S32G3 M7 Standby + QSPI Power Down Demo 说明

by John Li (nxa08200)

本文说明S32G3 M7核Standby MCAL demo 详细情况及定制，并在进入Standby之前调用QSPI 接口将QSPI NOR flash配置进入deep power down模式，以节省用电。

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

目录

1 参考资料说明

分类	名称	说明	备注
文档及代码	S32G_Standby_Demo_V*	S32G2 RDB2 板的 Demo 说明及代码	Download from: https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-M-kernel-Standby-demo-and-how-to-porting-to-Mcal/ta-p/1556313
代码	SW32G_RTD_4.4_4.0.2	MCAL	AutoSar MCAL 从www.nxp.com/s32g下载
文档	MX25UW51245G.pdf	芯片手册	Macronix QSPI NOR 数据手册
文档	S32G_ADD_GDFLASH_SUPPORT_V*	QSPI NOR flash 软件开发说明	Download from https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-ADD-GDFLASH-SUPPORT/ta-p/1763551

注意：由于本文是《S32G_Standby_Demo_V*》升级到 G3 的版本，并且增加了配置外部 QSPI NOR Flash 进入 deep power down 模式的功能，所以关于 S32G3+VR5510 相关 STANDBY 的文档要求，请参考前文。关于本文第三章的内容，必须参考前文相关内容。

2 G2 和 G3 Demo 的区别

- 硬件方面，参考文档：《AN12880-5510-standby》，如下：

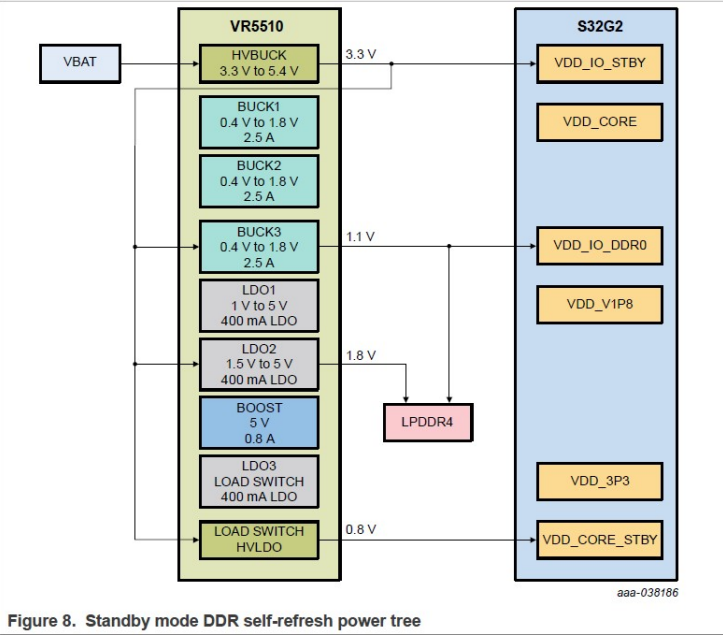


Figure 8. Standby mode DDR self-refresh power tree

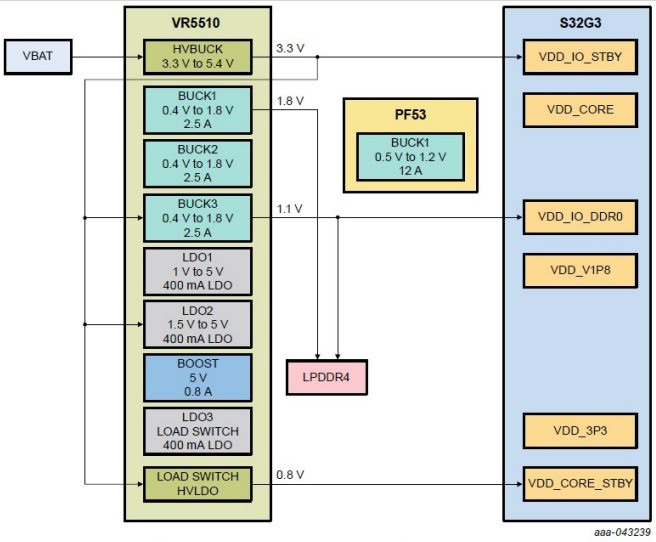


Figure 9. Standby mode DDR self-refresh power tree for S32G3 solution

所以 G2 与 G3 的区别在于对于 LPDDR4 自刷新状态的电源维持，G2 使用 LDO2 供电，而 G3 使用 BUCK1 供电，这样 PMIC 的驱动配置会不同。

- 软件方面：G2 Demo 基于 SW32G_RTD_4.4_3.0.2 开发的，G3 基于的 SW32G_RTD_4.4_4.0.2，对 3.0.2 原有的一些 Bug 进行了 Fix。G3 只提供 MCAL demo，不再提供 S32DS hard coding 的 demo，建议也参考此 demo。

3 G3 MCAL Demo 的实现

参考文档《S32G_Standby_Demo_V*》的第六章：修改为 MCAL demo，在 SW32G_RTD_4.4_4.0.2 创建 MCAL demo，所以本章绝大部分内容与前文相同，请参考前文，本文仅说明不同之处。

3.1 修改 UART 驱动

本节修改与 G2 demo 基本相同，请参考前文。注意点：

1. Uart_Example_S32G399A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0：

   ->McuCorePLL->McuPll_Configuration：

   原文遗漏：

   RDIV*=1 //change from 2。

2. Uart_Example_S32G399A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0：

   ->McuFXOSC：

   - FXOSC under MCU control*=checked//4.0.2 上为默认配置。

   自动生成代码时遇到的不相关的 Clock 配置错误采用自动计算方法来解决：

3. Uart_Example_S32G399A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0：

   ->McuCgm1PcsConfig_0:

