages(void);
	FLASH_EraseUserOptionBytes(void);	FLASH_OB_ERASE(void);
	FLASH_ProgramWord(uint32_t Address, uint32_t Data);	FLASH_ProgramWord(uint32_t Address,uint32_t Data);
	FLASH_ProgramHalfWord(uint32_t Address, uint16_t Data);	FLASH_ProgramHalfWord(uint32_t Address, uint16_t Data);

	AT32F421Flash 驱动程序 API	SXX32F030 Flash 驱动程序 API
选项字节编程	N/A	FLASH_OB_Unlock(void);
	N/A	FLASH_OB_Lock(void);
	FLASH_ProgramUserOptionByteData(uint32_t Address, uint8_t Data);	FLASH_ProgramOptionByteData(uint32_t Address, uint8_t Data);
	FLASH_EnableWriteProtect(uint32_t FLASH_Pages);	FLASH_OB_EnableWRP(uint32_t OB_WRP);
	FLASH_ReadProtectConfig(FunctionalState NewState);	FLASH_OB_RDPConfig(uint8_t OB_RDP);
	FLASH_UserOptionByteConfig(uint16_t UOB_IWDG, uint16_t UOB_STOP, uint16_t UOB_STDBY);	FLASH_OB_UserConfig(uint8_t OB_IWDG, uint8_t OB_STOP, uint8_t OB_STDBY);
	N/A	FLASH_OB_Launch(void);
	N/A	FLASH_OB_WriteUser(uint8_t OB_USER);
	N/A	FLASH_OB_BOOTConfig(uint8_t OB_BOOT1);
	N/A	FLASH_OB_VDDAConfig(uint8_t OB_VDDA_ANALOG);
	N/A	FLASH_OB_SRAMParityConfig(uint8_t OB_SRAM_Parity);
	FLASH_GetUserOptionByte(void);	FLASH_OB_GetUser(void);
	FLASH_GetWriteProtectStatus(void);	FLASH_OB_GetWRP(void);
	FLASH_GetReadProtectStatus(void);	FLASH_OB_GetRDP(void);
FLAG管理	FLASH_GetFlagStatus(uint32_t FLASH_FLAG);	FLASH_GetFlagStatus(uint32_t FLASH_FLAG);
	FLASH_ClearFlag(uint32_t FLASH_FLAG);	FLASH_ClearFlag(uint32_t FLASH_FLAG);
	FLASH_GetStatus(void);	FLASH_GetStatus(void);
	FLASH_WaitForProcess(uint32_t Timeout);	FLASH_WaitForLastOperation(void);
	FLASH_GetPrefetchBufferStatus(void);	FLASH_GetPrefetchBufferStatus(void);

4.4 CRC 驱动程序

下表介绍了 AT32F421 与 SXX32F030 CRC 驱动程序 API 对应关系。

表 12. AT32F421 与 SXX32F0xx CRC 驱动程序 API 对应关系

	AT32F421Flash 驱动程序 API	SXX32F030 Flash 驱动程序 API
配置	N/A	CRC_DeInit(void);
	CRC_ResetDT(void);	CRC_ResetDR(void);
	N/A	CRC_ReverseInputDataSelect(uint32_t CRC_ReverseInputData);
	N/A	CRC_ReverseOutputDataCmd(FunctionalState NewState);
	N/A	CRC_SetInitRegister(uint32_t CRC_InitValue);

计算	CRC_CalculateCRC(uint32_t Data);	CRC_CalcCRC(uint32_t CRC_Data);
	CRC_CalcBlockCRC(uint32_t pBuffer[], uint32_t BufferLength);	CRC_CalcBlockCRC(uint32_t pBuffer[], uint32_t BufferLength);
	CRC_GetCRC(void);	CRC_GetCRC(void);

