S32G M7 Standby Demo 加 A53 Suspend to RAM Demo 搭建说明

by John Li

本文说明如何在S32G2 RDB2板上搭建一个M7 MCAL Standby Fullboot GPIO resume Demo加A53 Suspend to RAM的Demo，主要的应用场景是电动汽车的快速启动。

G3与更新版本BSP的支持情况与此类似，不再另外说明，客户可以自行参考开发。

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 参考资料说明与声明

分类	名称	说明	备注
文档	S32G_Standby_Demo_V4-*.pdf	S32G M7 Standby Demo 说明文档	从 community 下载 https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-M-kernel-Standby-demo-and-how-to-porting-to-Mcal /ta-p/1556313
文档	S32G_Linux_STR_V1-*.pdf	S32G2 Linux STR 说明文档	从 community 下载 https://community.nxp.com /t5/NXP-Designs-Knowledge-Base/ S32G-Linux-STR/ta-p/1652680
资料	S32G2_LinuxBSP_32.0*	S32G2 Linux 开发包，注意 User Manual 为 BSP 详细说明，重要。	Download from www.nxp.com
资料	SW32G_RTD_4.4_3.0.2_D2203.exe	MCAL 安装包	Download from www.nxp.com
文档	S32G_Bootloader_V1-*.pdf	Bootloader 说明文档	从 community 下载 https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ ta-p/1519838

说明：

汽车网关有快速启动要求，而电动车因为驻车时有更大的电池提供待机电源，所以希望是使用 Linux 的 suspend to ram 的功能来实现 Linux 的快速启动，而在 S32G 上则需要考虑将 M 核的 Standby 功能 与 A 核的 STR 功能 结合起来，目前可用的资源包括：

- 从 BSP32 起支持 ATF，可以支持 Linux 端的 STR 功能，文档《S32G_Linux_STR_V1-*.pdf》(John.Li)说明 linux STR 的原理和与 M7 Standby Demo 结合时所需要的修改。

- NXP 的 M7 内部 standby demo，可以支持 M 核端的 standby 功能，支持 full boot 和 standby ram boot。文档《S32G_Standby_Demo_V4-*.pdf》(John.Li)有详细说明，本文使用 MCAL full boot+GPIO resume Demo。
- 本 Demo 与本文主要说明如何将这两个 Demo 结合起来，形成一个整体的 Demo。
- 由于需要 Boot M 核加 A 核，所以也需要 Bootloader 工程的支持，文档《S32G_Bootloader_V1-*.pdf》(John.Li)说明了如何创建一个 MCAL sample 加 Linux 的 Bootloader 工程。

声明： 请注意：

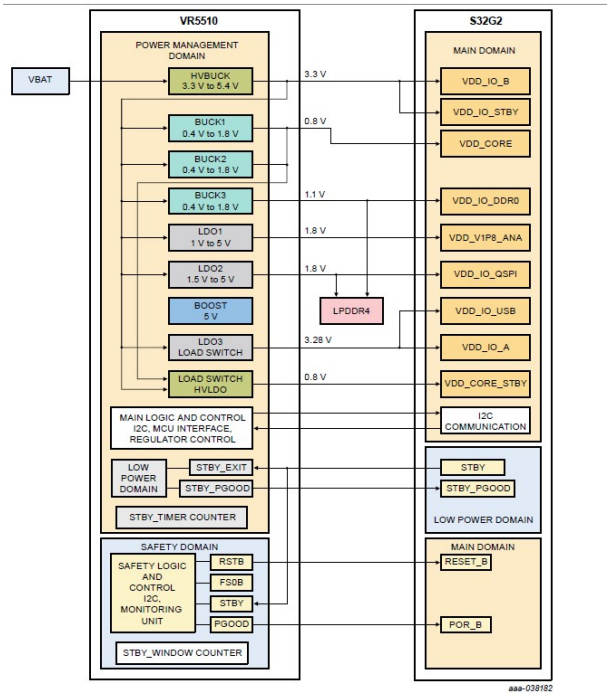
- **M7 standby demo 本来为 NXP 内部 Demo，不保证运行质量。而 Linux 本身也是 reference software。**
- **Linux STR 本身会引入比较复杂的电源管理切换，也会引起系统级的不稳定性。**
- **本文所说的方法也是实验性质，不保证运行质量。**

