# AN13120

## How to use SSARC to configure a peripheral after wake up on RT1170

Rev. 0 — February 3, 2021 Application Note

Contents



## 1 Introduction

The i.MX RT1170 crossover processor sets the speed records at 1 GHz. This ground-breaking family combines superior computing power and multiple media capabilities with more usability as well as real-time functionality.

This application note introduces how to use SSARC to configure a peripheral after wake-up.

The hardware is based on RT1170 EVK RevC1 (Hereafter referred to as EVK) and software is based on SDK 2.9.0 with IAR IDE.

## 2 SSARC overview

The State Save and Restore Controller (SSARC) saves the registers of functional modules in a memory, located in LPSR domain, before powering down and restoring the registers from memory after the module is powered up. This module is able to configure a peripheral before the CPU wakes up. Once the CPU wakes up, it can use the peripheral without initialization. Figure 1 shows the basic functions of SSRAC.

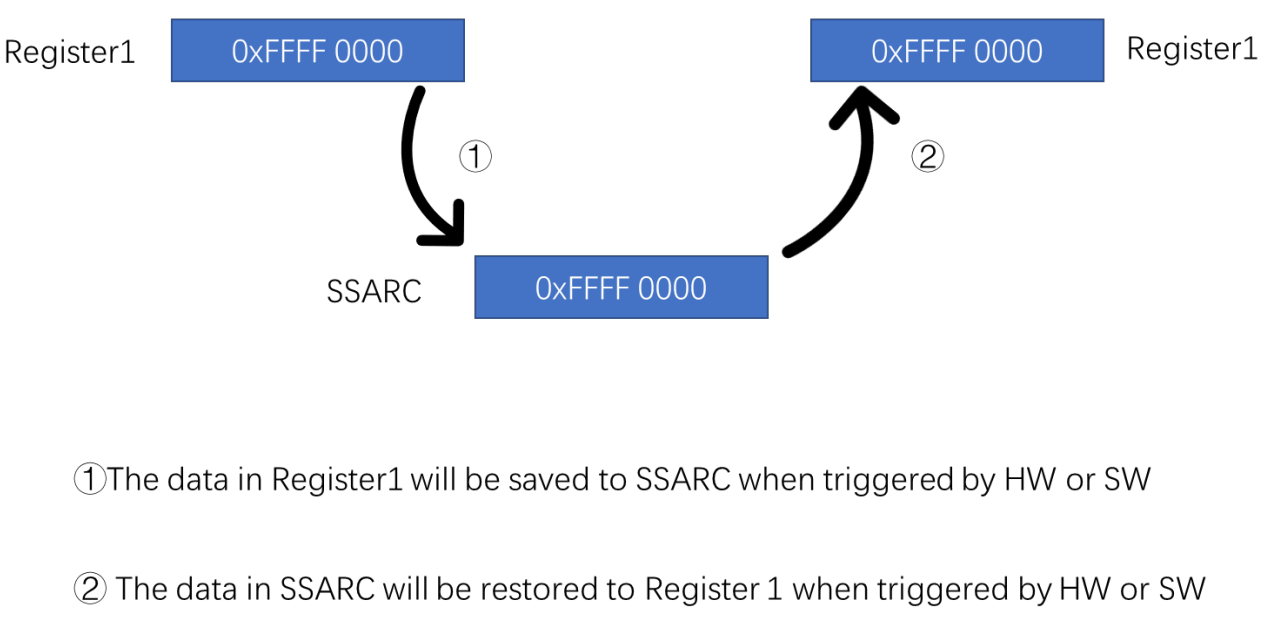


**Figure 1. SSARC basic function and work flow**

### 2.1 Descriptor

Descriptor is the most basic element in SSARC. It has up to 1024 descriptors which support most operations like CPU, such as write, read, delay and polling. It supports four operations and seven types of operation.