   - Source Frequency (Hz) (0 -> 2000000000) = 1.0E8 //自动生成。

3.2 实现时钟关闭代码

注意本处修改实际上有两种方式：

- 一种是不需要修改代码，而是再配置一个 McuClockSettingConfig_1，在此 Config 中将所有时钟使用 EB 配置到初始态，然后再调用函数 Mcu_InitClock(McuClockSettingConfig_1); 来实现，这样就不用编程。
- 考虑到关闭时钟代码有执行时间要求，所以本节仍然采用修改代码的方式来实现快速关闭时钟。

本节修改与 G2 demo 相同，请参考前文。

3.3 配置电源模式切换驱动

本节修改与 G2 demo 基本相同，请参考前文。注意点：

1. Uart_Example_S32G399A_M7->someid(…)->Mcu(…)->Mcu：

   ->General->McuGeneralConfiguration：

   ● Mcu Enter Low-Power Mode =checked //打开 standby 支持， 4.0.2 为默认配置。

2. ->General->McuModelConfiguration：

   ● Mcu Number of Mode Settings*=9//增加一个 MCU 模式，4.0.2 本来就有 0~8 个模式。

   ->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*=9 //自动修改为 9。

   此时 Mcu->General 中的

   ● Mcu Number of Mode Settings*=10 //自动计算。

   //其它参考前文。

3. ->McuPartition1Config：

   将 A53_1~3，4~7 全部设置为 Under MCU Control=unchecked，由 Linux 负责关闭。Core Reset Enable=checked。

   //其它参考前文。

3.4 配置唤醒源

本节修改与 G2 demo 基本相同，请参考前文。细节的不同点请参考工程原文件。注意点：

1. Uart_Example_S32G399A_M7->someid(…)->icu(…)->icu->IcuHWInterruptConfigList，点击+增加：IcuHwInterruptConfigList_0

   ● ICU Peripheral ISR Name= WKPU_CH_0

   ● IcuIsrEnable=checked //4.0.2 需要增加中断配置来打开 WKPU 中断，本处是考虑要处理 WKPU 中断的情况下需要打开，然后在 main 函数中注册此中断处理函数。

2. 3.0.2 版本 ICE 模块 Bug 比较多，所以前文中有很多源代码 Bug fix 的说明，4.0.2 版本已经 fix，所以不需要再修改源代码。

3.5 加入 PMIC 驱动

本节修改与 G2 Demo 基本相同，请参考前文。注意点：

1. Uart_Example_S32G399A_M7->someid(…)->I2c(…)->I2c->I2cChannel-> I2cChannel_0

由于 EB 界面修改，对于 Core PLL 2GHZ->DFS1 800MHZ->XBAR_DIV3_CLK 133Mhz，再到 I2C clock，根据 S32G RM 说明：

Table 308. I²C divider and hold values when glitch filter is disabled (continued)

IBC (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
29	384	33	190	193

先设置：

- I2c IBC Register Value (dynamic range)=0x29 //手动设置，然后之后的值使用自动计算获得：

2. Uart_Example_S32G399A_M7->someid(…)->Dio(…)->Dio

->DioPort:点击+号增加一项，修改为：

- Name*= DioPort_Pmic

  ->General:

- Dio Port Id = 2 // 2 - PortC - SIUL2_0

  ->DioChannel: 点击+号增加一项，修改：

- Dio Channel Id*=2 //因为 I2C4_CLK IOMUX 自 PC_02，为第 2 个 GPIO。

3. Uart_Example_S32G399A_M7->someid(…)->Pmic(…)->Pmic->PmicModeSettingConf->PmicModeSettingConf_1_Standby->PmicRegulatorsConfig:

根据之前的 G2 与 G3 供电不同分析，应该给 BUCK1 而不是 LDO2 保电。

- HVLDO Standby Mode Enable*=checked //HVLDO in Standby Mode is Enabled。
- BUCK3 Standby Mode Enable*=checked //enable the DDR 1v1 for DDR selfrefresh。
- BUCK1 Standby Mode Enable*=checked //enable the DDR 1v8 for DDR selfrefresh。
- BUCK1 Standby Output Voltage (V) (0.4 -> 1.8)*=1.8V。

4. 在文档《S32G_Standby_Demo_V*》，对 PMIC Setmode 函数进行了源代码修改，主要为了 feed watchdog 与 disable watchdog，而实际上这些操作理论上应该由 PMIC_Watchdog 驱动来完成，而且默认的 PMIC 驱动不修改也支持 Set to suspend mode，所以本文此处不再修改源代码。

3.6 主函数逻辑实现

本节修改与 G2 Demo 基本相同，请参考前文。注意点：

本新 Demo 增加一下串口输入控制功能：

```
#define STBY_MSG "Stdby demo: press y to enter, Full Boot,pull low of LLCE CAN0 RX to resume\r\n"
#define ENTER_MSG "Stdby demo: Enter Stdby\r\n"
…
uint8 Rx_Buffer[MSG_LEN];
…
    while (1)
   {
      memset(Rx_Buffer, 0 , 1);
      (void)Uart_SyncReceive(UART_CHANNEL, Rx_Buffer, 1, 10000);
       if(Rx_Buffer[0] == 'y')
      {
         break;
      }
   }
   (void)Uart_SyncSend(UART_CHANNEL, (const uint8 *)ENTER_MSG, strlen(ENTER_MSG), 10000);
                Pmic_InitClock(PmicConf_PmicDevice_PmicDevice_0,0);
```