4.5 GPIO 配置更新

如前文 *3.3.6* 所讲，AT32F421 与 SXX32F030 GPIO 基本相同，标准库结构相同，使用时调用正确库函数即可。但需要注意的是，AT32F421 在输出配置下不支持内部上下拉，这里与 SXX32F030 不同。

4.6 EXTI 线

AT32F421 与 SXX32F030 EXTI 相同，标准库结构相同，使用时调用正确 API 即可。

4.7 NVIC 中断配置

本节介绍了 NVIC 中断 (IRQ) 的配置。

1 在 AT32F421 系列中，NVIC 支持：

— 多达 28 个中断。

— 各中断的可编程优先级为 0-15（使用 4 个中断优先级位）。级别越高，优先级越低；级别 0 表示最高中断优先级。

— 将优先级值分为组优先级和子优先级两个域。

— 优先级动态变化。

Cortex-M4 异常由 CMSIS 功能管理：

根据优先级分组配置，使能并配置所选 IRQ 通道的抢占式优先级和子优先级。

下面的示例介绍了如何在 AT32F421 系列中配置 TMR3 溢出中断：

```
/* 为抢占式优先级配置两个位 */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2) ;
/* 使能 TMR3 全局中断（优先级更高）*/
  /* Enable the TMR3 global Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = TMR3_GLOBAL_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
```

2 在 SXX32F030 系列中，NVIC 支持：

— 多达 26 个中断

— 4 个可编程优先级（使用 2 个中断优先级位）

— 使能中断后，不得更改其优先级

Cortex-M0 异常由 CMSIS 功能管理：

使能并配置所选 IRQ 通道的优先级。优先级范围介于 0 到 3 之间。优先级值越小，优先级越高。

因此，应按如下步骤更新 TIM3 中断源代码的配置：

```
/* 使能 CEC 全局中断（优先级更高）*/
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

下表介绍了 AT32F421 与 SXX32F030 库之间的 NVIC 驱动程序 API 的对应关系。

表 13. AT32F421 与 SXX32F030 NVIC 驱动程序 API 对应关系

AT32F421MISC 驱动程序 API	SXX32F030 MISC 驱动程序 API
NVIC_Init(NVIC_InitType* NVIC_InitStruct);	NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
NVIC_SystemLPConfig(uint8_t LowPowerMode, FunctionalState NewState);	NVIC_SystemLPConfig(uint8_t LowPowerMode, FunctionalState NewState);
SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource);	SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource);
NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);	N/A
NVIC_SetVectorTable(uint32_t NVIC_VectTab, uint32_t Offset);	N/A

4.8 ADC 配置

本节介绍如何将现有代码从 SXX32F030 系列移植到 AT32F421 系列的示例。
下面的示例介绍了如何在 AT32F421 系列中将 ADC1 配置为连续转换通道 14：

```
/* ADCCLK = PCLK2/4 */
RCC_ADCCLKConfig(RCC_APB2CLK_Div4);
/* 使能 ADC 的 APB 接口时钟 */
RCC_APB2PeriphClockCmd(RCC_APB2PERIPH_ADC1, ENABLE);
/* 将 ADC1 配置为连续转换通道 14 */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrig_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 常规通道 14 配置 */
ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_55_5);
/* 使能 ADC1 的 DMA 接口 */
ADC_DMACtrl(ADC1, ENABLE);
/* 使能 ADC1 */
ADC_Ctrl(ADC1, ENABLE);
/* 使能 ADC1 复位校准寄存器 */
ADC_RstCalibration(ADC1);
/* 检查 ADC1 复位校准寄存器是否结束 */
while(ADC_GetResetCalibrationStatus(ADC1));
/* 启动 ADC1 校准 */
ADC_StartCalibration(ADC1);
/* 检查 ADC1 校准是否结束 */
while(ADC_GetCalibrationStatus(ADC1));
/* 启动 ADC1 软件转换 */
ADC_SoftwareStartConvCtrl(ADC1, ENABLE);
...
```

在 SXX32F030 系列中，用户必须按下列步骤更新此代码：

