S32G M7 Standby Demo combined A53 Suspend to RAM Demo Setup Guide

by John Li (nxa08200)

This article explains how to build a demo of M7 MCAL Standby Fullboot GPIO resume Demo plus A53 Suspend to RAM on the S32G2 RDB2 board. The main application scenario is the quick start of electric vehicles.

The support situation of G3 and the newer version of BSP is similar to this, no further explanation is given, customers can refer to it for development by themselves.

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	● Create the doc	John.Li
V2	● Translate to Eng	John.Li

Contents

1 Reference materials and statement 2
2 STBY+STR hardware checkpoints 3
3 Modified M7 MCAL Standby Demo codes 5
3.1 Clock modification 5
3.2 MCU related modification 6
3.3 UART Clock related modificaiton 7
3.4 Port related modification 8
3.5 I2C related modification 8
3.6 Enable the waiting function of M core entering STDY 9
3.7 Main function modification 9
4 Modify the Bootloader project to support simultaneous M/A core demo 11
4.1 I2C Clock related modification 11
4.2 Port related modifcaiton 11
4.3 Others modificaiton 13
5 Modify A53 Linux codes 14
6 Demo running and testing 14
6.1 Hardware link 14
6.2 Image burning 14
6.3 Demo running 15
7 Project release package 16
8 Suggestion for the future development 17
8.1 M/A core sync mechanism 17
8.2 Function safety and Information security 17
9 Remaining issues 18
9.1 IPCF STR support 18
9.2 PFE Slave STR support 18

1 Reference materials and statement

Class.	name	Comments	Remark
doc	S32G_Standby_Demo_V4-*.pdf	S32G M7 Standby Demo app doc	Download from community: https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-M-kernel-Standby-demo-and-how-to -porting-to-Mcal /ta-p/1556313
doc	S32G_Linux_STR_V1-*.pdf	S32G2 Linux STR app doc	Download from community: https://community.nxp.com /t5/NXP-Designs-Knowledge-Base/ S32G-Linux-STR/ta-p/1652680
materials	S32G2_LinuxBSP_32.0*	S32G2 Linux development kit, note that the User Manual is a detailed description of the BSP, which is important.	Download from www.nxp.com
materials	SW32G_RTD_4.4_3.0.2_D2203.exe	MCAL install package	Download from www.nxp.com
doc	S32G_Bootloader_V1-*.pdf	Bootloader app doc	Download from community: https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ ta-p/1519838

Notice:

The car gateway has a fast boot requirement, and the electric car has a larger battery to provide standby power when parking, so it is hoped to use the suspend to ram function of Linux to achieve a quick boot of Linux, and on the S32G, it is necessary to consider the M Standby function of the core is combined with the STR function of the A core. Currently available resources include:

- Since BSP32 supports ATF, it can support the STR function on the Linux side. The document <<S32G_Linux_STR_V1-*.pdf>> (John.Li) explains the principle of linux STR and the modification required when it is combined with M7 Standby Demo.
- NXP's M7 internal standby demo can support the standby function of the M core, and support full boot and standby ram boot. The document <<S32G_Standby_Demo_V4-*.pdf>> (John.Li) has detailed descriptions and how to rewrite it with Mcal standard. This article uses MCAL full boot+GPIO resume Demo.
- This demo and this article mainly explain how to combine the two demos to form a whole demo.
- Since Boot M core and A core are needed, the support of Bootloader project is also required. The document <<S32G_Bootloader_V1-*.pdf>> (John.Li) explains how to create a Bootloader project with MCAL sample plus Linux. Disclaimer:

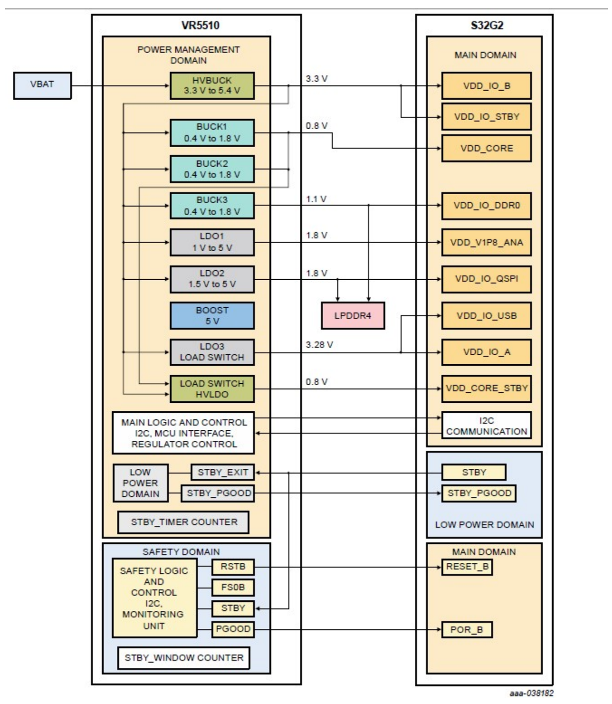
Statement:

- **The M7 standby demo is originally an internal demo of NXP, and the running quality is not guaranteed. And Linux itself is also reference software.**
- **Linux STR itself will introduce more complex power management switching, which will also cause system-level instability.**
- **The methods described in this article are also experimental in nature, and the running quality is not guaranteed. Therefore, customers should carefully determine their product functions and ensure their product quality. This article and this Demo are only Demos.**

2 STBY+STR hardware checkpoints

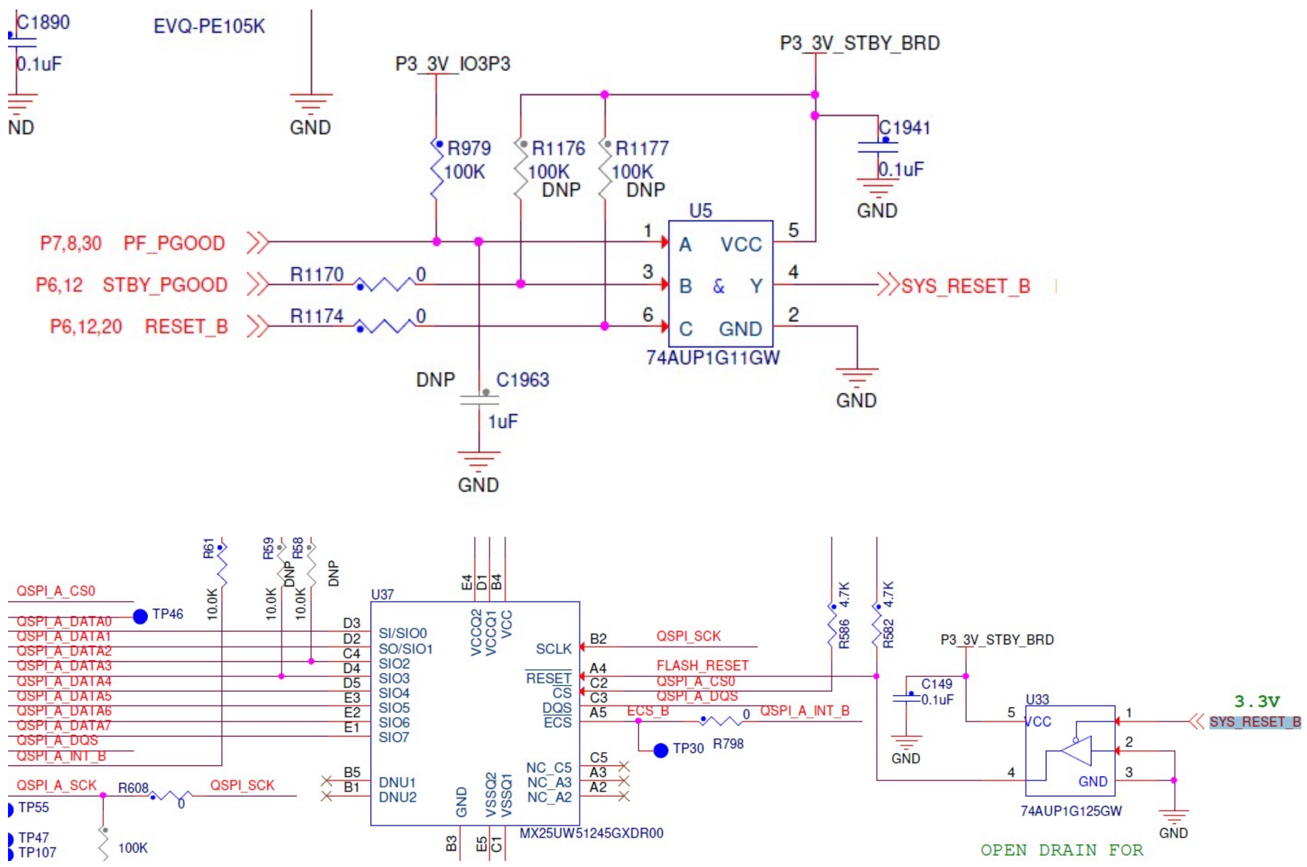
In order to support STBY+STR, you need to pay attention when designing hardware:

- VR5510 and S32G reference link as follows:



S32G notifies VR5510 to enter STBY by pulling STBY pin low, and VR5510 pulls STBY_GOOD low to notify S32G or other peripherals to enter STBY.

- When VR5510 enters STBY, it usually keeps VDD_IO_B/VDD_IO_STBY (3.3V) power supply, VDD_CORE_STBY (0.8V) power supply, and also keeps VDD_IO_DDR0 (1.1V) power supply when it supports Linux STR, and when entering STBY, Reduced voltage a little. Other power sources are off.
- Based on the above description, for the pins belonging to VDD_IO_B, the external pull-up does not affect the access to STBY, because this power supply still supplies power in STBY. However, the power of the pins belonging to VDD_IO_A will be turned off when entering STBY, so if these pins have an external pull-up power supply, it will backfeed and cause a floating voltage on VDD_IO_A. At this time, the PMIC will fail to enter STBY, so pay attention.
- For the reset design of QSPI NOR, the RDB board is as follows:



Therefore, after STBY exits, PMIC is powered on, and then pulls STBY_PGOOD to notify the peripherals to exit STBY. At this time, the external QSPI NOR will be reset, so that when STBY exits, when the M core is doing full boot, QSPI NOR will reset to the initialization state, ensuring bootup successfully.

3 Modified M7 MCAL Standby Demo codes

The default MCAL Sample used in the Bootloader project is UART, and the Standby Demo itself is also based on the UART project, so it is necessary to combine the two. The part required by Standby Demo can be added to the UART project already supported by Bootloader. This article uses the opposite method to modify the UART project of the Standby Demo to the parts that need to be modified in the Bootloader UART project. Please refer to the Bootloader documentation for relevant modifications, such as the Memory Mapping part.

The modification idea of the Mcal Standby Demo of the M core is: remove the clock and port related initialization, reconfigure the I2C Timing, and implement the main logic of the Main function.

3.1 Clock modification

In Main function removed MCU clock init：

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

Configure the I2C Clock and source to the same settings as in the Bootloader to provide the correct reference for the I2C module:

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

Note: After adjusting the clock, click Auto-Modify to solve the error related to the automatically generated clock.

3.2 MCU related modification

As mentioned above, the Mcu_Init function is commented out, you need to implement a simple function to initialize the MCU configuration variables, as follows:

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
```

Then the main function calls Mcu_Init_Simple to achieve.

Notice: In the mode setting function of the MCU module, Uart_Example_S32G274M_M7_STDY->someid()->Mcu()->Mcu->McuModeSettingConf->McuModeSettingConf_1->McuPartition1Config, check:

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

It is hoped that the switch of the A53_0 core can be controlled, but the value of the actual generated code is FALSE, so it needs to be modified manually:

\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\generate\src\ Power_Ip_VS_0_PBcfg.c

```
static const Power_Ip_MC_ME_CoreConfigType Power_Ip_MC_ME_aPartition1CoreConfigPB_1_VS_0[4U] =
{

   /* The configuration structure for Partition 1 Core 0. */
   {
       /* Specifies whether the given core is under MCU control. */
       (boolean)TRUE,
```

3.3 UART Clock related modificaiton

Due to the need to combine Standby Demo and Bootloader Demo, the Clock tree has changed. Refer to the document <<S32G_Bootloader_V1-2022.0909.pdf>> to modify the UART CLOCK TREE to meet the requirements of improving the UART source clock.

in addition:

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

- Uart Timeout Method= OSIF_COUNTER_DUMMY
- Uart Timeout Duration (0 -> 4294967295)= 100000 // Since the M core clock is increased to 400Mhz, it is recommended to modify and increase the Timeout duration by at least 10 times in the case of using the dummy timeout method.

Similarly, the following source code:

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);
```
// It is recommended to increase the timeout value of the synchronous sending function of these two places by 10 times

