S32G M7 Standby Demo 说明

by John Li

本文说明S32G M7核Standby demo 详细情况及定制。

请注意本文为培训和辅助文档，本文不是官方文档的替代，请一切以官方文档为准。

历史	说明	作者
V1	● 创建本文	John.Li
V2	● 增加UART1 支持 ● 增加时钟修改 ● 关掉 A53_0 核	John.Li
V3	● 修改为 Mcal sample	John.Li

目录

1 参考资料说明

分类	名称	说明	备注
文档	AN12880-5510-standby.pdf	VR5510 standby 说明文档	Search on www.nxp.com
文档	AN12952-Power Saving Techniques.pdf	S32G2 suspend 说明文档	Search on www.nxp.com 重要
文档	S32g274A_standby_demo_UG.pdf	Standby demo 使用说明	Check with your window FAE
代码	S32g274AstandbyRAMboot	Standy RAM 中运行代码	
代码	S32G274Astandbymode	Standy demo 代码	
文档	S32G2RM.pdf	芯片手册	Download from www.nxp.com
文档	ds649820 - VR5510 Datasheet Rev2 (2.0).pdf	芯片手册	Download from www.nxp.com
文档	S32G_RTD_MCAL_V*.pdf	Mcal 定制说明	Download from https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-MCAL-customization-application-doc/ta-p/1399899

2 Demo 创建运行过程

2.1 创建运行

参考文档《S32g274A_standby_demo_UG.pdf》，导入工程，编译及运行。注意：

- Import project into S32DS：打开 S32DS3.4.3+RTD3.0.2, File->Import->Existing Projects into Workspace/Next->Brows to “….\S32G274Astandbymode” /Finish。
- Build:在 S32DS 的 Project Explorer 里选择 S32G274Astandbymode:Debug 上右击->Build Project。

- Creating IVT images：此页选择的*.bin 有误，应该是 S32G274Astandbymode.bin，他的地址是 RAM start pointer=RAM entry porinter=0x34400000。
- 制造 IVT image 时，DCD 段选择 SRAM 初始化 DCD 脚本文件可以在 C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_DCD_InitSRAM.bin 里找到。
- 制造 IVT image 时，注意 application bootloader 的 size in bytes 会自动计算出来，但是如果其不是 8 bytes 的整数时，会报错，所以需要手动增大为 8 bytes 的整数倍。
- 制造 IVT image 时，原文是没有打包 qspi nor timing 头文件的，因为本 bin 文件本身比较小，所以用默认的低速 1 bit 模式来访问 qspi nor 不会导致 watchdog 超时失败，如果要增加也可以，在 interface selection 中，Boot device type=QuadSPI Serial Flash, checked Configure QuadSPI Serial parameters，然后 QuadSPI parameters 选择 C:\NXP\S32DS.3.4\eclipse\mcu_data\processors\S32G274A_Rev2\PlatformSDK_S32XX_2022_03\quadspi\default_boot_images 下的 mx25_sim133ddr.bin or mx25_sim133sdr.bin or mx25_sim200ddr.bin。
- 烧写镜像时，Algorithm 应当选择 MX25UW51245G

测试结果如下：

```
PMIC initial status Read:2006
PMIC status before entry:a
 Set the Boot model
  0: FULL BOOT;  1: Standby-RAM BOOT;
 Set the Boot model - FULL BOOT;
 ACK #2 :WAKE-UP Source number[31:0] is : #31
 ACK #3:RTC period is:5000 ms
PMIC initial status Read:6
PMIC status before entry:a
 Set the Boot model
  0: FULL BOOT;  1: Standby-RAM BOOT;
 Set the Boot model - Standby RAM BOOT;
 ACK #2 :WAKE-UP Source number[31:0] is : #31
 ACK #3:RTC period is:5000 ms
 boot In standby Ram
       I2C_4: Module Initialize
       I2C_4 Ready
PMIC initial status Read:6
PMIC status before entry:a
```

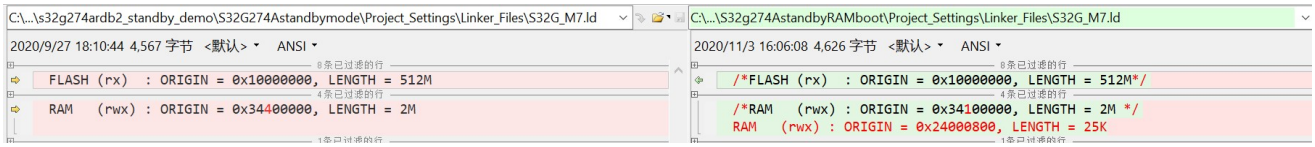
Set the Boot model

0: FULL BOOT; 1: Standby-RAM BOOT;

2.2 Standby 和 StandbyRAMboot 的区别

2.2.1 连接文件

…\Project_Settings\Linker_Files\S32G_M7.ld



可见 Standby demo 的连接地址是在 SRAM 中：

0x00_3400_0000	872415232	0x00_353F_FFFF	893386751	20480	Internal SRAM (first 8Mbyte)*

而 StandbyRAMboot 的连接地址是在 Standby RAM 中：

0x00_2400_0000	603979776	0x00_33FF_FFFF	872415231	262144	Standby RAM (32K)

2.2.2 主函数

step	Standbymode	StandbyRAMboot	
0	s32g_dfs_reset();	s32g_dfs_reset();	
1	s32g_clock_tree_init();	s32g_clock_tree_init();	
2	s32g_linflexd_init();	s32g_linflexd_init();	
3	Init_standbyram();	N/A	
4	qspi_init();	N/A	
5	s32g_i2c4_init();	s32g_i2c4_init();	
6	N/A	stbywkupconfig_pad();	
7	standby_entry();	standby_entry();	
7.1	standbyramc_cpy();	N/A	/* copy ivt and firmware into standby rom */
7.2 WKPU_set	配置了 CAN0_RX 和 RTC 作为唤醒源	仅配置了 RTC 作为唤醒源	

3 S32G Standby 原理与代码说明

3.1 外设初始化函数

- s32g_dfs_reset // clock dfs reset
- s32g_clock_tree_init// clock initialization，注意其对 QSPI NOR/uSDHC/UART 的时钟初始化
- s32g_linflexd_init // linflexd initialization 115200
- Init_standbyram // standby_rom initialization
- qspi_init // init Nor flash，注意此处初始化 AHB Buffer 模式。
- s32g_i2c4_init // i2c4 initialization,用于连接 PMIC

3.2 standbyramc_cpy(可选)

此功能只是 Standby RAM fast boot 有用。

退出待机模式会触发 BootROM 的执行。根据唤醒源，BootROM 从 Standby RAM 引导或执行完全引导。

Standby RAM 快速引导与普通 IVT 的格式相同（参见图像矢量表（IVT）），但其位置在待机启动时是固定的

Table 132. IVT image structure

Address	Size (bytes)	Name	Comments
0h	4	IVT header	Header showing the start of the IVT
4h	4	Reserved	Reserved
8h	4	Self-Test DCD pointer	Pointer to the start of the configuration data used for BIST
Ch	4	Self-Test DCD pointer (backup)	Pointer to the start of the backup configuration data used for BIST

Table continues on the next page...

Address	Size (bytes)	Name	Comments
10h	4	DCD pointer	Pointer to the start of DCD configuration data
14h	4	DCD pointer (backup)	Pointer to the start of backup DCD configuration data
18h	4	HSE_H firmware flash memory start pointer	Pointer to the start of the HSE_H firmware in flash memory
1Ch	4	HSE_H firmware flash memory start pointer (backup)	Pointer to the start of the backup HSE_H firmware in flash memory
20h	4	Application boot code flash memory start pointer	Pointer to the start of the application boot code in flash memory
24h	4	Application boot code flash memory start pointer (backup)	Pointer to the start of the backup application boot code in flash memory
28h	4	Boot configuration word	Configuration data used to select the boot configuration
2Ch	4	Life cycle configuration word	Configuration data used for advancing life cycle
30h	4	Reserved	Reserved
34h	32	Reserved for HSE_H firmware	Defined by the HSE_H firmware specification
54h	156	Reserved	Reserved
F0h	16	Galois Message Authentication Code (GMAC)	GMAC of first 240 bytes of IVT image structure

RAM（24000000h）。通过快速引导引导时，BootROM 仅支持：

•DCD 操作（从主和备份）

•应用程序引导（从主和备份）

快速引导不支持自检或 HSE_H 固件。它只支持（IVT 引导配置字字段）boot_SEQ=0，这表示要执行非安全引导。如果 BOOT_SEQ=1，BootROM 将发出重置。从快速引导引导时，BootROM 不支持从 QuadSPI 或 SD/MMC/eMMC 卡提取数据。在进入待机模式之前，应用程序应编程：

•IVT（24000000 小时）

•DCD（如果需要）

•备用 RAM 上的应用程序，其内容在待机模式下保持有效

IVT 中的所有指针必须指向备用 RAM 中的地址。BootROM 不在 fast 上执行任何身份验证通过快速引导引导时，引导、DCD 或应用程序。当请求完全引导时，BootROM 会执行正常引导操作，如本章所述，在任何重置之后。BootROM（启动只读存储器）在 Standby RAM 启动时不检查快速引导。有关唤醒源的更多详细信息，请参阅唤醒单元（WKPU）章执行完全引导或从 Standby RAM 快速引导引导的决定取决于触发唤醒的唤醒源配置。如果唤醒源配置为执行完全引导，BootROM 将执行完全引导。否则，它将从 Standby RAM 快速启动。

代码：

- IVT 头数组：

```
/* IVT binary */
const uint8_t ivt_binary[256] = {
    /* HEADER */
    0xd1, 0x1, 0x0, 0x60,
    /* reserved_1 */0x00, 0x00, 0x00, 0x00,
    /* Self-Test DCD */0x00, 0x00, 0x00, 0x00,
    /* Self-Test DCD (backup) */0x00, 0x00, 0x00, 0x00,
    /* DCD */0x00, 0x00, 0x00, 0x00,
    /* DCD (backup) */0x00, 0x00, 0x00, 0x00,
    /* HSE */0x00, 0x00, 0x00, 0x00,
    /* HSE (backup) */0x00, 0x00, 0x00, 0x00,
    /* Application bootloader */
    0x00, 0x04, 0x00, 0x24, //地址 0x24000400
    /* Application bootloader (backup) */0x00, 0x00, 0x00, 0x00,
    /* boot_config */0x0, 0x0, 0x0, 0x0,
    /* life_cycle_config */0x0, 0x0, 0x0, 0x0,
    /* reserved_2 */0x0, 0x0, 0x0, 0x0,
    /* hse_fw_reserved */0x0,…
    /* reserved_3 */0x0,…
    /* gmac */0x0,…
};
```

- 应用程序头数组：

```
            /* Image header */
            0xd5, 0x00, 0x00, 0x60,
            /* RAM start pointer */
            0x00, 0x08, 0x00, 0x24, //0x24000800
            /*RAM entry pointer*/
            0x00, 0x08, 0x00, 0x24, //0x24000800
            /*Code length*/
            0x00, 0x40, 0x00, 0x00, //0x4000
            0x0, 0x0, 0x0, 0x0, 0x0,
            0x0, 0x0, 0x0, 0x0, 0x0,
```

```
        0x0, 0x0, 0x0, 0x0, 0x0,
```

- 关键常量：

```
#define Standby_RAM_ADDRESS 0x24000000
#define Standby_RAM_APP_HEAD_ADDRESS 0x24000400
#define Standby_RAM_APP_ADDRESS 0x24000800
#define COPY_FLASH_ADDRESS 0x8000 //此处与 standbyramboot binary 烧在 QSPI NOR 中的偏移地址相同。
```

- 源代码：

```
    /* mv IVT to Standby RAM*/
    for(crtOffset = 0; crtOffset < 256; crtOffset++)
    {
        *((uint8_t *)(Standby_RAM_ADDRESS) + crtOffset) = ivt_binary[crtOffset];
    }
    /* mv app header to Standby RAM*/
    for(crtOffset = 0; crtOffset < 128; crtOffset++)
    {
        *((uint8_t *)(Standby_RAM_APP_HEAD_ADDRESS) + crtOffset) = application_binary[crtOffset];
    }
    /* mv app to Standby RAM*/
    for(crtOffset = 0; crtOffset < COPY_FLASH_SIZE; crtOffset++)
    {
        *((uint8_t *)(Standby_RAM_APP_ADDRESS) + crtOffset) = *((uint8_t
*)COPY_FLASH_ADDRESS + crtOffset);
    }
```

3.3 WKPU_set

参考文档《AN12952-Power Saving Techniques.pdf》，Table 6. Recommended configuration/setup sequence。

传入参数

BOOT_MODE=0: Standby RAM BOOT

BOOT_MODE=1 FULL BOOT

S32G_WKUP_SOURCE_NO=31// /*Wake-up source number 31 RTC*/ /* Wake-up source number, from 0-22, 31*/

Wakeup source 如下说明：

Table 61. WKUP expose pads

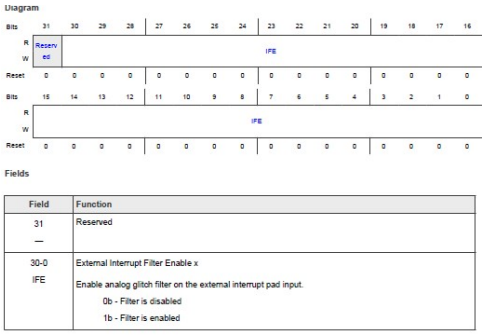
Number of	Value
NMI sources	1
Internal interrupt sources	1
External interrupt sources	23
Glitch filters for external interrupts	23
External interrupt vectors	1

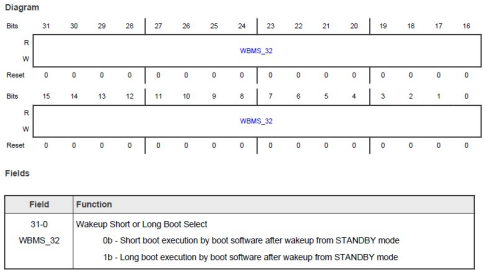
NOTE

Wake-up sources 0 to 22 map to the pads with the corresponding WKPUx signal and I/O supplies are described in the attached IOMUX spreadsheet, IO Signal tab. Wake-up source 31 is mapped to the internal RTC module. For the IO voltage supply range, refer to the Absolute Maximum Ratings section in the chip DataSheet.