3.7运行测试

生成 BLOB 镜像后，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：

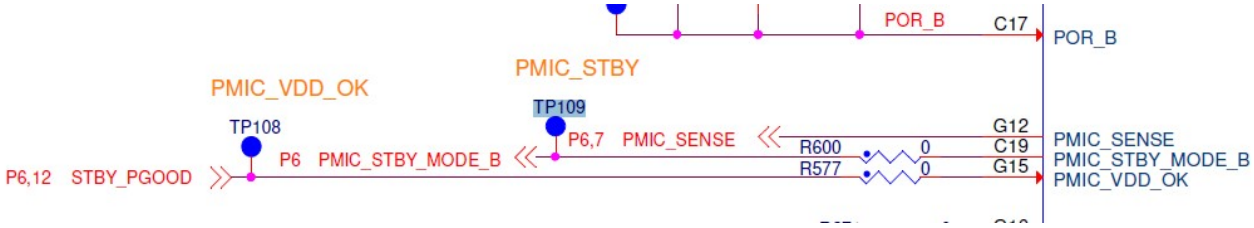
1. 串口打印出：

```
Stdby demo: press y to enter, Full Boot,pull low of LLCE CAN0 RX to resume
```

输入 y 然后进入 Standby 模式：

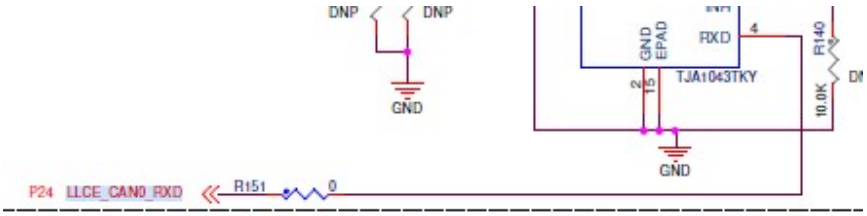
```
Stdby demo: Enter Stdby
```

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: press y to enter, Full Boot,pull low of LLCE CAN0 RX to resume
```

3.8 未来开发计划

本节修改与 G2 demo 相同。

4 将 QSPI NOR 设置进入 Deep Power Down 模式

根据 S32G3 原理图设计，如下：

所以在进入 STANDBY 时，如果需要 LPDDR4 进入自刷新模式，则需要保持住 BUCK1 的电源持续输出，这样会导致 QSPI NOR 在进入 STANDBY 后，并不会关闭电源，会增加更多耗电，减少 QSPI NOR 的耗电有硬件和软件的两个办法：

- 从硬件上，可以增加一个开关来控制 QSPI NOR 和 VDD_IO_QSPI 的电源关断。
- 从软件上：在 RDB3 的 Hardware Design Guide 中说明如下：

For use cases with DDR self-refresh mode:

- VDD_IO_DDR0 (1.1/1.35 V)
- LDO2 can be used as an additional supply ON during Standby for powering the LPDDR4 memory. VDD_IO_QSPI must also be powered by the same LDO in case QSPI is required to be powered in Standby. The QSPI flash must be configured in deep power-down state during Standby. LDO2 must be connected only to the domain/components that are required in Standby mode.

所以文档建议是使用软件的方式，在进入 STANDBY 之前，先发送命令给 QSPI，将其设置为 deep power down 模式。但是此功能在目前的软件中并没有实现。

参考 QSPI NOR 数据手册如下：

The Deep Power-down (DP) instruction is for setting the device to minimum power consumption (the standby current is reduced from ISB1 to ISB2). The Deep Power-down mode requires the Deep Power-down (DP) instruction to enter, during the Deep Power-down mode, the device is not active and all Write/Program/Erase instruction are ignored. When CS# goes high, it's only in deep power-down mode not standby mode. It's different from Standby mode.

The sequence of issuing DP instruction is: CS# goes low→sending DP instruction code→CS# goes high.

Once the DP instruction is set, all instruction will be ignored except the Release from Deep Power-down mode (RDP) and Read Electronic Signature (RES) instruction and softreset command. (those instructions allow the ID being reading out). When Power-down, or software reset command the deep power-down mode automatically stops, and when power-up, the device automatically is in standby mode. For DP instruction the CS# must go high exactly at the byte boundary (the latest eighth bit of instruction code been latched-in); otherwise, the instruction will not executed. As soon as Chip Select (CS#) goes high, a delay of tDP is required before entering the Deep Power-down mode.

ISB1	VCC Standby Current	1		20	100	uA	VIN = VCC or GND, CS# = VCC
ISB2	Deep Power-down Current			5	20	uA	VIN = VCC or GND, CS# = VCC

Symbol	Alt.	Parameter	Min.	Typ.	Max.	Unit
tDP[2]		CS# High to Deep Power-down Mode			10	us

所以需要在进入 STADNBY 之前，先让 QSPI NOR 时入到 deep power down 模式，而由于 full boot 是使用从头重启的方法，会执行 reset 命令，所以不需要实现退出 deep power down 模式指令。

Linux 的 QSPI-NOR 驱动，本身已经实现的 power management 函数，所以可以通过修改相关函数来实现：Linux\drivers\mtd\spi-nor\core.c：

```
/* mtd suspend handler */
static int spi_nor_suspend(struct mtd_info *mtd)
{…

  //可以考虑增加在此函数调用前，或者后(根据 DTR 是否使用的情况下)。
    /* Disable octal DTR mode if we enabled it. */
    ret = spi_nor_octal_dtr_enable(nor, false);
…
}
```