```
/* ADCCLK = PCLK/2 */
RCC_ADCCLKConfig(RCC_ADCCLK_PCLK_Div2);
/* 使能 ADC1 时钟 */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
/* ADC1 配置 */
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_TRGO;;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_ScanDirection = ADC_ScanDirection_Backward;
ADC_Init(ADC1, &ADC_InitStructure);
/* 以 55.5 个周期作为采样时间转换 ADC1 通道 1 */
ADC_ChannelConfig(ADC1, ADC_Channel_11 , ADC_SampleTime_55_5Cycles);
/* ADC 校准 */
ADC_GetCalibrationFactor(ADC1);
/* 循环模式下的 ADC DMA 请求 */
ADC_DMARequesxxodeConfig(ADC1, ADC_DMAMode_Circular);
/* 使能 ADC_DMA */
ADC_DMACmd(ADC1, ENABLE);
/* 使能 ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* 等待 ADCEN 标志 */
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_ADEN));
/* ADC1 常规软件启动转换 */
ADC_StartOfConversion(ADC1);
```

4.9 PWR 驱动程序

下表介绍了 AT32F421 与 SXX32F030 库之间的 PWR 驱动程序 API 的对应关系。利用 AT32F421 库中的函数替换 SXX32F030 中的对应函数后，便可轻松更新应用程序代码。

表 14. AT32F421 与 SXX32F030 PWR 驱动程序 API 对应关系

	AT32F421PWR 驱动程序 API	SXX32F030 PWR 驱动程序 API
接口配置	PWR_Reset(void);	PWR_DeInit(void);
	PWR_BackupAccessCtrl(FunctionalState NewState);	PWR_BackupAccessCmd(FunctionalState NewState);
PVD	PWR_PVDLevelConfig(uint32_t PWR_PVDLevel);	PWR_PVDLevelConfig(uint32_t PWR_PVDLevel);
	PWR_PVDCtrl(FunctionalState NewState);	PWR_PVDCmd(FunctionalState NewState);
唤醒	PWR_WakeUpPinCtrl(FunctionalState NewState);	PWR_WakeUpPinCmd(uint32_t PWR_WakeUpPin, FunctionalState NewState);
电源管理	N/A	PWR_EnterSleepMode(uint8_t PWR_SLEEPEntry);
	PWR_EnterSTOPMode(uint32_t PWR_Regulator, uint8_t PWR_STOPEntry);	PWR_EnterSTOPMode(uint32_t PWR_Regulator, uint8_t PWR_STOPEntry);
	PWR_EnterSTANDBYMode(void);	PWR_EnterSTANDBYMode(void);
标志管理	PWR_GetFlagStatus(uint32_t PWR_FLAG);	PWR_GetFlagStatus(uint32_t PWR_FLAG);
	PWR_ClearFlag(uint32_t PWR_FLAG);	PWR_ClearFlag(uint32_t PWR_FLAG);

4.10 I²C 驱动程序

AT32F421 器件中整合了全新的 I²C 特性。下表介绍了 SXX32F10x 与 AT32F421 库之间的 I²C 驱动程序 CRC 的对应关系。利用 AT32F421 库中的函数替换 SXX32F10x 中的对应函数后，便可更新应用程序代码。

表 15. AT32F421 与 SXX32F030 I²C 驱动程序 API 对应关系

	AT32F421I2C 驱动程序 API	SXX32F030 I2C 驱动程序 API
初始化和配置	I2C_DeInit(I2C_Type * I2Cx);	I2C_DeInit(I2C_TypeDef* I2Cx);
	I2C_Init(I2C_Type* I2Cx, I2C_InitType* I2C_InitStruct);	I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct);
	I2C_StructInit(I2C_InitType* I2C_InitStruct);	I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct);
	I2C_Cmd(I2C_Type* I2Cx, FunctionalState NewState);	I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_SoftwareResetCmd(I2C_Type* I2Cx, FunctionalState NewState);	I2C_SoftwareResetCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);

	AT32F421I2C 驱动程序 API	SXX32F030 I2C 驱动程序 API
	I2C_INTConfig(I2C_Type* I2Cx, uint16_t I2C_INT, FunctionalState NewState);	I2C_ITConfig(I2C_TypeDef* I2Cx, uint16_t I2C_IT,FunctionalState NewState);