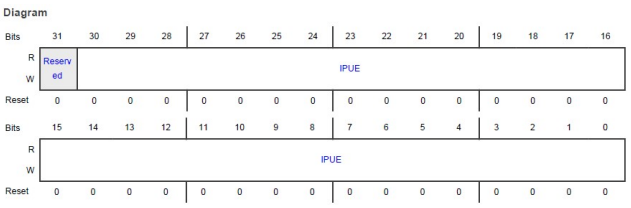
参考文档 S32G2_IOMUX.xlsx，搜索 WKUP 可以找到所有有 WKUP 功能的 GPIO， 如下：

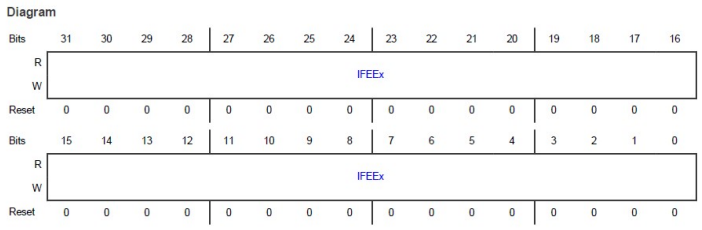
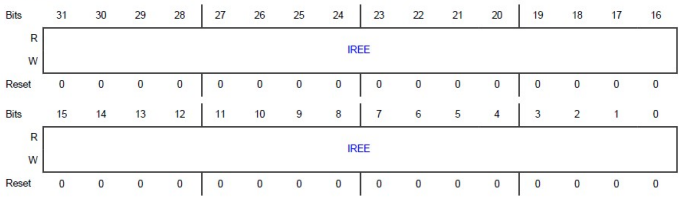
PC_11		43	SIUL2_0	0000_0000	0x4009C2EC	GPI[43]
PC_11		513	SIUL2_0	0000_0010	0x4009CA44	CAN0_RX
PC_11		745	SIUL2_1	0000_0010	0x44010DE4	LLCE_CAN0
PC_11	-			-		WKUP0

Step	代码	说明	寄存器
0	WKPU_WIFER = 0	Enable wakeup filers	

1	if(BOOT_MODE_SET)
	{
	WKPU_WBMSR = 1<<WKUP_SOURCE;
	WKPU_WBMSR |= 1;
	}
	else
	{
	WKPU_WBMSR = 0;
	}
		RAM/FULL boot configuration wbmsr：如果是 full boot，设置 RTC 和 CAN0_RX GPIO 为 full boot。其它为 stdram boot	

2	WKPU_IRER = 0x0; WKPU_WRER = 0x0;	Prepare Pad Pull activation Zero both registers. (Reason is to avoid capturing unintended events before WIPUER_WIPDER is written; see Step 5)	
3	SIUL2_0.MSCR[43] .B.PUE = 1; SIUL2_0.MSCR[43] .B.PUS = 1; SIUL2_0.MSCR[43]. B.IBE = 1;	管脚配置寄存器 SIUL2 的相关管脚配置要与 WKPU 和 STBY_GPR 的配置匹配：对于希望配置为唤醒源的管脚，配置为输入 IBE=1，可拉动 PUE=，拉高 PUS=1。其它不作唤醒源的管脚为 0。 需要确保：SIUL 拉动使能 MSCRn[PUE]=各自配置的 WKPU. WIPUER_WIPDER. IPUE[m] SIUL 拉动方向 MSCRn[PUS]=各自配置的 STDBY_GPR. WKUP_PUS. WKUP_PUS[y]	PC_11 GPI[43]，CAN0_RX[513]
4	STBY_GPR. WKUP_PUS.B. WKUP_PUS |= 0x1;	Configure STBY_GPR's pull directions Select pull direction for each pad that shall capture wakeup events during Standby Mode.// pull UP	

5	WKPU_ WIPUER_ WIPDER = 0x1	Enable Pulls in the Wakeup Units Configure pull enable/disable for each pad. This will become effective during Standby Mode after step 6. // ipue = 1, pull up	Diagram  Fields Field / Function 31 — Reserved 30-0 IPUE External Interrupt Pull Enable x 0b - Pull is disabled. 1b - Pull is enabled.
6	STBY_GPR. WKUP_PUS.B. WKUP_PU_OVERRIDE |= 0x1;	Move full pad control to Standby Domain elements Set to '1'. (This relieves SIUL2 from pad control and only elements within the standby domain take over pad control) clear wake-up status	

7	WKPU_WIFEER = 0x00000001; //can0 配置为下降沿触发
	WKPU_WIREER = 0x80000000;//RTC 配置为下升沿触发
		配置唤醒事件的边缘触发，注意
		• 要为所有唤醒事件配置触发边缘。
		• 确保所选触发边缘与管脚拉动方向不同。（背景：为了避免硬件内部 pull 启用被无意中捕获为唤醒事件。）
		避免同时使用上升和下降边缘触发 。
		也要避免同时关闭唤醒事件的两种边缘触
		注意：将“0”写入 IREE[x]和 IFEE[x'将禁用该管脚的外部中断功能（即，以下任何活动都不会产生系统唤醒或中断）。
		// falling down 配置为下降沿触发，因为管脚是配置
			Diagram
			 
			
			Fields
			Field / Function
			31-0 IFEEx External Interrupt Falling-edge Events Enable x
			0b - Falling-edge event is disabled
			1b - Falling-edge event is enabled
			Diagram
			 
			
			Fields
			Field / Function
			31-0 IREE External Interrupt Rising-edge Events Enable x
			0b - Rising-edge event is disabled
			1b - Rising-edge event is enabled

		为上拉输入的。	

8	WKPU_IRER = 0; //optionally 清除中断
	
	WKPU_WRER = 0x80000001; // LLCE CAN 0=0x1, RTC=0x80000000
		配置（使能）输入缓冲启用
		
		提示：确定管脚的有效输入缓冲使能可以通过以下逻辑方程：有效输入缓冲区使能=SIUL.MSCRn。IBE 或 WKPU IRER.EIRE[m]
		
		或 WKPU.WRER.WRE[m]。
		
		1.步骤 8.1 WKPU.IRER
		
		配置和/或清除 WKPU.IRER
		
		注意：在待机模式进入新中断之前应避免发生。然而，对于用户来说可能需要捕捉中断的应用程序在待机模式下请求以供以后使用。处于待机模式但无中断将发生，并且不会触发唤醒。对于所有人其他消息来源均清除了 WKPU.IRER。
		
		2.步骤 8.2 WKPU.WRER
		
		配置 WKPU。WRER 是输入缓冲器从 WKPU 侧启用。
			Diagram
			Bits 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
			R EIRE
			W
			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 EIRE
			W
			Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
			
			Fields
			Field / Function
			31-0 EIRE: External Interrupt Request Enable x
			0b - Interrupt requests from the corresponding EIF[x] bit are disabled
			1b - A set EIF[x] bit causes an interrupt request
			
			Diagram
			Bits 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
			R WRE
			W
			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 WRE
			W
			Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
			
			Fields
			Field / Function
			31-0 WRE: External Wakeup Request Enable x
			0b - System wakeup requests from corresponding EIF[x] bit are disabled
			1b - A set EIF[x] bit causes a system wakeup request

9	WKPU_WISR = 0xffffffff; //写 1 清除掉唤醒 状态寄存器	清除唤醒状态 清除寄存器状态。（注意，状态标志为 W1C）	Diagram Bits 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 R EIF W W1C 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 EIF W W1C Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Fields Field / Function 31-0 EIF: External Wakeup/Interrupt Status Flag x This flag can be cleared only by writing 1. Writing 0 has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request. 0b - No event has occurred on the pad 1b - An event as defined by WIREER and WIFEER has occurred

3.4 standby_modechange

参考文档《AN12952-Power Saving Techniques.pdf》，4.3.2.2 Main core shutdown and standby entry。

只有在 SoC 准备好进入待机状态后，并且只有在其他核心/核心簇处于待机 WFI 状态时，才能启动该序列

1.确保所有核心/核心簇的 WFI 状态仍然正确。

2.将唤醒模块的 WKUP_PU_OVERRIDE 编程为 1。

3.禁用主核心 IRQ。

4.使用 MSCM 禁用所有核心的 NMI。IRNMIC 寄存器。

5.在 MC_ME 中为最后一个运行的主内核编程一个有效的分区索引和主内核 ID。（参见 MC_ME MAIN_COREID 寄存器）

6.完成未完成的访问。

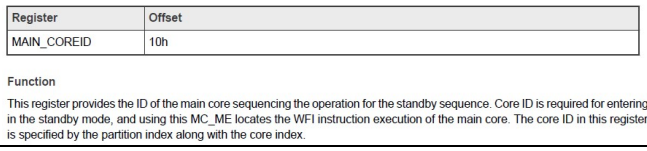
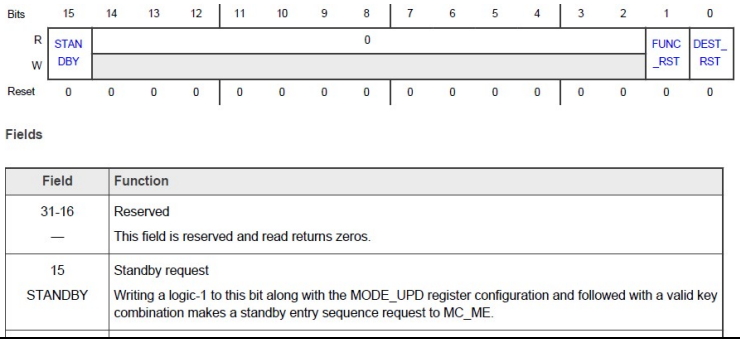
7.为备用模式转换编程 MC_ME。

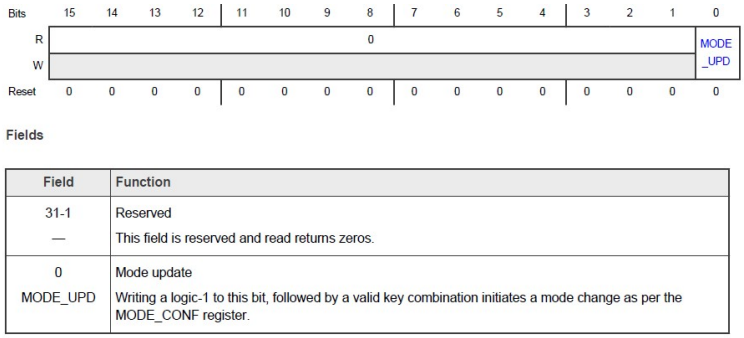
8.完成未完成的访问。

9.执行 WFI 指令。

除非转换被抢占，否则设备将进入待机模式。

代码如下：

Step	代码	说明	寄存器
0	MC_ME_MAIN_COREID = 0x0;	Main CoreID 配置为 M7_0	
1	MC_ME_MODE_CONF = (0x1<<15);	配置为进入 standby mode，需要写如 valid key 生效	

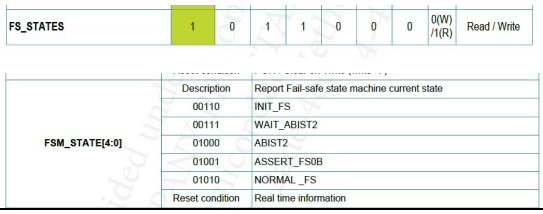
2	MC_ME_MODE_UPD = 1;	配置寄存器使能模式切换	 Fields Field / Function 31-1 — Reserved. This field is reserved and read returns zeros. 0 MODE_UPD — Mode update. Writing a logic-1 to this bit, followed by a valid key combination initiates a mode change as per the MODE_CONF register.
3	MC_ME_CTL_KEY = 0x5AF0; MC_ME_CTL_KEY = 0xA50F;	两次写入固定的 valid key，触发切换。	Register / Offset CTL_KEY / 0h S32G2 Reference Manual, Rev. 4, October, 2021 Reference Manual 1275 / 4654 NXP Semiconductors Mode Entry Module (MC_ME) Function This register provides the mechanism to MC_ME for starting the hardware processes for the partition(s) and standby entry sequence. The hardware processes for partitions are triggered through the corresponding PRTNn_PCONF register. The mechanism to trigger the hardware processes of the respective partitions require two write operations: first time with key and second time with inverted key. The hexadecimal value of key is 0x5AF0 whereas for inverted key is 0xA50F. For initiating a standby entry sequence, the MODE_CONF register is used for providing a standby entry request along with a valid key combination.

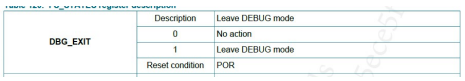
最后执行 WFI:

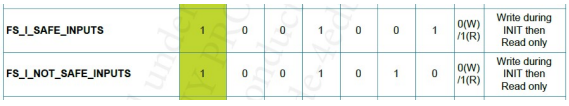
```
/****** S32G standby entry request **********/
    __asm("WFI");
    /***************************************/
    while(MC_ME_MODE_UPD == 1);
```

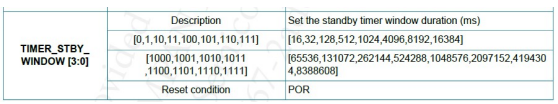
4 VR5510 PMIC Standby 原理与代码说明

4.1 PMIC_initConfig

Step	代码	说明	寄存器
0	status = PMIC_Read_FS(0x18, &read_data);	/*****Read the FSM_STATE field in FS_STATES REG for debug purpose * 00110 -- init_fs * 00111 --	

		wait_ABIST2 * 01000 -- ABIST2 * 01001 -- ASSERT_FS0B * 01010 -- NORMAL_FS**********/	
1	PMIC_Write_FS(0x18,0x4000);	/*****write the DBG_EXIT field in FS_STATES REG request PMIC exit debug status.**********/	 DBG_EXIT: Description: Leave DEBUG mode; 0: No action; 1: Leave DEBUG mode; Reset condition: POR
2	PMIC_Write(0x05, 0x100);	/**** Setup VR5510 for standby entry *****/ /*In the Main Memory map (0x20), enable HVLDO and PREV by setting these bits in the M_REG_CTRL3 register (0x05) * configure low power mode VPRE voltage, w0 to set as 3.3V (w1 set as 3V) */ Bucks 和 LDO 必须在待机模式下禁用，除了 HVLDO 和 VPRE。 根据客户的电源树，也可以启用其他稳压器。例如，如果 DDR 是自刷新的，那么 LDO2 和 BUCK3 也应该被使能。 如果客户使用 VPREV_STBY_EN_OTP 位启用该选项，则输出 VPRE 的电压可以配置为 3.3 V 或 3 V	 M_REG_CTRL3: 0 0 0 0 1 0 1; 0(W)/1(R); Read / Write HVLDO_STBY: Description: Enable/Disable HVLDO in standby mode; 0: Disabled; 1: Enabled; Reset condition: POR

3	PMIC_Write_FS(0x9, 0x0F);
	// configure TIMING_WINDOW_STBY[3:0] in FS_I_SAFE_INPUTS register
	PMIC_Write_FS(0xa, ~(0x0F)); // configure TIMING_WINDOW_STBY[3:0] in FS_I_SAFE_INPUTS register
		/* Configure time window 1111b equals 8ms, PMIC wait time window for STBY_S32G_PIN pull down */	
			FS_I_SAFE_INPUTS: 1 0 0 1 0 0 1; 0(W)/1(R); Write during INIT then Read only
			FS_I_NOT_SAFE_INPUTS: 1 0 0 1 0 1 0; 0(W)/1(R); Write during INIT then Read only
			Table 73. Standby Timing Window
			TIMING_WINDOW_STBY[3:0]: 1111; Configure the window duration: 10 ms