```
/* mtd resume handler */
static void spi_nor_resume(struct mtd_info *mtd)
{…
    /* re-initialize the nor chip */
    ret = spi_nor_init(nor);
    …
}
```

而与 Linux 不同，MCAL 的 FLS 驱动架构 API 中并没有考虑到 power management 的功能，所以需要想办法借用现有的 API 来实现相关功能。

先使用 Fls 工程来开发测试 API。

根据文档《S32G_ADD_GDFLASH_SUPPORT_xxxx》创建一个 Fls 工程。

4.1 Fls 层的修改

在文件 fls_tx_xxx\src\fls.c 中定义了与 Autosar 规范匹配的 Flash 上层函数，所以我们可以考虑借用 Fls_SetMode 函数，增加相应的功能如下：

```
void Fls_SetMode(MemIf_ModeType Mode)
{…
 switch (Mode)
     {…
//johnli add for deep power down mode
        case MEMIF_MODE_DEEP_PWRDWN: //增加此宏
          Fls_IPW_Deeppwrdwn(); //增加此 API.
//end
…
```

Memif_ts_xxx\include\Memif_Types.h

```
typedef enum
{…
  MEMIF_MODE_DEEP_PWRDWN  /** johnli add the deep power down mode **/
}MemIf_ModeType;
```

4.2 中间层的修改

fls_tx_xxxx\include\fls_ipw.h

```
void  Fls_IPW_Deeppwrdwn(void); //johnli add for deep power down mode
```

fls_tx_xxxx\src\fls_ipw.c

```
void Fls_IPW_Deeppwrdwn(void)
    {
    //          Fls_SectorIndexType SectorIndexIter;
                Qspi_Ip_StatusType Status = STATUS_QSPI_IP_SUCCESS;
                uint32 flashInstance;
                uint32 ControllerInstance;
                uint32 lastFlashInstance = FLS_DEVICE_INSTANCE_INVALID;

    //          for (SectorIndexIter = Fls_u32JobSectorIt; SectorIndexIter <= Fls_u32JobSectorEnd;
SectorIndexIter++ )
    //          {
                        /* Get external flash instance */
                        flashInstance = (*(Fls_pConfigPtr->pFlsQspiCfgConfig->u8SectFlashUnit))[0];

                        MCAL_DATA_SYNC_BARRIER();
                        MCAL_INSTRUCTION_SYNC_BARRIER();
                        /* Check if this channel already was checked before */
                        if (flashInstance != lastFlashInstance)
                        {
                                lastFlashInstance = flashInstance;
                                /* Get controller instance */
                                ControllerInstance =
((*(Fls_pConfigPtr->pFlsQspiCfgConfig->paFlashConnectionCfg))[flashInstance]).qspiInstance;

                                /* Prepare timeout counter */
                                Fls_Qspi_u32ElapsedTicks = 0U;
                                Fls_Qspi_u32TimeoutTicks =
OsIf_MicrosToTicks(QSPI_IP_CMD_COMPLETE_TIMEOUT, (OsIf_CounterType)QSPI_IP_TIMEOUT_TYPE);
                                Fls_Qspi_u32CurrentTicks =
OsIf_GetCounter((OsIf_CounterType)QSPI_IP_TIMEOUT_TYPE);
                                /* Wait for the controller to become idle */
                                do
                                {
                                        /* Add Fault Injection point for FR_ILLINE flag */
```

```
        MCAL_FAULT_INJECTION_POINT(FLS_FIP_FR_ERROR_ABORTSUSPEND);