	I2C_StretchClockCmd(I2C_Type* I2Cx, FunctionalState NewState);	I2C_StretchClockCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_StopModeCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_DualAddressCmd(I2C_Type* I2Cx, FunctionalState NewState);	I2C_DualAddressCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_OwnAddress2Config(I2C_Type* I2Cx, uint8_t Address);	I2C_OwnAddress2Config(I2C_TypeDef* I2Cx,uint16_t Address, uint8_t Mask);
	I2C_GeneralCallCmd(I2C_Type* I2Cx, FunctionalState NewState);	I2C_GeneralCallCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_SlaveByteControlCmd(I2C_TypeDef* I2Cx,FunctionalState NewState);
	N/A	I2C_SlaveAddressConfig(I2C_TypeDef* I2Cx,uint16_t Address);
	N/A	I2C_10BitAddressingModeCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_NACKPositionConfig(I2C_Type* I2Cx, uint16_t I2C_NACKPosition);	N/A
	I2C_ARPCmd(I2C_Type* I2Cx, FunctionalState NewState);	N/A
通信处理	N/A	I2C_AutoEndCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_ReloadCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_NumberOfBytesConfig(I2C_TypeDef* I2Cx, uint8_t Number_Bytes);
	N/A	I2C_MasterRequestConfig(I2C_TypeDef* I2Cx, uint16_t I2C_Direction);
	I2C_GenerateSTART(I2C_Type* I2Cx, FunctionalState NewState);	I2C_GenerateSTART(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_GenerateSTOP(I2C_Type* I2Cx, FunctionalState NewState);	I2C_GenerateSTOP(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_10BitAddressHeaderCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_AcknowledgeConfig(I2C_Type* I2Cx, FunctionalState NewState);	I2C_AcknowledgeConfig(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_GetAddressMatched(I2C_TypeDef* I2Cx);
	N/A	I2C_GetTransferDirection(I2C_TypeDef* I2Cx);
	N/A	I2C_TransferHandling(I2C_TypeDef* I2Cx, uint16_t Address, uint8_t Number_Bytes, uint32_t ReloadEndMode, uint32_t StartStopMode);

	AT32F421I2C 驱动程序 API	SXX32F030 I2C 驱动程序 API
	I2C_CheckEvent(I2C_Type* I2Cx, uint32_t I2C_EVENT);	N/A
	I2C_Send7bitAddress(I2C_Type* I2Cx, uint8_t Address, uint8_t I2C_Direction);	N/A
SMBUS 管理	I2C_SMBusAlertConfig(I2C_Type* I2Cx, uint16_t I2C_SMBusAlert);	I2C_SMBusAlertCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_ClockTimeoutCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_ExtendedClockTimeoutCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_IdleClockTimeoutCmd(I2C_TypeDef* I2Cx,FunctionalState NewState);
	N/A	I2C_TimeoutAConfig(I2C_TypeDef* I2Cx, uint16_t Timeout);
	N/A	I2C_TimeoutBConfig(I2C_TypeDef* I2Cx, uint16_t Timeout);
	I2C_CalculatePEC(I2C_Type* I2Cx, FunctionalState NewState);	I2C_CalculatePEC(I2C_TypeDef* I2Cx, FunctionalState NewState);
	N/A	I2C_PECRequestCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