4	PMIC_Write(0x2, 0xF400); // configure TIMER_STBY_WINDOW[3:0] in M_SM_CTRL1 register	/* Configure time window 1111b equals 8388608ms, STBY_TIMER_EN=1 ; PMIC MAIN logic window - move to deep fail safe during STBY */ 客户可以通过 I2C 总线启用或	 TIMER_STBY_WINDOW [3:0]: Description: Set the standby timer window duration (ms); [0,1,10,11,100,101,110,111]: [16,32,128,512,1024,4096,8192,16384]; [1000,1001,1010,1011,1100,1101,1110,1111]: [65536,131072,262144,524288,1048576,2097152,4194304,8388608]; Reset condition: POR

		禁用待机定时器。 这个计时器防止设备陷入待机模式。因为，当计时器到期时，两种设备，VR5510 和 S32G，被复位。 定时器持续时间可以配置为 TIMER_STBY_WINDOW[3:0]	
5	PMIC_Write_FS(0x0F, 0x0200); PMIC_Write_FS(0x10, 0xFDFF);	/*config WDW_period [3:0] = 0, disable WD*/	Table 44. Watchdog Window period configuration WD_WINDOW[3:0] | Watchdog Window Period 0000 | DISABLE (during INIT_FS only)
6	PMIC_Write_FS(0x12, 0xA54D);	/*Feed watchdog once in order to disable it*/	Table 114. FS_WD_ANSWER register description WD_ANSWER [15:0] | Description | Watchdog answer value from the MCU 0… | Challenger WD Answer = (NOT(((LFSR x 4)+6)–4))/4 (refer to **Chapter 21.4.2**) …1 | Simple WD Answer = 0x5AB2 (refer to **Chapter 21.4.1**) Reset condition | POR
7	PMIC_Write_FS(0x12, 0x4A9A); PMIC_Write_FS(0x12, 0x9535); PMIC_Write_FS(0x12, 0x2A6A); PMIC_Write_FS(0x12, 0x54D4); PMIC_Write_FS(0x12, 0xA9A9); PMIC_Write_FS(0x12, 0x5353);	/*Feed watchdog once in order to disable it*/	
8	PMIC_Write_FS(0x14, 0x6565);		Table 116. FS_RELEASE_FS0B register description RELEASE_ FS0B [15:0] | Description | Secure 16bits word to release FS0B 0… | Depend on WD_SEED value and calculation …1 Reset condition | POR
9	status = PMIC_Read_FS(0x18, &read_data);	/*Read the status again. the status must be in NORMAL_FS status right before * standby entry */	

4.2 PMIC_standbyEntry

Step	代码	说明	寄存器

<table>
<tr>
<td>0</td>
<td>PMIC_Write(0x9, 0x3);</td>
<td>/* Configure low power clock frequency for VPRE in M_CLOCK2 register /0b11 600Mhz _ 20mA capacity
*
need to be wrote 40us before PMIC move to STBY*/</td>
<td>
Table 90. M_CLOCK2 register description
<table>
<tr><td rowspan="6">LOW_POWER_ CLK [1:0]</td><td>Description</td><td>Low Power Clock frequency selection</td></tr>
<tr><td>00</td><td>100kHz</td></tr>
<tr><td>01</td><td>100kHz</td></tr>
<tr><td>10</td><td>300kHz</td></tr>
<tr><td>11</td><td>600kHz</td></tr>
<tr><td>Reset condition</td><td>POR</td></tr>
</table>
</td>
</tr>
<tr>
<td>1</td>
<td>PMIC_Write_FS(0x15, 0x02);</td>
<td>/*PMIC standby entry request from MPU*/</td>
<td>
<table>
<tr><td>FS_SAFE_IOS</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0(W) /1(R)</td><td>Read / Write</td></tr>
</table>
<table>
<tr><td>RSTB_REQ</td><td>Description</td><td>Request assertion of RSTB (Pulse)</td></tr>
</table>
VR5510 All information provided in this document is subject to legal disclaimers.
Product data sheet Rev. 2 — 22 December 2020
NXP Semiconductors
Multi-Output
<table>
<tr><td></td><td>0</td><td>No Assertion</td></tr>
<tr><td></td><td>1</td><td>RSTB Assertion (Pulse)</td></tr>
<tr><td></td><td>Reset condition</td><td>POR</td></tr>
</table>
</td>
</tr>
</table>

5 定制修改

5.1 关闭 RTC 唤醒功能

如果只考虑使用 CAN 的 RX 的 GPIO 功能来唤醒，不需要使用 RTC 功能，如下修改测试：

S32G274Astandbymode\src\ S32G_Standby.c

unsigned char S32G_WKUP_SOURCE_NO = 0; // Wake-up source number, from 0-22, 31; 0 means WKUP0=CAN0_RX
void standby_entry()

{

//S32G_WKUP_SOURCE_NO = 31; // Wake-up source number 31 RTC

// th_printf("\n ACK #3:RTC period is:%d ms\n", RTC_PERIOD_CFG_ms);

 // RTC_set(RTC_PERIOD_CFG_ms);

void WKPU_set(uint8_t BOOT_MODE_SET, uint8_t WKUP_SOURCE)

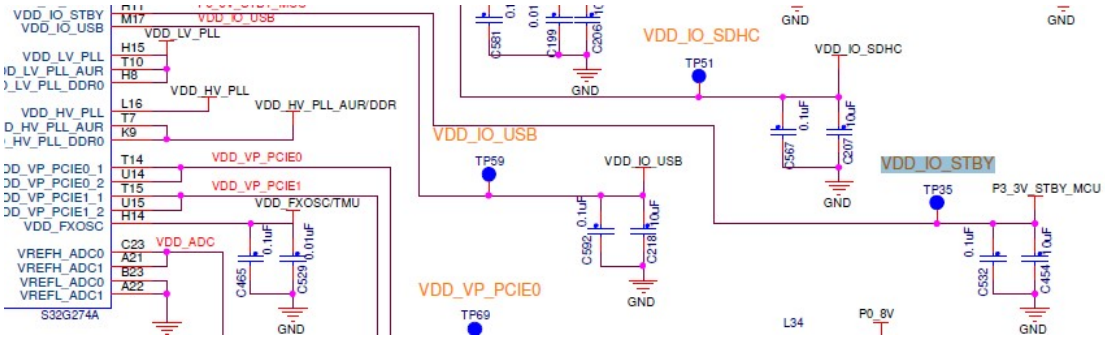
{

// WKPU_WIREER = 0x80000000;
  //WKPU_WRER = 0x80000001; // LLCE CAN 0

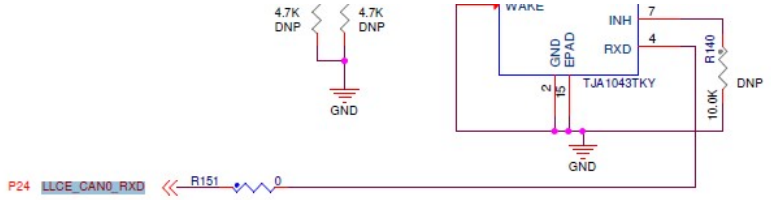
  WKPU_WRER = 0x00000001; // LLCE CAN 0

测试：

LLCE_CAN0_RX 由 VDD_IO_STBY=3.3V 供电，如下：



LLCE_CAN0_RX 测试点为：



使用 full boot 来测试：R151 是上拉至 3.3V 的，的以只需要把 R151 对地轻触一下，就可以唤醒，如下：

PMIC initial status Read:2006

PMIC status before entry:a

Set the Boot model

0: FULL BOOT; 1: Standby-RAM BOOT;

Set the Boot model - FULL BOOT;

ACK #2 :WAKE-UP Source number[31:0] is : #0

PMIC initial status Read:6

PMIC status before entry:a

Set the Boot model

0: FULL BOOT; 1: Standby-RAM BOOT;

5.2 打开 CAN1_RX 唤醒功能

本节说明如何支持更多的唤醒源，比如说增加 LLCE_CAN1_RX 的 GPIO 唤醒，则在以下修改的基础上增加：

S32G274Astandbymode\src\ S32G_Standby.c

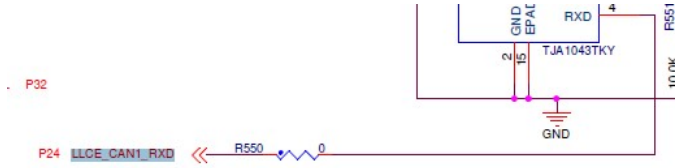
```
//unsigned char S32G_WKUP_SOURCE_NO = 0; // Wake-up source number, from 0-22, 31; 0 means WKUP0=CAN0_RX
    unsigned char S32G_WKUP_SOURCE_NO = 1; // Wake-up source number, from 0-22, 31; 1 means WKUP1=CAN1_RX, CAN0_RX already set in function WKPU_set
```

PJ_02	146	SIUL2_1	0000_0000	0x44010488	GPI[146]
PJ_02	746	SIUL2_1	0000_0010	0x44010DE8	LLCE_CAN1
PJ_02	-		-		WKUP1

```
void WKPU_set(uint8_t BOOT_MODE_SET, uint8_t WKUP_SOURCE)
{
    /*step 4 Configure SIUL2*/…

    /*add can1_rx*/
    SIUL2_0.MSCR[146].B.PUE = 1;
    SIUL2_0.MSCR[146].B.PUS = 1;
    SIUL2_0.MSCR[146].B.IBE = 1;
    /*step 8 */

    //WKPU_WIFEER = 0x00000001; // falling down
    WKPU_WIFEER = 0x00000003; // falling down

    /*step 9 */
    //WKPU_WRER = 0x80000001; // LLCE CAN 0
    //WKPU_WRER = 0x00000001; // LLCE CAN 0
        WKPU_WRER = 0x00000003; // LLCE CAN 0/1
```

测试：

LLCE_CAN1_RX 测试点为：



使用 full boot 来测试：所以将 R550 对地轻触，结果为：

PMIC initial status Read:2006

PMIC status before entry:a

Set the Boot model

0: FULL BOOT; 1: Standby-RAM BOOT;

Set the Boot model - FULL BOOT;

ACK #2 :WAKE-UP Source number[31:0] is : #1

PMIC initial status Read:6

PMIC status before entry:a

Set the Boot model

0: FULL BOOT; 1: Standby-RAM BOOT;

5.3只支持 full boot

一般情况下使用 full boot 的情况较多，所以可以不考虑 standby ram fast boot 的情况下可以如下去掉一些代码，（注意 standby ram fast boot 本身会用到 standby ram，所以和 ATF 中使用到的 standby ram 存在冲突，需要处理 memory mapping，但是因为我们只考虑 full boot 的情况，所以本文不做介绍）。

S32G274Astandbymode\src\main.c

```
int main(void)
{
// Init_standbyram();
// qspi_init();
```

S32G274Astandbymode\src\ S32G_Standby.c

```
void standby_entry()
{
//    standbyramc_cpy();
```

Full boot 不需要初始化 standbyram，也不需要初始化 qspi nor 来拷贝 standby ivt 头，standby ram copy 也可以去掉，一些逻辑控制代码也可以去掉，测试方法与前两节相同。

5.4 打开 DDR 相关电源

支持 DDR 自刷新需要给 LPDDR4 和 VDD_IO_DDR0 供电，这样可以加快上电顺序，因为内容 之前已经保存在 DDR 内存中。DDR 刷新模式下 VR5510 开启的电源是 VPRE、HVLDO、LDO2 和 BUCK3。相关功耗数据请参考 VR5510 的应用笔记：《AN12880-5510-standby.pdf》和 LPDDR4 数据手册。

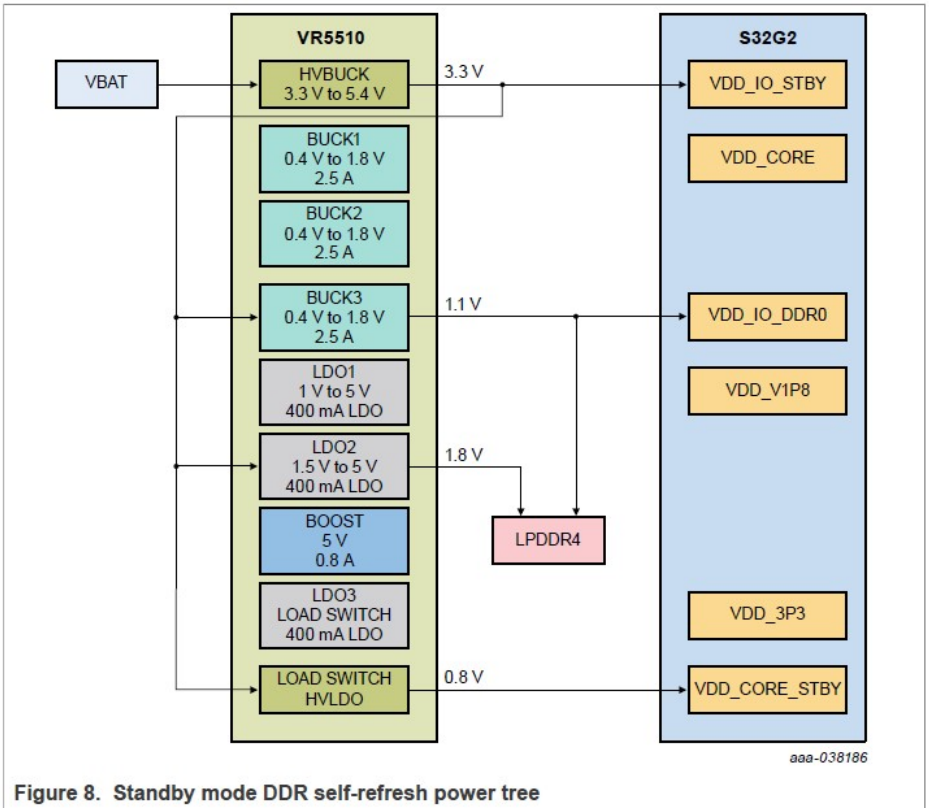


Figure 8. Standby mode DDR self-refresh power tree

如下代码修改：

S32G274Astandbymode\src\ S32G_Standby.c

```
void PMIC_initConfig(void)
{
  //still add buck3 1.1v for vdd_io_ddr0 and lpddr4, and LDO2 for lpddr4
  // PMIC_Write(0x05, 0x100);
  PMIC_Write(0x05, 0x1110);
```

26.6 M_REG_CTRL3 register

Bits	BIT23	BIT22	BIT21	BIT20	BIT19	BIT18	BIT17	BIT16
Write	0	**LDO3 _ STBY**	0	**LDO2 _ STBY**	0	**LDO1 _ STBY**	0	**HVLDO _STBY**
Read	RESERVED	**LDO3 _ STBY**	RESERVED	**LDO2 _ STBY**	RESERVED	**LDO1 _ STBY**	RESERVED	**HVLDO _STBY**
Reset	0	1	0	1	0	1	0	1

