AN13120
如何用 SSARC 在 RT1170 唤醒之后配置外设

Rev. 0 — February 3, 2021　　Application Note

by:　NXP Semiconductors

目录

1 介绍

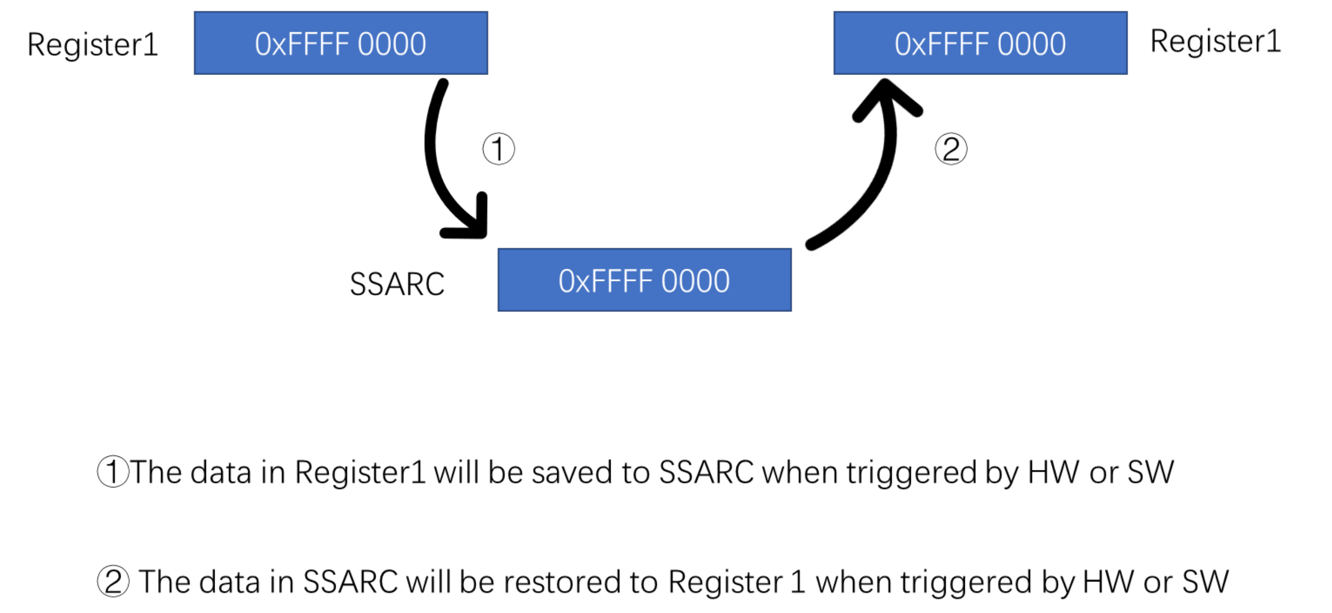
i.MX RT1170 是一款突破性的跨界处理器，它把最高运行频率提高到了 1 GHz，而且还在结合了高性能运算和多媒体功能的同时，增强了可用性及实时功能。

本应用文档将会介绍如何在唤醒之后用 SSARC 配置外设。

本文档内所使用的硬件是 RT1170 EVK RevC1 (本文档简称为 EVK)， 软件是基于 IAR IDE 的 SDK 2.9.0 。

2 SSARC 概述

State Save and Restore Controller（SSARC 状态保存和恢复控制器）能在断电前可以把一个功能寄存器的数值保存到一个内存里（在 LPSR 域）然后在通电后从内存中恢复这个寄存器。这种模式可以在 CPU 唤醒之前配置外设，当 CPU 唤醒之后，就可以直接使用外设而不必初始化了。图 1 展示了 SSARC 的基本功能。



Register1
0xFFFF 0000
0xFFFF 0000
Register1
①
②
SSARC
0xFFFF 0000
①The data in Register1 will be saved to SSARC when triggered by HW or SW
② The data in SSARC will be restored to Register 1 when triggered by HW or SW

图 1. SSARC 基本功能和工作流程

2.1 描述符（descriptor）

描述符是 SSARC 里最基本的元素, SSARC 有最多 1024 个描述符。描述符执行的操作都与 CPU 相似，比如读写(read and write)，延迟(delay)和轮询(polling)。它支持 4 组操作和 7 种操作类型。

- **操作（operations）**

    — 不保存和不恢复（Save Disable and Restore Disable）
    — 保存和不恢复 （Save Enable and Restore Disable）
    — 不保存和恢复（Save Disable and Restore Enable）
    — 保存和恢复（Save Enable and Restore Enable）
- **操作类型（operation types）**
    — 读取数据然后写入（Read Value and Write Back）
    — 写入一个固定值（Write a Fixed Value）
    — 或逻辑（Read a register or with a value then write it back (OR)）
    — 与逻辑（Read a register and with a value then write it back (AND)）
    — 延迟周期（Delay)
    — 轮询 0（Read a register until the bit or bits in it changed to 0）
    — 轮询 1（Read a register until the bit or bits in it changed to 1）