	I2C_GetPEC(I2C_Type* I2Cx);	I2C_GetPEC(I2C_TypeDef* I2Cx);
数据传输	I2C_ReadRegister(I2C_Type* I2Cx, uint8_t I2C_Register);	I2C_ReadRegister(I2C_TypeDef* I2Cx, uint8_t I2C_Register);
	I2C_SendData(I2C_Type* I2Cx, uint8_t Data);	I2C_SendData(I2C_TypeDef* I2Cx, uint8_t Data);
	I2C_ReceiveData(I2C_Type* I2Cx);	I2C_ReceiveData(I2C_TypeDef* I2Cx);
DMA管理	I2C_DMACmd(I2C_Type* I2Cx, FunctionalState NewState);	I2C_DMACmd(I2C_TypeDef* I2Cx, uint32_t I2C_DMAReq, FunctionalState NewState);
	I2C_DMALastTransferCmd(I2C_Type* I2Cx, FunctionalState NewState);	N/A
中断和标志管理	I2C_GetFlagStatus(I2C_Type* I2Cx, uint32_t I2C_FLAG);	I2C_GetFlagStatus(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG);
	I2C_ClearFlag(I2C_Type* I2Cx, uint32_t I2C_FLAG);	I2C_ClearFlag(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG);
	I2C_GetINTStatus(I2C_Type* I2Cx, uint32_t I2C_INT);	I2C_GetITStatus(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
	I2C_ClearITPendingBit(I2C_Type* I2Cx, uint32_t I2C_INT);	I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, uint32_t I2C_IT);

4.11 SPI 驱动程序

与 SXX32F030 SPI 相比，AT32F421SPI 有一些区别。下表介绍了 AT32F421 与 SXX32F10x 库之间的 SPI 驱动程序 API 的对应关系。

表 16. AT32F421 与 SXX32F030 SPI 驱动程序 API 对应关系

	AT32F421SPI 驱动程序 API	SXX32F030 SPI 驱动程序 API
初始化和配置	SPI_I2S_Reset(SPI_Type* SPIx);	SPI_I2S_DeInit(SPI_TypeDef* SPIx);
	SPI_Init(SPI_Type* SPIx, SPI_InitType* SPI_InitStruct);	SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct);
	I2S_Init(SPI_Type* SPIx, I2S_InitType* I2S_InitStruct);	I2S_Init(SPI_TypeDef* SPIx, I2S_InitTypeDef* I2S_InitStruct);
	SPI_DefaultInitParaConfig(SPI_InitType* SPI_InitStruct);	SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct);
	I2S_DefaultInit(I2S_InitType* I2S_InitStruct);	I2S_StructInit(I2S_InitTypeDef* I2S_InitStruct);
	N/A	SPI_TIModeCmd(SPI_TypeDef* SPIx, FunctionalState NewState);
	N/A	SPI_NSSPulseModeCmd(SPI_TypeDef* SPIx, FunctionalState NewState);
	SPI_Enable(SPI_Type* SPIx, FunctionalState NewState);	SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState);
	I2S_Enable(SPI_Type* SPIx, FunctionalState NewState);	I2S_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState);
	SPI_FrameSizeConfig(SPI_Type* SPIx, uint16_t SPI_DataSize);	SPI_DataSizeConfig(SPI_TypeDef* SPIx,uint16_t SPI_DataSize);
	N/A	SPI_RxFIFOThresholdConfig(SPI_TypeDef* SPIx, uint16_t SPI_RxFIFOThreshold);
	N/A	SPI_BiDirectionalLineConfig(SPI_TypeDef* SPIx, uint16_t SPI_Direction);
	SPI_NSSInternalSoftwareConfig(SPI_Type* SPIx, uint16_t SPI_NSSInternalSoft);	SPI_NSSInternalSoftwareConfig(SPI_TypeDef* SPIx, uint16_t SPI_NSSInternalSoft);
	SPI_NSSHardwareOutputEnable(SPI_Type* SPIx, FunctionalState NewState);	SPI_SSOutputCmd(SPI_TypeDef* SPIx, FunctionalState NewState);