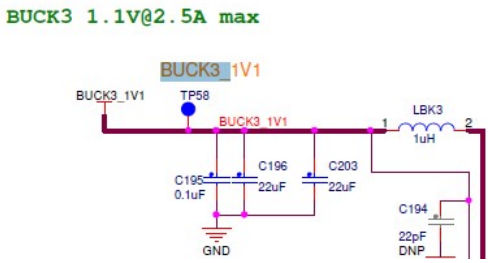
BIT15	BIT14	BIT13	BIT12	BIT11	BIT10	BIT9	BIT8
0	**VPREV_ STBY**	0	**BUCK3_ STBY**	0	**BUCK2_ STBY**	0	**BUCK1_ STBY**
RESERVED	**VPREV_ STBY**	RESERVED	**BUCK3_ STBY**	RESERVED	**BUCK2_ STBY**	RESERVED	**BUCK1_ STBY**
0	1	0	1	0	1	0	1

BUCK3_STBY	Description	Enable/Disable BUCK3 in standby mode
	0	Disabled
	1	Enabled
	Reset condition	POR
VPREV_STBY	Description	Set the VPRE voltage in standby mode (only if VPREV_STBY_EN_OTP = 1)

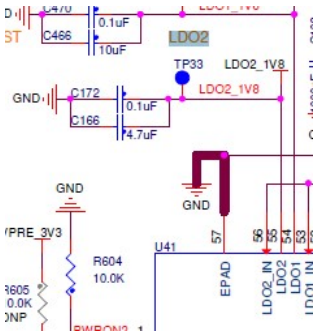
LDO2_STBY	Description	Enable/Disable LDO2 in standby mode
	0	Disabled
	1	Enabled
	Reset condition	

测试结果：

Buck3 测试点为：



LDO2 的测试点为：



所以在进入 stand by mode 后，量测一下相关测试点电平是否保留，测得 TP33=1.8V，TP58=3.3V。可以保留。

5.5 修改调试串口为 UART1

因为 Linux 会用到 UART0，所以 M7 standby demo 修改为 UART1，如下代码:

S32g274astandbymode\src\s32g_linflexd\s32g_linflexd.c

```
bool s32g_linflexd_init()
{
#if 1 /*从#if 0 修改为#if 1，变为初始化 UART1*/
        …/*初始化 UART1*/
#else
        … /*初始化 UART0*/
#endif
```

write_char 和 get_char 函数相应的编译宏控制开关也同理切换。

```
#if 1
char get_char(void)
{…/*使用 UART1*/
#else
      …/*使用 UART0*/
#endif

#if 1
void write_char(char value)
{…/*使用 UART1*/
#else
      …/*使用 UART0*/
```

```
#endif
```

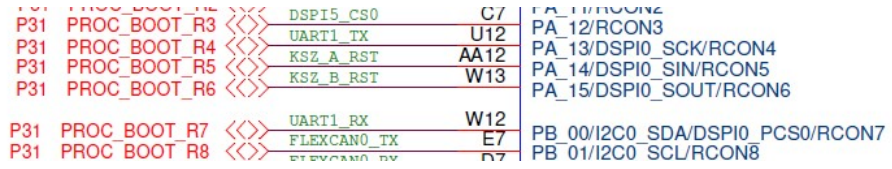
注意，demo 中 write_char 的写法，以 UART1 为例：

```
void write_char(char value)
{
    uint32_t delay;

    LINFLEXD_1.BDRL.B.DATA0 = value;
    // do{}while ( LINFLEXD_0.UARTSR.B.DTFTFF == 0 );
    while((LINFLEXD_1.UARTSR.R & 0x2) == 0); //打开这个，修改为 LINFLEXD_1
    // for(delay=0;delay<0xFFFF;delay++); //default 是使用 delay 来实现的，但是如果 CPU 时钟频率有变化，则使用这种办法就有风险，所以建议修改为上面这种检查寄存器 bit 位的方法。
    LINFLEXD_1.UARTSR.R = 0x00000002;                    // clear DTF bit, only
    if ( value == '\n' ) write_char('\r');
}
```

S32g274astandbymode\src\boards\nxp_s32g_vnp_rdb\s32g_vnp_rdb_siul.c

```
void s32g_vnp_rdb_uart_siul()
{
#if 1 /*从#if 0 修改为#if 1，变为 UART1 iomux*/

/**********PA13 and PB00*****LINFLEX1***/
    SIUL2_0.MSCR[13].R = 0x200002;

    SIUL2_0.MSCR[16].R = 0x80000;
    SIUL2_1.IMCR[224].R = 0x2;
#else
    …/*UART0 IOMUX*/
#endif
```

如下硬件设计：



及 IOMUX 寄存器配置：

PA_13		13	SIUL2_0	0000_0000	0x4009C274	GPIO[13]

PA_13				0000_0001		SPI0_SCK_0
PA_13				0000_0010		LIN1_TX

LINFlexD_1	LIN1_RX	SIUL2_1	736	0000_0000	-	disable_low
				0000_0001	-	disable_high
				0000_0010	IO_PAD	PB_00
				0000_0011	IO_PAD	PB_10
				0000_0100	IO_PAD	PC_04

测试：

将 M7 核调试串串口连接在 J1，UART1 上，可以看到调试信息从 UART1 上打印出来。

5.6 修改设备驱动时钟

以 UART 为例，目前 standby demo 中的时钟树如下：

代码：

S32g274astandbymode\src\s32g_clock\s32g_pll\s32g_pll.c

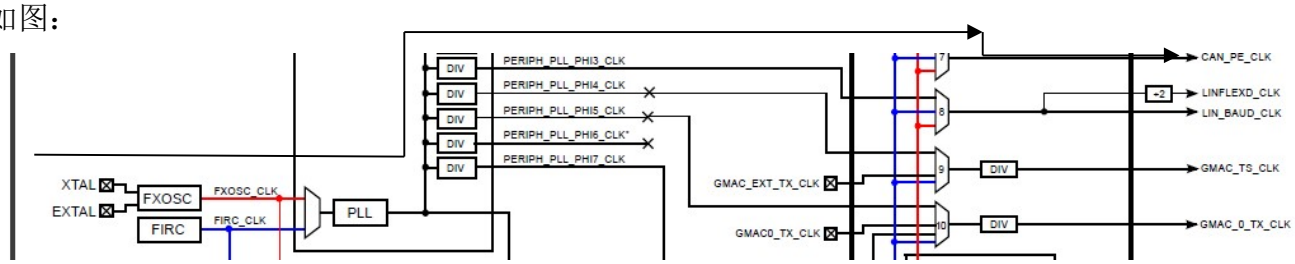
```
void s32g_pll_peripheral()
{
    /* * f(pll_vco) = (f(pll_ref)/PLLDV[RDIV]) * ((PLLDV[MFI]) + (PLLFD[MFN])/18432)
     * PLL_VCO = 2000MHz*/
/* Divider value = 24+1, 2Ghz/25 = 80 MHz, LIN clk set to 80MHz */
    /* Selecting FXOSC as the source for PLL = 1
     * Selecting FIRC as the source for PLL = 0
     * FIRC_FREQ = 48MHz (fixed)
     * FXOSC_FREQ = 40MHz */
```

S32g274astandbymode\src\s32g_clock\s32g_clocking.c

```
void s32g_periph_clock_selector_configure(void)
{
    /* Clock divider for MC_CGM_0 MUX_8 :
     * Divider: Enable
     * Division Factor: 1
```

```
    * Phase shift: 0
    * Clock source = 15 (PERIPH_PLL_PHI3_CLK)
    * LIN CLK = 80MHz
    */
```

如图：



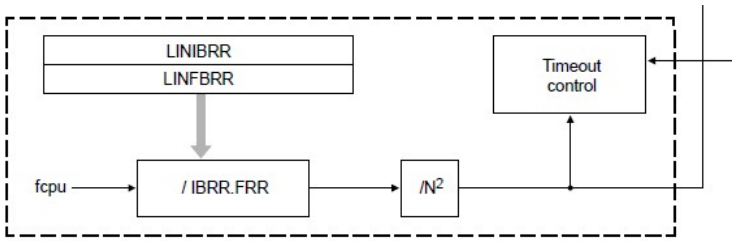
UART 内部的波特率配置如下：

代码：

S32g274astandbymode\src\s32g_linflexd\s32g_linflexd.c

```
bool s32g_linflexd_init()
{
/* UART, rx enable, tx enable, Tx buffer-mode, Rx buffer-mode, 1byte buffer, no Parity, 8bit data */
    LINFLEXD_0.UARTCR.B.UART = 0x1;
    LINFLEXD_0.UARTCR.B.TXEN = 0x1;
    LINFLEXD_0.UARTCR.B.RXEN = 0x1;
    LINFLEXD_0.UARTCR.B.WL0 = 0x1;

    /* LINFLEX_0 to use LIN_CLK as 80MHz */
    /* 115200 Baud */
    LINFLEXD_0.LINIBRR.R = 43;
    LINFLEXD_0.LINFBRR.R = 6;
```

说明：



Baud rate is calculated with the following formula for both receiver and transmitter.

When UARTCR[ROSE] = 1, Tx = Rx = LIN_CLK / (OSR × LDIV). When UARTCR[ROSE = 0, Tx = Rx = LIN_CLK / (16 × LDIV),

where LIN_CLK is the frequency of the baud clock.

LDIV is an unsigned fixed point number. The mantissa is coded into 20 bits of LINIBRR[IBR] and the fraction is coded on 4 bits of LINFBRR[FBR].

When reduced oversampling is enabled, LINFBRR must not be used and programmed to zero, and LDIV contains only the integer part of LINIBRR.

Table 307. Examples of baud rate calculations

UARTCR1[ROSE]	Mode(s)	LDIV	LIN_CLK	LINIBRR[IBR]	LINFBRR[FBR]	Baud rate
ROSE = 0	LIN and UART	468.75 d	36 MHz	468 d	12	36 MHz/ (16 × 468.75) = 4.8 Kbit/s
ROSE = 1	UART	5 d	80 MHz	5 d	4	LIN_CLK / (OSR × LDIV) = 80 MHz / (4 × 5) = 4 Mbit/s

27-24 OSR	Over Sampling Rate Configures the number of samples taken for a bit when reduced oversampling is enabled. In most cases, OSR can be 4, 5, 6 or 8. When idle state monitoring is enabled (MIS=1), OSR can be only 4 or 8. 4 or 8.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	Register bit can be read in any mode, written only in initialization mode when UART bit is set.
23 ROSE	Reduced Over Sampling Enable Register bit can be read in any mode, written only in initialization mode when UART bit is set. 0b - Each bit is over sampled sixteen times. 1b - OSR determines the oversampling rate.

由于代码中 ROSE 未配置=0， 所以 over sampling=16。

的以 80Mhz/((43.(6/16))/16)=115274 baudrate。

以下我们将 LIN_CLK 根时钟配置为 125Mhz，与 Linux 内部相同，再配置 baudrate 为 115200。

代码：

S32g274astandbymode\src\s32g_clock\s32g_pll\s32g_pll.c

```
void s32g_pll_peripheral()
{
/* Divider value = 24+1, 2Ghz/25 = 80 MHz, LIN clk set to 80MHz */
    //PERIPH_PLL.PLLODIV[3].R = 0x180000; //LIN_CLK
    /* Divider value = 15+1, 2Ghz/16 = 125 MHz, LIN clk set to 125 MHz */
    PERIPH_PLL.PLLODIV[3].R = 0xf0000; //LIN_CLK
```

S32g274astandbymode\src\s32g_linflexd\s32g_linflexd.c

```
bool s32g_linflexd_init()
{
    /* LINFLEX_0 to use LIN_CLK as 80MHz */
    /* 115200 Baud */
    //LINFLEXD_0.LINIBRR.R = 43;
    //LINFLEXD_0.LINFBRR.R = 6;
    /* LINFLEX_0 to use LIN_CLK as 125MHz */
    /* 115200 Baud */
    LINFLEXD_0.LINIBRR.R = 67; //125Mhz/((67.(13/16))/16)=115207
    LINFLEXD_0.LINFBRR.R = 13;
```

测试结果可以正常打印串口消息。

注意：如果有其它地方已经初始化了时钟数，则 standby demo 中的时钟初始化可以注掉，则设备驱动中的 divider 根据实际时钟树配置。

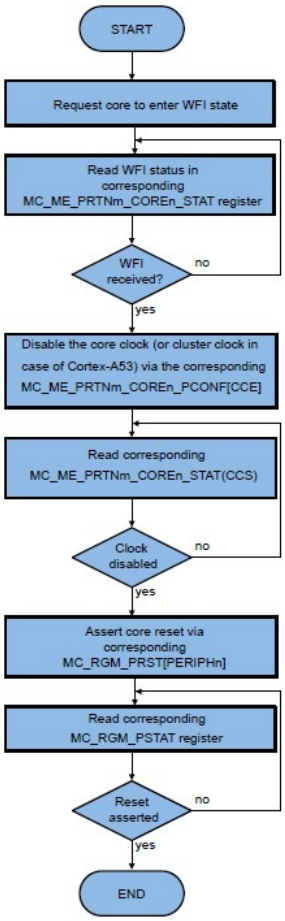
```
int main(void)
{
    //s32g_dfs_reset();
    /* clock initialization */
    //s32g_clock_tree_init();
```

5.7 事先关掉所有其它的非主核

本 demo 目前只考虑了一个 M7_0 核运行 demo 的情况，实际应用中需要将其它核先置于 WFI，然后关掉，最后再关掉最后一个 M7_0 主核。所以以下代码给出了关掉 A53_0 核的示例。

另外，非 M7_0 核的其它 M 核也建议由 M7_0 核来关闭，而不由 A 核 Linux 来关闭，代码与下类似，本文不再说明。

关闭一个核流程如下：



Core turn-off sequence

32.1.9 Disable core clocks

1. Indicate that you want to disable the core clock by writing 0 to PRTN*m*_CORE*n*_PCONF[CCE].
2. Enable the clock hardware-update processes by writing 1 to PRTN*m*_CORE*n*_PUPD[CCUPD].
3. Start the enabled hardware-update processes by writing the control key to CTL_KEY[KEY].

   MC_ME requests closing of the clock gates for the specified core.
4. Ensure the clock hardware process has been triggered by writing 1 to PRTN*m*_PUPD[PCUD].
5. Ensure the core clock gates are closed, indicated by reading 0 from PRTN*m*_CORE*n*_STAT[CCS].

