S32G M7 Standby Demo

by John Li

This article explains the details and customization of the S32G M7 core Standby demo. And how to porting to Autosar Mcal demo.

Please note that this article is a training and auxiliary document. This article is not a substitute for the official document. Please refer to the official document.

History	Comments	Author
V1	● Creae the doc	John.Li
V2	● Add the customization chapter 5	John.Li
V3	● Modify to Mcal sample chapter 6	John.Li
V4	● Translate to Eng.	John.Li

Contents

1 Description of reference materials

Classfication	Name	Illustration	Comments
Doc	AN12880-5510-standby.pdf	VR5510 standby application doc	Search on www.nxp.com
Doc	AN12952-Power Saving Techniques.pdf	S32G2 suspendapplication doc	Search on www.nxp.com (Important!)
Doc	S32g274A_standby_demo_UG.pdf	Standby demo user guide	Check with your window FAE
Code	S32g274AstandbyRAMboot	Standy RAM short boot code	
Code	S32G274Astandbymode	Standy demo full boot code	
Code	Mcal RTD 3.0.2	Autosar Mcal	Download from www.nxp.com
Doc	S32G2RM.pdf	Chipset reference manual	Download from www.nxp.com
Doc	ds649820 - VR5510 Datasheet Rev2 (2.0).pdf	Chipset reference manual	Download from www.nxp.com
Doc	S32G_RTD_MCAL_V*.pdf	Mcal customization application doc(Chinese version)	Download from https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-MCAL-customization-application-doc/ta-p/1399899

2 Demo creation and running process

2.1 Demo checkpoints

Refer to the document <<S32g274A_standby_demo_UG.pdf>>, import the project, compile and run. Notice:

- Import project into S32DS： Open S32DS3.4.3+RTD3.0.2, File->Import->Existing Projects into Workspace/Next->Brows to "….\S32G274Astandbymode" /Finish.
- Build: In S32DS IDE Project Explorer window, choose S32G274Astandbymode:Debug right click and ->Build Project.

- Creating IVT images: The *.bin selected on this page is wrong, it should be S32G274Astandbymode.bin, and its address is RAM start pointer=RAM entry porter=0x34400000.
- When create the IVT image, the DCD segment selection SRAM initialization DCD script file can be found in C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_DCD_InitSRAM.bin. which is bootloader sample project.
- When create the IVT image, note that the size in bytes of the application bootloader will be automatically calculated, but if it is not an integer of 8 bytes, an error will be reported, so it needs to be manually increased to an integer multiple of 8 bytes.
- When create the IVT image, The original image does not pack the qspi nor timing header file, because the bin file itself is relatively small, so using the default low-speed 1 bit mode to access qspi nor will not cause watchdog timeout failure, if you want to increase it, you can: In interface selection, Boot device type=QuadSPI Serial Flash, checked Configure QuadSPI Serial parameters and then in QuadSPI parameters to choose: 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.
- When burning the image, Algorithm should select MX25UW51245G

The test results are as follows:

```
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

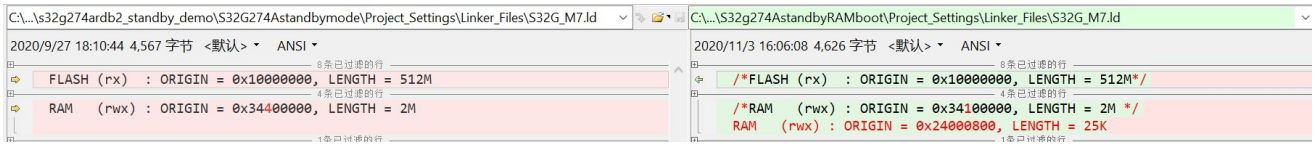
Set the Boot model

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

2.2 The difference between Standby and StandbyRAMboot

2.2.1 Link file

…\Project_Settings\Linker_Files\S32G_M7.ld



So Standby demo's link address in the SRAM:

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

But StandbyRAMboot's link address in Standby RAM:

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

2.2.2 Main function

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	Configure CAN0_RX and RTC as wakeup sources	Only configure RTC as wakeup source	

3 S32G Standby principle and Code Description

3.1 Peripheral initialization function

- s32g_dfs_reset // clock dfs reset
- s32g_clock_tree_init// clock initialization ,Pay attention to its clock initialization for QSPI NOR/uSDHC/UART
- s32g_linflexd_init // linflexd initialization 115200
- Init_standbyram // standby_rom initialization
- qspi_init // init Nor flash, Note that the AHB Buffer mode is initialized here.
- s32g_i2c4_init // i2c4 initialization, use to link PMIC

3.2 standbyramc_cpy(optional)

This function is only useful for Standby RAM fast boot.

Exiting standby mode triggers the execution of BootROM. Depending on the wakeup source, BootROM boots from Standby RAM or performs a full boot.

Standby RAM fast boot has the same format as normal IVT (see Image Vector Table (IVT)), but its position is fixed when booting from standby

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

- DCD operation (from primary and backup)
- Application bootstrapping (from primary and backup)

Fastboot does not support self-test or HSE_H firmware. It only supports (IVT Boot Configuration Word field) boot_SEQ=0, which means to perform non-secure boot. If BOOT_SEQ=1, BootROM will issue a reset. BootROM does not support extracting data from QuadSPI or SD/MMC/eMMC cards when booting from Fastboot. Before entering standby mode, the application should program:

- IVT (24000000 hours)
- DCD (if required)
- Applications on Standby RAM whose contents remain valid in standby mode

All pointers in the IVT must point to addresses in backup RAM. BootROM does not perform any authentication on fast, boot, DCD or applications when booting via fastboot. When a full boot is requested, the BootROM performs normal boot operations, as described in this chapter, after any resets. BootROM (Boot Read-Only Memory) does not check Fastboot when Standby RAM boots. For more details on wakeup sources, refer to the Wakeup Unit (WKPU) chapter. The decision to perform a full boot or fast boot from Standby RAM depends on the wakeup source configuration that triggers the wakeup. If the wakeup source is configured to perform a full boot, BootROM will perform a full boot. Otherwise, it will boot quickly from Standby RAM.

Codes:

- IVT header array:

```
/* 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, //address 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,…
};
```

- Application header array:

```
            /* 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,
```

- Important constants:

```
#define Standby_RAM_ADDRESS 0x24000000
#define Standby_RAM_APP_HEAD_ADDRESS 0x24000400
#define Standby_RAM_APP_ADDRESS 0x24000800
#define COPY_FLASH_ADDRESS 0x8000 // This is the same as the offset address burned in QSPI NOR by standbyramboot binary.
```

- Source codes:

```
    /* 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

Refer the doc <<AN12952-Power Saving Techniques.pdf>>, Table 6. Recommended configuration/setup sequence.