2.2 组（Group）

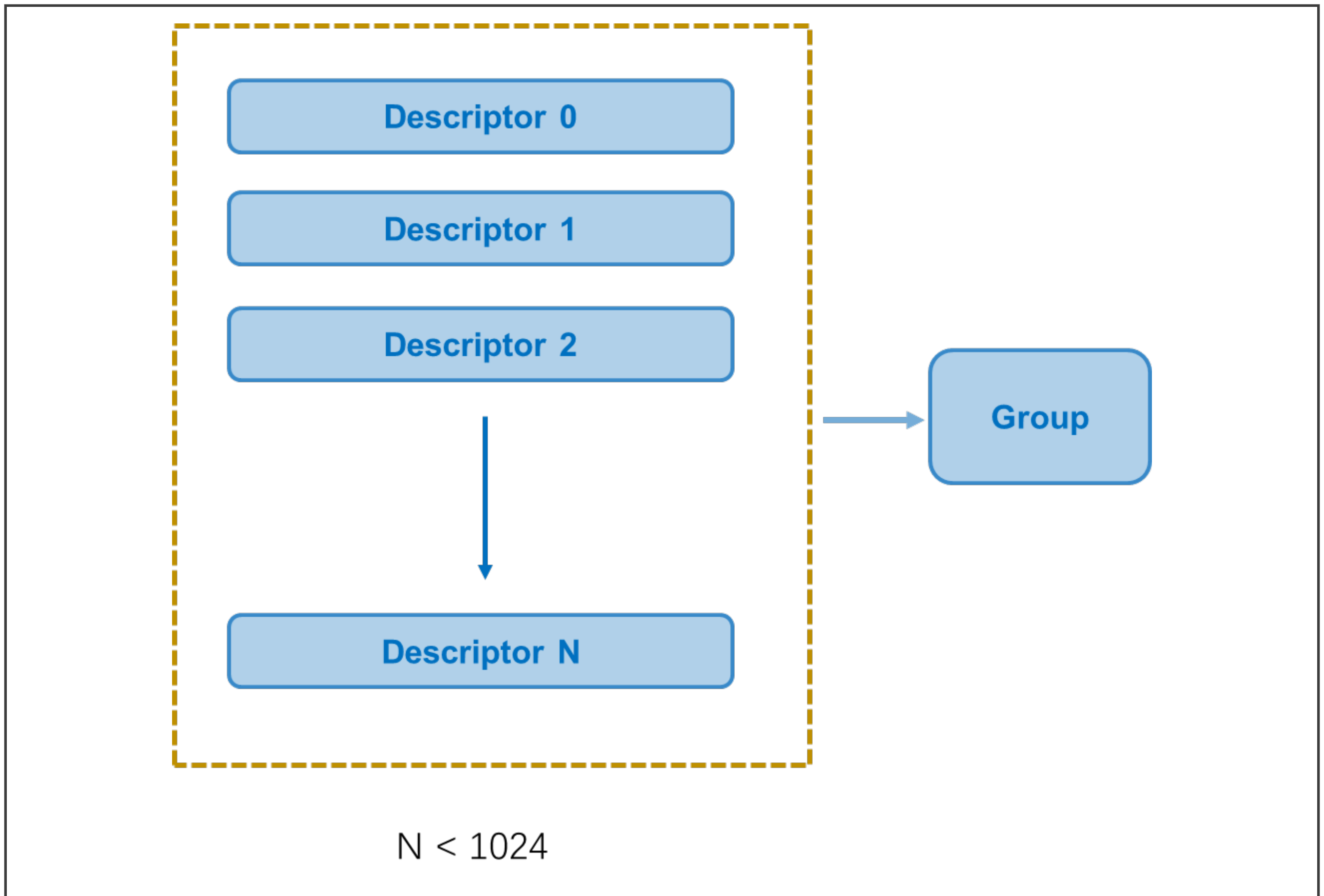


图 2. **一些描述符组成一个组**

1024 个描述符最多被分成 16 个组，每个组里面都包含着几个连续的描述符。描述符是由组控制的，组可以由软件或硬件触发。每个组的保存和恢复都有它自己的优先级，分为从 0 到 15 这 16 个优先级，其中 0 为最高优先级。通常保存操作不需要注意优先级，但是恢复操作的优先级是很重要的。比如，当一个外设需要用管脚，那么这个管脚在初始化的时候就应该要有更高的优先级。

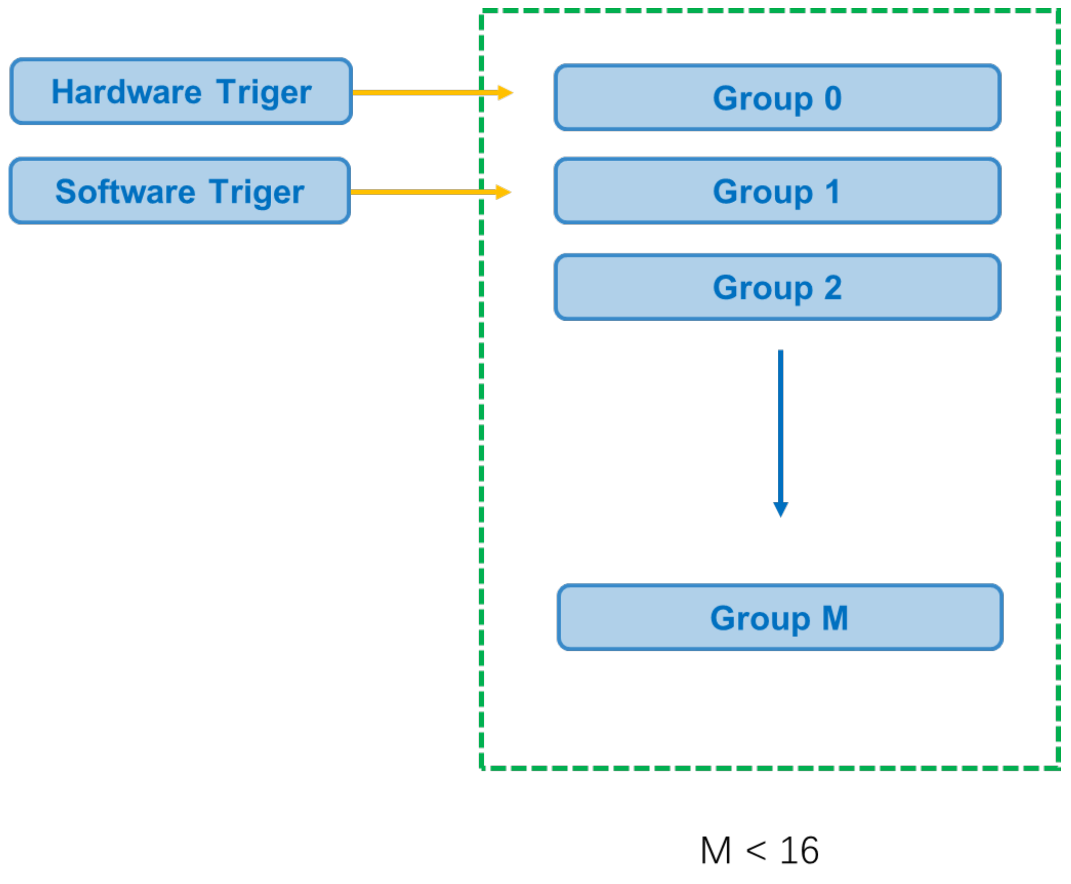
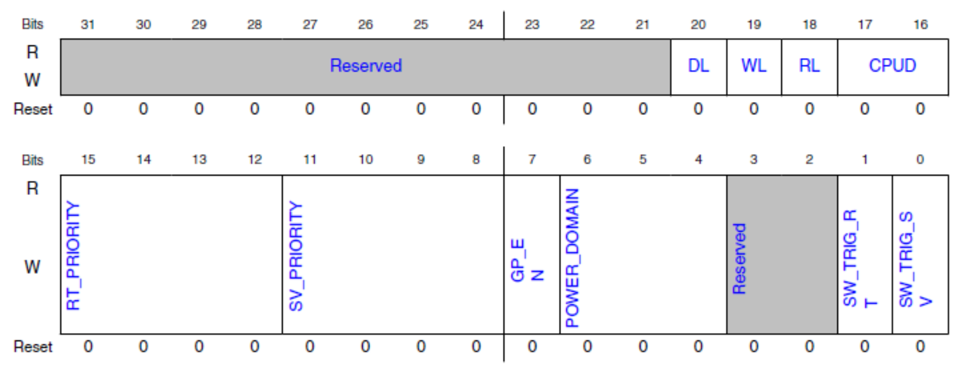