S32G274Astandbymode\src\ S32G_Standby.c

```
//johnli add

#define BIT_32(nr)                          ((1) << (nr))
#define BIT                                 BIT_32
#define S32_MC_ME_PRTN_N_CORE_M_STAT_CCS_MASK          BIT(0)
#define S32_MC_ME_PRTN_N_CORE_M_STAT_WFI_MASK          BIT(31)
```

S32G Standby Demo

```
/* Apply changes to MC_ME partitions 此函数用于正式触发partition状态切换 */
void mc_me_apply_hw_changes(void)
{
        MC_ME_CTL_KEY = 0x5AF0;
        MC_ME_CTL_KEY = 0xA50F;
}
/*决定关闭核clock*/
```

0 CCE	Core 0 clock enable This bit controls whether the clock to Core 0 in partition 1 should be enabled or disabled. 0b - Disable the core clock 1b - Enable the core clock

```
static void mc_me_part_core_pconf_write_cce_a53_0(void)
{
   MC_ME_PRTN1_CORE0_PCONF=0x0;
}
/*使能clock硬件更新*/
```

0 CCUPD	Core 0 clock update This bit controls whether the hardware processes for enabling/disabling the clock to Core 0 in the partition 1 should be triggered or not. 0b - Do not trigger the hardware process 1b - Trigger the hardware process

```
static void mc_me_part_core_pupd_write_ccupd_a53_0(void)
{
        MC_ME_PRTN1_CORE0_PUPD=0x1;
}

/*检查 clock gate是否成功*/
```

0 CCS	Core 0 clock process status This bit provides the status of the clock corresponding to core clock enablement/disablement. 0b - Clock is inactive. 1b - Clock is active.

```
static bool s32_core_clock_running_a53_0(void)
{
        uint32_t stat;

        stat = MC_ME_PRTN1_CORE0_STAT;
        return ((stat & S32_MC_ME_PRTN_N_CORE_M_STAT_CCS_MASK) != 0);
}

uint8_t get_rgm_a53_bit(uint8_t core)
{
        static uint8_t periph_rgm_coresp[] = {
                /** Cluster 0, core 0*/
                [0] = 65,
                [1] = 66,
```

```
                [2] = 69,
                [3] = 70,
                /** Cluster 1, core 0*/
                [4] = 67,
                [5] = 68,
                [6] = 71,
                [7] = 72,
        };

        return periph_rgm_coresp[core] % 64;
}
/*A53 partition reset如下*/
```

32.1.12 Assert partition reset signals

1. Request assertion of the partition's reset signals by writing the appropriate value to MC_RGM.PRST*n*_0.
2. Ensure the partition reset signals are asserted by reading MC_RGM.PSTAT*n*_0.

```
void s32_reset_core_a53_0(void)
{
        uint32_t resetc;
        uint32_t statv;

   resetc = BIT(get_rgm_a53_bit(0));
        statv = resetc;

        /* Assert the core reset */
   resetc |= MC_RGM_PRST1_0;
   MC_RGM_PRST1_0 = resetc;

        /* Wait reset status */
        while (!(MC_RGM_PSTAT1_0 & statv))
                ;
}


void s32_turn_off_core_a53_0(void)
{
        uint32_t stat;

        /* Assumption : The core is already in WFI */
        stat = MC_ME_PRTN1_CORE0_STAT;

        /* The clock isn't enabled */
        if (!(stat & S32_MC_ME_PRTN_N_CORE_M_STAT_CCS_MASK))
                return;

        /* Wait for WFI */
        do {
                stat = MC_ME_PRTN1_CORE0_STAT;
        } while (!(stat & S32_MC_ME_PRTN_N_CORE_M_STAT_WFI_MASK));
```

```
        /* Disable the core clock */
        mc_me_part_core_pconf_write_cce_a53_0();
        mc_me_part_core_pupd_write_ccupd_a53_0();

        /* Write valid key sequence to trigger the update. */
        mc_me_apply_hw_changes();

        /* Wait for the core clock to become inactive */
        while (s32_core_clock_running_a53_0())
                ;

        s32_reset_core_a53_0();
}

    //end
    void standby_entry()
    {…
        PMIC_standbyEntry();
        s32_turn_off_core_a53_0(); //johnli add
        clock_shutdown();
    …
```

6 修改为 MCAL Demo

本 Demo 本身是 S32DS 环境的 Hard Coding 代码，一般客户考虑集成的问题是希望实现成一个 MCAL Demo，本章以 Uart_Example_S32G274A_M7 工程为基础说明如何修改为一个 MCAL 的 Standby Demo，使用 full boot+GPIO 唤醒的方式，主要的修改包括：

- 修改 UART 的波特率，及 M 核时钟以及一些时钟配置。
- 实现关闭时钟的 API。
- 配置 MCU 模块的 Standby 模式。
- 配置唤醒源，需要增加并配置 ICU，EcuM 模块，并修改相关代码。
- 加入 PMIC 驱动，需要增加并配置 PMIC，I2C 和 DIO 模块，并修改 PMIC 进入 Standby 的代码。
- 主函数调用逻辑。

根据文档《S32G_RTD_MCAL_V*.pdf》说明创建一个 UART example 工程，并编译通过。本文使用的 RTD 版本为 SW32G_RTD_4.4_3.0.2。

6.1 修改 UART 驱动

1　修改 Uart 的波特率(将波特率设置为 115200)：

Uart_Example_S32G274A_M7->someid(…)->Uart(…)-> Uart->UartChannel-> UartChannel_1：

- DesireBaudrate= LINFLEXD_UART_BAUDRATE_115200

2　修改 M7 Core 的时钟(将 M7 核时钟设置为正常使用的 400Mhz)：

Uart_Example_S32G274A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0：

->McuCorePLL->McuPll_Configuration：

- CORE PLL under MCU control=checked
- Core PLL Enabled=checked
- MFD (1 -> 255) =50 //然后 McuPll_parameter 中的 PLL_VCO 自动计算为 2G

->McuCoreDFS->McuDfs_1：

- DFS1 Output Port Enable=checked
- DFS1 MFI=1
- DFS MFN=9
- DFS1_CLK Frequency=8.0E8 //自动计算为

->McuCgm0ClockMux0

- CGM0 Clock Mux0 Source*=CORE_PLL_DFS1_CLK
- Clock Mux0 Frequency (XBAR_2X_CLK)= 8.0E8 //自动计算为 8.0E8，所以 M7 clock=400Mhz。
- CGM0 Clock Mux0 Divider0 Enable (LBIST_CLK)=unchecked //bug fix
- Clock Mux0 Divider1 Frequency (DAPB_CLK)=1.3E8//自动计算为

3　修改工程的一些时钟配置(使 M7 可以控制 FXOSC 的开关)：

Uart_Example_S32G274A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0：

->McuFXOSC：

- FXOSC under MCU control*=checked//方便之后控制 FXOSC 的开关

4.　生成代码：

```
Mcu_InitClock(mcu.c)
|->Mcu_Ipw_InitClock
|  |->Clock_Ip_InitClock
|  |  |->Clock_Ip_SpecificPlatformInitClock /* DFS reset, FIRC_CLK configuration etc. */
|  |  |-> Clock_Ip_ResetClockConfiguration
/*********************************************************************
  *** Ramp down to safe configuration. Reset elements from clock tree:
  *** selectors, fractional dividers, plls and xoscs
  *********************************************************************/
|  |  |->  /*******************************************************
  *** Load the new configuration. Selectors that might
  *** be clocked from PLLs shouldn't be configured.
  *******************************************************/
|  |  |->  /* Initialize clock objects, internal driver data */
|  |  |-> /* Configure the PCFS  */
|  |  |->  /* Configure the clock divider triggers that are under MCU control */
|  |  |-> /* Configure the clock dividers that are under MCU control */
|  |  |-> /* Trigger update for all divider trigger that are under MCU control */
|  |  |->  /* Configure PLL clock generators */
|  |  |->  /* Configure PLL clock generators */
|  |  |-> /* Configure fractional dividers */
|  |  |->  /* Switch the clock multiplexers under MCU control to the configured source clocks */
  /* Note: if the configured source clock of a ClockMux is the output clock of a PLL/DFS,
   * the configuration will be skipped and the respective ClockMux will be switched in
   * the "Clock_Ip_DistributePllClock" function instead, when the source clock will have
   * stabilized already. */
|  |  |-> /* Enable the Clock Monitoring Units ( CMU0 .. n ) according to configuration. */
```

6.2 实现时钟关闭代码

在函数 mcu_ts_*\src\clock_ip.c 中，有自己的本地函数：

```
void Clock_Ip_ResetClockConfiguration(Clock_Ip_ClockConfigType const * Config)
{ …/* Ramp down all selectors from configuration to SAFE_CLOCK */
  …/* Put in reset state all fractional dividers from configuration */
```

```
…/* Power down all plls from configuration */
…/* Power down all xoscs from configuration */
```

此函数实行了时钟的彻底反初始化到最初安全的状态，所以我们需要将这个函数暴露出来实现时钟关闭的功能，如下修改：

mcu_ts_*\src\clock_ip.c

```
// static void Clock_Ip_ResetClockConfiguration(Clock_Ip_ClockConfigType const * Config);
//static void Clock_Ip_ResetClockConfiguration(Clock_Ip_ClockConfigType const * Config)
void Clock_Ip_ResetClockConfiguration(Clock_Ip_ClockConfigType const * Config)
```

mcu_ts_*\include\clock_ip.h

```
void Clock_Ip_ResetClockConfiguration(Clock_Ip_ClockConfigType const * Config); //johnli add
```

mcu_ts_*\src\Mcu_ipw.c

```
void Mcu_Ipw_ResetClockConfiguration(const Mcu_ClockConfigType * ClockConfigPtr)
{
  Clock_Ip_ResetClockConfiguration(ClockConfigPtr);
}
```

mcu_ts_*\include\mcu_ipw.h

```
void Mcu_Ipw_ResetClockConfiguration(const Mcu_ClockConfigType * ClockConfigPtr);
```

mcu_ts_*\src\Mcu.c

```
void Mcu_ResetClockConfiguration(Mcu_ClockType ClockSetting)
{

Mcu_Ipw_ResetClockConfiguration(&(*Mcu_pConfigPtr->ClockConfigArrayPtr)[Mcu_au8ClockConfigIds[ClockSetting]]);
}
```

mcu_ts_*\include\mcu.h

```
void Mcu_ResetClockConfiguration(Mcu_ClockType ClockSetting);
```

然后 main 函数调用：Mcu_ResetClockConfiguration(McuClockSettingConfig_0);就可以关闭时钟了。

6.3 配置电源模式切换驱动

Uart_Example_S32G274A_M7->someid(…)->Mcu(…)->Mcu：

->General->McuGeneralConfiguration：

- Mcu Enter Low-Power Mode =checked //打开 standby 支持

->General->McuModelConfiguration：

- Mcu Number of Mode Settings*=2//增加一个 MCU 模式

->McuModeSettingConf：

在 McuModeSettingConf_0 上右击，选择 duplicate element 复制一个配置来修改。

打开 McuModeSettingConf_1：

->General:

- Mode ID*=1
- Operating Mode*=SOC_STANDBY //表示此模式为 Standby
- Main Core Select*=CM7_0
- Mcu Enable Sleep On Exit*=checked //支持唤醒

M7 Demo 主要负责 M7 核与 A53_0 核的 Standby，所以只需要配置 McuPartition0Config 和 McuPartition1Config：

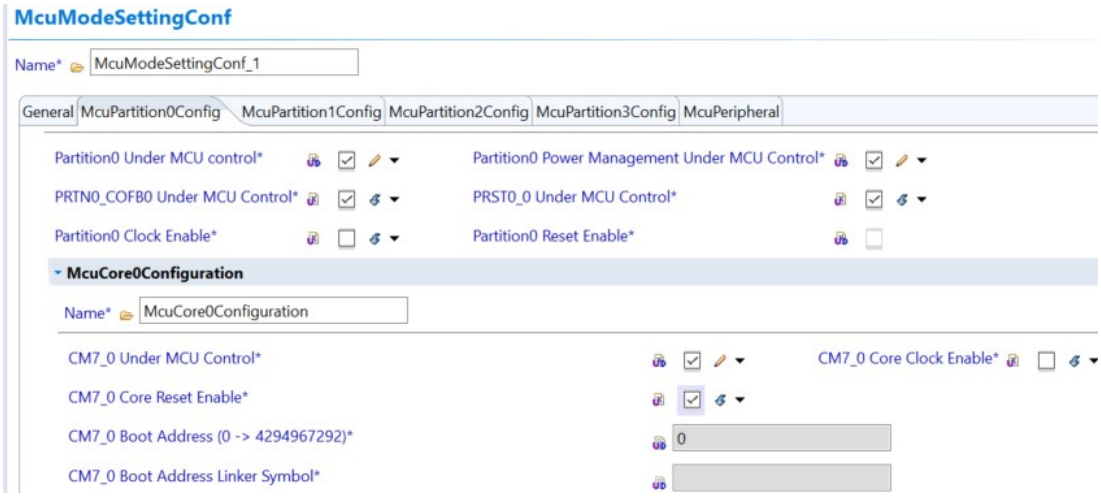
->McuPartition0Config：

- Partition0 Under MCU control*=checked //default
- Partition0 Power Management Under MCU Control*=checked //default
- PRTN0_COFB0 Under MCU Control*=checked //修改增加
- PRST0_0 Under MCU Control*=checked //修改增加
- Partition0 Clock Enable*=checked//修改保留此 partition 的时钟
- Partition0 Reset Enable*=unchecked //default，是否通知 reset signal，由于 M7_0 是最后一个主核，所以不需要，此项不可以配置

->McuCore0Configuration：

- CM7_0 Under MCU Control*=checked //default
- CM7_0 Core Clock Enable*=checked //修改，保留此 M7_0 时钟 enabled。
- CM7_0 Core Reset Enable*=unchecked//修改，通知不要 reset 此核。

剩余的 M7_1~2，也可以配置来管理，本文不配置。但是因为 bug fix 需要把 CM7_1/2 的 clock enable 关闭。Reset enable 打开。



->McuPartition1Config：

- Partition1 Under MCU control* =checked //default
- Partition1 Power Management Under MCU Control*=checked //default
- PRTN1_COFB0 Under MCU Control*=checked //修改
- PRST1_0 Under MCU Control*=checked //修改
- Partition1 Clock Enable*=unchecked //修改，关闭 A53 Partition 时钟。
- Partition1 Reset Enable*=checked //default

->McuCore0Configuration：

- Cortex-A53 CORE 0 cluster 0 Under MCU Control*=checked//修改，需要使用 M 核代码来关闭 A53_0
- CA53 cluster0 Core Clock Enable*=unchecked //default，关闭 A53_0 时钟 enabled。
- Cortex-A53 CORE 0 cluster 0 Reset Enable*=checked //default

其余的 A53_1~3 不要修改，由 Linux 负责关闭。

生成代码在：

Uart_ts_*/examples/ebt/uart_example_s32g274a_m7/generate/src/power_ip_vs_0_pbcfg.c

```
Power_Ip_aModeConfigPB_VS_0
/* Start of Mcu_aModeConfig[1] */
  {
    /* Mode Configuration ID. */
    (Power_Ip_ModeType)1U,
```

```
        /* The Power Mode name (code). */
        POWER_IP_SOC_STANDBY_MODE, //soc standby mode

        /* The Sleep On Exit configuration */
        (boolean)TRUE,

        /* MC_ME IP Mode settings. */
        &Power_Ip_MC_ME_ModeConfigPB_1_VS_0,
        /* MC_RGM IP Mode settings. */
        &Power_Ip_MC_RGM_ModeConfigPB_1_VS_0
    } /* End of Mcu_aModeConfig[1] */
```