Input parameters:

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 as follows:

Table 61. WKUP expose pads

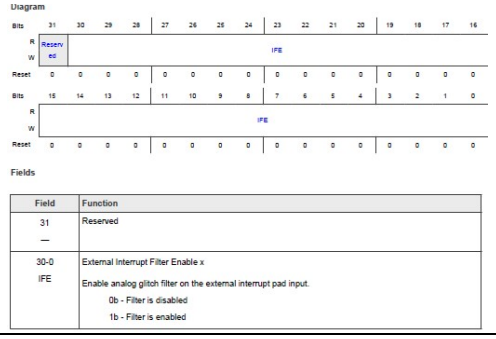
Number of	Value
NMI sources	1
Internal interrupt sources	1
External interrupt sources	23
Glitch filters for external interrupts	23
External interrupt vectors	1

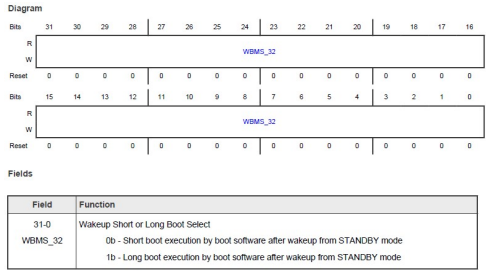
NOTE

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.

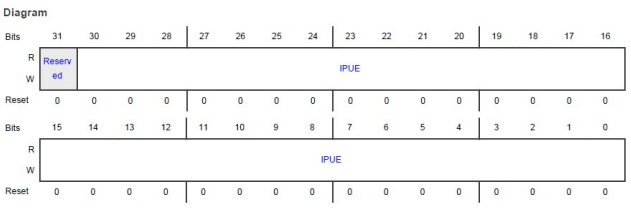
Refer to the document <<S32G2_IOMUX.xlsx>>, search WKUP to find all GPIOs with WKUP function, as follows:

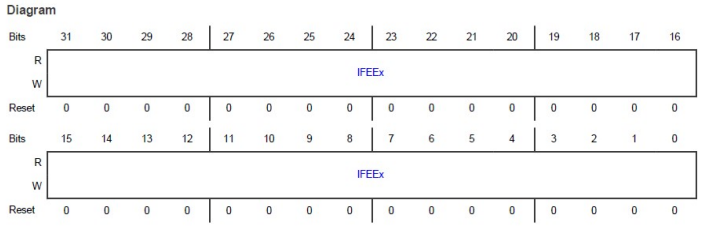
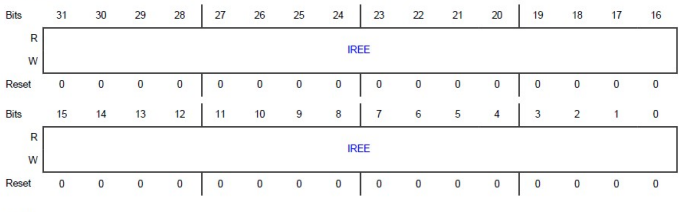
PC_11	43	SIUL2_0	0000_0000	0x4009C2EC	GPI[43]
PC_11	513	SIUL2_0	0000_0010	0x4009CA44	CAN0_RX
PC_11	745	SIUL2_1	0000_0010	0x44010DE4	LLCE_CAN0
PC_11	-		-		WKUP0

Step	Codes	Comments	Registers Settings
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：if it is full boot, set the RTC and CAN0_RX GPIO as follow boot, the other are std ram boot	

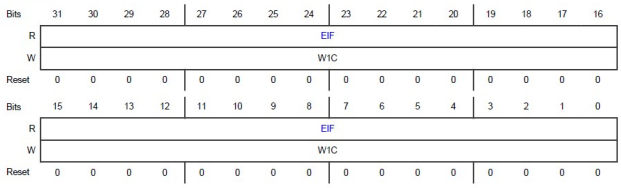
	}		
2	WKPU_IRER = 0x0; WKPU_WRER = 0x0;	Prepare Pad Pull activation Zero both registers. (Reason is to avoid capturing unintended events before WIPUER_WIPDER is written; see Step 5)	
3	SIUL2_0.MSCR[43] .B.PUE = 1; SIUL2_0.MSCR[43] .B.PUS = 1; SIUL2_0.MSCR[43]. B.IBE = 1;	The relevant pin configuration of the pin configuration register SIUL2 should match the configuration of WKPU and STBY_GPR: For the pins that want to be configured as wake-up sources, configure as input IBE=1, pull PUE=, and pull PUS=1. Other pins that are not used as wake-up sources are 0. Need to ensure: SIUL pull enable MSCRn[PUE] = respective configured WKPU. WIPUER_WIPDER. IPUE[m] SIUL pull direction MSCRn[PUS] = STDBY_GPR of respective configuration. 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 configure to falling edge trigger.
	WKPU_WIREER = 0x80000000;//RTC configure to rising edge trigger.
		Configure the edge trigger of the wake-up event, pay attention
		• To configure a trigger edge for all wakeup events.
		• Make sure that the selected trigger edge is not the same as the pin pull direction.
		(Background: To avoid hardware internal pull enable being inadvertently caught as a wakeup event.)
		Avoid using both rising and falling edge triggers at the same time.
		Also avoid turning off both edge touches for wakeup events at the same time.
		Note: Writing '0' to IREE[x] and IFEE[x' will disable the external interrupt function of that pin (i.e. any following activity will not generate
			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

		system wake-up or interrupt). // falling down is configured as a falling edge trigger, because the pin is configured as a pull-up input.	

8	WKPU_IRER = 0; //optionally clear interrupt
	WKPU_WRER = 0x80000001; // LLCE CAN 0=0x1, RTC=0x80000000
		Configure (enable) input buffer enable Tip:
		To determine the effective input buffer enable of the pin, the following logic equation can be used: Effective input buffer enable = SIUL.MSCRn. IBE or WKPU IRER.EIRE[m] or WKPU.WRER.WRE[m].
		1. Step 8.1 WKPU.IRER configure and/or clear WKPU.IRER
		NOTE: A new interrupt should be avoided before entering standby mode. However, it may be desirable for the user to capture interrupted application requests in standby mode for later use. In standby mode but no interrupt will occur and no wakeup will be triggered. WKPU.IRER cleared for all other sources.
		2. Step 8.2 WKPU.WRER Configure WKPUs. WRER is input buffer enable from WKPU side.
			Diagram
			Bits 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
			R / W EIRE
			Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
			Bits 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
			R / W EIRE
			Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
			Fields
			Field: 31-0 EIRE — Function: 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 / W WRE
			Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
			Bits 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
			R / W WRE
			Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
			Fields
			Field: 31-0 WRE — Function: 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

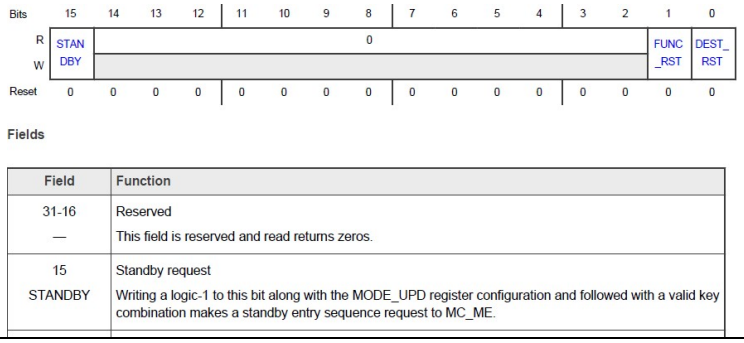
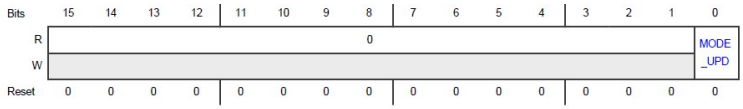
9	WKPU_WISR = 0xffffffff; // Write 1 to clear the wake-up status register	clear wake state Clear register status. (Note, the status flag is W1C)	Diagram  Fields Field: 31-0 / EIF Function: External Wakeup/Interrupt Status Flag x This flag can be cleared only by writing 1. Writing 0 has no effect. If 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