图 3. 组可以由软件或者硬件来触发

2.2.1 用软件触发组

软件触发组的方法可以被用来检查在进入低功耗模式之前 SSARC 是否被设置正确。 DESC_CTRL1_x (x 代表组的序号) 中的 bit0 和 bit1 可以用软件来触发保存和恢复。



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved											DL	WL	RL	CPUD	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RT_PRIORITY				SV_PRIORITY				GP_EN	POWER_DOMAIN			Reserved		SW_TRIG_RT	SW_TRIG_SV
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 4. 软件触发在 DESC_CTRL1_x 中的组

除此之外，以下的 API 也可以用来保存和恢复：

*void SSARC_TriggerSoftwareRequest(SSARC_LP_Type *base, uint8_t groupID, ssarc_software_trigger_mode_t mode)*

在 SDK 中有一个软件触发的例子：

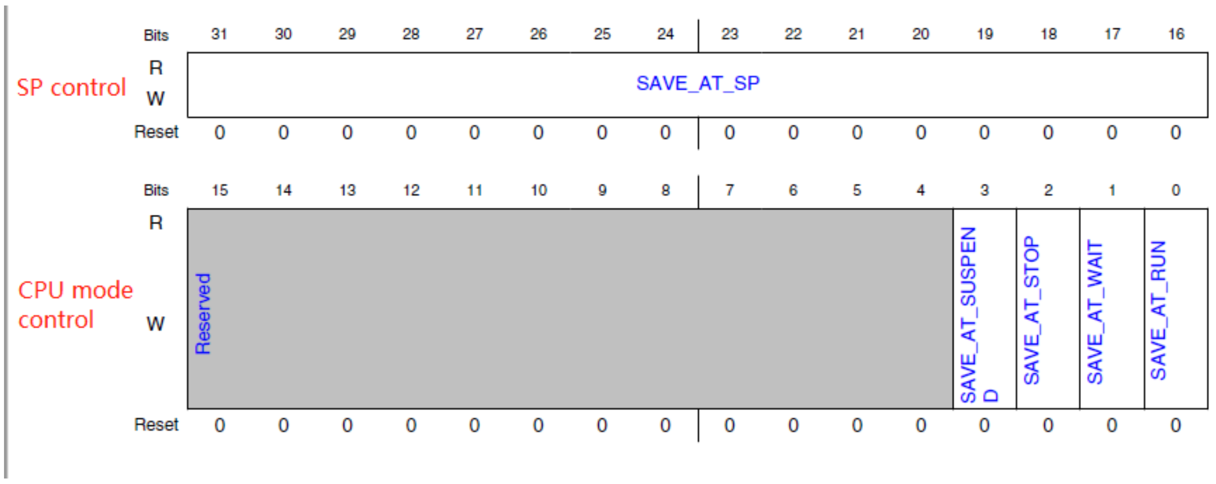
boards\evkmimxrt1170\driver_examples\ssarc\software_trigger

2.2.2 用硬件触发组

组可以由 BPC 触发。当某一个电源域关闭的时候，这个域内的外设设置信息就会丢失，在这种情况下，SSARC 就会自动保存数据然后在通电后恢复数据。BPC 会给 SSARC 发送保存和恢复的请求然后完成握手。因为 SP 和 CPU 模式可以触发 BPC 断电，所以触发信号可以由 SP 和 CPU 发出。

以 SP 模式下的 BPC2 控制 WAKEUP MIXSP 为例子，通常 WAKEUP MIX 会在 SP11 之后关闭，当其他的 SP 转换为 SP11 或在 SP11 之后，BPC 会向 SSARC 发送保存请求，接着组就会收到这个请求然后在描述符里进行操作。当 SP 转换之后，比如转换成 SP1，BPC2 会向 SSARC 发送恢复的请求，然后就会执行描述符里的操作进行恢复。

如果 BPC0 中的 MEGA MIX 是被 CM7 的 CPU 模式控制的，当 CM7 进入低功耗模式的时候，BPC0 会向 SSARC 发送保存请求。如果 CM7 已经唤醒，BPC0 会向 SSARC 发送恢复请求。对于低功耗模式，有 WAIT,STOP 和 SUSPEND 三种情况，进入这些模式需要根据 BPC_SSAR_SAVE_CTRL 的设定来发送请求。



SP control
Bits 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
R
W
SAVE_AT_SP
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Bits 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CPU mode control
R
W
Reserved
SAVE_AT_SUSPEND
SAVE_AT_STOP
SAVE_AT_WAIT
SAVE_AT_RUN
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

图 5. BPC_SSAR_SAVE_CTRL

寄存器 `BPC_SSAR_SAVE_CTRL` 和 `BPC_SSAR_RESTORE_CTRL` 可以控制 SSARC 的请求。寄存器 **BPC_SSAR_ACK_CTRL** 可以用来控制延迟和等待请求的行为。

表 1. 电源域与 BPC 的映射关系

Power domain	Assignment
BPC0	MEGA MIX
BPC1	DISPLAY MIX
BPC2	WAKEUP MIX
BPC3	LPSRMIX
BPC4	MIPIPHY
BPC5	Virtual
BPC6	Virtual
BPC7	Virtual

NOTE

虚拟的电源域并没有物理意义，但是可以用来触发 SSARC 的保存和恢复，状态和数据。

3 恢复的顺序和技巧

3.1 外设的时钟源

因为大多数 SSARC 的情况是在低功耗模式下进行的，所以第一步需要检查时钟。检查时钟的基本步骤是：

1. 检查时钟源是否在目标唤醒 SP 下可用。
2. 检查 LPCG 的状态，LPCG 是由 SP 或 CPU 模式控制的。
    - 如果是由 SP 控制的，就检查在目标唤醒 SP 下是否可用。
    - 如果是 CPU 模式控制的，就需要检查 CPU 模式的睡眠设定。
3. 如 图 6 所示，唤醒流程的最后一步就是 CPU 模式，这就表示当 SSARC 要初始化或者恢复一个外设时，LPCG 可能会屏蔽掉时钟，是因为 CPU 仍处于低功耗模式。因此，请确保 LPCG 在低功耗模式下是可用的。