- **Operations**
  - Save Disable and Restore Disable
  - Save Enable and Restore Disable
  - Save Disable and Restore Enable
  - Save Enable and Restore Enable
- **Type of operation**
  - Read Value and Write Back
  - Write a Fixed Value
  - Read a register or with a value then write it back (OR)
  - Read a register and with a value then write it back (AND)
  - Delay several cycles (Delay)
  - Read a register until the bit or bits in it changed to 0 (Polling 0)
  - Read a register until the bit or bits in it changed to 1 (Polling 1)

## 2.2 Group

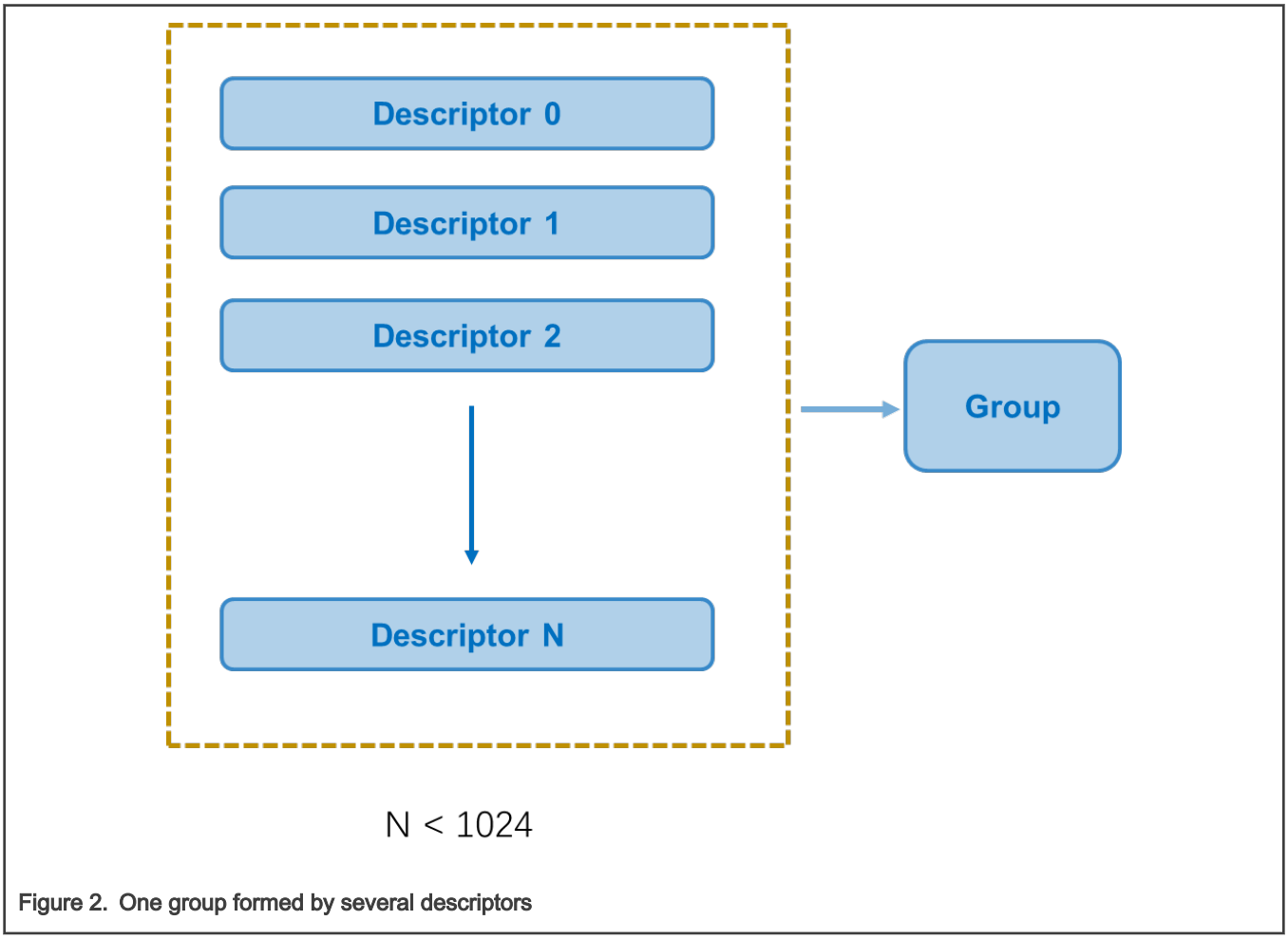


**Figure 2. One group formed by several descriptors**

The 1024 descriptors falls into 16 groups, with each group containing a number of continuous descriptors. The descriptor is controlled by group. The group can be trigger by either software or hardware. Each group has its own priority for save and restore.