所以客户应该谨慎决定其产品功能并自行保证其产品质量，本文及本 Demo 仅为 Demo 性质。

2 STBY+STR 的硬件注意点

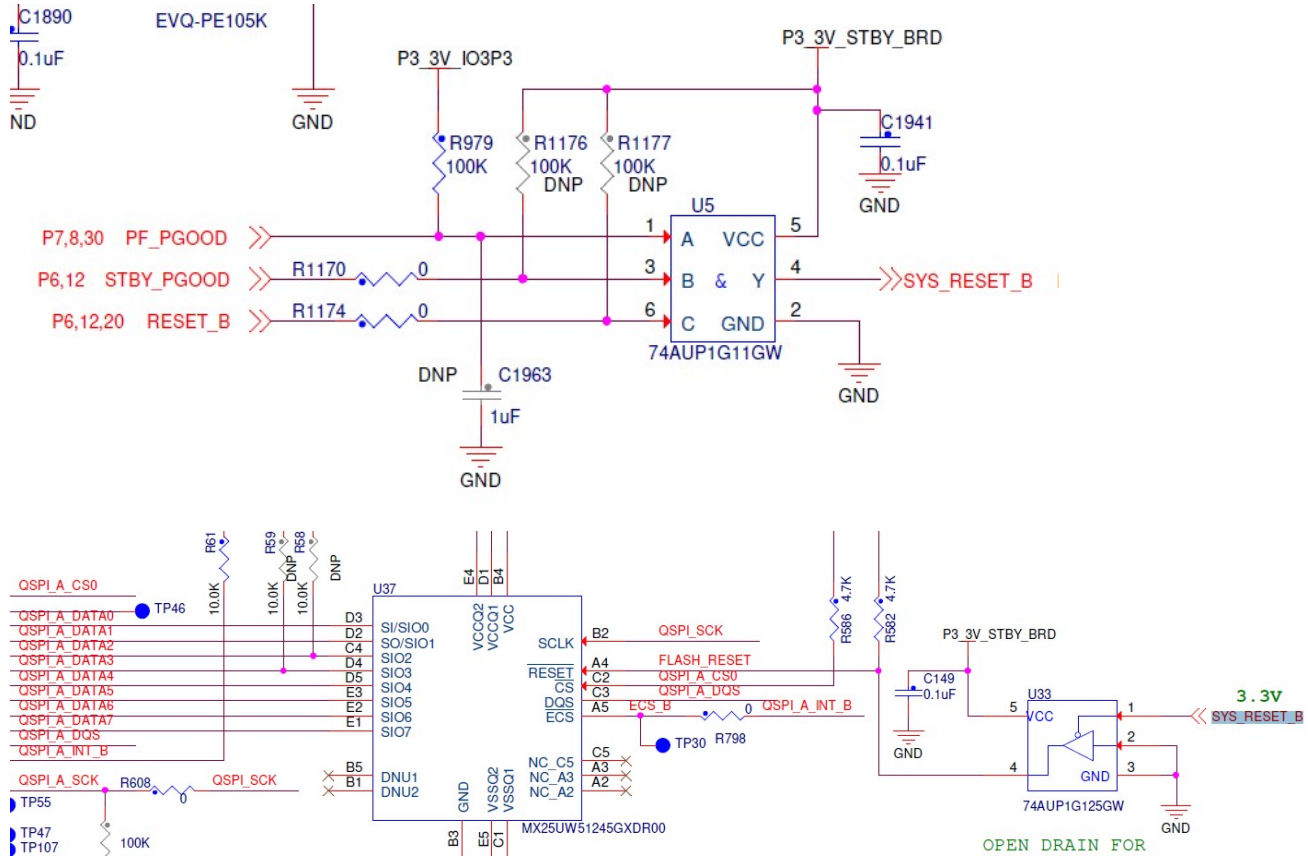
为了支持 STBY+STR，设计硬件时需要注意：

- VR5510 与 S32G 的连接图如下：



S32G 通过 STBY 管脚拉低通知 VR5510 进入 STBY，VR5510 拉低 STBY_GOOD，通知 S32G 或其它外设进入了 STBY。

- 在 VR5510 进入 STBY 时，通常情况下会保持 VDD_IO_B/VDD_IO_STBY(3.3V)电源，VDD_CORE_STBY(0.8V)电源，在支持 Linux STR 时还会保持住 VDD_IO_DDR0(1.1V)电源，并在进入 STBY 时，有所降压。其它电源关闭。
- 基于以上说明，对属于 VDD_IO_B 的管脚，外接上拉不影响进入 STBY，因为此电源在 STBY 时仍然供电。但属于 VDD_IO_A 的管脚，其电源在进入 STBY 是会关闭，所以如果这些管脚有外接的上拉电源，会反灌导致 VDD_IO_A 上有浮动电压，此时 PMIC 时入 STBY 会失败，须注意。
- 对 QSPI NOR 的 reset 设计， RDB 板如下：



所以在 STBY 退出后，PMIC 上电，然后拉动 STBY_PGOOD 通知外设退出 STBY，此时会 reset 外部 QSPI NOR，这样保证 STBY 退出时，M 核做 full boot 的时候，QSPI NOR 会 reset 到初始化状态，保证启动成功。

3 修改 M7 MCAL Standby Demo 代码

Bootloader 工程中默认使用的 MCAL Sample 是 UART，而 Standby Demo 本身也是基于 UART 工程做出来的，所以需要将两者做一下结合。可以在 Bootloader 已经支持的 UART 工程上将 Standby Demo 需要的部分增加上去。本文是采用相反的方法将 Standby Demo 的 UART 工程修改 Bootloader UART 工程需要修改的部分，注意参考 Bootloader 文档了解相关修改，比如 Memory Mapping 部分等。

M 核的 Mcal Standby Demo 的修改思路为：将 clock 和 port 相关初始化移除掉，I2C Timing 重配置，以及 Main 函数主逻辑实现。

3.1 Clock 相关修改

在 Main 函数中将 MCU clock 初始化注掉：

/* Initialize the clock tree and apply PLL as system clock */

// Mcu_InitClock(McuClockSettingConfig_0);

将 I2C Clock 和源配置为与 Bootloader 中相同的设置，以提供 I2C 模块正确的引用：

Uart_Example_*->someid->Mcu(*)->Mcu->McuClockSettingConfig->

McuClockSettingConfig_0:

McuCgm0ClockMux0:

- CGM0 Clock Mux0 Source*= CORE_PLL_DFS1_CLK
- Clock Mux0 Frequency (XBAR_2X_CLK) (dynamic range)*= 8.0E8//自动计算为

  McuClockReferencePoint，点击+号增加：
- Name= I2C_CLK
- Mcu Clock Frequency Select= XBAR_DIV3_CLK
- Mcu Clock Reference Point Frequency (0 -> 5000000000) = 1.3333333333333333E8//自动计算为

注意：调节时钟后相关自动生成 clock 的错误点击自动修改解决。

3.2 MCU 相关修改

如上所述，Mcu_Init 函数被注释掉了，则需要实现一个简单函数来实现 MCU 配置变量的初始化，如下：

mcu_ts_*\src\mcu.c

```
//johnli
void Mcu_Init_Simple(const Mcu_ConfigType * ConfigPtr)
{
uint32 NoConfigs;
  Std_ReturnType CheckStatus;
  CheckStatus = (Std_ReturnType) Mcu_HLDChecksEntry(MCU_INIT_ID);
        Mcu_pConfigPtr = &Mcu_PreCompileConfig;
        MCU_PARAM_UNUSED(ConfigPtr);
/* Save the Mcu Mode IDs configurations. */
        for(NoConfigs = (uint32)0U; NoConfigs < Mcu_pConfigPtr->NoModeConfigs; NoConfigs++)
        {
          Mcu_au8ModeConfigIds[(*Mcu_pConfigPtr->ModeConfigArrayPtr)[NoConfigs].ModeConfigId] =
(uint8)NoConfigs;
        }
        /* Save the Mcu Clock IDs configurations. */
        for(NoConfigs = (uint32)0U; NoConfigs < Mcu_pConfigPtr->NoClkConfigs; NoConfigs++)
        {
          Mcu_au8ClockConfigIds[(*Mcu_pConfigPtr->ClockConfigArrayPtr)[NoConfigs].ClkConfigId] =
(uint8)NoConfigs;
        }
        Mcu_HLDChecksExit(CheckStatus, MCU_INIT_ID);
}
//end
```

mcu_ts_*\include\mcu.h

```
void Mcu_Init_Simple(const Mcu_ConfigType * ConfigPtr);  //johnli
```