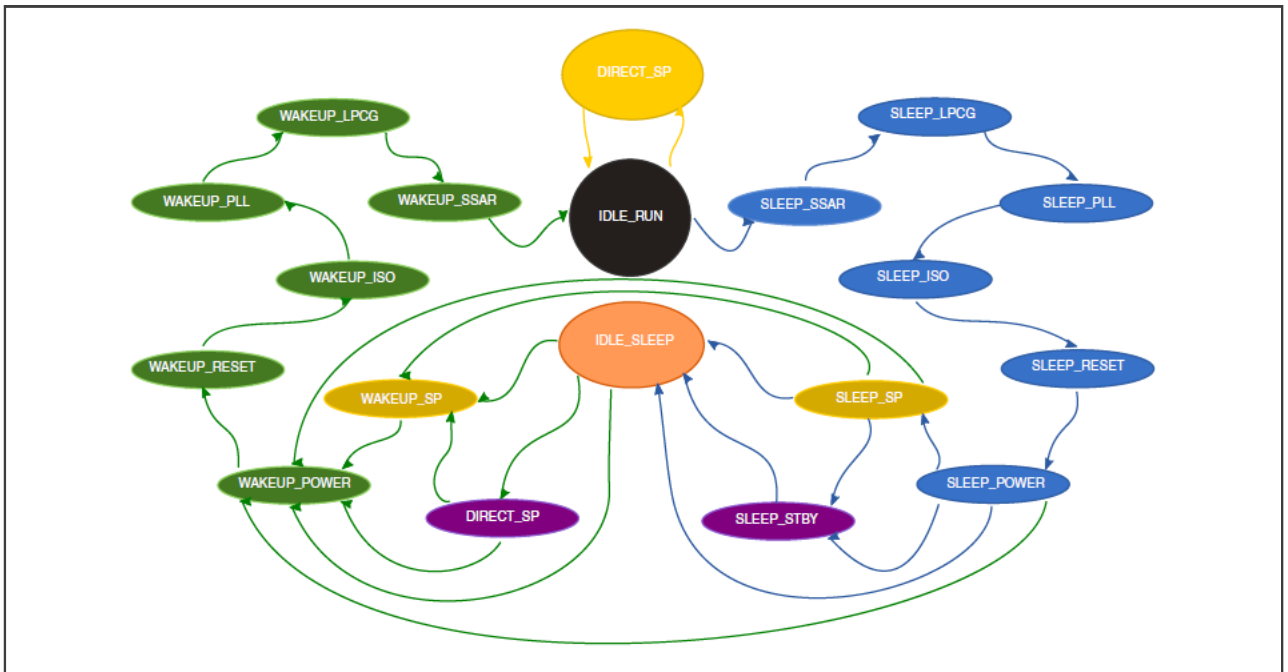


图 6. CPU 模式转换流程

更多关于步骤的细节可以参考《RT1170 时钟和低功耗特性》（文档 AN13104）的 4.13 节。

3.2 恢复的方法和顺序

如描述符（descriptor）所描述的，有 7 种描述符的操作。对于不同的外设，有不同种类的操作。比如 GPIO PIN，可以用保存和恢复使能下的数据读写，因为 GPIO 的初始化不需要考虑操作的顺序。基本的流程是:

- 恢复在 IOMUXC_SW_MUX_CTRL 寄存器和 IOMUXC_SW_PAD_CTRL 寄存器的 PIN。这些寄存器通常都可以使用保存和恢复使能下的数据读写。
- 检查外设初始化的顺序。对于一些外设，需要按时间的顺序来初始化，比如像 FlexSPI，需要给外设配置为运行模式，然后配置控制寄存器，解锁 LUT，更新 LUT，最后锁住 LUT。所以，对于 FlexSPI 的恢复，更多的工作是用来初始化。因此对于类似这样的外设，使用不保存和恢复操作中的写入固定值是更好的选择。

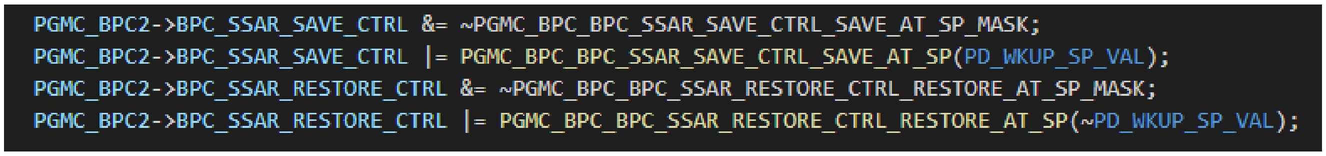
4 示例

举两个例子来展示如何在唤醒之后用 SSARC 来恢复外设。以 FlexSPI 为例，进入 SP11，Suspend STBY 模式，在 SP11 WAKEUP MIX 之下会断电，外设也会随之断电。示例中的 FlexSPI 是属于 WAKEUP MIX。在唤醒之后，CPU 会在 SP1 运行，这时 WAKEUP MIX 是通电的，然后 CPU 会跳转到一个在 FlexSPI 地址中的一个运行测试函数并用 GPIO 使 LED 闪烁。

关于如何跳转到这个测试函数，请参考《RT1170 时钟和低功耗特性》（文档 AN13104）的 4.15 节。

4.1 BPC2 配置为保存和恢复的触发源

配置 BPC 为 SP 控制模式然后发送基于 SP 设定的保存和恢复请求。PD_WKUP_SP_VAL 是 0xf800 表示 WAKEUP MIX 在 SP11-SP15 之间会断电。



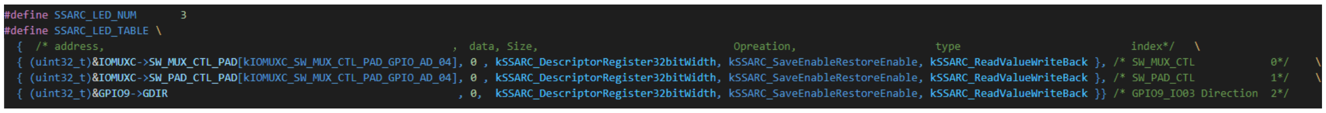
```
PGMC_BPC2->BPC_SSAR_SAVE_CTRL &= ~PGMC_BPC_BPC_SSAR_SAVE_CTRL_SAVE_AT_SP_MASK;
PGMC_BPC2->BPC_SSAR_SAVE_CTRL |= PGMC_BPC_BPC_SSAR_SAVE_CTRL_SAVE_AT_SP(PD_WKUP_SP_VAL);
PGMC_BPC2->BPC_SSAR_RESTORE_CTRL &= ~PGMC_BPC_BPC_SSAR_RESTORE_CTRL_RESTORE_AT_SP_MASK;
PGMC_BPC2->BPC_SSAR_RESTORE_CTRL |= PGMC_BPC_BPC_SSAR_RESTORE_CTRL_RESTORE_AT_SP(~PD_WKUP_SP_VAL);
```

图 7. BPC2 配置为保存和恢复的触发源

4.2 唤醒后恢复 GPIO 管脚

在 RT1170 EVK 上有一个连接到 GPIO_AD_04 的板载 LED。这个管脚可以用来控制 LED 的闪烁。在 GPIO_AD_04 里的 GPIO 是 GPIO09_IO03。关于 pad 和 pin 的设定，可以用保存和恢复操作下的读写数据.