There are 16 priority settings from 0 to 15 and 0 has the highest priority. Usually the priority is not required to be cared about during the save operation but it is important for the restore operation. For example, if to use pins for a peripheral, the initialization of the pins has a higher priority than other settings.

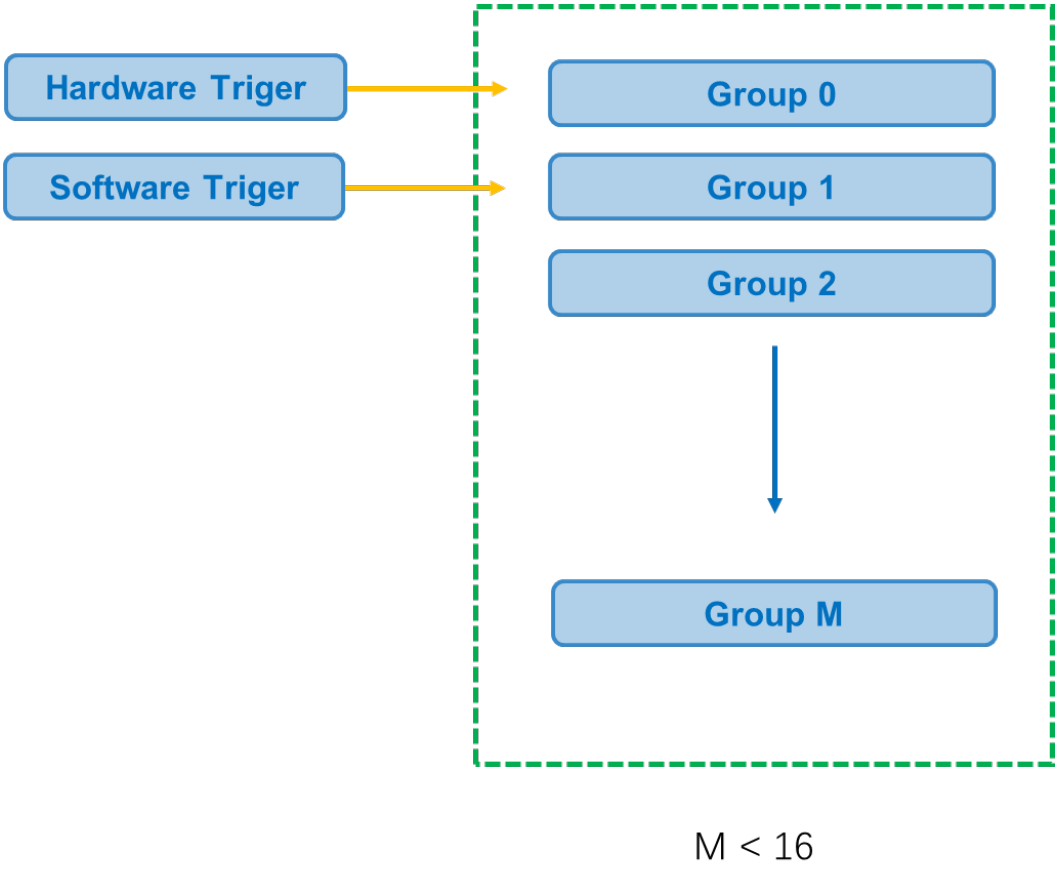


Figure 3. Group triggered by hardware or software

### 2.2.1 Group triggered by software

The software trigger method can be used to verify whether the SSARC settings are correct or not before entering low power. The bit 0 and bit 1 in `DESC_CTRL1_x`, `x` representing index for each group, are able to use the software to trigger the save and restore operations.