然后主函数调用 Mcu_Init_Simple 来实现。

注意：

MCU 模块的 mode 设置功能中，Uart_Example_S32G274M_M7_STDY->someid()->Mcu()->Mcu->McuModeSettingConf->McuModeSettingConf_1->McuPartition1Config，勾选了：

- Cortex-A53 CORE 0 cluster 0 Under MCU Control=checked

是希望可以控制 A53_0 核的开关，但是实际生成代码这个值为 FALSE，所以需要手动修改：

3.3 UART Clock 相关修改

由于需要结合 Standby Demo 和 Bootloader Demo，则 Clock 树有所变化，参考文档《S32G_Bootloader_V1-2022.0909.pdf》说明，将 UART CLOCK TREE 修改一下，以满足提高 UART 源时钟的要求。

另外：

Uart_Example_S32G274M_M7_STDY->someid()->Uart()->Uart->Gerneal:

- Uart Timeout Method= OSIF_COUNTER_DUMMY
- Uart Timeout Duration (0 -> 4294967295)= 100000 //由于 M 核时钟提升到 400Mhz，在使用 dummy timeout 方式的情况下建议修改提高 Timeout duration 至少 10 倍。

同理如下源代码：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\src\ main.c

```
(void)Uart_SyncSend(UART_CHANNEL, (const uint8 *)STBY_MSG, strlen(STBY_MSG), 100000);
 (void)Uart_SyncSend(UART_CHANNEL, (const uint8 *)ENTER_MSG, strlen(ENTER_MSG), 100000); //此两处的同步 发送函数的超时值建议升高 10 倍
```

3.4 Port 相关修改

在 Main 函数中将 PORT 初始化注掉：

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

3.5 I2C 相关修改

因为I2C的根时钟XBAR_DIV3_CLK由之前的8M变成了133Mhz，所以I2C的SCL Divider应该由之前的80M/400K=200变成133M/400K=333。通过调节：

Uart_Example_*->someid->I2c(*)->I2c->I2cChannel-> I2cChannel_0->Master Configuration:

- I2cClockRef= /Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig_0/I2C_CLK //XBAR_DIV3_CLK=133Mhz
- I2c Prescaled Shift (0 -> 2) =2
- I2c Prescaler Divider (0 -> 7) =2
- I2c Shift Tap Point (0 -> 7)=5

通过以上值调节得到 I2C IBFD 寄存器值，从而自动配置 SCL 为接近并小于 400K 的一个值：

- I2c SCL Divider (cycles) (20 -> 15360)= 352 //自动计算为，>333 的某一个值

- I2c SDA Hold Delay (cycles) (7 -> 2052)= 68 //自动计算为
- I2c Hold Start Delay (cycles) (6 -> 7672)= 152 //自动计算为
- I2c Hold Stop Delay (cycles) (11 -> 7684)= 180 //自动计算为
- I2c Baud Rate (0 -> 1000000) =378787.8787878788 //自动计算为 <400K 的某一个值，注意一般 I2C 设备都支持 400K 的速率，所以此处设置为接近并小于 400K 的一个值，但是某些设备可以支持 400K 以上的速率，可以通过提高速率来加速启动时间。

以上 IBFD 与 timing 的对应关系可以参考芯片手册的“Table ⋆. I2C divider and hold values when glitch filter is disabled”，注意其值与 EB IDE 不尽相同，以 EB 为准。

3.6 实现 M 核进入 STDY 状态等待功能

本 Demo 使用串口输入来控制是否让 M 核进入 STDY，所以需要 A 核先执行 STR 命令后，再输入确认：

所以使用串口输入命令功能如下：

```
uint8 Rx_Buffer[1];
while (1)
  {
     memset(Rx_Buffer, 0 , 1);
     (void)Uart_SyncReceive(UART_CHANNEL, Rx_Buffer, 1, 10000);
      if(Rx_Buffer[0] == 'y')
     {
        break;
     }
  }
```