- 添加一个描述符到 IOMUXC_SW_MUX_CTL_PAD_GPIO_AD_04。
- 添加一个描述符到 IOMUXC_SW_PAD_CTL_PAD_GPIO_AD_04。
- 添加一个描述符到 GPIO9_GDIR 用来配置方向寄存器。



```
#define SSARC_LED_NUM       3
#define SSARC_LED_TABLE \
  {  /* address,                                              , data, Size,                               Opreation,                          type                                 index*/   \
  { (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_AD_04], 0 , kSSARC_DescriptorRegister32bitWidth, kSSARC_SaveEnableRestoreEnable, kSSARC_ReadValueWriteBack }, /* SW_MUX_CTL           0*/    \
  { (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_AD_04], 0 , kSSARC_DescriptorRegister32bitWidth, kSSARC_SaveEnableRestoreEnable, kSSARC_ReadValueWriteBack }, /* SW_PAD_CTL           1*/    \
  { (uint32_t)&GPIO9->GDIR                                              , 0,  kSSARC_DescriptorRegister32bitWidth, kSSARC_SaveEnableRestoreEnable, kSSARC_ReadValueWriteBack }} /* GPIO9_IO03 Direction  2*/
```

图 8. 为 GPIO 管脚配置 SSARC

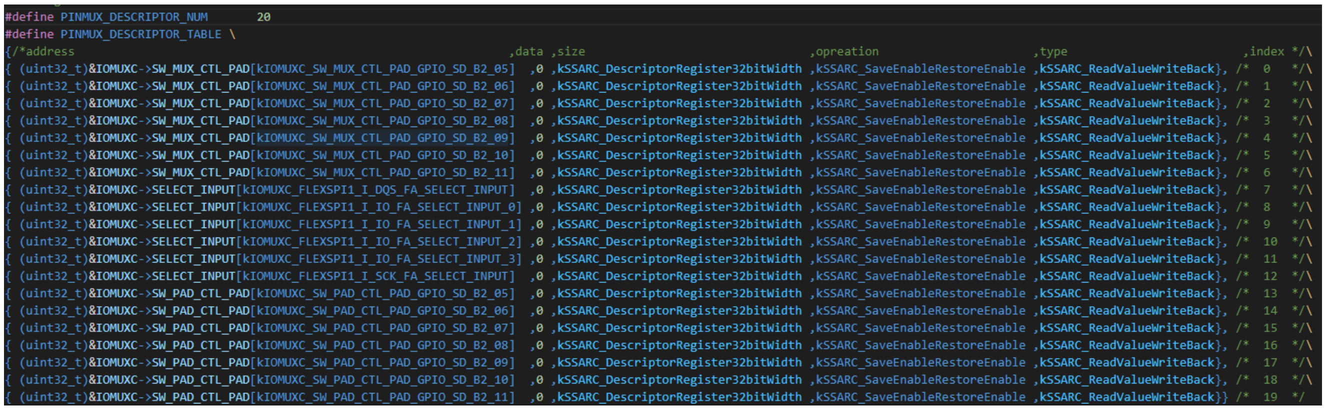
这三个描述符组成了一个组（在本文档称为 GroupA），这个组可以被 BPC2 触发。当系统被唤醒的时候，GPIO9_IO03 会被恢复，然后用触发寄存器使 LED 闪烁。

BOARD_USER_LED_GPIO->DR_TOGGLE = 1<<BOARD_USER_LED_GPIO_PIN;

4.3 唤醒后恢复 FlexSPI

4.3.1 恢复管脚设定

恢复 FlexSPI 的第一步是恢复 IOMUXC_SW_MUX_CTRL, IOMUXC_SW_PAD_CTRL 和 SELECT_INPUT 的管脚。在 SELECT_INPUT 中，不是所有的管脚都需要被恢复，但在 IOMUXC_SW_MUX_CTRL 和 IOMUXC_SW_PAD_CTRL 中，所有的管脚都需要恢复。管脚的恢复可以用保存和恢复操作下的读写数据来完成。图 9 表示了这些描述符形成一个组（本文档称为 GroupB）。

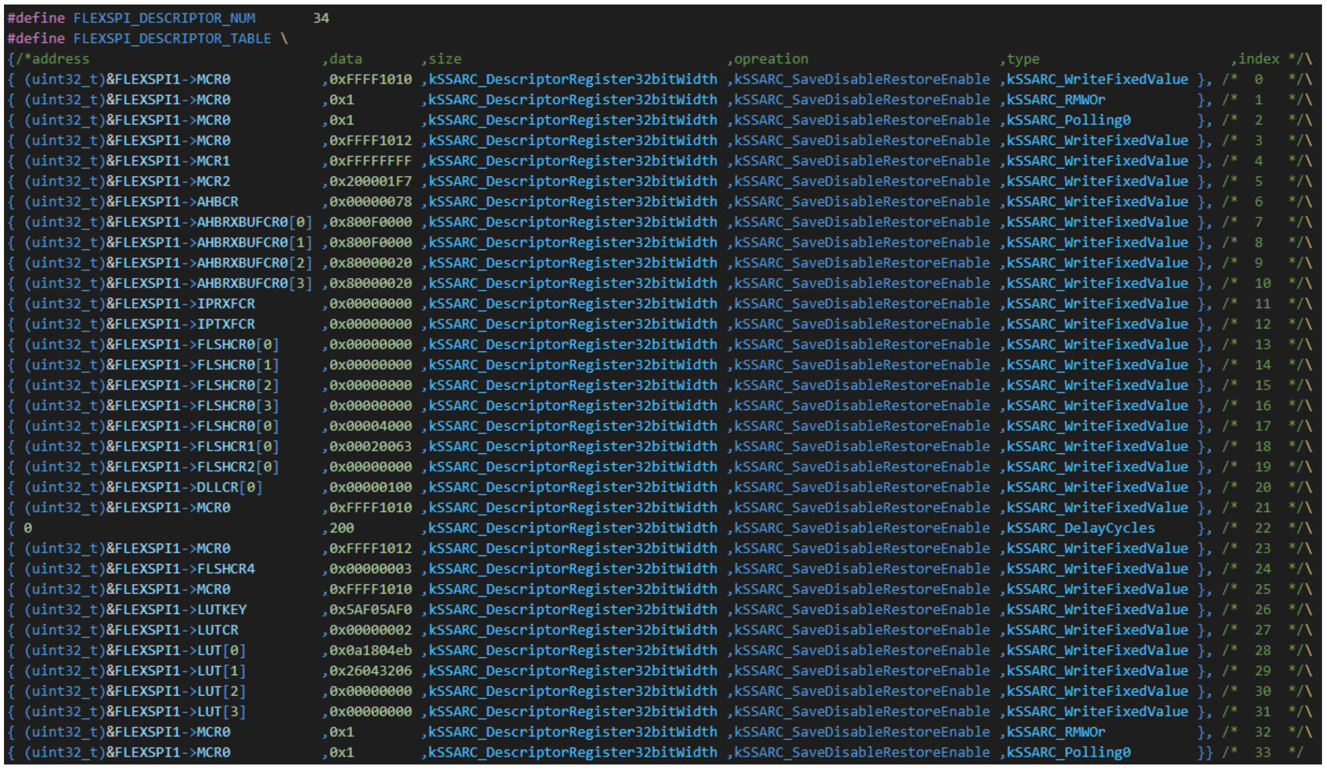


```
#define PINMUX_DESCRIPTOR_NUM       20
#define PINMUX_DESCRIPTOR_TABLE \
{/*address                                                           ,data ,size                              ,opreation                        ,type                        ,index */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_05]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  0   */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_06]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  1   */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_07]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  2   */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_08]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  3   */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_09]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  4   */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_10]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  5   */\
{ (uint32_t)&IOMUXC->SW_MUX_CTL_PAD[kIOMUXC_SW_MUX_CTL_PAD_GPIO_SD_B2_11]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  6   */\
{ (uint32_t)&IOMUXC->SELECT_INPUT[kIOMUXC_FLEXSPI1_I_DQS_FA_SELECT_INPUT]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  7   */\
{ (uint32_t)&IOMUXC->SELECT_INPUT[kIOMUXC_FLEXSPI1_I_IO_FA_SELECT_INPUT_0] ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  8   */\
{ (uint32_t)&IOMUXC->SELECT_INPUT[kIOMUXC_FLEXSPI1_I_IO_FA_SELECT_INPUT_1] ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  9   */\
{ (uint32_t)&IOMUXC->SELECT_INPUT[kIOMUXC_FLEXSPI1_I_IO_FA_SELECT_INPUT_2] ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  10  */\
{ (uint32_t)&IOMUXC->SELECT_INPUT[kIOMUXC_FLEXSPI1_I_IO_FA_SELECT_INPUT_3] ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  11  */\
{ (uint32_t)&IOMUXC->SELECT_INPUT[kIOMUXC_FLEXSPI1_I_SCK_FA_SELECT_INPUT]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  12  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_05]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  13  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_06]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  14  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_07]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  15  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_08]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  16  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_09]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  17  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_10]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}, /*  18  */\
{ (uint32_t)&IOMUXC->SW_PAD_CTL_PAD[kIOMUXC_SW_PAD_CTL_PAD_GPIO_SD_B2_11]  ,0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveEnableRestoreEnable ,kSSARC_ReadValueWriteBack}} /*  19  */
```