                                    Status = Qspi_Ip_ControllerGetStatus(ControllerInstance);
                                    Fls_Qspi_u32ElapsedTicks += OsIf_GetElapsed(&Fls_Qspi_u32CurrentTicks, (OsIf_CounterType)QSPI_IP_TIMEOUT_TYPE);
                                    if ((STATUS_QSPI_IP_BUSY == Status) && (Fls_Qspi_u32ElapsedTicks >= Fls_Qspi_u32TimeoutTicks))
                                    {
                                        (void) Det_ReportRuntimeError((uint16)FLS_MODULE_ID, FLS_INSTANCE_ID, FLS_MAINFUNCTION_ID, FLS_E_TIMEOUT);
                                        Status = STATUS_QSPI_IP_TIMEOUT;
                                    }
                                }
                                while (STATUS_QSPI_IP_BUSY == Status);
    #if (FLS_QSPI_HANG_RECOVERY == STD_ON)
                                if (STATUS_QSPI_IP_TIMEOUT == Status)
                                {
                                    /* The controller is being stuck in BUSY state, perform the abort sequence */
                                    Status = Qspi_Ip_Abort(ControllerInstance);
                                }
    #endif /* (FLS_QSPI_HANG_RECOVERY == STD_ON) */
                                if (STATUS_QSPI_IP_SUCCESS == Status)
                                {
                                    /* Check that external memory is idle */
                                    if (STATUS_QSPI_IP_SUCCESS == Qspi_Ip_GetMemoryStatus(flashInstance)) //之前的操作全部完成
                                    {
                                        /* put it into deep power down mode */调用底层函数发送进入 DP 模式命令
                                        Status = Qspi_Ip_DeepPwrdwn(flashInstance);
                                    }
                                }
                            }
        //          }
```

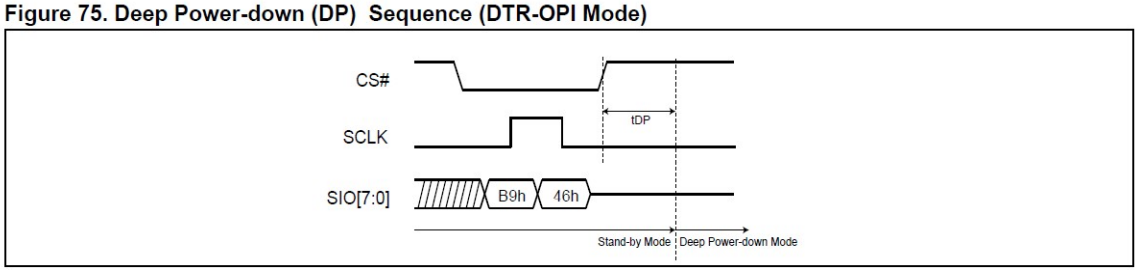
```
            if (STATUS_QSPI_IP_SUCCESS != Status)
            {
                Fls_eLLDJob = FLASH_JOB_NONE;
                Fls_eLLDJobResult = MEMIF_JOB_FAILED;
            }
    }
```

4.3 QSPI_IP 层的修改

1. EB 界面配置：

   根据 QSPI NOR 数据手册说明：

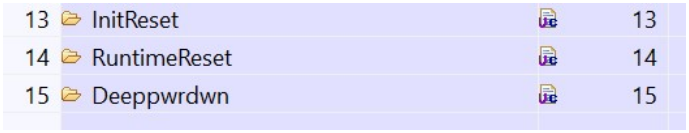
Figure 75. Deep Power-down (DP) Sequence (DTR-OPI Mode)

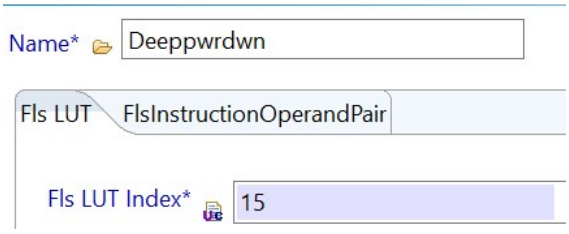


   增加新的 LUT:

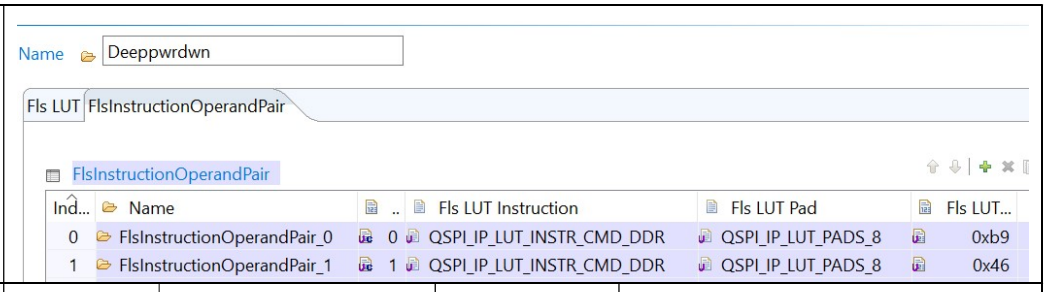
Fls_Example_S32G399A_M7->somId(…)->Fls(…)->Fls->MemCfg-> MemCfg_DOPI->FlsLUT：

   复制 RuntimeReset 修改为：





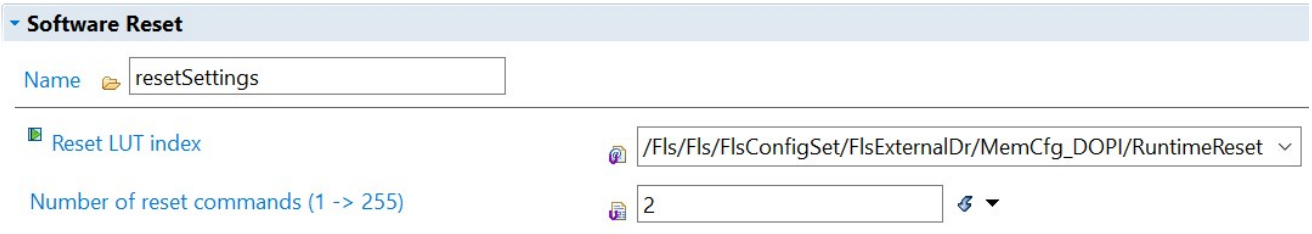
   然后修改其 FlsInstructionOperandPair 配置为：

<table>
<tr><td>Deeppwrdwn
EB 配置</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="3">Deeppwrdwn
具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>47b9</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0xB9 0x46(DP DTR-OPI mode)</td></tr>
<tr><td>4746</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td></tr>
</table>

然后生成代码：
\Fls_TS_xxxx\examples\EBT\S32G3\Fls_Example_S32G399A_M7\generate\src\Qspi_Ip_VS_0_PBcfg.c

```
  /* paLutOperations */
static Qspi_Ip_InstrOpType MemCfg_DOPI_VS_0_paLutOperations_0[65U] =
{…
  /*  58: RuntimeReset  */
  …
  /*  62: Deeppwrdwn  */
  (Qspi_Ip_InstrOpType)((Qspi_Ip_InstrOpType)QSPI_IP_LUT_INSTR_CMD_DDR | (Qspi_Ip_InstrOpType)QSPI_IP_LUT_PADS_8 | (Qspi_Ip_InstrOpType)185U),
  (Qspi_Ip_InstrOpType)((Qspi_Ip_InstrOpType)QSPI_IP_LUT_INSTR_CMD_DDR | (Qspi_Ip_InstrOpType)QSPI_IP_LUT_PADS_8 | (Qspi_Ip_InstrOpType)70U),
  (Qspi_Ip_InstrOpType)(QSPI_IP_LUT_INSTR_STOP)
};
```

注意：在 Fls_Example_S32G399A_M7->somId(…)->Fls(…)->Fls->MemCfg->MemCfg_DOPI->Fls_External_Flash 中，有对一些操作的 LUT 序号引用，比如说：



但是我们新增加的功能并没有此配置界面，所以需要手动修改源代码：

Fls_ts_xxxx\include\Qspi_Ip_Types.h:

```
typedef struct
{
  uint16 deeppwrdwnCmdLut;                /*!< First command in reset sequence          */
  uint8 deeppwrdwnCmdCount;               /*!< Number of commands in reset sequence     */
} Qspi_Ip_DeeppwrdwnConfigType;

typedef struct
{…
  Qspi_Ip_ResetConfigType resetSettings;          /*!< Soft Reset settings, used at runtime          */
  Qspi_Ip_DeeppwrdwnConfigType deeppwrdwnSettings; //johnli add for deep power down
…
} Qspi_Ip_MemoryConfigType;
```

以下生成的源代码也需要每次重新生成后修改：
Fls_ts_xxxx\examples\ebt\s32g3\fls_example_s32g399a_m7/generate/src/Qspi_Ip_VS_0_PBcfg.c
：

```
const Qspi_Ip_MemoryConfigType FlsConfigSet_VS_0_paFlashCfg[1U] =
{…
    {
        /* resetSettings */
        58U,  /* resetCmdLut */
        2U                          /* resetCmdCount */
    },
    //johnli add for deep power down
    {
        /* deeppwrdwnSettings */
        62U,  /* deeppwrdwnCmdLut */ //指向之前生成代码中 deepwrdwn 的 lut 序号
        1U                          /* deeppwrdwnCmdCount */
    },
    //end
```

2. 实现 Qspi_Ip_DeepPwrdwn 函数:

在 fls_ts_xxxx\src\Qspi_Ip.c 中，实现以下函数：

```
Qspi_Ip_StatusType Qspi_Ip_DeepPwrdwn(uint32 instance)
{
  uint16 deeppwrdwnCmdLut;      /*!< First command in deep power down sequence          */
```

```
    uint8  deeppwrdwnCmdCount;       /*!< Number of commands in reset sequence       */
    const Qspi_Ip_StateType * state = &(Qspi_Ip_MemoryStateStructure[instance]);
    Qspi_Ip_StatusType status = STATUS_QSPI_IP_ERROR;

    DEV_ASSERT_QSPI(instance < QSPI_IP_MEM_INSTANCE_COUNT);
    deeppwrdwnCmdLut = state->configuration->deeppwrdwnSettings.deeppwrdwnCmdLut;
    if (QSPI_IP_LUT_INVALID != deeppwrdwnCmdLut)
    {
        deeppwrdwnCmdCount = state->configuration->deeppwrdwnSettings.deeppwrdwnCmdCount;
        /* Perform deep power down*/此外借用了 Qspi_Ip_InitReset 函数，但是传入 deep power down 的 lut 序号
        status = Qspi_Ip_InitReset(instance, deeppwrdwnCmdLut, deeppwrdwnCmdCount, state);
    }

    return status;
}
```

fls_ts_xxxx\include\Qspi_Ip.h

```
Qspi_Ip_StatusType Qspi_Ip_DeepPwrdwn(uint32 instance);
```

4.4 主测试函数调用

在 fls_ts_xxxx\examples\ebt\s32g3\fls_example_s32g399a_m7\src\main.c 中

```
int main(void)
{…
    /* Write data to external sector */
    Fls_Write(LOGICAL_START_ADDR, TxBuffer, FLS_BUF_SIZE);
    while (MEMIF_IDLE != Fls_GetStatus())
    {
        Fls_MainFunction();
    }
    /* Check last job */
    ExampleAssert(MEMIF_JOB_OK == Fls_GetJobResult());
    //在写与读之间增加 deep power down 操作：
    Fls_SetMode(MEMIF_MODE_DEEP_PWRDWN); //johnli for deep power down mode test
    /* Read data from external sector */
    Fls_Read(LOGICAL_START_ADDR, RxBuffer_IP, FLS_BUF_SIZE);
```