Power_Ip_MC_ME_ModeConfigPB_1_VS_0->Power_Ip_MC_ME_aPartitionConfigPB_1_VS_0->/* The configuration structure for Partition 0. */->Power_Ip_MC_ME_aPartition0CoreConfigPB_1_VS_0

```
/* The configuration structure for Partition 0 Core 0. */
    {
        /* Specifies whether the given core is under MCU control. */
        (boolean)TRUE,

        /* The index of the core within the partition. */
        (uint8)0U,

        /* The boot address of the core. */
        (uint32 *)0x00000000U,

        /* The process configuration register value of the core. */
        MC_ME_PRTN0_CORE0_PCONF_CCE
        (
            MC_ME_PRTNX_COREX_PCONF_CCE_DIS_U32
        )
    },
```

Power_Ip_MC_RGM_ModeConfigPB_1_VS_0->Power_Ip_MC_RGM_aDomainConfigPB_1_VS_0->Power_Ip_MC_RGM_aDomain0CoreConfigPB_1_VS_0

```
/* The configuration structure for Domain 0 Core 0. */
    {
        /* Specifies whether the given core is under MCU control. */
```

```
    (boolean)TRUE,

    /* The index of the core within the domain. */
    (uint8)0U,

    /* The reset enable register value of the core. */
    MC_RGM_PRST0_COFB0_RSTEN
    (
        ((uint32)0x00000000U)
        | MC_RGM_PRST0_COFB0_RSTEN_CORES_MASK(0U)
    ),

    /* Mask containing the Core blocks to be updated. */
    MC_RGM_PRST0_COFB0_RSTEN_CORES_MASK(0U)
  }
```

所以通过如下 API 就可以进入 soc_standby 模式：

```
/* Apply a mode configuration */
  Mcu_SetMode(McuModeSettingConf_1); 函数调用关系为：
|->Mcu_Ipw_SetMode
|   |->Power_Ip_SetMode
|   |   |->Power_Ip_OnOffPartCoreCofb/* turn on/off partitions, cores and COFBs */
|   |   |->Power_Ip_MC_ME_SocStandbyEntry
|   |   |     |-> OsIf_SuspendAllInterrupts   /* Disable all IRQs */
|   |   |     |->Power_Ip_pxMC_ME->MAIN_COREID =
ModeConfigPtr->McMeModeConfigPtr->MainCoreIdRegValue;/* Program MC_ME for valid main core id */
|   |   |     |->Power_Ip_pxMC_ME->MODE_CONF =
MC_ME_MODE_CONF(MC_ME_MODE_CONF_STANDBY_MASK); /* Makes a request to go to Standby mode */
|   |   |     |->Power_Ip_MC_ME_TriggerModeUpdate();/* Trigger the update in hardware */
|   |   |     |     |->Power_Ip_pxMC_ME->MODE_UPD =
MC_ME_MODE_UPD_MODE_UPD(MC_ME_MODE_UPD_MODE_UPD_MASK);
|   |   |     |     |->Power_Ip_MC_ME_WriteControlKeys();
  /* Starting the hardware processes */

  /* Write key to MC_ME_CTL_KEY */
```

```
    Power_Ip_pxMC_ME->CTL_KEY = MC_ME_CTL_KEY_KEY(MC_ME_CTL_KEY_DIRECT_KEY_U32);

    /* Write inverted key to MC_ME_CTL_KEY */
    Power_Ip_pxMC_ME->CTL_KEY =
MC_ME_CTL_KEY_KEY(MC_ME_CTL_KEY_INVERTED_KEY_U32);
    |  |  |    |->Call_Power_Ip_CM7_EnableSleepOnExit
    |  |  |    |-> EXECUTE_WAIT/* Execute WFI */
```

6.4 配置唤醒源

1. 首先配置 ICU 模块：

在 Uart_Example_S32G274A_M7 上右击，然后点 Module Configurations，在 Available Modules 中选中 icu(…)，点击右移箭头加入工程中：

Uart_Example_S32G274A_M7->someid(…)->icu(…)->icu->General:

- IcuDevErrorDetect=unchecked//关掉一些参数检查
- Post Build Variant Used*=checked ////修改为
- Config Variant= VariantPostBuild

->IcuConfigSet：

- IcuMaxChannel =1 //修改为 1

-> IcuAutosarExt：

- IcuWkpuStandbyWakeupSupport=checked ////修改为 Icu_Init() will not clear the wakeup flags (WISR register) if it is already set during init.

Uart_Example_S32G274A_M7->someid(…)->icu(…)->icu->icuwkup

点击加号增加 IcuWkpu_0，点击进入，再点击加号增加 IcuWkpuChannles_0

- Wkpu Channel (dynamic range)*=0
- ICU Wakeup Filter Enable*=checked //修改打开。
- ICU Wakeup Pullup Enable*= checked //修改打开，注意此功能目前有 bug，需要 Hard Code 解决。

Uart_Example_S32G274A_M7->someid(…)->icu(…)->icu->IcuChannel：

点击加号增加 IcuChannel_0，点击进入：

->General:

- IcuChannelRef*= /Icu/Icu/IcuConfigSet/IcuWkpu_0/IcuWkpuChannels_0
- IcuDefaultStartEdge*= ICU_FALLING_EDGE\\由于管脚配置选择上拉，所以这个的触发条件一定要选择为下降沿
- IcuMeasurementMode*= ICU_MODE_SIGNAL_EDGE_DETECT
- IcuWakeupCapability*=checked//修改 Channel is wakeup capable

注意，选择此项后，需要打开 IceWakeup，然后：

- Name=IcuWakeup
- IcuChannelWakupInfo=//此项需要加入 EcuM 模块来配置，之后说明如何配置其中的 wakeup source，由于 EcuM 是上层模块，NXP MCAL 里也只是 sample 模块，所以不详细说明。

If the wakeup-capability is true the wakeup source referenced is transmitted to the ECU State Manager (EcuM) .Implementation Type: reference to EcuM_WakeupSourceType

2. 再配置 EcuM 模块(实验性质，给 Icu 引用)：

在 Uart_Example_S32G274A_M7 上右击，然后点 Module Configurations，在 Available Modules 中选中 EcuM(…)，点击右移箭头加入工程中：此处只是简单配置一个 EcuM 的唤醒源给 ICE 去引用，并不详细说明。

Uart_Example_S32G274A_M7->someid(…)->EcuM(…)->EcuM：

->General->EcuMConfiguration-> EcuMCommonConfiguration:

- EcuMConfigConsistencyHash=2 //配置一个值

-> EcuMDefaultsShutdownTarget:

- EcuMDefaultState= EcuMStateSleep

->EcuMGeneral：

- EcuMDevErrorDetect*，EcuMIncludeDet*，EcuMVersionInfoApi*=checked//bug fix
- EcuMMainFunctionPeriod=10//bug fix

->EcuMSleepMode:点击+号增加一个 EcuMSleepMode_0，点击进入

- EcuMSleepModeId=0//
- EcuMSleepModeSuspend*=checked//
- EcuMSleepModeMcuModeRef*=/Mcu/Mcu/McuModuleConfiguration/McuModeSettingConf_1//选择我们在 sleep 模式时的配置。

EcuMWakeupSourceMask:点击+号增加一个。选择下面这个 EcuMWakeupSource_0

->EcuMWakeupSource:点击+号增加一个 EcuMWakeupSource_0，点击进入：

- EcuMWakeupSourceId=0
- EcuMWakeupSourcePolling*=checked

->EcuMOSResource:点击+号增加一个//bug fix

再将 ICE 模块中的 IceChannelWakeupInfo=/EcuM/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource_0

3. 最后配置 PORT 的唤醒源：

Uart_Example_S32G274A_M7->someid(…)->Port(…)->Port->PortContainer->PortContainer_0：

->General：

- PortNumberOfPortPins=//增加一个管脚

-> PortPin：点击+号增加一个管脚，点击进入：配置 LLCE_CAN0_RX 为 WKUP0，pull up。

- Name= LLCE_CAN0_RX_WKUP
- PortPin Pull Enable=checked// pull enable
- PortPin Pull Select =checked// pull up
- PortPin Mode Changeable=checked//default
- PortPin SIUL2 Instance = SIUL2_0
- PortPin Id=3
- PortPin Mscr (dynamic range) =43
- PortPin Direction = PORT_PIN_IN
- PortPin Mode= WKPU_WKUP0
- PortPin Level Value = PORT_PIN_LEVEL_LOW
- PortPin Output Slew Rate= SRE_3_3V_50MHZ

4. 生成代码：

```
Ice_init
|-> Icu_Ipw_Init
|    |->Wkpu_Ip_Init
|    |    |->Wkpu_Ip_Filter/* Set Wakeup/Interrupt Filter Enable Register */

 /** @brief Wkpu HW Channel Filter enable */
      (boolean)TRUE,
```

```
    /* Enables Wakeup/Interrupt Filter Enable Register */
    if (enable)
    {
        base->WIFER |= channelMaskLo;
#ifdef WKPU_IP_64_CH_USED
        base->WIFER_64 |= channelMaskHi;
#endif
    }
|   |   |->Wkpu_Ip_PullUp
/** @brief Wkpu HW Channel Pullup enable */
        (boolean)FALSE,
        /** @brief Wkpu Default Start Edge */
        WKPU_IP_FALLING_EDGE,
    if (enable)
    {
        base->WIPER |= channelMaskLo;
#ifdef WKPU_IP_64_CH_USED
        base->WIPER_64 |= channelMaskHi;
#endif
    }
|   |   |->Wkpu_Ip_SetActivationCondition
        /** @brief Wkpu Default Start Edge */
        WKPU_IP_FALLING_EDGE,
|   |   |   |->Wkpu_Ip_EnableFallingEdge
    /* Enables Wakeup/Interrupt Falling edge event enable Register */
    if (enable)
    {
        base->WIFEER |= channelMaskLo;
#ifdef WKPU_IP_64_CH_USED
        base->WIFEER_64 |= channelMaskHi;
#endif
    }
```

5. 修改 PULL UP ENABLE 的 bug，有两点：

- WKPU_IP_SUPPORT_PULLUP 编译控制宏不起作用，注销掉。
- WIPER 名字错，应该为 WIPUER_WIPDER。

icu_ts_*\src\wkpu_ip.c

```
Wkpu_Ip_StatusType Wkpu_Ip_Init (uint8 instance, const Wkpu_Ip_IrqConfigType* userConfig)
{…
//#ifdef WKPU_IP_SUPPORT_PULLUP
      /* Set Wakeup/Interrupt Pull-up Enable Register */…
                  Wkpu_Ip_PullUp(base,
                  channelMaskLo,
#ifdef WKPU_IP_64_CH_USED
                  channelMaskHi,
#endif
                  (*userConfig->pChannelsConfig)[index].pullEn);
//#endif
…
//#ifdef WKPU_IP_SUPPORT_PULLUP
static inline void Wkpu_Ip_PullUp(…)
{
   if (enable)
   {
      base->WIPUER_WIPDER |= channelMaskLo; //register error, modified
…
   }
   else
   {
      base->WIPUER_WIPDER &= ~channelMaskLo; //register error, modified
…
   }
}
//#endif
```

6. 增加 STBY GPR 寄存器访问：

如下所示，在进入 standby 之前，需要将由 SIUL 寄存器上拉的管脚切换为由 WKPU 上拉，因为只有 STBY GPR WKPU 寄存器在 Standby 模式下可以保持。

31.5.5.1.1 Standby mode entry flow diagram

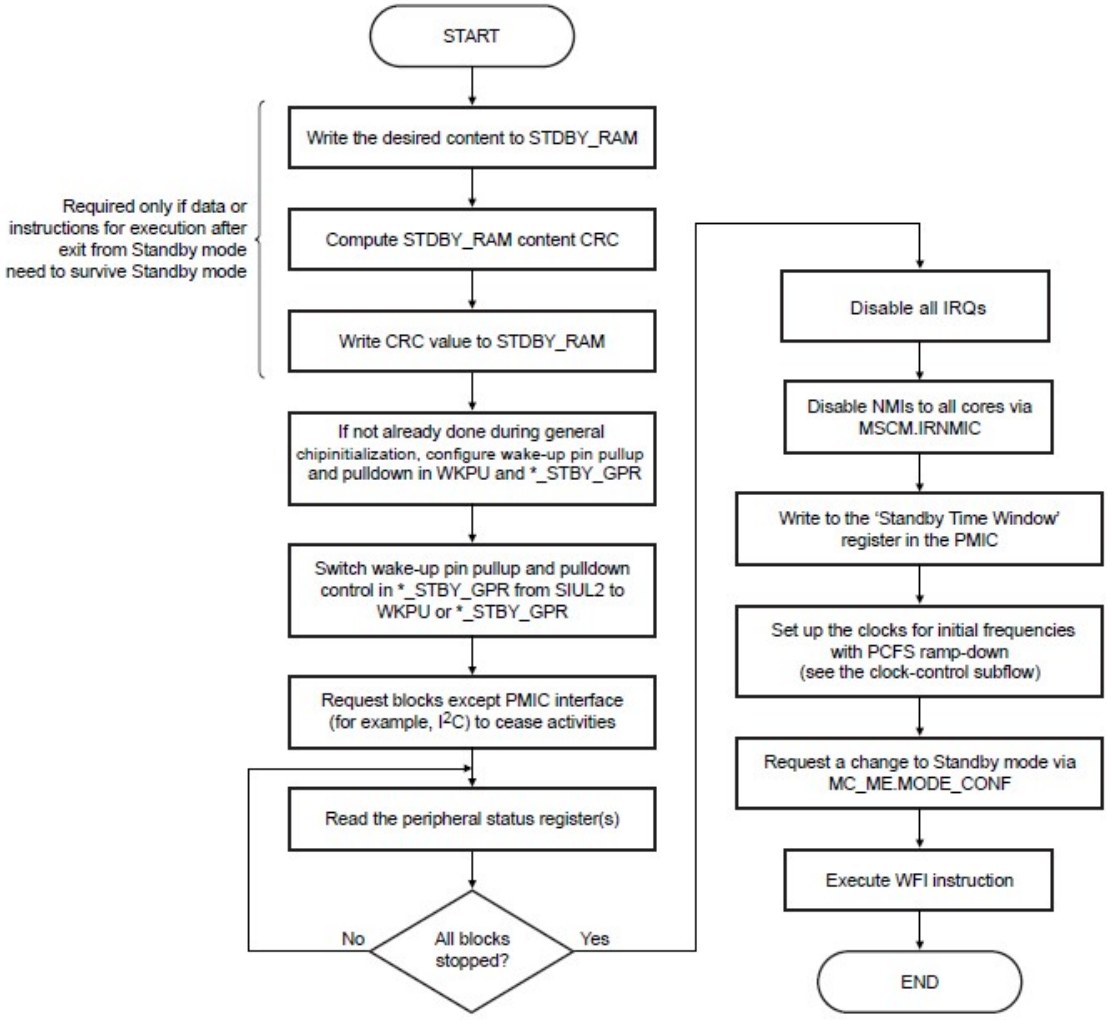


Figure 178. Standby mode entry flow diagram

Device Mode	SIUL2_MSCRn[PUE]	SIUL2_MSCRn[PUS]	WKUP_PUS[WKUP_PU_OVERRIDE]	WKPU_WIPER[IPUE]	WKPU_PUS[WKUP_PUS]	Affect on WKPU PAD
All except STANDBY modes	0	x	0	x	x	Highz
	x	x	1	0	x	Highz
	x	x	1	1	0	Pull down
	x	x	1	1	1	Pull up
	1	0	0	x	x	Pull down
	1	1	0	x	x	Pull up

NOTE

Wake-up pads pull controls must be configured via the WKPU and not by the SIUL during STANDBY. The proper procedure for managing the wake-up pads pulls is described in the "Wake-up pin pullup/pulldown control during Run and Standby modes" section of the Power chapter.