数据传输	SPI_I2S_TxData(SPI_Type* SPIx, uint16_t Data); uint16_t Data);	SPI_SendData8(SPI_TypeDef* SPIx, uint8_t Data); SPI_I2S_SendData16(SPI_TypeDef* SPIx, uint16_t Data);
	SPI_I2S_RxData(SPI_Type* SPIx);	SPI_ReceiveData8(SPI_TypeDef* SPIx); SPI_I2S_ReceiveData16(SPI_TypeDef* SPIx);
硬件 CRC 计算函数	N/A	SPI_CRCLengthConfig(SPI_TypeDef* SPIx, uint16_t SPI_CRCLength);
	SPI_TxCRC(SPI_Type* SPIx);	SPI_TransmitCRC(SPI_TypeDef* SPIx);
	SPI_CRCEN(SPI_Type* SPIx, FunctionalState NewState);	SPI_CalculateCRC(SPI_TypeDef* SPIx,FunctionalState NewState);

	AT32F421SPI 驱动程序 API	SXX32F030 SPI 驱动程序 API
	SPI_GetCRC(SPI_Type* SPIx, uint8_t SPI_CRC);	SPI_GetCRC(SPI_TypeDef* SPIx, uint8_t SPI_CRC);
	SPI_GetCRCPolynomial(SPI_Type* SPIx);	uint16_t SPI_GetCRCPolynomial(SPI_TypeDef* SPIx);
DMA传输	SPI_I2S_DMAEnable(SPI_Type* SPIx, uint16_t SPI_I2S_DMAReq, FunctionalState NewState);	SPI_I2S_DMACmd(SPI_TypeDef* SPIx, uint16_t SPI_I2S_DMAReq, FunctionalState NewState);
	N/A	SPI_LastDMATransferCmd(SPI_TypeDef* SPIx,uint16_t SPI_LastDMATransfer);
中断和标志管理	SPI_I2S_INTConfig(SPI_Type* SPIx, uint8_t SPI_I2S_INT, FunctionalState NewState);	SPI_I2S_ITConfig(SPI_TypeDef* SPIx, uint8_t SPI_I2S_IT, FunctionalState NewState);
	N/A	SPI_GetTransmissionFIFOStatus(SPI_TypeDef* SPIx);
	N/A	SPI_GetReceptionFIFOStatus(SPI_TypeDef* SPIx);
	SPI_I2S_GetFlagStatus(SPI_Type* SPIx, uint16_t SPI_I2S_FLAG);	SPI_I2S_GetFlagStatus(SPI_TypeDef* SPIx, uint16_t SPI_I2S_FLAG);(*)
	SPI_I2S_ClearFlag(SPI_Type* SPIx, uint16_t SPI_I2S_FLAG); void SPI_I2S_ClearINTPendingBit(SPI_Type* SPIx, uint8_t SPI_I2S_INT);	SPI_I2S_ClearFlag(SPI_TypeDef* SPIx, uint16_t SPI_I2S_FLAG);(*)
	SPI_I2S_GetITStatus(SPI_Type* SPIx, uint8_t SPI_I2S_INT);	SPI_I2S_GetITStatus(SPI_TypeDef* SPIx, uint8_t SPI_I2S_IT);

4.12 USART 驱动程序

与 SXX32F030 USART 相比，AT32F421USART 中有一些区别。下表介绍了 AT32F421 与 SXX32F030 库之间的 USART 驱动程序 API 的对应关系。

表 17. AT32F421 与 SXX32F030 USART 驱动程序 API 的对应关系

	AT32F421USART 驱动程序 API	SXX32F030 USART 驱动程序 API
初始化和配置	USART_Reset(USART_Type* USARTx);	USART_DeInit(USART_TypeDef* USARTx);
	USART_Init(USART_Type* USARTx, USART_InitType* USART_InitStruct);	USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct);
	USART_StructInit(USART_InitType* USART_InitStruct);	USART_StructInit(USART_InitTypeDef* USART_InitStruct);
	USART_ClockInit(USART_Type* USARTx, USART_ClockInitType* USART_ClockInitStruct);	USART_ClockInit(USART_TypeDef* USARTx, USART_ClockInitTypeDef* USART_ClockInitStruct);
	USART_ClockStructInit(USART_ClockInitType* USART_ClockInitStruct);	USART_ClockStructInit(USART_ClockInitTypeDef * USART_ClockInitStruct);
	USART_Cmd(USART_Type* USARTx, FunctionalState NewState);	USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);

	AT32F421USART 驱动程序 API	SXX32F030 USART 驱动程序 API
	N/A	USART_DirectionModeCmd(USART_TypeDef* USARTx, uint32_t USART_DirectionMode,FunctionalState NewState);