3.7 Main 函数的修改

根据文档《S32G_Bootloader_V1-*.pdf》(John.Li)和《S32G_Standby_Demo_V4-*.pdf》(John.Li)说明，最终将 standby UART 工程 main 函数修改为：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\src\main.c

```
#define STBY_MSG "Stdby demo: Full Boot,pull low of LLCE CAN0 RX to resume\r\n"
#define ENTER_MSG "Stdby demo: Enter Stdby\r\n"
…
```

```
int main(void)
{
  volatile Uart_StatusType Uart_Status;
  volatile Std_ReturnType Std_Uart_Status;
  uint8 Rx_Buffer[1];
  Std_Uart_Status = E_NOT_OK;

 Mcu_Init_Simple(NULL_PTR);
    /* Initialize I2c driver */
    I2c_Init(NULL_PTR);

    /* Initialize Pmic driver */
    Pmic_Init(NULL_PTR);
     /* Initialize Vr5510 device */
     Pmic_InitDevice(PmicConf_PmicDevice_PmicDevice_0);

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

 /* Initialize IRQs with api */
Platform_InstallIrqHandler(WKPU_GRP_IRQn, & WKPU_EXT_IRQ_SINGLE_ISR, NULL_PTR);
              Platform_SetIrq(WKPU_GRP_IRQn, TRUE);

 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);
      Pmic_InitClock(PmicConf_PmicDevice_PmicDevice_0,0);

      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_SetMode(PmicConf_PmicDevice_PmicDevice_0,1);
    Mcu_ResetClockConfiguration(McuClockSettingConfig_0);
    Mcu_SetMode(McuModeSettingConf_1);

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

4 修改 Bootloader 工程来支持同时 Boot M/A 核 Demo

根据文档《S32G_Bootloader_V1-*.pdf》说明，先配置出一个可以同时启动 A 核 Linux BSP32+M 核 UART MCAL sample 的 Bootloader 工程，验证可以运行。

然后根据文档《S32G_Standby_Demo_V4-*.pdf》(John.Li)说明，将 MCAL Standby Demo 中的 clock, port 相关初始化移值到 Bootloader 中。

4.1 I2C Clock 相关修改

根据文档《S32G_Bootloader_V1-*.pdf》说明，针对 UART MCAL sample 和 Linux 配置完时钟后，还需要考虑 Mcal standby demo 的时钟配置。参考文档《S32G_Standby_Demo_V4-*.pdf》(John.Li)说明，增加 I2C 相关配置：

在 Bootloader…->EcuC(…)->Mcu(…)->Mcu->McuClockSettingConfig-> McuClockSettingConfig_0->McuClockReferencePoint 中，点击+号增加一项，进入：

- Name= I2C_CLK
- Mcu Clock Frequency Select= XBAR_DIV3_CLK
- Mcu Clock Reference Point Frequency= 1.3E8//自动计算

注意：调节时钟后相关自动生成 clock 的错误点击自动修改解决。

4.2 Port 相关修改

Bootloader…->EcuC(…)->Port(…) -> Port->PortContainer-> PortContainer_0->General : PortNumberOfPortPins=+3

-> PortPin：点击+号增加一个管脚，点击进入：

- Name= I2C4_CLK
- PortPin Ode* =checked//
- PortPin Id=44
- PortPin Mscr (dynamic range) =34
- PortPin Direction = PORT_PIN_INOUT
- PortPin Mode= I2C_4_I2C4_SCL_INOUT
- PortPin Level Value = PORT_PIN_LEVEL_NOTCHANGE
- PortPin Output Slew Rate= SRE_3_3V_50MHZ

同理增加 SDA

- Name= I2C4_SDA
- PortPin Pull Enable /PortPin Pull Select =checked// pull enable，注意需要上拉，不然会导致 I2C 连接失败
- PortPin Ode* =checked
- PortPin Id=45
- 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

同理增加 WKUP0：

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

- Name= LLCE_CAN0_RX_WKUP
- PortPin Pull Enable=checked// pull enable
- PortPin Pull Select =checked// pull up
- PortPin Mode Changeable=checked//default
- PortPin SIUL2 Instance = SIUL2_0
- PortPin Id=46

- 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

然后在 PortContainer_0->General 里自动调节：

- PortNumberOfPortPins=46

4.3 其它修改

如下将 M 核部分修改为支持 M 核 standby Demo：

主要修改：配置 M7 Boot sources 部分：

打开 Bootloader(…)->Bootloader->Boot Sources->BootSources_M7_LightingApp:

修改名字：Name= BootSources_Standby Demo。

参考链接文件：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Platform_TS_T40D11M30I2R0\build_files\gcc\linker_ram.ld