图 9. 给每个管脚加一个描述符，包括 IOMUXC_SW_MUX_CTRL，IOMUXC_SW_PAD_ACTRL 和 SELECT_INPUT

4.3.2 恢复 FlexSPI

因为 WAKEUP MIX 被断电了，所以 FlexSPI 需要被重新初始化。为了实现这个功能，简单的保存和恢复并不有效因为配置过程有恢复顺序的要求。比如，它需要在运行模式下配置，需要软件恢复，LUT 配置时需要输入一个密码去解锁和锁住 LUT。在配置的过程中，一些标记位需要用来检查轮询的工作状态是否稳定。所以在保存和恢复使能的操作下是不能恢复 FlexSPI 的。关于如何配置 FlexSPI，请参见 AN13112。图 10 展示了在不同的操作种类下的 FlexSPI 恢复流程，这些描述符形成一个组（本文档称为 GroupC）。



```
#define FLEXSPI_DESCRIPTOR_NUM       34
#define FLEXSPI_DESCRIPTOR_TABLE \
{/*address                        ,data       ,size                                 ,opreation                        ,type                        ,index */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0xFFFF1010 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  0   */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0x1        ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_RMWOr           }, /*  1   */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0x1        ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_Polling0        }, /*  2   */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0xFFFF1012 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  3   */\
{ (uint32_t)&FLEXSPI1->MCR1           ,0xFFFFFFFF ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  4   */\
{ (uint32_t)&FLEXSPI1->MCR2           ,0x200001F7 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  5   */\
{ (uint32_t)&FLEXSPI1->AHBCR          ,0x00000078 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  6   */\
{ (uint32_t)&FLEXSPI1->AHBRXBUFCR0[0] ,0x800F0000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  7   */\
{ (uint32_t)&FLEXSPI1->AHBRXBUFCR0[1] ,0x800F0000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  8   */\
{ (uint32_t)&FLEXSPI1->AHBRXBUFCR0[2] ,0x80000020 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  9   */\
{ (uint32_t)&FLEXSPI1->AHBRXBUFCR0[3] ,0x80000020 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  10  */\
{ (uint32_t)&FLEXSPI1->IPRXFCR        ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  11  */\
{ (uint32_t)&FLEXSPI1->IPTXFCR        ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  12  */\
{ (uint32_t)&FLEXSPI1->FLSHCR0[0]     ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  13  */\
{ (uint32_t)&FLEXSPI1->FLSHCR0[1]     ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  14  */\
{ (uint32_t)&FLEXSPI1->FLSHCR0[2]     ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  15  */\
{ (uint32_t)&FLEXSPI1->FLSHCR0[3]     ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  16  */\
{ (uint32_t)&FLEXSPI1->FLSHCR0[0]     ,0x00004000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  17  */\
{ (uint32_t)&FLEXSPI1->FLSHCR1[0]     ,0x00020063 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  18  */\
{ (uint32_t)&FLEXSPI1->FLSHCR2[0]     ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  19  */\
{ (uint32_t)&FLEXSPI1->DLLCR[0]       ,0x00000100 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  20  */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0xFFFF1010 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  21  */\
{ 0                                   ,200        ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_DelayCycles     }, /*  22  */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0xFFFF1012 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  23  */\
{ (uint32_t)&FLEXSPI1->FLSHCR4        ,0x00000003 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  24  */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0xFFFF1010 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  25  */\
{ (uint32_t)&FLEXSPI1->LUTKEY         ,0x5AF05AF0 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  26  */\
{ (uint32_t)&FLEXSPI1->LUTCR          ,0x00000002 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  27  */\
{ (uint32_t)&FLEXSPI1->LUT[0]         ,0x0a1804eb ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  28  */\
{ (uint32_t)&FLEXSPI1->LUT[1]         ,0x26043206 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  29  */\
{ (uint32_t)&FLEXSPI1->LUT[2]         ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  30  */\
{ (uint32_t)&FLEXSPI1->LUT[3]         ,0x00000000 ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_WriteFixedValue }, /*  31  */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0x1        ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_RMWOr           }, /*  32  */\
{ (uint32_t)&FLEXSPI1->MCR0           ,0x1        ,kSSARC_DescriptorRegister32bitWidth ,kSSARC_SaveDisableRestoreEnable ,kSSARC_Polling0        }} /*  33  */
```