Refer to the document <<AN12952-Power Saving Techniques.pdf>>, 4.3.2.2 Main core shutdown and standby entry. The sequence can only be started after the SoC is ready to enter standby, and only if other cores/core clusters are in standby WFI

1. Make sure the WFI status of all cores/core clusters is still correct.
2. Program the WKUP_PU_OVERRIDE of the wake-up module to 1.
3. Disable the main core IRQ.
4. Disable NMI for all cores using MSCM. IRNMIC register.
5. Program a valid partition index and master ID in MC_ME for the last running master. (see MC_ME MAIN_COREID register)
6. Complete unfinished visits.
7. Program MC_ME for alternate mode transition.
8. Complete unfinished visits.
9. Execute the WFI command. The device will enter standby mode unless the transition is preempted. code as below:

Step	Code	Comments	Regsiters settings
0	MC_ME_MAIN_COREID = 0x0;	Main CoreID configure to M7_0	Register: MAIN_COREID Offset: 10h Function This register provides the ID of the main core sequencing the operation for the standby sequence. Core ID is required for entering in the standby mode, and using this MC_ME locates the WFI instruction execution of the main core. The core ID in this register is specified by the partition index along with the core index.

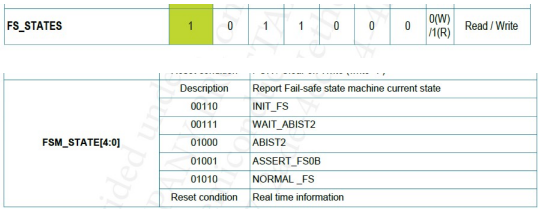
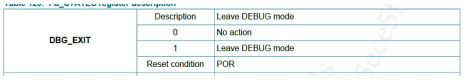
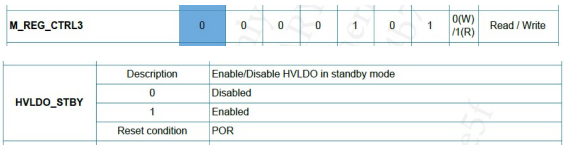
1	MC_ME_MODE_CONF = (0x1<<15);	Configured to enter standby mode, you need to write such as valid key to take effect	 Fields Field / Function 31-16 — Reserved. This field is reserved and read returns zeros. 15 STANDBY — Standby request. Writing a logic-1 to this bit along with the MODE_UPD register configuration and followed with a valid key combination makes a standby entry sequence request to MC_ME.
2	MC_ME_MODE_UPD = 1;	Configure the register to enable mode switching	 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;	Write the fixed valid key twice to trigger switching.	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.

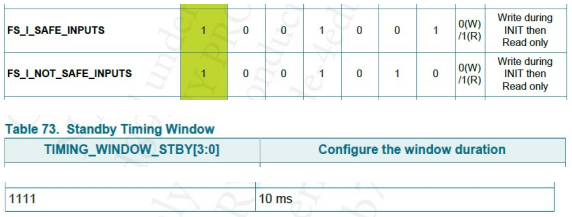
At last, execute WFI:

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

4 VR5510 PMIC Standby principle and code description

4.1 PMIC_initConfig

Step	Codes	Comments	Registers settings
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.**********/	
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 and LDOs must be disabled in standby mode, except HVLDO and VPRE. Depending on the customer's power tree, other regulators may also be enabled. For example, if the DDR is self-refresh, then LDO2 and BUCK3 should also be enabled. If the customer enables this option using the VPREV_STBY_EN_OTP bit, the output VPRE voltage can be configured as 3.3 V or 3 V	

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] | Configure the window duration
			1111 | 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 */ TIMER_STBY_WINDOW[3:0] Customers can enable or disable the standby timer through the I2C bus. This timer prevents the device from going into standby mode. Because, when the timer expires, both devices, VR5510 and S32G, are reset. The timer duration can be configured as TIMER_STBY_WINDOW[3:0]	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
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…/…1: Depend on WD_SEED value and calculation; 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	Codes	Comments	Registers settings
0	PMIC_Write(0x9, 0x3);	/* 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*/	Table 90. M_CLOCK2 register description LOW_POWER_CLK [1:0]: Description: Low Power Clock frequency selection 00: 100kHz 01: 100kHz 10: 300kHz 11: 600kHz Reset condition: POR
1	PMIC_Write_FS(0x15, 0x02);	/*PMIC standby entry request from MPU*/	FS_SAFE_IOS: 1 0 1 0 1 0 1 0(W) /1(R) Read / Write

			RSTB_REQ | Description | Request assertion of RSTB (Pulse) VR5510 All information provided in this document is subject to legal disclaimers. Product data sheet Rev. 2 — 22 December 2020 NXP Semiconductors Multi-Output 0 | No Assertion 1 | RSTB Assertion (Pulse) Reset condition | POR

5 Customization modification

5.1 Do not enable RTC wakeup feaure

If you only consider using the RX GPIO function of CAN to wake up, you don't need to use the RTC function, modify the test as follows:

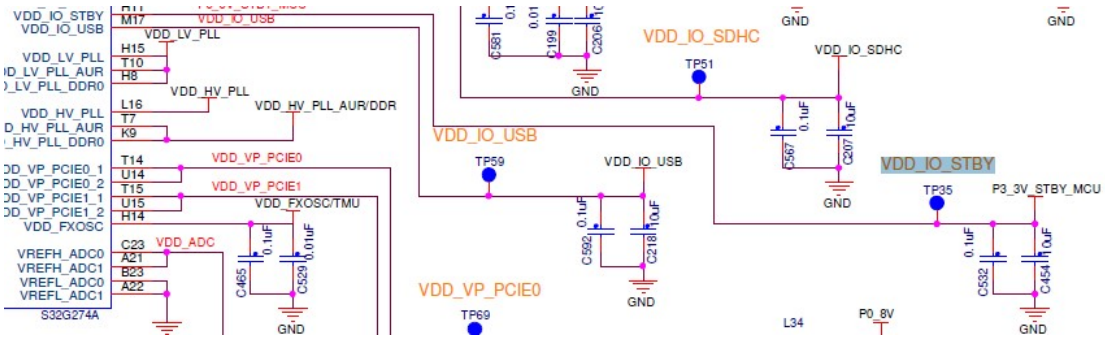
S32G274Astandbymode\src\ S32G_Standby.c

```
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
```

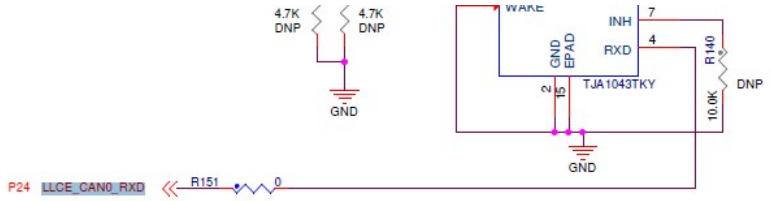
WKPU_WRER = 0x00000001; // LLCE CAN 0

Test:

LLCE_CAN0_RX powered by VDD_IO_STBY=3.3V as follows:



LLCE_CAN0_RX test point is:



Use full boot to test: R151 is pulled up to 3.3V, so you only need to touch R151 to the ground to wake it up, as follows:

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 Eable CAN1_RX wakeup feature

This section explains how to support more wake-up sources. For example, to increase the GPIO wake-up of LLCE_CAN1_RX, it is added on the basis of the following modifications:

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

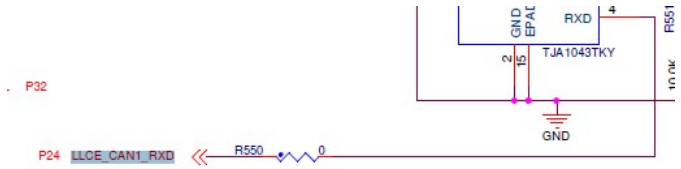
unsigned char S32G_WKUP_SOURCE_NO = 1; // Wake-up source number, from 0-22, 31; 1 means WKUP1=CAN1_RX, CAN0_RX already set in function WKPU_set

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
```

Test:

LLCE_CAN1_RX test point is:



Use full boot to test: so touch the R550 to the ground, the result is:

```
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 Only support full boot

In general, full boot is often used, so you can remove some codes as follows without considering standby ram fast boot (note that standby ram fast boot itself will use standby ram, so it is the same as the standby ram used in ATF, There is a conflict and memory mapping needs to be deal with, but because we only consider the case of full boot, this article does not introduce it).

S32G274Astandbymode\src\main.c

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

S32G274Astandbymode\src\ S32G_Standby.c

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

Full boot does not need to initialize standby ram, nor does it need to initialize qspi nor to copy the standby ivt header. The standby ram copy can also be removed, and some logic control codes can also be removed. The test method is the same as the previous two sections.

5.4 Open the DDR related power

Supporting DDR self-refresh requires powering LPDDR4 and VDD_IO_DDR0, which speeds up the power-on sequence because the content has been saved in the DDR memory before. In DDR refresh mode, the power supplies for VR5510 are VPRE, HVLDO, LDO2 and BUCK3. For related power consumption data, please refer to the VR5510 application note: <<AN12880-5510-standby.pdf>> and the LPDDR4 data sheet.

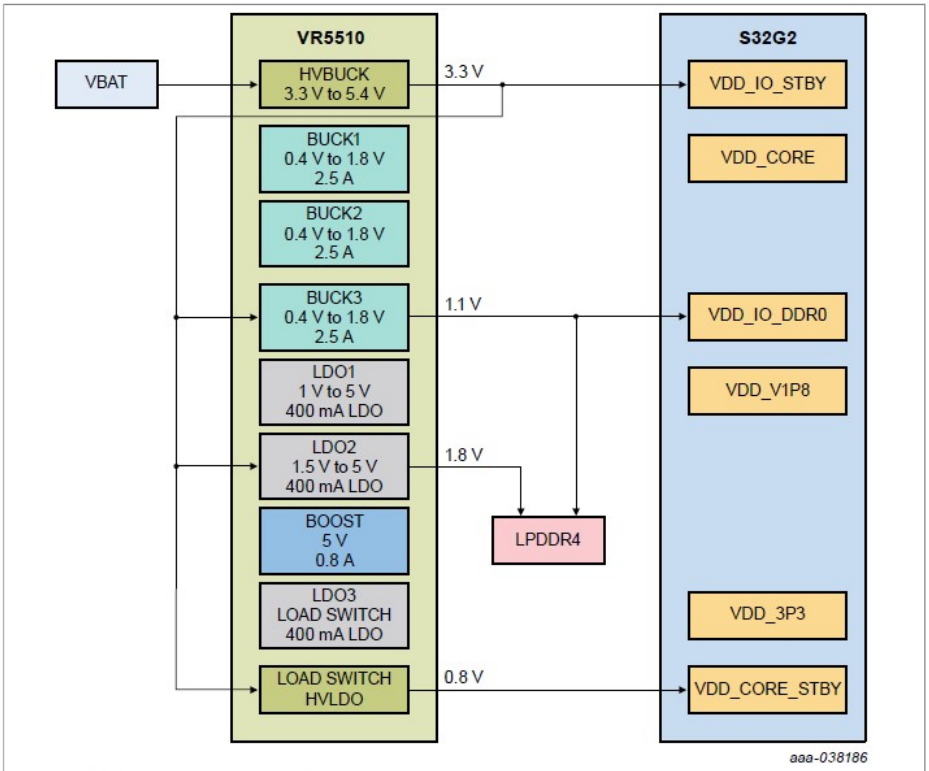


Figure 8. Standby mode DDR self-refresh power tree

Modify the code as follows:

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

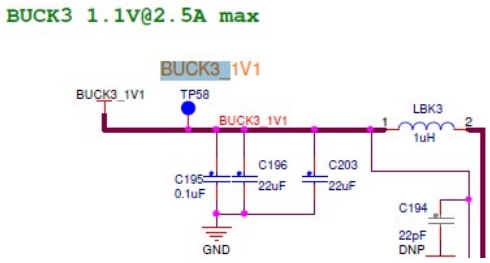
BIT15	BIT14	BIT13	BIT12	BIT11	BIT10	BIT9	BIT8
0	**VPREV_ STBY**	0	**BUCK3_ STBY**	0	**BUCK2_ STBY**	0	**BUCK1_ STBY**
RESERVED	**VPREV_ STBY**	RESERVED	**BUCK3_ STBY**	RESERVED	**BUCK2_ STBY**	RESERVED	**BUCK1_ STBY**
0	1	0	1	0	1	0	1

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)

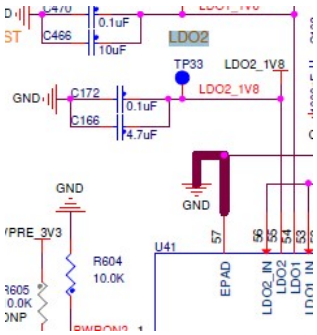
LDO2_STBY	Description	Enable/Disable LDO2 in standby mode
	0	Disabled
	1	Enabled
	Reset condition	

Test result:

Buck3 test point is:



LDO2 test point is:



So after entering the stand by mode, measure whether the level of the relevant test points is preserved, and the measured TP33=1.8V, TP58=3.3V. can be reserved.

5.5 Modify debug serial port to UART1

Because Linux will use UART0, the M7 standby demo is changed to UART1, as follows:

S32g274astandbymode\src\s32g_linflexd\s32g_linflexd.c

```
bool s32g_linflexd_init()
{
#if 1 /* from #if 0 modify to #if 1，init UART1*/
        …/* init UART1*/
#else
        … /* init UART0*/
#endif
```

write_char and get_char function, The corresponding compilation macro control switch of the function is also switched in the same way.