```
int_sram              : ORIGIN = 0x34400000, LENGTH = 0x00080000 /* 4MB offset, 512KB size */
```

参考编译生成Mapping文件：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\out\main.map

```
.startup 0x34400010 0x1d0 tmp/startup_cm7.o
0x34400010 Reset_Handler
0x34400010 _start
```

得到 bin 文件大小为：1280Kbytes=1,310,720。

综上所述：

1. BootSources_Standby Demo ->General-> Boot souce=QSPI //保持不变。
2. BootSources_Standby Demo ->General ->Reset handler address =0x34400010
3. BootSources_Standby Demo ->Boot image fragments->ImageFragments_0->Load image at address (RAM)= 0x34400000
4. BootSources_Standby Demo ->Boot image fragments->ImageFragments_0-> Image size(bytes)= 1314816 > 1,310,720
5. BootSources_Standby Demo ->Boot image fragments->ImageFragments_0-> Image CRC value=0x0
6. 由于修改了名字，在Bootloader(…)->Bootloader->Core Configuration:在A53_0_BSP_ATF 下修改或增加一项，名称改为：M7_Standby_Demo, Core ID 修改为：M7_0，Boot source configuraiton 指向/Bootloader/Bootloader/ BootSources_Standby Demo。

5 修改 A53 Linux 代码

A 核的 Standby Demo 的修改思路为：

- 将 A53_0 核作为主核关闭的代码去掉，也不要去关闭 M 核。
- 不需要去关闭时钟，时钟由 M 核来关闭。
- 不需要访问 PMIC，PMIC 驱动在 M 核这边。
- 唤醒相关控制也由 M 核负责

根据文档《S32G_Linux_STR_V1 -*.pdf》说明，来修改 Linux 代码，主要是在 ATF 中的修改。参考文档第 5 章：定制修改。

6 Demo 运行测试

6.1 硬件连接

- 12V 电源连接到 J176 上，SW15 为电源开关
- 连接 USB 线从 UART0 J2 到电脑上，作为烧写线与 Linux 的调试串口 115200 8n1
- 连接 USB 线从 UART1 J1 到电脑上，作为 M7 核的调试串口 115200 8n1
- SW9=OFFOFF，SW10=OFFOFF 为下载模式，用于镜像烧写，SW10_1=ON 为正常启动模式
- Bootmode 全 OFF 表示从 QSPI NOR 启动

6.2 镜像烧写

切换成下载模式，上电，根据《S32G_Bootloader_V1-*.pdf》说明烧写操作好的 bootloader 带 IVT 头的镜像，fip.bin 和 mcal sample。

运行C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\GUI\s32ft.exe，Target 选择S32G2xxx，Algorithm 选择MX25UW51245G。

COM 口的port names:中设置为设备管理器中看到的串口：COM22(注意要为串口0对应PC串口号)。

然后点击Upload target and algorthim to hardware…，烧写工具就会加载算法镜像并配置烧写设备：

```
Configuring target
Progress: 100
Flash algo is loaded.
```

Device: Macronix MX25UW51245G
Capacity: 64 MiB (67108864 bytes)

之后再点击Erase memory range…，选择0x0-0x500000

- 使用flash tools 烧写bootloader 镜像到QSPI 中：

  点击 Upload file to device…，将“bt_blob.bin”烧写到地址 0x0 处。

- 使用flash tools 烧写A53 fip.bin 到QSPI 中：

  点击Upload file to device…，将“fip.bin”烧写到地址0x100000 处，烧写地址参考之前Bootloader MCAL 配置的QSPI source address，烧写时注意是烧写fip.bin 文件，这个是不带IVT 头的A53 Bootloader fip.s32。

- 使用flash tools 烧写M7_0 的main.bin 到QSPI 中：

  点击Upload file to device…，将“main.bin”烧写到地址0x200000 处，这个是M7 standby Demo 测试镜像。

- 烧写A53 Linux 镜像到SDcard 中：

  根据文档《S32G_Kernel_BSP32_V4-20220513.pdf》，说明，使用SDcard 读卡器在Ubuntu 中将整个镜像烧写到TFcard 中：