	USART_SetPrescaler(USART_Type* USARTx, uint8_t USART_Prescaler);	USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler);
	USART_OverSampling8Cmd(USART_Type* USARTx, FunctionalState NewState);	USART_OverSampling8Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
	USART_OneBitMethodCmd(USART_Type* USARTx, FunctionalState NewState);	USART_OneBitMethodCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_MSBFirstCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_DataInvCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_InvPinCmd(USART_TypeDef* USARTx, uint32_t USART_InvPin, FunctionalState NewState);
	N/A	USART_SWAPPinCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_ReceiverTimeOutCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_SetReceiverTimeOut(USART_TypeDef* USARTx, uint32_t USART_ReceiverTimeOut);
停机模式	N/A	USART_AutoBaudRateCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_AutoBaudRateConfig(USART_TypeDef* USARTx, uint32_t USART_AutoBaudRate);
波特率自动检	N/A	USART_AutoBaudRateCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_AutoBaudRateConfig(USART_TypeDef* USARTx, uint32_t USART_AutoBaudRate);
数据传输	USART_SendData(USART_Type* USARTx, uint16_t Data);	USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
	USART_ReceiveData(USART_Type* USARTx);	USART_ReceiveData(USART_TypeDef* USARTx);
多处理器通信	USART_SetAddress(USART_Type* USARTx, uint8_t USART_Address);	USART_SetAddress(USART_TypeDef* USARTx, uint8_t USART_Address);
	N/A	USART_MuteModeWakeUpConfig(USART_TypeDef* USARTx, uint32_t USART_WakeUp);
	N/A	USART_MuteModeCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_AddressDetectionConfig(USART_TypeDef * USARTx, uint32_t USART_AddressLength);

	AT32F421USART 驱动程序 API	SXX32F030 USART 驱动程序 API
LIN 模式	USART_LINBreakDetectLengthConfig(USART_Type* USARTx, uint16_t USART_LINBreakDetectLength);	USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, uint32_t USART_LINBreakDetectLength);
	USART_LINCmd(USART_Type* USARTx, FunctionalState NewState);	USART_LINCmd(USART_TypeDef* USARTx, FunctionalState NewState);
半双工模式	USART_HalfDuplexCmd(USART_Type* USARTx, FunctionalState NewState);	USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState NewState);
智能卡模式	USART_SmartCardCmd(USART_Type* USARTx, FunctionalState NewState);	void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	USART_SmartCardNACKCmd(USART_Type* USARTx, FunctionalState NewState);	USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState NewState);
	USART_SetGuardTime(USART_Type* USARTx, uint8_t USART_GuardTime);	USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime);
	N/A	USART_SetAutoRetryCount(USART_TypeDef* USARTx, uint8_t USART_AutoCount);
	N/A	void USART_SetBlockLength(USART_TypeDef* USARTx, uint8_t USART_BlockLength);