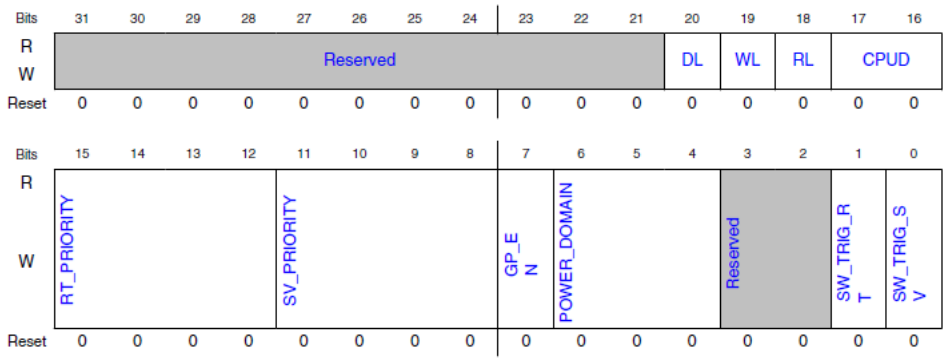


Figure 4. Group register in DESC_CTRL1_x triggered by software

Besides, the API as below can be used for save and restore operations.

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

There is one software trigger example in SDK as below:

*boards\evkmimxrt1170\driver_examples\ssarc\software_trigger*

### 2.2.2 Group triggered by hardware

The group is able to triggered by BPC. When a power domain is powered off, the settings of this domain are lost. If it happens, SSARC automatically saves the data before power-off and restores the data after power-on. The BPC sends save and restore request to SSARC and completes the handshake. Because SP and CPU mode can trigger the BPC power-down, the trigger signal can be sent by SP or CPU mode.

Take BPC2, WAKEUP MIX with SP mode as example, usually the WAKEUP MIX is powered down after SP11. Once the SP switches to SP11 or other SP after SP11, BPC sends the save request to SSARC, the Group will be triggered by it and execute the operations in descriptors. If the SP switched to an SP, such as SP1, once the SP has been switched, the BPC2 will send the restore request to SSARC, then the SSARC will execute the restore operations in descriptors.

If BPC0, MEGA MIX, is controlled by CM7's CPU mode, once the CM7 entered a low power mode BPC0 will send the save request to SSARC. If the CM7 wake up, BPC0 will send the restore request to SSARC. For the low power mode, it can be WAIT, STOP or SUSPEND, but entering such mode will send the request that depends on the settings in `BPC_SSAR_SAVE_CTRL`.