```
sudo dd if= fsl-image-base-s32g274ardb2.sdcard of=/dev/sd<partition> bs=1M conv=fsync
```

并更新一下修改后的fip.s32：

```
sudo dd if=<path/to/fip.s32> of=/dev/sdc bs=512 skip=1 seek=1 conv=fsync,notrunc
```

fsl-s32g274a-rdb2.dtb 可以直接在 windows 环境拷贝到 TFcard 的 FAT 分区里。

然后将 TFcard 插入到 RDB2 板上的 J3 TFcard 插槽内，并将 SW3=On，切换成 TFcard 启动。

6.3 Demo 运行

切换成正常模式，然后在 PC 上打开两个串口：115200-8n1，上电启动 M 核串口 1 打印如下：

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

然后在 Linux 终端串口 0 执行命令：

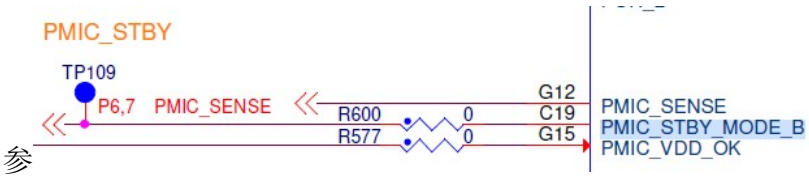
```
echo disabled > /sys/devices/platform/40060000.rtc/power/wakeup
echo mem > /sys/power/state
```

然后在 M7 串口上输入 y。

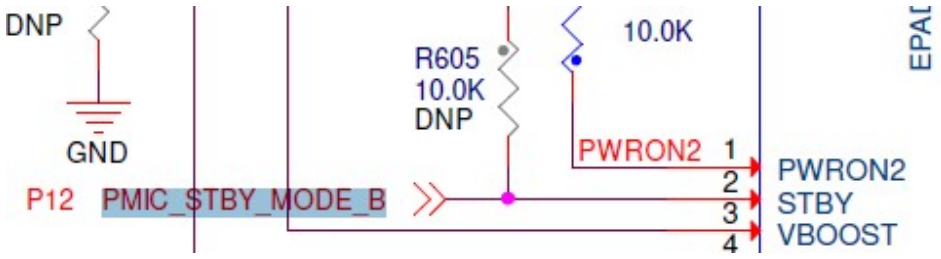
打印为：

```
Stdby demo: Enter Stdby
```

是否进入 Standby 模式的测试点如下：

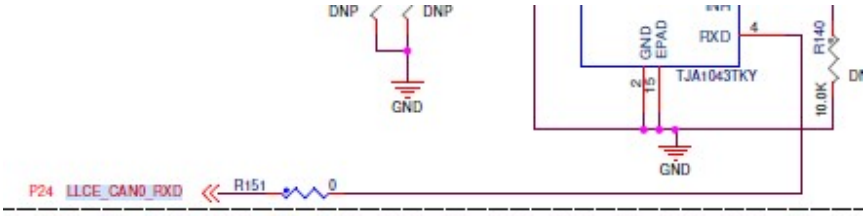


参



S32G 通过拉动 PMIC_STBY_MODE_B 管脚来通知 PMIC 进入或退出 Standby，所以可以通过量测 TP109 的电平变化来看 S32G 是否进入了 Standby 状态，此管脚电平切换是一个硬件行为。

根据文档《S32G_Standby_Demo_V*-*.pdf》说明，将 CAN0_RX 管脚 R151 对地轻触，来触发中断，退出 Standby 模式：



M 核串口 1 再次打印：

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

A 核串口 0 再次打印