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


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

```
#endif
```

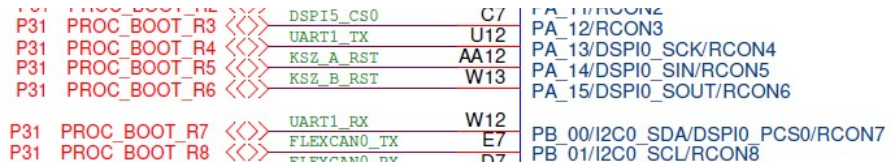
Note, the writing method of write_char in the demo, take UART1 as an example:

```
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++); // The default is implemented using delay, but if the CPU clock frequency changes, there is a risk in using this method, so it is recommended to modify it to the above method of checking the register bit.
    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 /*from #if 0 modify to #if 1, change to 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
```

The following hardware design:



And IOMUX register config:

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

Test:

Connect the M7 core debugging serial port to J1 and UART1, and you can see the debugging information is printed out from UART1.

5.6 Modify the device drive clock

Taking UART as an example, the current clock tree in the standby demo is as follows:

code:

S32g274astandbymode\src\s32g_clock\s32g_pll\s32g_pll.c

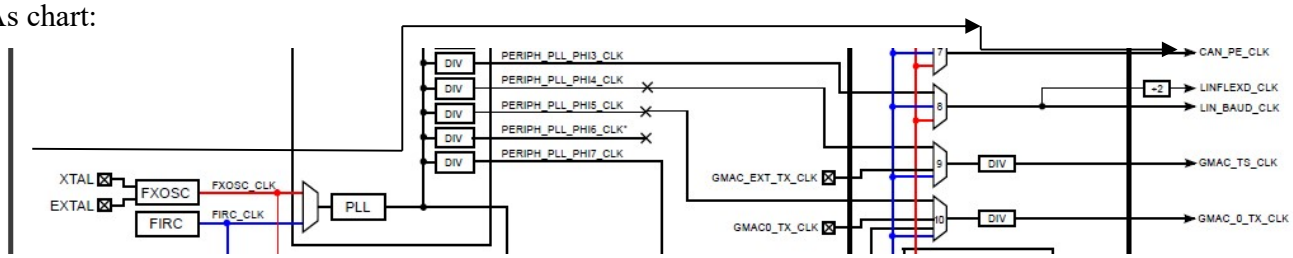
```
void s32g_pll_peripheral()
{
    /* * f(pll_vco) = (f(pll_ref)/PLLDV[RDIV]) * ((PLLDV[MFI]) + (PLLFD[MFN])/18432)
     * PLL_VCO = 2000MHz*/
/* Divider value = 24+1, 2Ghz/25 = 80 MHz, LIN clk set to 80MHz */
    /* Selecting FXOSC as the source for PLL = 1
     * Selecting FIRC as the source for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

As chart:



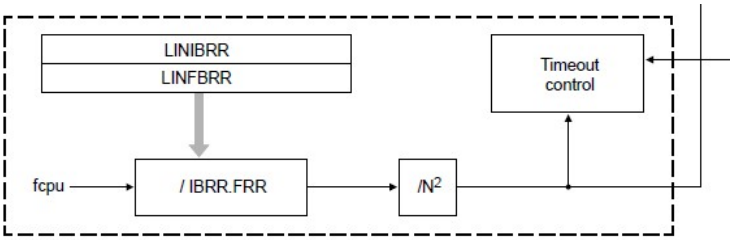
The baud rate configuration inside the UART is as follows:

code:

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;
```

As follows:



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.

Since ROSE is not configured=0 in the code, and over sampling=16. So baudrate=80Mhz/((43.(6/16))/16)=115274 baudrate. Below we configure the LIN_CLK root clock to be 125Mhz, which is the same as inside Linux, and then configure the baudrate to be 115200.

code:

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;
```

The test result can print the serial port message normally.

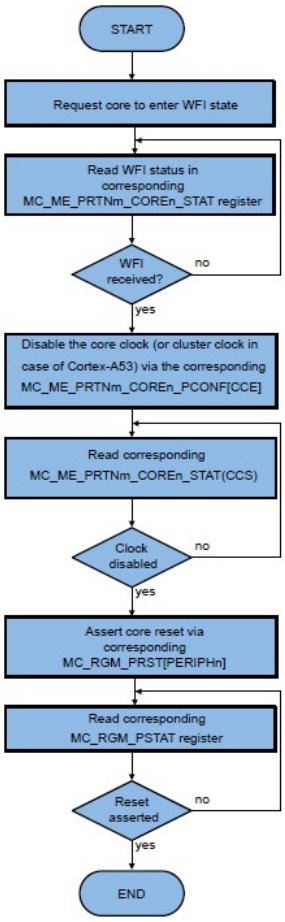
Note: If the number of clocks has been initialized elsewhere, the clock initialization in the standby demo can be commented out, and the divider in the device driver is configured according to the actual clock tree.

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

5.7 close other non-main core

This demo currently only considers the case of one M7_0 core running the demo. In practical applications, other cores need to be placed in WFI first, then turned off, and finally the last M7_0 main core is turned off. So the following code gives an example of turning off the A53_0 core.

In addition, other M cores other than the M7_0 core are also recommended to be shut down by the M7_0 core instead of the A core Linux. The code is similar to the following, and will not be explained in this article. The process of shutting down a kernel is as follows:



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)
```

```
/* Apply changes to MC_ME partitions This function is used to officially trigger the partition state switch */
void mc_me_apply_hw_changes(void)
{
        MC_ME_CTL_KEY = 0x5AF0;
        MC_ME_CTL_KEY = 0xA50F;
}
/*close core 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;
}
/*Enable clock hardware update */
```

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;
}

/* Check whether the clock gate is successful */
```

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 Build a new MCAL demo

This Demo itself is the register Hard Coding code of the S32DS environment. Generally, the customer considers the integration problem and hopes to realize a MCAL Demo. This chapter uses the Uart_Example_S32G274A_M7 project as the basis to explain how to modify it into a MCAL Standby Demo, using the full boot+GPIO wake-up method. Major modifications include:

- Modify the baud rate of UART, M core clock and some clock configurations.
- Implement the API to turn off the clock.
- Configure the Standby mode of the MCU module.
- To configure the wake-up source, you need to add and configure the ICU and EcuM modules, and modify the relevant codes.
- To add PMIC driver, you need to add and configure PMIC, I2C and DIO modules, and modify the code for PMIC to enter Standby.
- Main function call logic.

Create a UART example project according to the document <<S32G_RTD_MCAL_V*.pdf>> and compile it. The RTD version used in this article is SW32G_RTD_4.4_3.0.2.

6.1 Modify the UART driver

1 Modify the baud rate of Uart (set the baud rate to 115200):

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

- DesireBaudrate= LINFLEXD_UART_BAUDRATE_115200

2 Modify the clock of M7 Core (set the M7 core clock to 400Mhz for normal use):

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 // Then PLL_VCO in McuPll_parameter is automatically calculated as 2G

->McuCoreDFS->McuDfs_1：

- DFS1 Output Port Enable=checked
- DFS1 MFI=1
- DFS MFN=9
- DFS1_CLK Frequency=8.0^{E}8 // automatically calculated as

->McuCgm0ClockMux0

- CGM0 Clock Mux0 Source*=CORE_PLL_DFS1_CLK
- Clock Mux0 Frequency (XBAR_2X_CLK)= 8.0E8 // automatically calculated as 8.0E8, so M7 clock=400Mhz。
- CGM0 Clock Mux0 Divider0 Enable (LBIST_CLK)=unchecked //bug fix
- Clock Mux0 Divider1 Frequency (DAPB_CLK)=1.3E8// automatically calculated as

3 Modify some clock configurations of the project (so that M7 can control the switcher of FXOSC):

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

->McuFXOSC：

- FXOSC under MCU control*=checked// It is convenient to control the switch of FXOSC later

4. Generate code:

```
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 Implement the clock shutdown code

In the function mcu_ts_*\src\clock_ip.c, there is its own local function:

```
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 */
```

This function implements the complete deinitialization of the clock to the initial safe state, so we need to expose this function to realize the function of closing the clock, modify as follows:

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);
```

Then the main function calls: Mcu_ResetClockConfiguration(McuClockSettingConfig_0); to turn off the clock.

6.3 Configure the power mode switching driver

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

->General->McuGeneralConfiguration：

- Mcu Enter Low-Power Mode =checked //open standby support

->General->McuModelConfiguration：

- Mcu Number of Mode Settings*=2//add a MCU mode

->McuModeSettingConf：

On McuModeSettingConf_0 right click，choose duplicate element to copy a configuration to modify:

Open McuModeSettingConf_1：

->General:

- Mode ID*=1
- Operating Mode*=SOC_STANDBY //it is Standby mode
- Main Core Select*=CM7_0
- Mcu Enable Sleep On Exit*=checked //support wakeup

M7 Demo is mainly responsible for the Standby of M7 core and A53_0 core, so you only need to configure McuPartition0Config and McuPartition1Config:

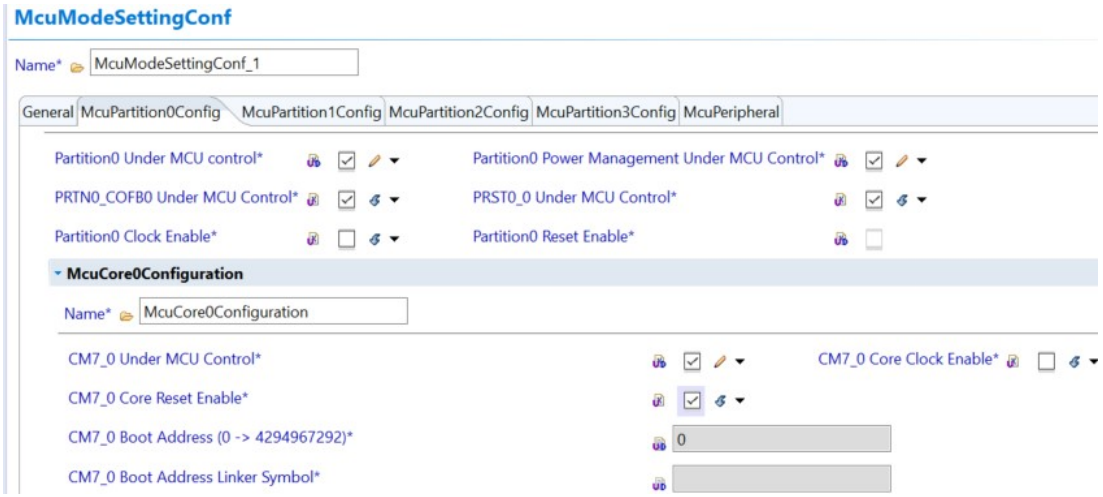
->McuPartition0Config：

- Partition0 Under MCU control*=checked //default
- Partition0 Power Management Under MCU Control*=checked //default
- PRTN0_COFB0 Under MCU Control*=checked // Modified
- PRST0_0 Under MCU Control*=checked // Modified
- Partition0 Clock Enable*=checked//keep this partition clock
- Partition0 Reset Enable*=unchecked //default， Whether to notify the reset signal, since M7_0 is the last main core, so it is not necessary, this item cannot be configured

->McuCore0Configuration：

- CM7_0 Under MCU Control*=checked //default
- CM7_0 Core Clock Enable*=checked //keep the M7_0 clock
- CM7_0 Core Reset Enable*=unchecked//do not reset this main core。

The remaining M7_1~2 core can also be configured for management, which is not configured in this article. But because of the bug fix, the clock enable of CM7_1/2 needs to be turned off. Reset enable is turned on.



->McuPartition1Config：

- Partition1 Under MCU control* =checked //default
- Partition1 Power Management Under MCU Control*=checked //default
- PRTN1_COFB0 Under MCU Control*=checked //modified
- PRST1_0 Under MCU Control*=checked // modified
- Partition1 Clock Enable*=unchecked // modified, close A53 Partition clock.
- Partition1 Reset Enable*=checked //default

->McuCore0Configuration：

- Cortex-A53 CORE 0 cluster 0 Under MCU Control*=checked// Modifid, need to use M core code to turn off A53_0
- CA53 cluster0 Core Clock Enable*=unchecked //default，close A53_0 clock.
- Cortex-A53 CORE 0 cluster 0 Reset Enable*=checked //default

The rest of A53_1~3 should not be modified normally, and Linux is responsible for closing them.

Generate codes:

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)
  }
```

So you can enter the soc_standby mode through the following API:

```
/* Apply a mode configuration */
  Mcu_SetMode(McuModeSettingConf_1); //the call routine as follows:
|->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 Confgure the wakeup source

1. Firstly, configure the ICE module:

On Uart_Example_S32G274A_M7 right clock, then click Module Configurations, in Available Modules choose icu(…), click the right arrow to join the project:

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

- IcuDevErrorDetect=unchecked// Turn off some parameter checking
- Post Build Variant Used*=checked //modified
- Config Variant= VariantPostBuild

->IcuConfigSet：

- IcuMaxChannel =1 //modified to 1

-> IcuAutosarExt：

- IcuWkpuStandbyWakeupSupport=checked //modifed to 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

Click the + to add IcuWkpu_0, click to enter, and then click the + to add IcuWkpuChannles_0:

- Wkpu Channel (dynamic range)*=0
- ICU Wakeup Filter Enable*=checked //Open it 。
- ICU Wakeup Pullup Enable*= checked //Open it, note that this feature currently has bugs, need modify the source codes to fix it.

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

Click the + to add IcuChannel_0, click to enter：

->General:

- IcuChannelRef*= /Icu/Icu/IcuConfigSet/IcuWkpu_0/IcuWkpuChannels_0

- IcuDefaultStartEdge*= ICU_FALLING_EDGE\\由 Since the pin configuration selects pull-up, the trigger condition of this must be selected as falling edge
- IcuMeasurementMode*= ICU_MODE_SIGNAL_EDGE_DETECT
- IcuWakeupCapability*=checked//modifed: Channel is wakeup capable

Note that after selecting this item, you need to open IceWakeup, and then

- Name=IcuWakeup
- IcuChannelWakupInfo=// This item needs to be configured by adding the EcuM module, and then explain how to configure the wakeup source. Since EcuM is an upper-level module, and NXP MCAL is only a sample module, it will not be described in detail.

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. Configure the EcuM module (experimental example, refer to ICU):

On Uart_Example_S32G274A_M7 right clock, and then clock Module Configurations, in Available Modules choose EcuM(…), clock the right arrow to join into this project. here is just a simple configuration of an EcuM wake-up source for ICE to reference, and no details are given.

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

->General->EcuMConfiguration-> EcuMCommonConfiguration:

- EcuMConfigConsistencyHash=2 //config a value

-> EcuMDefaultsShutdownTarget:

- EcuMDefaultState= EcuMStateSleep

->EcuMGeneral：

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

->EcuMSleepMode:click + to add a EcuMSleepMode_0, click to enter:

- EcuMSleepModeId=0//
- EcuMSleepModeSuspend*=checked//
- EcuMSleepModeMcuModeRef*=/Mcu/Mcu/McuModuleConfiguration/McuModeSettingConf_1// Choose our configuration in sleep mode.

EcuMWakeupSourceMask: click + to add one。Choose the follow EcuMWakeupSource_0.

->EcuMWakeupSource: click + to add aEcuMWakeupSource_0, click to enter:

- EcuMWakeupSourceId=0
- EcuMWakeupSourcePolling*=checked

->EcuMOSResource: click + to add one //bug fix

Then configure the ICE module:

IceChannelWakeupInfo=/EcuM/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource_0

3. Finally configure the wakeup source in PORT Module:

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

   ->General：

   - PortNumberOfPortPins=//add a pin

   -> PortPin：click + to add a pin, click to enter, configure LLCE_CAN0_RX to 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. Generate codes:

```
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. Modify the bug of PULL UP ENABLE, there are two points:
   - WKPU_IP_SUPPORT_PULLUP Compile control macros do not work, delete them.
   - WIPER register is the wrong name, change to 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. Add STBY GPR register access codes:

As shown below, before entering standby, the pins pulled up by the SIUL register need to be switched to be pulled up by WKPU, because only the STBY GPR WKPU register can be kept in Standby mode.

31.5.5.1.1 Standby mode entry flow diagram

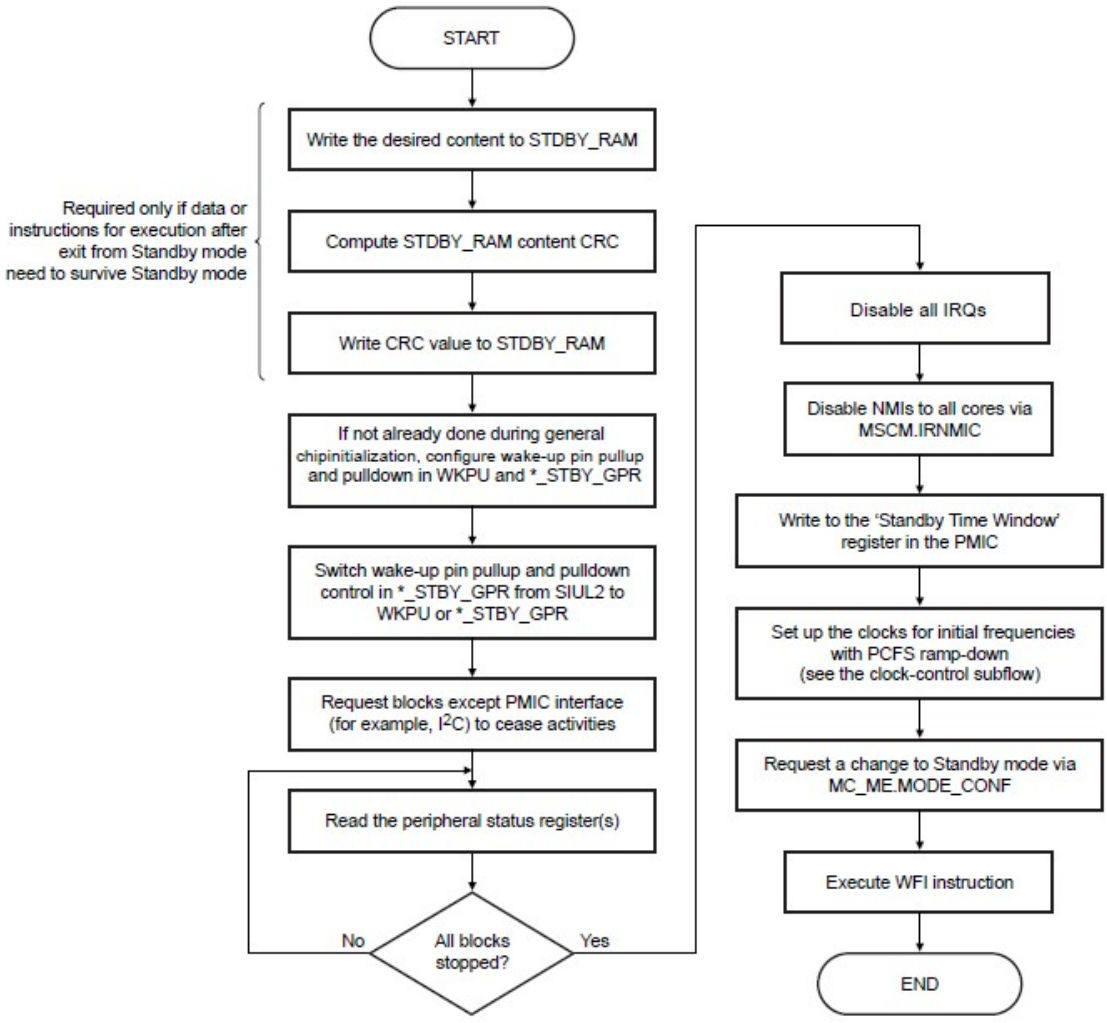


Figure 178. Standby mode entry flow diagram

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. Modify the optimization problem (channelMaskLo/Hi will be optimized and changed to volatile variables):

   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. Add WBMS register access APIs:

   At present, the IDE and source code of the entire ice module do not provide the configuration and access codes of the WBMS registers, so it is temporarily implemented with local functions.

   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
  }
}
```

Then call it as follows:

```
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 function call:

In addition to calling Icu_Init to initialize filter, Pullup, and activation, you also need to call Icu_EnableEdgeDetection to initialize the wakeup source.

6.5 Add PMIC driver

1. Add I2C module(used to link PMIC)：

On Uart_Example_S32G274A_M7 right clock, and then click Module Configurations, in Available Modules choose I2C(…), clock the right arrow to join in the project:

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

->General

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

->I2cChannel:click + to add I2cChannel_0, click to enter:

->I2cChannel_0：

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

Then on Uart_Example_S32G274A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0->McuClockReferencePoint, click + to add one item, enter:Name= I2C_CLK

- Mcu Clock Frequency Select= XBAR_DIV3_CLK
- Mcu Clock Reference Point Frequency= 8000000.0// Automatic calculation

Back to I2C moduel, Open I2cClockRef:

- I2cClockRef= /Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig_0/I2C_CLK
- I2c Asynchronous Method= I2C_USING_INTERRUPTS
- I2c Baud Rate (0 -> 1000000)= 400000.0 // Automatic calculation, related timing setting are:
- 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. Modify the Port module to add I2C pins

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

   ->General :

   - PortNumberOfPortPins=//add two pins

   -> PortPin：click + to add one pin, enter it :

   - 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

   Add SDA pin with the same way:

   - Name= I2C4_SDA
   - PortPin Pull Enable /PortPin Pull Select =checked// pull enable, Note that it needs to be pulled up, otherwise it will cause the I2C connection to fail
   - 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. Add DIO module (provide I2C GPIO module release function, use GPIO function to simulate release algorithm):

   On Uart_Example_S32G274A_M7 right click, then click Module Configurations, in Available Modules choose Dio(…), click right arrow to join in the project:

   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//Modifed

->DioPort: click + to add one item, enter it:

- Name*= DioPort_Pmic

->DioChannel: click + to add one item, enter it:

- Dio Channel Id*=2 //because I2C4 IOMUX from PC_01/02, which is the second group of GPIOs.