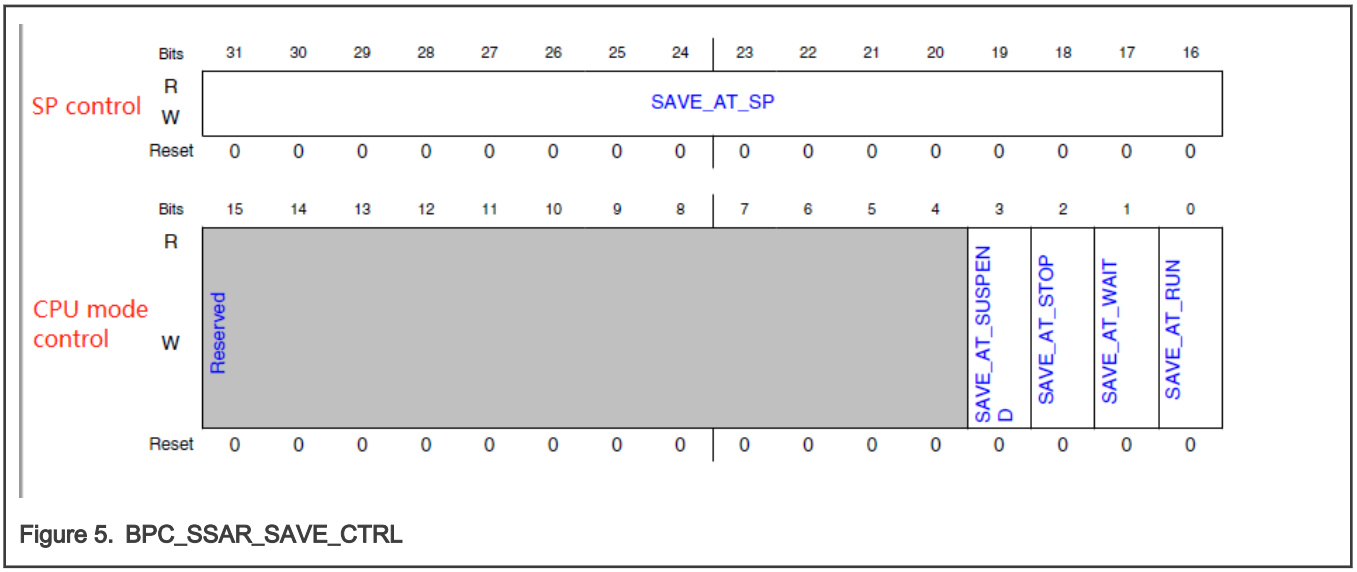


**Figure 5. BPC_SSAR_SAVE_CTRL**

The registers `BPC_SSAR_SAVE_CTRL` and `BPC_SSAR_RESTORE_CTRL` can control the SSAR request. The register `BPC_SSAR_ACK_CTRL` can be used to control the delay and wait response behavior.

**Table 1. Power controller assignment**

| Power domain | Assignment |
|---|---|
| BPC0 | MEGA MIX |
| BPC1 | DISPLAY MIX |
| BPC2 | WAKEUP MIX |
| BPC3 | LPSRMIX |

*Table continues on the next page...*

Table 1. Power controller assignment (continued)

| Power domain | Assignment |
|---|---|
| BPC4 | MIPIPHY |
| BPC5 | Virtual |
| BPC6 | Virtual |
| BPC7 | Virtual |

**NOTE**

Virtual power domains do not physically exist, but are used to trigger SSARC to save and restore, status and data.

# 3 Restore sequence and skills

## 3.1 Clock source for a peripheral

Because most use cases for SSARC relates with low power, the first step is to check the clock. The basic steps is as below.

1. Check whether the clock source for clock root is available or not in the target wakeup SP.
2. Check the LPCG status. LPCG can be controlled by SP or CPU mode.
   - If LPCG is controlled by SP, just check whether it is available or not in the target wakeup SP.
   - If LPCG is controlled by CPU mode, check the sleep CPU mode settings.
3. As shown in Figure 6, the last step of wakeup flow is CPU mode. When SSARC is going to initialize or restore a peripheral, the LPCG may block the clock, because the CPU is still under low power mode. Therefore, make sure the LPCG is available under a low power mode.

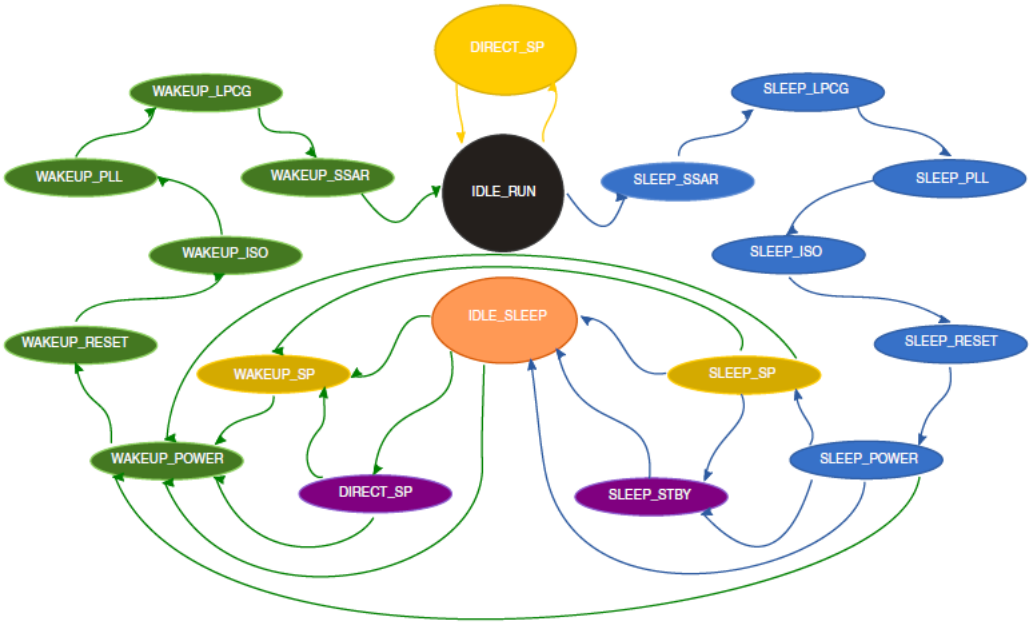


Figure 6. CPU mode transition flow

For more detailed steps, see **Section 4.13** in *Debug and Application for RT1170 Clock and Low Power Feature* (document AN13104).