```
  while (MEMIF_IDLE != Fls_GetStatus())
  {
      Fls_MainFunction();
  }
  /* Check last job */
  ExampleAssert(MEMIF_JOB_OK == Fls_GetJobResult());
```

4.5 Fls 驱动的测试

为了测试，我们添加代码来防止 Fls_MainFunction 去 reset 外部 QSPI Nor：

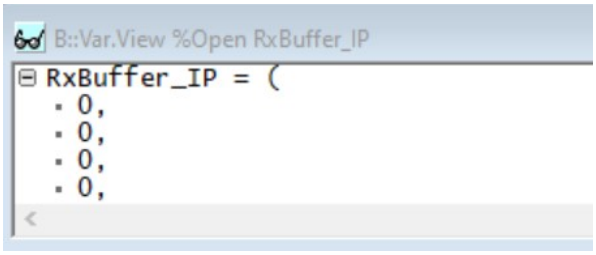
```
static uint8 Fls_u8Deeppwrdwn; //johnli deep power down debug
…
void Fls_SetMode(MemIf_ModeType Mode)
{…
        case MEMIF_MODE_DEEP_PWRDWN:
                                Fls_IPW_Deeppwrdwn();
                                Fls_u8Deeppwrdwn =1; //johnli debug
…
void Fls_MainFunction(void)
{…
  if(1 != Fls_u8Deeppwrdwn) //johnli debug
            {
          Fls_IPW_AbortSuspended(); //此函数会调用 Qspi_Ip_Reset 去 reset qspi nor
            }
```

以上改造事实上不需要，因为进入 deep power down 后，实际应用场景中没有 QSPI NOR 的访问了，直到下次 resume 的时候 reset。本节为了测试增加以上代码来测试进入 deep power down 后是否可以访问成功。

根据文档《S32G_ADD_GDFLASH_SUPPORT_xxxx》，使用 Lauterbach 加载脚本：C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\Fls_Example_S32G399A_M7\debug\run.cmm，停止在 main 函数入口外。使用 Lauterbach 检查 Fls_Init，Fls_Write，的执行结果。

然后再执行完：Fls_SetMode(MEMIF_MODE_DEEP_PWRDWN);后。

再确认一下 Fls_Read 的执行结果：



可以看到读不到内容了，证明 QSPI Nor Flash 已经进入了 Deep power down 模式。

5 将 Deep Power Down 功能集成到 STANDBY 工程中并测试

5.1 EB 配置

1. 增加 Fls 驱动：

在 Uart_Example_S32G399A_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。增加的 Fls 驱动配置界面基本为空，所以先关闭 EB tresos，将 C:\EB\tresos\workspace\Fls_Example_S32G399A_M7\config\Fls.xdm 拷贝到 C:\EB\tresos\workspace\Uart_Example_S32G399A_M7\config\Fls.xdm，重新打开 EB tresos，就可以看 Fls 驱动已经配置好了。

2. 修改 CLK 配置：

Uart_Example_S32G399A_M7->someid(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0

-> McuPeriphPLL：

- RDIV=1
- MFD (1 -> 255)=50

则：

- PLL_VCO Frequency (Calculated) (dynamic range)= 2.0E9 //自动计算为。

其它不相关时钟关掉：

- PHI0 Divider enable=unchecked
- PHI1~7 Divider enable=unchecked
- PLL_PHI0~7 Frequency (Calculated) (dynamic range)=0 //自动计算为。

-> McuPeriphDFS：

- DFS1 MFI (1 -> 255) =1
- DFS1 MFN (0 -> 35) =9 //修改。

- DFS1_CLK Frequency (Calculated) (dynamic range)= 8.0E8 //自动计算为。

同时将以下 DFS 时钟关掉，Bug fix：

- DFS3/2/5 Output Port Enable*=uncheck
- DFS3/2/5_CLK Frequency (Calculated) (dynamic range)*=0//自动计算为。

-> McuCgm0ClockMux12：

- CGM0 Clock Mux12 Source= PERIPH_PLL_DFS1_CLK //修改为此时钟。
- Clock Mux12 Divider0 Frequency (QSPI_2X_CLK) (dynamic range) =4.0E8 //自动计算为。

->McuClockReferencePoint：

增加：

- Name=QSPI_AHB_CLK
- Mcu Clock Frequency Select =XBAR_CLK
- Mcu Clock Reference Point Frequency (0 -> 5000000000)= 4.0E8

如下 S32G3 芯片手册说明，本节点应该也可以不增加，Fls 驱动本身没有引用。

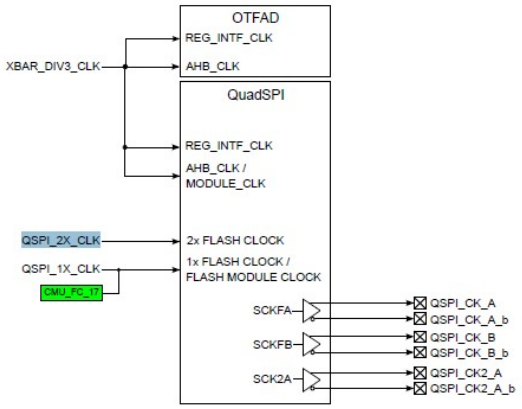


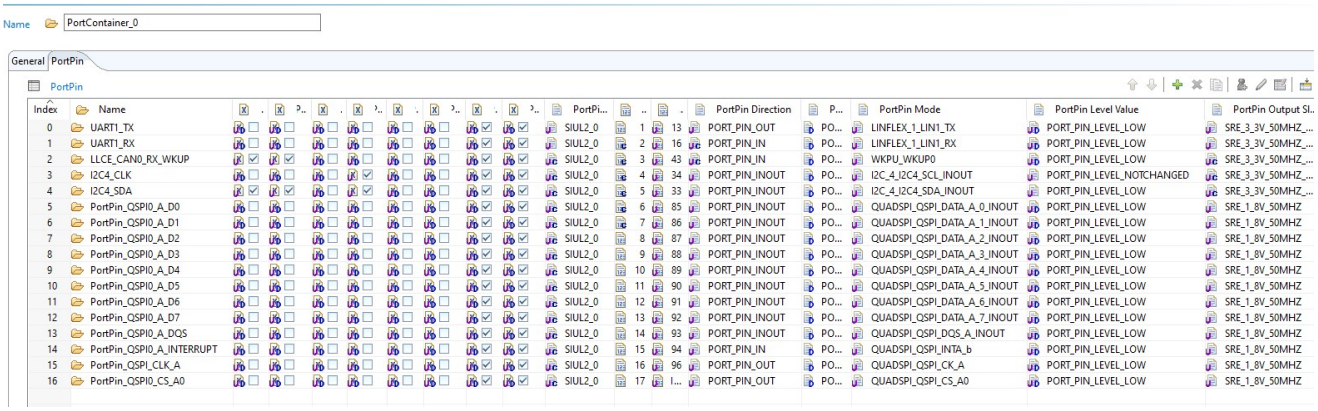
Figure 120. QuadSPI clocking

3. 修改 Port 配置：

为了批量添加 Port，将 C:\EB\tresos\workspace\Fls_Example_S32G399A_M7\config\port.xdm 中的管脚配置部分拷贝到 C:\EB\tresos\workspace\Uart_Example_S32G399A_M7\config\port.xdm 中，然后重启 EB tresos 来修改，主要是修改：

- PortPin Id
- PortNumberOfPortPins

基本都可以通过自动计算来修改，修改后的结果为：



Index	Name	PortPi...	PortPin Direction	PortPin Mode	PortPin Level Value	PortPin Output Sl..
0	UART1_TX	SIUL2_0	PORT_PIN_OUT	LINFLEX_1_LIN1_TX	PORT_PIN_LEVEL_LOW	SRE_3_3V_50MHZ_...
1	UART1_RX	SIUL2_0	PORT_PIN_IN	LINFLEX_1_LIN1_RX	PORT_PIN_LEVEL_LOW	SRE_3_3V_50MHZ_...
2	LLCE_CAN0_RX_WKUP	SIUL2_0	PORT_PIN_IN	WKPU_WKUP0	PORT_PIN_LEVEL_LOW	SRE_3_3V_50MHZ_...
3	I2C4_CLK	SIUL2_0	PORT_PIN_INOUT	I2C_4_I2C4_SCL_INOUT	PORT_PIN_LEVEL_NOTCHANGED	SRE_3_3V_50MHZ_...
4	I2C4_SDA	SIUL2_0	PORT_PIN_INOUT	I2C_4_I2C4_SDA_INOUT	PORT_PIN_LEVEL_LOW	SRE_3_3V_50MHZ_...
5	PortPin_QSPI0_A_D0	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_0_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
6	PortPin_QSPI0_A_D1	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_1_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
7	PortPin_QSPI0_A_D2	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_2_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
8	PortPin_QSPI0_A_D3	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_3_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
9	PortPin_QSPI0_A_D4	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_4_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
10	PortPin_QSPI0_A_D5	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_5_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
11	PortPin_QSPI0_A_D6	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_6_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
12	PortPin_QSPI0_A_D7	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DATA_A_7_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
13	PortPin_QSPI0_A_DQS	SIUL2_0	PORT_PIN_INOUT	QUADSPI_QSPI_DQS_A_INOUT	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
14	PortPin_QSPI0_A_INTERRUPT	SIUL2_0	PORT_PIN_IN	QUADSPI_QSPI_INTA_b	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
15	PortPin_QSPI_CLK_A	SIUL2_0	PORT_PIN_OUT	QUADSPI_QSPI_CK_A	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ
16	PortPin_QSPI0_CS_A0	SIUL2_0	PORT_PIN_OUT	QUADSPI_QSPI_CS_A0	PORT_PIN_LEVEL_LOW	SRE_1_8V_50MHZ

5.2 主测试函数与编译修改

生成代码后拷贝到 C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Uart_TS_T40D11M40I2R0\examples\EBT\S32G3\Uart_Example_S32G399A_M7\generate，按照前章说明修改 Fls 驱动相关配置源代码。

1. 修改编译文件：

   C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Uart_TS_T40D11M40I2R0\examples\EBT\S32G3\Uart_Example_S32G399A_M7\project_parameters.mk

   将 Fls MemIf 加入：

   ```
   MCAL_MODULE_LIST := BaseNXP Det Platform Mcu Port Resource EcuC Rte I2c Icu Pmic Dio EcuM Uart Fls MemIf
   ```

2. 修改主测试函数：

   C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Uart_TS_T40D11M40I2R0\examples\EBT\S32G3\Uart_Example_S32G399A_M7\src\main.c

   ```
   #include "Fls.h"
   …
   #define FLS_MSG "Stdby demo: put flash into deep power down\r\n"
   …
     /* Initialize the Mcu driver */
     Mcu_Init(NULL_PTR);