4. Add PMIC module:

On Uart_Example_S32G274A_M7 right click , then click Module Configurations, in Available Modules choose Pmic(…), click right arrow to join in the project:

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

->General :

- Post Build Variant Used*=checked//modified
- Config Variant*= VariantPreCompile// modified
- Provide PMIC Watchdog API=checked// modified, need configure watchdog
- PMIC Default Error Detection*=unchecked// modified, Prevent some parameter checking bugs in the code itself.

->PmicDevice： click + to add a new PmicDevice_0, click to enter:

->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

Except for PmicAmuxChannel and PmicSVSSettingConf, click the + to add default items for all other items.

Then in PmicClockSettingConf-> PmicClockSettingConfig_0->PmicClockReferencePoint: click + to add:

PmicClockSettingConfig

Name PmicClockSettingConfig_0

General | PmicClockReferencePoint

PmicClockReferencePoint

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

Then in PmicClockSettingConfig_0->General-> :

- Low Power Oscillator Frequency [Hz]= F_600000 // Pmic_InitClock needs to be called to initialize the configuration.

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

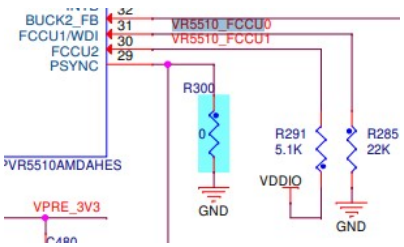
And Then in PmicFailSafeConfigure:

->PmicWatchdogConfiguration:

- Watchdog API Enable*=checked //bug fix
- Watchdog Window Period Enable=unchecked // Turn off the watchdog cycle to feed the dog here, even if the dog is not fed in Normal mode, it will not cause a restart.

->PmicSafeInputsConfiguration：

- FCCU Monitoring Configuration= DISABLED// Modified, as the following schematic design:



S32G Standby Demo

Therefore, the default hardware configuration is in the fault state of FCCU1=low, FCCU2=high. It is necessary for the S32G to pull the reverse pull to exit the fault state after configuring the FCCU. Otherwise, after feeding the dog for the first time, the PMIC will cause a restart after entering the NORMAL_FS from INIT_FS , so we turn off the FCCU Monitoring function to prevent the restart caused by feeding the dog.

- Safety Standby Window Duration=TIME_8MS // It is used to configure the time window from i2c writing stdby request to S32G pulling the standby pin. The setting here is relatively large. Note that the time of these two operations should be less than 8ms, so no large delays such as serial port debugging can be added between them. operation, or use a debugger.

Then click the + in PmicModeSettingConf to add another configuration, enter, and configure it as standby mode:

- Name*=PmicModeSettingConf_1_Standby

  ->General:

- Operating Mode*=STANDBY
- Standby Timer Enable=checked//Open
- Standby Timer Window Duration= TIME_8388608MS// Set to the maximum, this is the maximum retention time of the Standby state, so set to the maximum.

  ->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. Generate codes:

```
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. Modify the setmode function (rewrite the setmode function according to the requirements of PMIC to enter standby):

   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 function call:

According to the above analysis: In addition to using PMIC_Init and PMIC_InitDevice for initialization, the Main function needs to call Pmic_InitClock to set the clock in Standby mode, and call

PMIC_SetMode to enter Standby mode. The time for calling this function to S32G to pull the Standby pin cannot exceed 8ms.

6.6 Main function call routine

The modifications mainly include:

- Implement the initialization of newly added modules and include header files.
- Change the original asynchronous serial interface transceiver mode to synchronous interface.
- Switch the PMIC mode according to the serial port software.
- Turn off CLOCK.
- Switch to STANDBY mode.

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);
}
```

According to the document <<S32G_RTD_MCAL_V*.pdf>>(JohnLi Chinese version), modify and compile related files. Note that because related modules have been added, the following files also need to be modified

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
```

Then execute “make build” command to compile the main.elf file.

6.7 Test

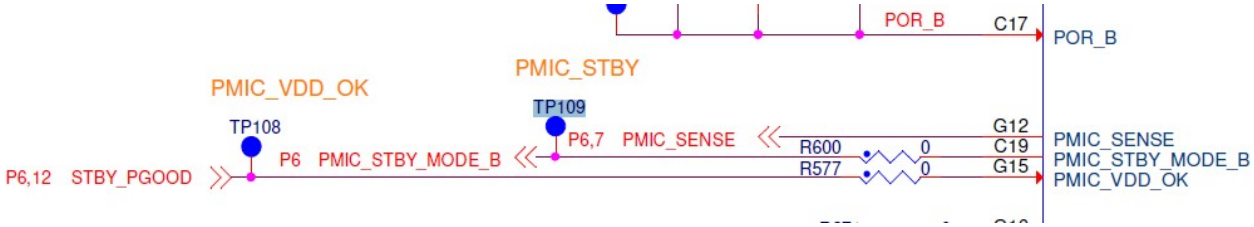
According to the document <<S32G_RTD_MCAL_V*.pdf>>(JohnLi Chinese version), convert the compiled “main.elf” into a “main.bin” file, and then use the IVT tool of S32DS to mark the IVT header, pay attention to add the timing header of QSPI NOR. Then use the Flash tools of S32DS to burn the image into QSPI NOR through UART0 in the download mode, then switch back to the normal startup mode, change the serial port to UART1, and start:

1. UART1 output:

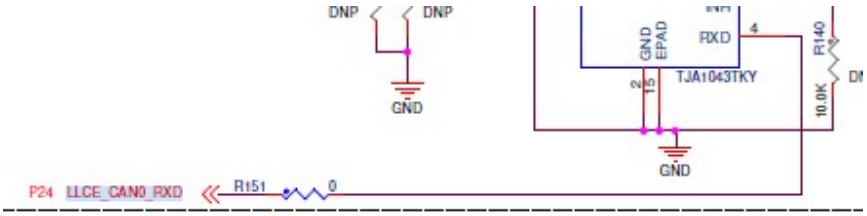
   ```
   Stdby demo: Full Boot, pull LLCE CAN0 RX to low to  resume
   ```

   And then enter Standby mode.

2. Measure the S32G Standby pin TP109 and the PMIC Stby_pgood pin, both are low, indicating that the Standby signal output from the S32G to the PMIC and the standby good signal returned by the PMIC are normal:



3. Other power lights are off, indicating that the relevant power is off.
4. As follow LLCE_CAN0_RX test point:



So touch the R151 to the ground, the system will resume to full boot and run again, and the serial port will print again:

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

6.8 Future development plan

- This chapter only describes the full boot method. The SSRAM short standby method is not described because of its limited application scenarios.
- This chapter only explains the way of GPIO resume, RTC resume is not explained yet, customers can refer to self-development.