图 10. 恢复 FlexSPI

4.3.3 配置 Group

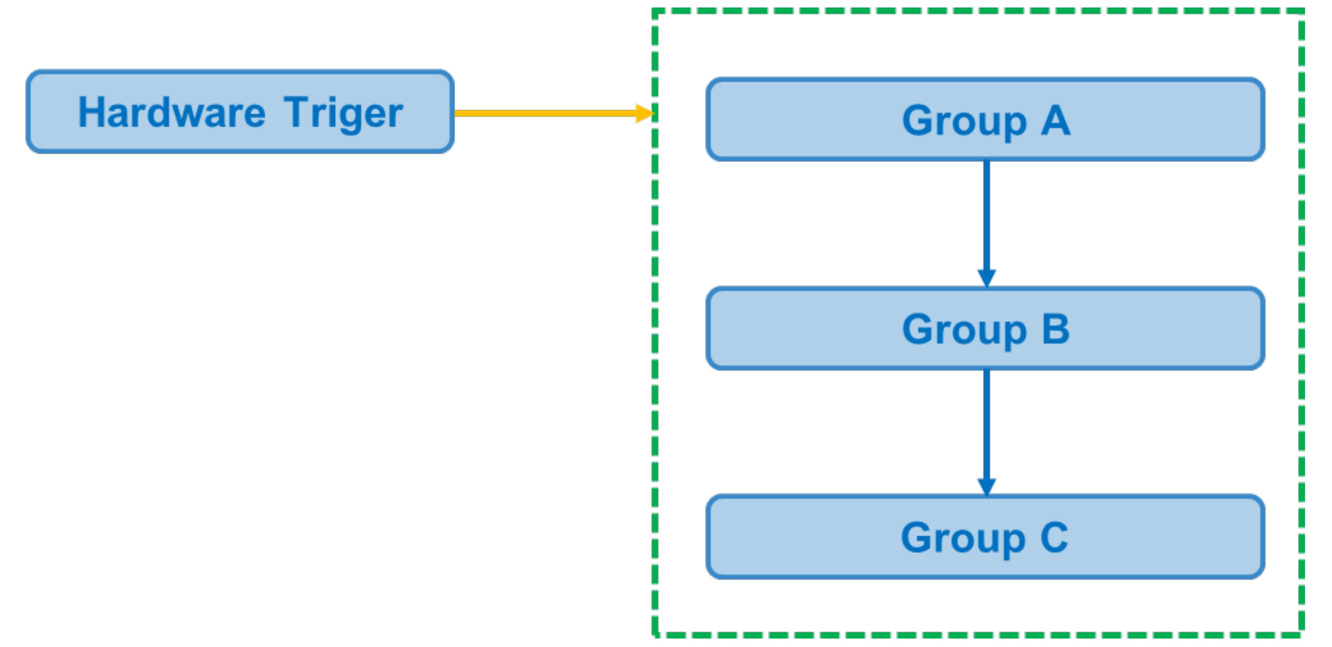
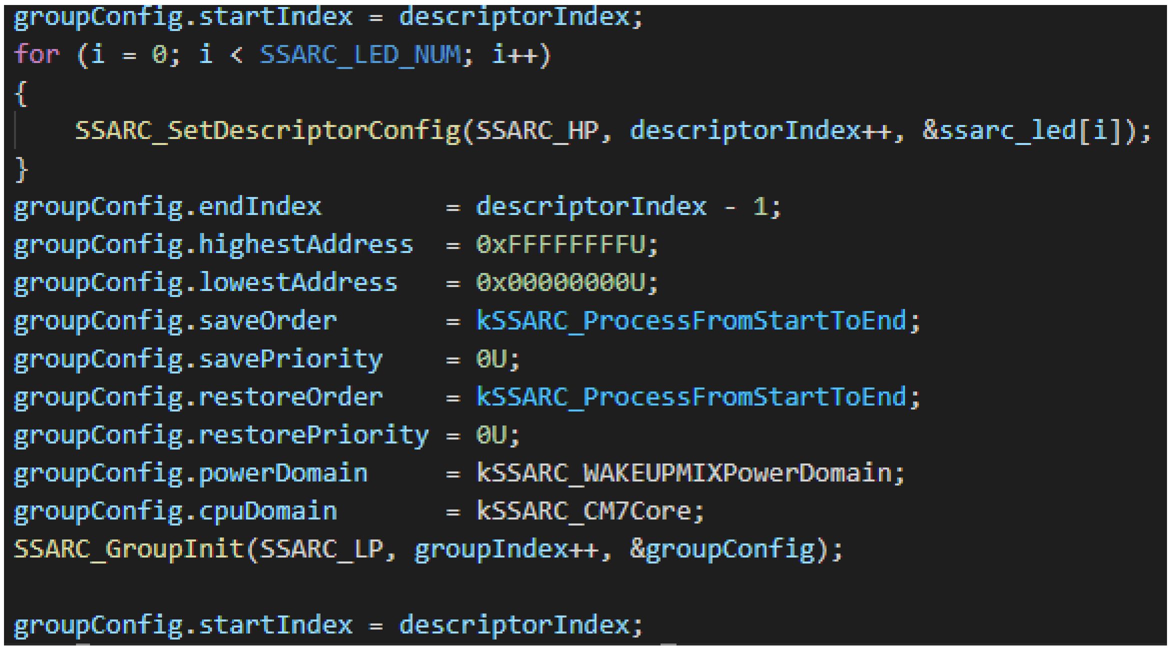