Field	Function
31 WKUP_PU_OVERRIDE	WKUP Pullup Override This field selects one of two methods to control the WKUP pins. If the value is 1, that overrides SIUL2's MSCR[PUE] and MSCR[PUS] fields. NOTE It is observed that the wakeup pads configured pulled up or pulled down from the SIUL2 module retains their pull in standby mode also. This occurs even when the WKUP_PUS[WKIP_PU_OVERRIDE] = 0, which disabled the WKPU to drive the pull on the wakeup pads. 0b - Use SIUL2's MSCR[PUE] and MSCR[PUS] fields 1b - Use WKPU and GPR modules
30-23 —	Reserved
22-0 WKUP_PUSn	WKUP Pad n Pullup Select Selects pullup or pulldown on WKUP pad n if SIUL2's MSCR[PUE]=1 on the pad. 0b - Pulldown 1b - Pullup

icu_ts_*\include\wkpu_ip.h

```
#include "S32G274A_S32G_STDBY_GPR.h"
```

icu_ts_*\src\wkpu_ip.c

```
//#ifdef WKPU_IP_SUPPORT_PULLUP
static inline void Wkpu_Ip_PullUp(…)
```

```
{
  if (enable)
  {
    /*step 5 Handover control to siul until WKUP and STBY_BPR are setup*/
    /* johnli Set WKUP_PUS register, hand over siul pull up to wkup pull up */
    IP_S32G_STDBY_GPR->WKUP_PUS |= S32G_STDBY_GPR_WKUP_PUS_WKUP_PUS4_MASK;// 1b - Pullup
    base->WIPUER_WIPDER |= channelMaskLo; /*step 6 Enable pulls*/
    IP_S32G_STDBY_GPR->WKUP_PUS |= S32G_STDBY_GPR_WKUP_PUS_WKUP_PU_OVERRIDE_MASK; // 1b - Use WKPU and GPR modules
    /*step 7 clear wake-up status*/
…
//#endif
```

7. 修改优化问题(channelMaskLo/Hi 会被优化掉，修改为 volatile 变量)：

icu_ts_*\src\wkpu_ip.c

```
void Wkpu_Ip_SetActivationCondition (uint8 instance, uint8 hwChannel, Wkpu_Ip_EdgeType edge)
{
  WKPU_Type * base;
  volatile uint32 channelMaskLo;
  volatile uint32 channelMaskHi;

  channelMaskLo=0U;
  channelMaskHi=0U;
```

8. 增加 WBMS 寄存器 APIs：

目前整个 ice 模块的 IDE 和源代码，没有提供 WBMS 寄存器的配置和访问代码，所以临时用本地函数实现一下。

icu_ts_*\src\wkpu_ip.c

```
static inline void Wkpu_Ip_WakeupMethod(WKPU_Type * const base,
                                        uint32 channelMaskLo,
#ifdef WKPU_IP_64_CH_USED
                                        uint32 channelMaskHi,
#endif
                                        boolean fullbootenable)
```

```
{
   /* Enable wake-up method*/
   if (fullbootenable)
   {
      base->WBMSR |= channelMaskLo;/*step 9 */
#ifdef WKPU_IP_64_CH_USED
      base->WBMSR_64 |= channelMaskHi;
#endif
   }
   /* Disable wake-up method */
   else
   {
      base->WBMSR &= ~channelMaskLo;
#ifdef WKPU_IP_64_CH_USED
      base->WBMSR_64 &= ~channelMaskHi;
#endif
   }
}
```

然后如下调用：

```
void Wkpu_Ip_EnableInterrupt(uint8 instance, uint8 hwChannel)
{…
   Wkpu_Ip_WakeupRequest(…);

//johnli add
   Wkpu_Ip_WakeupMethod(base,
                        channelMaskLo,
#ifdef WKPU_IP_64_CH_USED
                        channelMaskHi,
#endif
                        TRUE);

//end
```

9. Main 函数调用：

除了调用 Icu_Init 来初始化 filter，Pullup，activation 外，还需要调用 Icu_EnableEdgeDetection 来初始化唤醒源。

6.5 加入 PMIC 驱动

1. 增加 I2C 模块(用于连接外部 PMIC)：

在 Uart_Example_S32G274A_M7 上右击，然后点 Module Configurations，在 Available Modules 中选中 I2C(…)，点击右移箭头加入工程中。

Uart_Example_S32G274A_M7->someid(…)->I2c(…)->I2c：

->General

- Post Build Variant Used*=checked//修改
- Config Variant*= VariantPreCompile//修改
- I2c Development Error Detection =unchecked//修改
- I2c Disable Production Error Reporting*=checked//修改
- I2c Timeout Duration=10000//修改

->I2cChannel:点击+增加一条 I2cChannel_0，点击进入：

->I2cChannel_0：

- I2C Hardware Channel=IIC_4//i2c4 连接 PMIC
- I2c Master/Slave configuration =MASTER_MODE

然后在 Uart_Example_S32G274A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0->McuClockReferencePoint 中，点击+号增加一项，进入：

- Name= I2C_CLK
- Mcu Clock Frequency Select= XBAR_DIV3_CLK
- Mcu Clock Reference Point Frequency= 8000000.0//自动计算

然后回到 I2C 模块：打开 I2cClockRef。

- I2cClockRef= /Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig_0/I2C_CLK
- I2c Asynchronous Method= I2C_USING_INTERRUPTS
- I2c Baud Rate (0 -> 1000000)= 400000.0 //自动计算，相应 timing 配置如下：
- I2c Prescaled Shift (0 -> 2)=0；I2c Prescaler Divider (0 -> 7)=0；I2c Shift Tap Point (0 -> 7) =0；

I2c SCL Divider (cycles) (20 -> 15360)=20；I2c SDA Hold Delay (cycles) (7 -> 2052) =7；I2c Hold Start Delay (cycles) (6 -> 7672) =6；I2c Hold Stop Delay (cycles) (11 -> 7684)=11。

2. 修改 Port 模块增加 I2C 管脚

Uart_Example_S32G274A_M7->someid(…)->Port(…)->Port->PortContainer->PortContainer_0 :

->General :

- PortNumberOfPortPins=//增加两个管脚

-> PortPin：点击+号增加一个管脚，点击进入：

- Name= I2C4_CLK
- PortPin Ode* =checked//
- PortPin Id=4
- PortPin Mscr (dynamic range) =34
- PortPin Direction = PORT_PIN_INOUT
- PortPin Mode= I2C_4_I2C4_SCL_INOUT
- PortPin Level Value = PORT_PIN_LEVEL_NOTCHANGE
- PortPin Output Slew Rate= SRE_3_3V_50MHZ

  同理增加 SDA

- Name= I2C4_SDA
- PortPin Pull Enable /PortPin Pull Select =checked// pull enable，注意需要上拉，不然会导致 I2C 连接失败
- PortPin Ode* =checked
- PortPin Id=4
- PortPin Mscr (dynamic range) =33
- PortPin Direction = PORT_PIN_INOUT
- PortPin Mode= I2C_4_I2C4_SDA_INOUT
- PortPin Level Value = PORT_PIN_LEVEL_LOW
- PortPin Output Slew Rate= SRE_3_3V_50MHZ

3. 增加 DIO 模块(提供 I2C GPIO 模块解拉死功能，用 GPIO 功能模拟解拉死算法)：

在 Uart_Example_S32G274A_M7 上右击，然后点 Module Configurations，在 Available Modules 中选中 Dio(…)，点击右移箭头加入工程中。

Uart_Example_S32G274A_M7->someid(…)->Dio(…)->Dio

->General:

- Dio Development Error Detect；Dio Version Info Api；Dio Flip Channel Api ；Dio Masked Write Port Api； Dio Read Zero For Undefined Port Pins=checked//修改

->DioPort:点击+号增加一项，修改为：

- Name*= DioPort_Pmic

->DioChannel: 点击+号增加一项，修改：

- Dio Channel Id*=2 //因为 I2C4 IOMUX 自 PC_01/02，为第 2 组 GPIO。

4. 增加 PMIC 模块

在 Uart_Example_S32G274A_M7 上右击，然后点 Module Configurations，在 Available Modules 中选中 Pmic(…)，点击右移箭头加入工程中。

Uart_Example_S32G274A_M7->someid(…)->Pmic(…)->Pmic->

->General :

- Post Build Variant Used*=checked//修改
- Config Variant*= VariantPreCompile//修改
- Provide PMIC Watchdog API=checked//修改，需要配置 watchdog
- PMIC Default Error Detection*=unchecked//修改，防止代码本身一些参数检查的 Bug

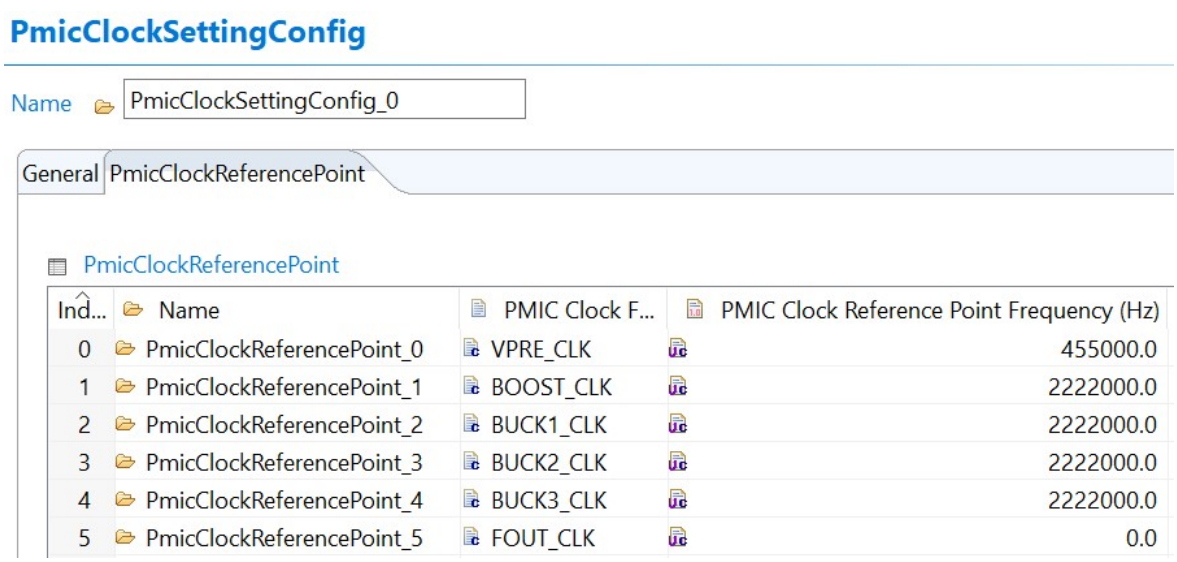
->PmicDevice：点击+号增加 PmicDevice_0，点击进入：

->General :

- I2C Channel Reference*= /I2c/I2c/I2cGlobalConfig/I2cChannel_0
- I2C SCL Pin Reference*= /Port/Port/PortConfigSet/PortContainer_0/I2C4_CLK
- I2C SCL Dio Reference*= /Dio/Dio/DioConfig/DioPort_Pmic/DioChannel_0

除了 PmicAmuxChannel 和 PmicSVSSettingConf 外，其它所有项都点击+号增加默认项。

然后在 PmicClockSettingConf-> PmicClockSettingConfig_0->PmicClockReferencePoint：点击+号增加：



Ind...	Name	PMIC Clock F...	PMIC Clock Reference Point Frequency (Hz)
0	PmicClockReferencePoint_0	VPRE_CLK	455000.0
1	PmicClockReferencePoint_1	BOOST_CLK	2222000.0
2	PmicClockReferencePoint_2	BUCK1_CLK	2222000.0
3	PmicClockReferencePoint_3	BUCK2_CLK	2222000.0
4	PmicClockReferencePoint_4	BUCK3_CLK	2222000.0
5	PmicClockReferencePoint_5	FOUT_CLK	0.0

然后在 PmicClockSettingConfig_0->General->

- Low Power Oscillator Frequency [Hz]= F_600000 //需要调用 Pmic_InitClock 来实现 对配置的初始化。

Table 90. M_CLOCK2 register description

	Description	Low Power Clock frequency selection
LOW_POWER_CLK [1:0]	00	100kHz
	01	100kHz
	10	300kHz
	11	600kHz
	Reset condition	POR

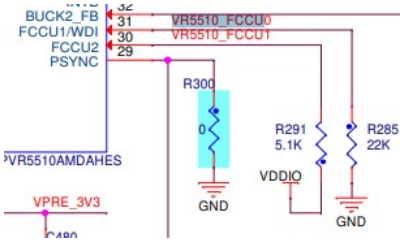
然后在 PmicFailSafeConfigure 中：

->PmicWatchdogConfiguration:

- Watchdog API Enable*=checked //bug fix
- Watchdog Window Period Enable=unchecked // 此处关闭 watchdog 周期喂狗，即使在 Normal 模式下不喂狗，也不会导致重启。

->PmicSafeInputsConfiguration：

- FCCU Monitoring Configuration= DISABLED//修改，如下原理图设计：



所以默认硬件配置是处于 FCCU1=low，FCCU2=high 的故障态，需要 S32G 配置完 FCCU 后反向拉动退出故障态，要不然在第一次喂狗后，PMIC 从 INIT_FS 进入到 NORMAL_FS 后会导致重启，所以我们将 FCCU Monitoring 功能关闭，防止因为喂狗导致的重启。

- Safety Standby Window Duration=TIME_8MS //用于配置从 i2c 写 stdby request 到 S32G 拉 standby 管脚的时间窗口，此处设置的比较大，注意要保证这两个操作时间要小于 8ms，所以之间不能加串口 debug 之类的大延时操作，或是使用调试器。

然后在 PmicModeSettingConf 中点击+号再增加一个配置，进入，配置为 standby 模式：

- Name*=PmicModeSettingConf_1_Standby

  ->General:
- Operating Mode*=STANDBY
- Standby Timer Enable=checked//打开
- Standby Timer Window Duration= TIME_8388608MS//设置为最大，此为 Standby 状态的最大保持时间，所以设置为最大。