## 3.2 Restore method and sequence

As shown in Descriptor, there are seven types of operation descriptor. For different peripheral, different types of operations are needed. For example, for a GPIO PIN, use the Read Value and Write Back with save and restore function enabled, because the initialization of GPIO does not need to consider the order of operations while a peripheral needs. The basic flow is:

- Restore the PIN with `IOMUXC_SW_MUX_CTRL` register and `IOMUXC_SW_PAD_CTRL` register. These registers usually can use the Read Value and Write Back with save and restore function enabled.
- Check the peripheral initialization sequence. Some peripherals need to initialize in the order of time. For example, FlexSPI needs to configure the peripheral to the run modem, configure the control register, unlock LUT table, update the LUT, and finally lock the LUT. Therefore, the tore function of FlexSPI is most for initialization. For these peripherals, it is a better operation to write a Fixed Value with save and restore function disabled.

# 4 Example

There are two examples to show how to use SSARC to restore a peripheral after wake up. Taking FlexSPI as examples, the low power mode is SP11 Suspend STBY. Under SP11 WAKEUP MIX will power down, which means its peripherals will also power down. The FlexSPI used in the example belongs to WAKEUP MIX. After wakeup, CPU runs at SP1 when the WAKEUP MIX is power on and CPU jumps to a running test function located at the FlexSPI address and toggles a GPIO with LED.

For how to jump to this test function, see **Chapter 4.15** in *Debug and Application for RT1170 Clock and Low Power Feature* (document AN13104).

## 4.1 Configure BPC2 as save and restore trigger source

Configure the BPC as SP control mode and send the save and restore request based on the SP settings. `PD_WKUP_SP_VAL` is `0xf800`, which means WAKEUP MIX will power down at SP11-SP15.

```
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);
```

Figure 7. Configure BPC2 as the save and restore trigger source

## 4.2 Restore a GPIO pin after wakeup

There is an on-board LED which connected to `GPIO_AD_04` on RT1170 EVK. This pin is used to toggle LED blinking. The GPIO in `GPIO_AD_04` is `GPIO09_IO03`. For the pad and pin settings, the Read Value and Write Back with save enable and restore enable operation can be used.

- Add a descriptor for `IOMUXC_SW_MUX_CTL_PAD_GPIO_AD_04`.
- Add a descriptor for `IOMUXC_SW_PAD_CTL_PAD_GPIO_AD_04`.
- Add a descriptor for `GPIO9_GDIR`, which is used to configure the direction register as output for toggle LED.

```
#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*/
```

Figure 8. Configure the SSARC for a GPIO pin

These three descriptors form a group, named as **Group A** in this applictaion note, which can be triggered by BPC2. After the system wakes up, the `GPIO9_IO03` is restored and the register is toggled to enable the LED blink.

*BOARD_USER_LED_GPIO->DR_TOGGLE = 1<<BOARD_USER_LED_GPIO_PIN;*

## 4.3 Restore FlexSPI after wakeup

### 4.3.1 Restore the PIN settings

The first step to restore FlexSPI is that restore the `IOMUXC_SW_MUX_CTRL`, `IOMUXC_SW_PAD_CTRL` and the `SELECT_INPUT` for function pin. Not all pins need to configure the `SELECT_INPUT`, but all need to configure `IOMUXC_SW_MUX_CTRL` and `IOMUXC_SW_PAD_CTRL`. For pins restore Read Value and Write Back with save enable and restore enable operation can be used. The descriptors form a group, as shown in Figure 9, named as **Group B** in this application note.

```
#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  */
```

Figure 9. Add descriptor for each pin IOMUXC_SW_MUX_CTRL，IOMUXC_SW_PAD_ACTRL and SELECT_INPUT