IrDA模式	USART_IrDAConfig(USART_Type* USARTx, uint16_t USART_IrDAMode);	USART_IrDAConfig(USART_TypeDef* USARTx, uint32_t USART_IrDAMode);
	USART_IrDACmd(USART_Type* USARTx, FunctionalState NewState);	USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState);
RS485 模式	N/A	USART_DECmd(USART_TypeDef* USARTx, FunctionalState NewState);
	N/A	USART_DEPolarityConfig(USART_TypeDef* USARTx, uint32_t USART_DEPolarity);
	N/A	USART_SetDEAssertionTime(USART_TypeDef* USARTx, uint32_t USART_DEAssertionTime);
	N/A	USART_SetDEDeassertionTime(USART_TypeDef * USARTx, uint32_t USART_DEDeassertionTime);
DMA传输	USART_DMACmd(USART_Type* USARTx, uint16_t USART_DMAReq, FunctionalState NewState);	USART_DMACmd(USART_TypeDef* USARTx, uint32_t USART_DMAReq, FunctionalState NewState);
中断和标志管理	v USART_INTConfig(USART_Type* USARTx, uint16_t USART_INT, FunctionalState NewState);	USART_ITConfig(USART_TypeDef* USARTx, uint32_t USART_IT, FunctionalState NewState);
	N/A	USART_RequestCmd(USART_TypeDef* USARTx, uint32_t USART_Request, FunctionalState NewState);
	N/A	USART_OverrunDetectionConfig(USART_TypeDef * USARTx, uint32_t USART_OVRDetection);

	AT32F421USART 驱动程序 API	SXX32F030 USART 驱动程序 API
	USART_GetFlagStatus(USART_Type* USARTx, uint16_t USART_FLAG);	USART_GetFlagStatus(USART_TypeDef* USARTx, uint32_t USART_FLAG);
	USART_ClearFlag(USART_Type* USARTx, uint16_t USART_FLAG);	USART_ClearFlag(USART_TypeDef* USARTx, uint32_t USART_FLAG);
	USART_GetITStatus(USART_Type* USARTx, uint16_t USART_INT);	USART_GetITStatus(USART_TypeDef* USARTx, uint32_t USART_IT);
	USART_ClearITPendingBit(USART_Type* USARTx, uint16_t USART_INT);	USART_ClearITPendingBit(USART_TypeDef* USARTx, uint32_t USART_IT);

4.13 IWDG 驱动程序

AT32F421 和 SXX32F030 器件上的现有 IWDG 具有相同规格，不过，SXX32F030 系列中具备窗口功能特性，可用于检测外部振荡器上的过频率。下表列出了 IWDG 驱动程序 API。

表 18. AT32F421 与 SXX32F030 IWDG 驱动程序 API 对应关系

	AT32F421IWDG 驱动程序 API	SXX32F030 IWDG 驱动程序 API
预分频器和计数器配置	IWDG_KeyRegWrite(uint16_t IWDG_WriteAccess);	IWDG_WriteAccessCmd(uint16_t IWDG_WriteAccess);
	IWDG_SetPrescaler(uint8_t IWDG_Prescaler);	IWDG_SetPrescaler(uint8_t IWDG_Prescaler);
	IWDG_SetReload(uint16_t Reload);	IWDG_SetReload(uint16_t Reload);
	void IWDG_ReloadCounter(void);	IWDG_ReloadCounter(void);
	N/A	IWDG_SetWindowValue(uint16_t WindowValue);
IWDG激活	IWDG_Enable(void);	IWDG_Enable(void);
标志管理	IWDG_GetFlagStatus(uint16_t IWDG_FLAG);	IWDG_GetFlagStatus(uint16_t IWDG_FLAG);

5 版本历史

表 19.文档版本历史

日期	版本	变更
2020.08.03	1.0.0	最初版本
2020.10.21	1.0.1	格式调整
2021.03.04	1.0.2	新增*PWR*

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，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) 汽车应用或汽车环境；(D) 航天应用或航天环境，且/或(E) 武器。因雅特力产品不是为前述应用设计的，而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，风险由购买者单独承担，并且独力负责在此类相关使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。