  ->PmicRegulatorsConfig:
- HVLDO Standby Mode Enable*=checked //HVLDO in Standby Mode is Enabled
- BUCK3 Standby Mode Enable*=checked //enable the DDR 1v1 for DDR selfrefresh
- LDO2 Standby Mode Enable*=checked //enable the DDR 1v1 for DDR selfrefresh

5. 生成代码：

```
static const Pmic_VR55XX_ModeConfigType Pmic_VR55XX_ModeConfigPB_0_1_VS_0 =
{…
  /* The selected target mode.*/
  PMIC_STANDBY_MODE, …
  /* The M_REG_CTRL3 register configuration. */
  ((uint16)0x0000U)
  | PMIC_VR55XX_M_REG_CTRL3_BUCK3_STBY_MASK16
  | PMIC_VR55XX_M_REG_CTRL3_VPREV_STBY_MASK16
```

```
   | PMIC_VR55XX_M_REG_CTRL3_HVLDO_STBY_MASK16
   | PMIC_VR55XX_M_REG_CTRL3_LDO2_STBY_MASK16…
Pmic_VR55XX_SetMode
|->
      case PMIC_STANDBY_MODE:
      {
          eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_MAIN_UNIT,
PMIC_VR55XX_M_SM_CTRL1_ADDR8, pModeConfig->u16MainStateMachineReg);
          eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_MAIN_UNIT,
PMIC_VR55XX_M_REG_CTRL3_ADDR8, pModeConfig->u16MainControlReg3 );

          if (PMIC_VR55XX_CFG_2_OTP_STBY_SAFE_DIS_OTP_MASK8 !=
( PMIC_VR55XX_CFG_2_OTP_STBY_SAFE_DIS_OTP_MASK8 &
(*pOtpRegisterConfig->paOtpFailSafeConfig)[PMIC_VR55XX_FS_GET_OTP_REG_INDEX_U8(PMIC_VR55XX_
FS_CFG_2_OTP_ADDR8)].u8RegisterData))
          {
              eInternalStatus |= Pmic_VR55XX_I2C_ReadRegister(DeviceId, PMIC_FAIL_SAFE_UNIT,
PMIC_VR55XX_FS_SAFE_IOS_ADDR8, &u16RegValue );
              eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_FAIL_SAFE_UNIT,
PMIC_VR55XX_FS_SAFE_IOS_ADDR8,(uint16)((uint16)(u16RegValue &
PMIC_VR55XX_FS_SAFE_IOS_W_BITS_MASK16) | PMIC_VR55XX_FS_SAFE_IOS_STBY_REQ_MASK16));
          }
```

6. 修改 setmode 函数(按照 PMIC 进入 standby 要求，重写 setmode 函数)：

   Pmic_ts_*/pmic_vr55xx.c

```
Pmic_ReturnType Pmic_VR55XX_SetMode(const Pmic_DeviceIndexType DeviceId, const
Pmic_ModeIndexType ModeSettingID)
{…
      case PMIC_STANDBY_MODE:
      {
//step 1 /*****Read the FSM_STATE field in FS_STATES REG for debug purpose
      * 00110 -- init_fs
      * 00111 -- wait_ABIST2
      * 01000 -- ABIST2
      * 01001 -- ASSERT_FS0B
      * 01010 -- NORMAL_FS**************/
```

```
eInternalStatus |= Pmic_VR55XX_I2C_ReadRegister(DeviceId, PMIC_FAIL_SAFE_UNIT,
PMIC_VR55XX_FS_STATES_ADDR8, &u16RegValue);

if ((PMIC_VR55XX_FS_STATES_FSM_STATES_NORMAL_FS_U16 == (u16RegValue &
PMIC_VR55XX_FS_STATES_FSM_STATES_MASK16))||(PMIC_VR55XX_FS_STATES_FSM_STATES_IN
IT_FS_U16 == (u16RegValue & PMIC_VR55XX_FS_STATES_FSM_STATES_MASK16))) //just in normal_fs
or init_fs can enter standby

 {

//step 2/*****write the DBG_EXIT field in FS_STATES REG  request PMIC exit debug status.**********/

    eInternalStatus |= Pmic_VR55XX_I2C_ReadRegister(DeviceId, PMIC_FAIL_SAFE_UNIT,
PMIC_VR55XX_FS_STATES_ADDR8, &u16RegValue);

   if (PMIC_VR55XX_FS_STATES_DBG_MODE_MASK16 == (u16RegValue &
PMIC_VR55XX_FS_STATES_DBG_MODE_MASK16)) //if in debug mode, exit debug mode

 {

    u16RegValue |=PMIC_VR55XX_FS_STATES_DBG_EXIT_MASK16;

    eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_FAIL_SAFE_UNIT,
PMIC_VR55XX_FS_STATES_ADDR8, u16RegValue);

 }


//step 3 /*In the Main Memory map (0x20), enable HVLDO and PREV by setting these bits in the M_REG_CTRL3
register (0x05)  * configure low power mode VPRE voltage, w0 to set as 3.3V (w1 set as 3V) */

//still add buck3 1.1v for vdd_io_ddr0 and lpddr4, and LDO2 for lpddr4

// PMIC_Write(0x05, 0x100);

//PMIC_Write(0x05, 0x1110);


eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_MAIN_UNIT,
PMIC_VR55XX_M_REG_CTRL3_ADDR8, pModeConfig->u16MainControlReg3 );


//step4/* Configure time window 1111b          equals 8ms, PMIC wait time window for STBY_S32G_PIN pull
down */

// configure TIMING_WINDOW_STBY[3:0] in FS_I_SAFE_INPUTS register

//already configured in function Pmic_VR55xx_ConfigureFailSafe

//step 5

/* Configure time window 1111b equals 8388608ms, STBY_TIMER_EN=1 ; PMIC MAIN logic window - move to
deep fail safe during STBY  */

// configure TIMER_STBY_WINDOW[3:0] in M_SM_CTRL1 register

eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_MAIN_UNIT,
PMIC_VR55XX_M_SM_CTRL1_ADDR8, pModeConfig->u16MainStateMachineReg);
```

```
//step 6 disable wdg: /*config WDW_period [3:0] = 0, disable WD*/->/*Feed watchdog once in order to disable it*/->          /*Feed watchdog 6 times  in order to clear flt cnt */
eInternalStatus |=Pmic_VR55XX_DisableWatchdog(DeviceId);
//step 7
/*Read the status again. the status must be in NORMAL_FS status right before
     * standby entry */

eInternalStatus |= Pmic_VR55XX_I2C_ReadRegister(DeviceId, PMIC_FAIL_SAFE_UNIT, PMIC_VR55XX_FS_STATES_ADDR8, &u16RegValue);
if (PMIC_VR55XX_FS_STATES_FSM_STATES_NORMAL_FS_U16 == (u16RegValue & PMIC_VR55XX_FS_STATES_FSM_STATES_MASK16)) //just in normal_fs or init_fs can enter standby
 {
    //step 8
       /* Configure low power clock frequency for VPRE in M_CLOCK2 register /0b11 600Mhz _ 20mA capacity
    * need to be wrote 40us before PMIC move to STBY*/
    //need call in main Pmic_InitClock(const Pmic_DeviceIndexType DeviceId, const Pmic_ClockSettingIndexType ClockSettingId);
   //step 9
   ///*PMIC standby entry request from MPU*/ PMIC_Write_FS(0x15, 0x02);

    eInternalStatus |= Pmic_VR55XX_I2C_WriteRegister(DeviceId, PMIC_FAIL_SAFE_UNIT, PMIC_VR55XX_FS_SAFE_IOS_ADDR8,(uint16)((uint16)(u16RegValue & PMIC_VR55XX_FS_SAFE_IOS_W_BITS_MASK16) | PMIC_VR55XX_FS_SAFE_IOS_STBY_REQ_MASK16));
 }
else
{
  return eInternalStatus;
}
}
else
{
  return eInternalStatus;
}
```

7. Main 函数调用：

根据以上分析：Main 函数除开使用 PMIC_Init 和 PMIC_InitDevice 来做初始化为，需要调用 Pmic_InitClock 来设置 Standby 模式下的时钟，以及调用 PMIC_SetMode 来进入 Standby 模式，这个函数调用到 S32G 拉动 Standby 管脚的时间不能超过 8ms。

6.6主函数逻辑实现

修改主要包括：

- 实现新加入模块的初始化和包括进头文件。
- 将原异步串口收发模式改为同步接口。
- 根据串口软件来切换 PMIC 模式。
- 关闭 CLOCK。
- 切换为 STANDBY 模式。

Uart_example_s32g274a_m7/src/main.c：

```
…
#if 1
#include "Mcal.h"
#include "CDD_Pmic.h"
#include "CDD_I2c.h"
#include "Dio.h"
#include "Osif.h"
#include "Icu.h"
#endif
…
#if 1
#define STBY_MSG "Stdby demo: Full Boot, pull LLCE CAN0 RX to low to  resume\r\n"
extern ISR(WKPU_EXT_IRQ_SINGLE_ISR);
#endif
…
int main(void)
{
  volatile Uart_StatusType Uart_Status;
  volatile Std_ReturnType Std_Uart_Status;

  Std_Uart_Status = E_NOT_OK;
```

```
  /* Initialize the Mcu driver */
  Mcu_Init(NULL_PTR);

  /* Initialize the clock tree and apply PLL as system clock */
  Mcu_InitClock(McuClockSettingConfig_0);

  /* Apply a mode configuration */
  Mcu_SetMode(McuModeSettingConf_0);

  /* Initialize all pins using the Port driver */
  Port_Init(NULL_PTR);

  /* Initialize IRQs */
//  Platform_Init(NULL_PTR);
//  Platform_InstallIrqHandler(LINFLEXD1_IRQn, LINFLEXD1_UART_IRQHandler, NULL_PTR);

    /* Initialize I2c driver */
            I2c_Init(NULL_PTR);
            /* Initialize Pmic driver */
            Pmic_Init(NULL_PTR);
            /* Install Gpio ISR */

              /* Initialize Vr5510 device */
     Pmic_InitDevice(PmicConf_PmicDevice_PmicDevice_0);

            Platform_InstallIrqHandler(WKPU_GRP_IRQn, & WKPU_EXT_IRQ_SINGLE_ISR, NULL_PTR);
            Platform_SetIrq(WKPU_GRP_IRQn, TRUE);

    /* Initialize the Icu driver */
            Icu_Init(&Icu_Config_VS_0);
             Icu_EnableEdgeDetection(IcuChannel_0);

  /* Initializes an UART driver*/
  Uart_Init(&Uart_xConfig_VS_0);
```

```
(void)Uart_SyncSend(UART_CHANNEL, (const uint8 *)STBY_MSG, strlen(STBY_MSG), 10000);
   /* Configure low power clock frequency for VPRE in M_CLOCK2 register /0b11 600Mhz _ 20mA capacity
    * need to be wrote 40us before PMIC move to STBY*/
    Pmic_InitClock(PmicConf_PmicDevice_PmicDevice_0,0);
   /*set pmic to standby mode*/
     Pmic_SetMode(PmicConf_PmicDevice_PmicDevice_0,1);
   /*reset clock to initial*/
     Mcu_ResetClockConfiguration(McuClockSettingConfig_0);
     /*set s32g to standby mode*/
     Mcu_SetMode(McuModeSettingConf_1);

   Uart_Deinit();
   Exit_Example((Uart_Status == UART_STATUS_NO_ERROR) && (Std_Uart_Status == E_OK));
   return (0U);
}
```

根据文档《S32G_RTD_MCAL_V*.pdf》说明，修改编译相关文件，注意因为增加了相关模块，所以如下文件也需要修改：

C:\Work\soureinsight\S32G\s32g_mcal_3.0.2\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\project_parameters.mk

```
MCAL_MODULE_LIST := Base Det Platform Mcu Uart Os Port Resource EcuC Rte I2c Icu Pmic Dio EcuM
```

然后执行 make build 编译出 main.elf 文件。

6.7 运行测试

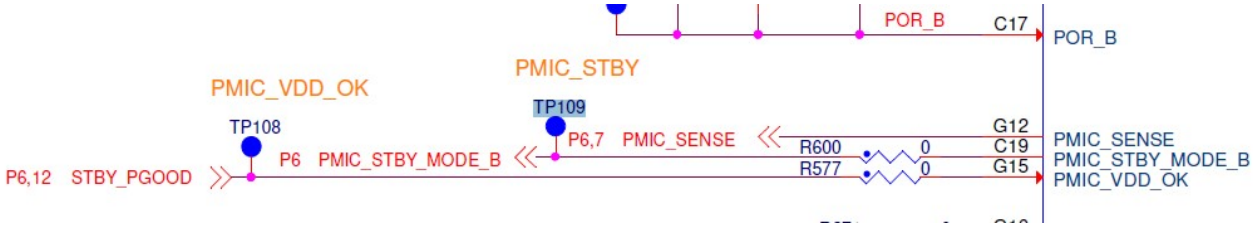
根据文档《S32G_RTD_MCAL_V*.pdf》说明，将编译出来的 main.elf 转成 main.bin 文件，然后再用 S32DS 的 IVT 工具打上 IVT 头，注意要打入 QSPI　NOR 的 timing 头。然后使用 S32DS 的 Flash tools 将镜像在下载模式下通过 UART0 烧写到 QSPI NOR 中，然后切换回正常启动模式，串口修改为 UART1，启动：

1. 串口打印出：

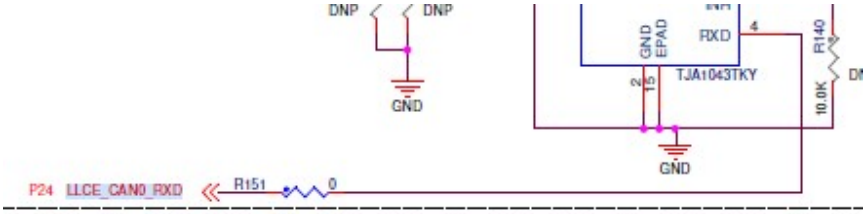
   ```
   Stdby demo: Full Boot, pull LLCE CAN0 RX to low to  resume
   ```

   然后进入 Standby 模式。

2. 量测 S32G Standby 管脚 TP109 和 PMIC 的 Stby_pgood 管脚，均为低，说明 S32G 输出给 PMIC 的 Standby 信号和 PMIC 返回的 standby good 信号均为正常：



3. 其它电源灯关闭，说明相关电源关闭。
4. 如下 LLCE_CAN0_RX 测试点：



所以将 R151 对地轻触，系统就会 resume full boot 重新运行，串口再次打印：

```
Stdby demo: Full Boot, pull LLCE CAN0 RX to low to  resume
```

6.8 未来开发计划

- 本章仅说明 full boot 的方式，SSRAM short standby 的方式由于受限导致应用场景少，所以不再说明。
- 本章仅说明 GPIO resume 的方式，RTC resume 暂时没有说明，客户可以参考自行开发。