### 4.3.2 Restore the FlexSPI

Because the WAKEUP MIX is power down, FlexSPI needs to initialize again. To achieve that, the simple save and restore cannot be used because there are timing requirements in the configuration process. For example, it should be configured under run mode, it needs software reset and when configure the Look Up Table (LUT), it needs to enter a key and then unlock and lock the LUT. During the configuration process, some flag bits are needed to determine whether the working state is stable by polling. So that Read Value and Write Back with save enable and restore enable operation is unable to restore the FlexSPI. For how to configure the FlexSPI, see AN13112. Figure 10 shows the FlexSPI restore flow with different operation types and these descriptor forms a group, named as Group C in this application note.

```
#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  */
```

Figure 10. Restore the FlexSPI

### 4.3.3 Configure the group

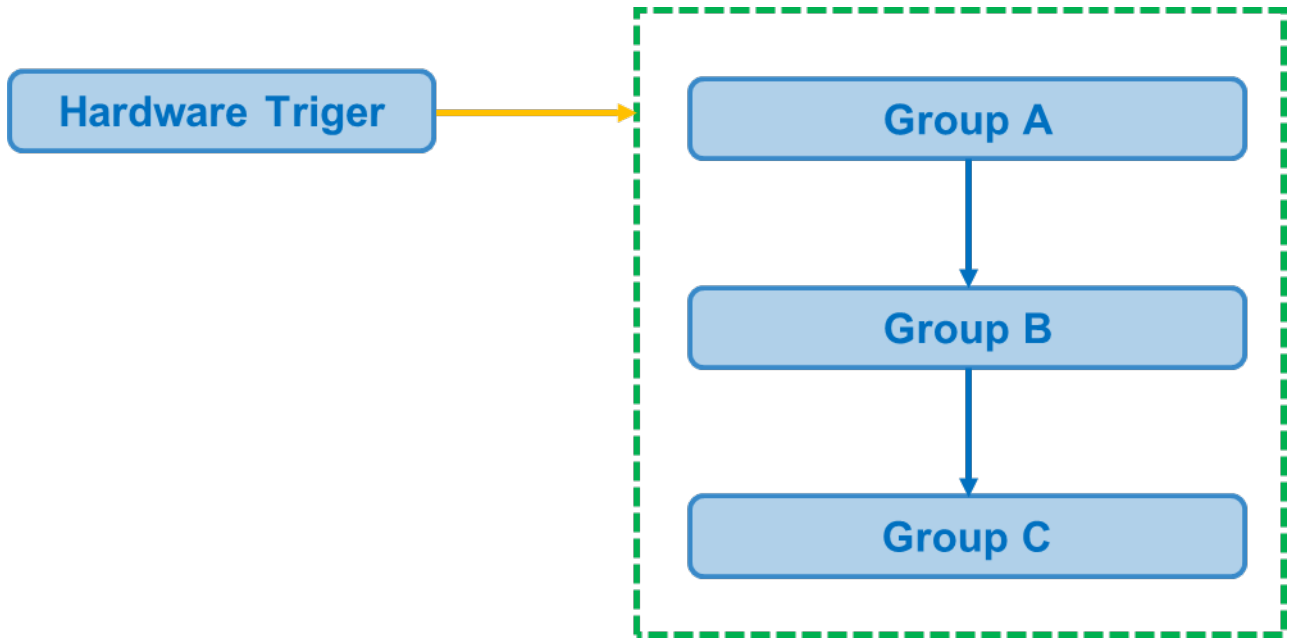


Figure 11. Group Restore Sequence

Several descriptors can form a group and then the group can be triggered by hardware or software. The restore priority is an important setting, because it impacts the restore sequence. Usually, the PIN restore should be the first step followed by the

peripheral. For this example, Group A has the highest priority, Group B has the second highest priority, and the priority for Group C is the lowest.

```
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;
```

Figure 12. Group A settings with Restore Priority 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);
```

Figure 13. Group B settings with Restore Priority 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);
```

Figure 14. Group C settings with Restore Priority 2

These groups are triggered according to the priority. Then, FlexSPI is restored before the CPU wakes up. Once the CPU wakes up, CPU can acquire and execute instructions from Flash by FlexSPI and LED can be toggled by the GPIO toggle function.

*How To Reach Us*

**Home Page:**

nxp.com

**Web Support:**

nxp.com/support



For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

**Date of release: February 3, 2021**

**Document identifier: AN13120**

arm