     /* Initialize the clock tree and apply PLL as system clock */
     Mcu_InitClock(McuClockSettingConfig_0);
   while ( MCU_PLL_LOCKED != Mcu_GetPllStatus() )
   ```

```
  {
      /* Busy wait until the System PLL is locked */
  }
  /* Distribute the PLL across the clock tree */
  Mcu_DistributePllClock(); //注意 Mcu_InitClock 后这两句调用必须要，实际测试中发现没有会影响 QSPI autoupdate 模式的锁定。
…
  /* Initialize Fls driver */
  Fls_Init(NULL_PTR);
…
(void)Uart_SyncSend(UART_CHANNEL, (const uint8 *)FLS_MSG, strlen(ENTER_MSG), 10000);
Fls_SetMode(MEMIF_MODE_DEEP_PWRDWN); //johnli for deep power down mode test
```

5.3 运行测试

运行测试方法参考 4.5 节，如果需要使用 Lauterbach 跟踪 main.bin 的执行情况，可以修改：

C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\cmm\connect_s32g3xx_m7.cmm 为：

```
sys.cpu S32G399A-M7-0
SYStem.config.debugporttype JTAG
SYStem.Option TRST OFF
SYStem.Option DisMode THUMB
sys.attach

Data.Load.Elf C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Uart_TS_T40D11M40I2R0\examples\EBT\S32G3\Uart_Example_S32G399A_M7\out\main.elf  /GLOBTYPES /NoCode
break
list
ENDDO
```

然后 main.c 修改为：

```
volatile int debug;
int main(void)
{
debug =1;
while(debug);
```

则代码从QSPI NOR启动后就会停在while 处，这个时候用以上脚本连接上lauterbach,手工将 debug 改为0，就可以继续运行调试。注意Jtag在进入Standby前需要断掉。