```
[   23.765016] fsl_fccu 4030c000.fccu: FCCU status is 0 (normal)
…
[   26.101480] OOM killer enabled.
[   26.101484] Restarting tasks ... done.
[   26.113346] PM: suspend exit
```

7 工程发布包

Release

|->Linux

| |->atf

| | |->atf_str.patch, fip.bin, fip.s32

| |->uboot

| | |->uboot_str.patch

| |->kernel

| | |->kernel-str.patch， fsl-s32g274a-rdb2.dtb

|->mcal

| |->EB

| | |-> Tresos

| | | |->generate //Uart sample 修改的 stdy demo 的生成文件

| | | |->workspace

| | | | |->Bootloader_S32G2XX_ASR_4.4_M7 //bootloader 工程文件

| | | | |->Uart_Example_S32G274A_M7_STDY // Uart sample 修改的 stdy demo 的工程文件

| |->NXP

| | |->Integration_Reference_Examples_S32G2_2022_06. Code. Framework. Realtime. Swc. Bootloader. Platforms. S32G2XX. Build //bootloader 的编译目录

| | | |->bin_bootloader //编译生成目录

| | | | |->Bootloader.bin/elf/map //生成的 bootloader 镜像

| | | |->cmm //调试的 lauterbach 脚本

| | | |->launch.bat //编译配置文件

| | |->SW32G_RTD_4.4_3.0.2. eclipse. Plugins //stdy demo 目录

| | | |->Icu_TS_T40D11M30I2R0 //ICE 模块修改

| | | |->Mcu_TS_T40D11M30I2R0 //MCU 模块修改

| | | |->Pmic_TS_T40D11M30I2R0 //PMIC 模块修改

| | | |->Uart_TS_T40D11M30I2R0 //基于 UART sample 的 standby demo 主目录

| | | |->Platform_TS_T40D11M30I2R0 //链接文件

|->test_binary

| |->bt_blob.bin //bootloader 镜像，用 IVT 工具打上了 IVT 头

| |->fip.bin/s32 //atf 镜像

| |->fsl-s32g274a-rdb2.dtb //内核 dtb

| |->main.bin/elf/map //standby demo 镜像

8 未来开发建议

8.1 M/A 核同步机制

M/A 在 Standby 时的同步机制需要客户自行开发，如果是在上层实现，可以考虑如 IPCF 驱动之类的同步办法，本文在进入 STDBY&STR 时是采用轮流在 M/A 串口输入命令的办法，只为 Demo 所用。

如果是要在底层实现，因为 Standby 时会将所有驱动 suspend 掉，所以不能考虑用需要驱动支持的办法，可以考虑在 standby SRAM 中设置标志和轮询标志之类的办法。

8.2 功能安全与信息安全

由于 Linux STR 本身会引入的不稳定性，建议 M 核端要考虑 A 核端退出 STR 时不成功的可能性，可以考虑通过 M 核与 A 核的心跳机制(通过 IPCF 或者网口)来判断 A 核是否退出 STR 成功，如果不成功时要做重新引导。

由于本 Demo 使用 M7 standby demo Full boot 的流程，所以 secure boot 可以在 full boot 里进行，但是如果是使用 standby sram short boot 的办法，则要考虑 standby sram 中镜像的签名与验签。

9 遗留问题

参考 Linux BSP 的 release notes 可以看到 Linux 支持 STR 的驱动列表，如下示例：《S32G2_LinuxBSP_32.0_Release_Notes.pdf》。

– Core Linux Suspend-to-RAM (STR)

– STR for the following Linux drivers: Real-Time Clock (RTC), LinFLexD UART, uSDHC

– Add STR features to BSP drivers for QSPI, FlexCAN, PIT, STM timers, FCCU, TM drivers ADC, SerDes, PFE,RTC, I2C, GPIO, PinCtr, Clks, LPDDR4, eMMC, SDHC DSPI and GMAC (with RGMII interface only).

– Add SerDes driver with STR capabilities

STR	Linux	S32GEN1	After three or more resets from U-Boot console, a rtcwake command issued from Linux will not suspend with the specified time.	N/A	ALB-8238

9.1 IPCF STR 支持

截止目前的 Linux BSP 发布(2023/5/4)，没有宣称 IPCF 的 Linux 驱动支持 STR，不过 IPCF 驱动相对简单，对硬件依赖小，所以是否在 STR 时会遇到问题需要实测，或者等待 NXP 确认 IPCF 的 STR 支持。

9.2 PFE Slave STR 支持

截止目前的 Linux BSP 发布(2023/5/4)，PFE 的 Linux Slave 驱动支持的 STR，还不满足实际情况的需求，需要对其中的 suspend/resume 两个 STR 依赖函数进行修改，或者等待 NXP 的修复确认。