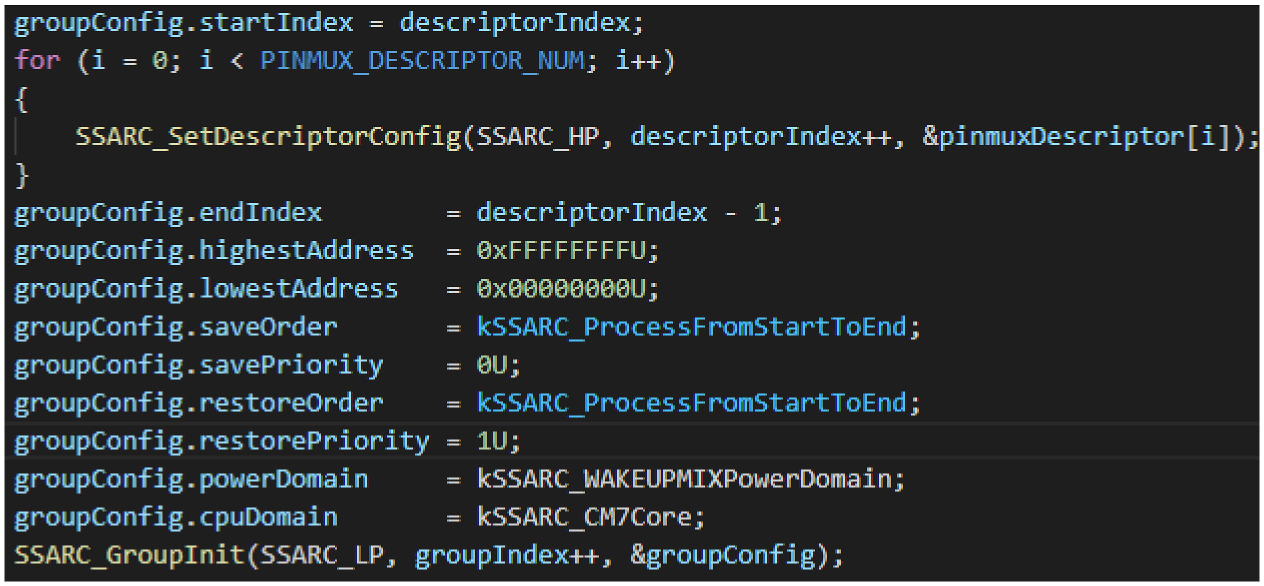
图 11. Group 恢复顺序

组的恢复优先级是一个非常重要的设定，这会影响到恢复的顺序，通常管脚恢复通常是第一步，接下来是外设。举例说明，Group A 有最高优先级，接下来依次是 Group B，Group C。



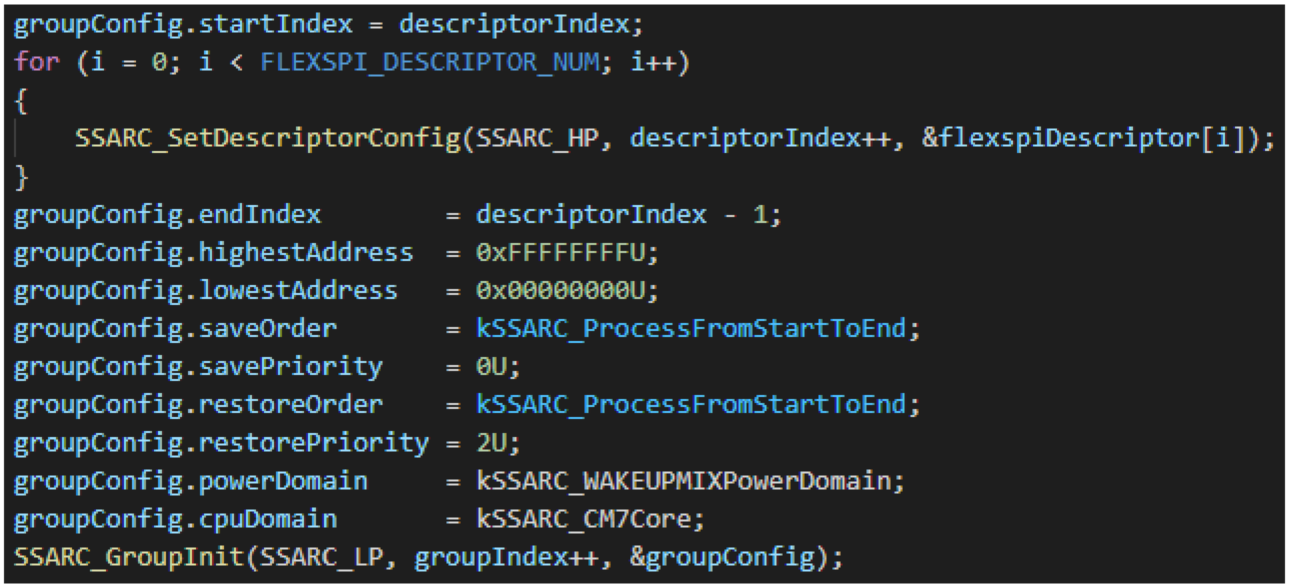
```
groupConfig.startIndex = descriptorIndex;
for (i = 0; i < SSARC_LED_NUM; i++)
{
    SSARC_SetDescriptorConfig(SSARC_HP, descriptorIndex++, &ssarc_led[i]);
}
groupConfig.endIndex        = descriptorIndex - 1;
groupConfig.highestAddress  = 0xFFFFFFFFU;
groupConfig.lowestAddress   = 0x00000000U;
groupConfig.saveOrder       = kSSARC_ProcessFromStartToEnd;
groupConfig.savePriority    = 0U;
groupConfig.restoreOrder    = kSSARC_ProcessFromStartToEnd;
groupConfig.restorePriority = 0U;
groupConfig.powerDomain     = kSSARC_WAKEUPMIXPowerDomain;
groupConfig.cpuDomain       = kSSARC_CM7Core;
SSARC_GroupInit(SSARC_LP, groupIndex++, &groupConfig);

groupConfig.startIndex = descriptorIndex;
```

图 12. Group A 的恢复优先级为 0



```
groupConfig.startIndex = descriptorIndex;
for (i = 0; i < PINMUX_DESCRIPTOR_NUM; i++)
{
    SSARC_SetDescriptorConfig(SSARC_HP, descriptorIndex++, &pinmuxDescriptor[i]);
}
groupConfig.endIndex         = descriptorIndex - 1;
groupConfig.highestAddress   = 0xFFFFFFFFU;
groupConfig.lowestAddress    = 0x00000000U;
groupConfig.saveOrder        = kSSARC_ProcessFromStartToEnd;
groupConfig.savePriority     = 0U;
groupConfig.restoreOrder     = kSSARC_ProcessFromStartToEnd;
groupConfig.restorePriority = 1U;
groupConfig.powerDomain      = kSSARC_WAKEUPMIXPowerDomain;
groupConfig.cpuDomain        = kSSARC_CM7Core;
SSARC_GroupInit(SSARC_LP, groupIndex++, &groupConfig);
```

图 13. Group B 的恢复优先级设为 1



```
groupConfig.startIndex = descriptorIndex;
for (i = 0; i < FLEXSPI_DESCRIPTOR_NUM; i++)
{
    SSARC_SetDescriptorConfig(SSARC_HP, descriptorIndex++, &flexspiDescriptor[i]);
}
groupConfig.endIndex         = descriptorIndex - 1;
groupConfig.highestAddress   = 0xFFFFFFFFU;
groupConfig.lowestAddress    = 0x00000000U;
groupConfig.saveOrder        = kSSARC_ProcessFromStartToEnd;
groupConfig.savePriority     = 0U;
groupConfig.restoreOrder     = kSSARC_ProcessFromStartToEnd;
groupConfig.restorePriority = 2U;
groupConfig.powerDomain      = kSSARC_WAKEUPMIXPowerDomain;
groupConfig.cpuDomain        = kSSARC_CM7Core;
SSARC_GroupInit(SSARC_LP, groupIndex++, &groupConfig);
```

图 14. Group C 的恢复优先级设为 2

这些组会按照优先级依次被触发，FlexSPI 会在 CPU 被唤醒之前恢复。当 CPU 唤醒之后，CPU 可以获得并运行由 FlexSPI 读取的指令，然后 GPIO 函数会使 LED 闪烁。

Date of release: February 3, 2021
Document identifier: AN13120