3.4 Port related modification

Comment out the PORT initialization in the Main function:

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

3.5 I2C related modification

Because the root clock XBAR_DIV3_CLK of I2C has changed from 8M to 133Mhz, the SCL Divider of I2C should change from 80M/400K=200 to 133M/400K=333. By adjusting:

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

The I2C IBFD register value is obtained by adjusting the above value, so that SCL is automatically configured to a value close to and less than 400K:

- I2c SCL Divider (cycles) (20 -> 15360)= 352 // Automatically calculated as a value >333
- I2c SDA Hold Delay (cycles) (7 -> 2052)= 68 // Automatically calculated
- I2c Hold Start Delay (cycles) (6 -> 7672)= 152 // Automatically calculated
- I2c Hold Stop Delay (cycles) (11 -> 7684)= 180 // Automatically calculated
- I2c Baud Rate (0 -> 1000000) =378787.8787878788 //

It is automatically calculated as a value <400K. Note that generally I2C devices support a rate of 400K, so here is set to a value close to and less than 400K, but some devices can support a rate above 400K, which can be accelerated by increasing the rate Start Time.

For the above correspondence between IBFD and timing, please refer to "Table *. I2C divider and hold values when glitch filter is disabled" in the chip manual. Note that the values are different from EB IDE, and EB shall prevail.

3.6 Enable the waiting function of M core entering STDY

This Demo uses serial port input to control whether the M core enters STDY, so the A core needs to execute the STR command first, and then enter the confirmation:

Therefore, the function of using the serial port input command is as follows:

```
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 function modification

According to the documents <<S32G_Bootloader_V1-*.pdf>> (John.Li) and <<S32G_Standby_Demo_V4-*.pdf>> (John.Li), the main function of the standby UART project is finally modified to:

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 Modify the Bootloader project to support simultaneous M/A core demo

According to the document <<S32G_Bootloader_V1-*.pdf>>(John.Li), first configure a Bootloader project that can start the A-core Linux BSP32+M-core UART MCAL sample at the same time, and verify that it can run.

Then according to the document <<S32G_Standby_Demo_V4-*.pdf>> (John.Li), transfer the clock and port related initialization in MCAL Standby Demo to the Bootloader.

4.1 I2C Clock related modification

According to the document <<S32G_Bootloader_V1-*.pdf>>, after configuring the clock for the UART MCAL sample and Linux, you also need to consider the clock configuration of the Mcal standby demo. Refer to the document <<S32G_Standby_Demo_V4-*.pdf>> (John.Li) to add I2C related configuration:

In Bootloader…->EcuC(…)->Mcu(…)->Mcu->McuClockSettingConfig->McuClockSettingConfig_0->McuClockReferencePoint, click + to add one item and click to enter:

- Name= I2C_CLK
- Mcu Clock Frequency Select= XBAR_DIV3_CLK
- Mcu Clock Reference Point Frequency= 1.3E8// Automatic calculation

Note: After adjusting the clock, click Auto-Modify to solve the error related to the automatically generated clock.

4.2 Port related modifcaiton

Bootloader…->EcuC(…)->Port(…) -> Port->PortContainer-> PortContainer_0->General :

PortNumberOfPortPins=+3

-> PortPin： Click the + to add a pin, click to enter:

- 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

  Same way to add SDA:

- 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=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

Same way to add WKUP0：

-> PortPin： Click the + to add a pin, click to enter: configuration: 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

And then in: PortContainer_0->General Automatic adjustment:

- PortNumberOfPortPins=46

4.3 Others modificaiton

Modify the M core part to support the M core standby Demo as follows:

Main modification: Configure the M7 Boot sources part:

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

Modify the name: Name= BootSources_Standby Demo。

Reference link file:
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 */
```

Refer to compiling to generate a Mapping file:
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
```

The obtained bin file size is: 1280Kbytes=1,310,720.

In summary:

1. BootSources_Standby Demo ->General-> Boot souce=QSPI
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. Because modify the name: In Bootloader(…)->Bootloader->Core Configuration: after A53_0_BSP_ATF, modify or add one item:
   - Name is:M7_Standby_Demo,
   - Core ID modify to：M7_0，
   - Boot source configuraiton to /Bootloader/Bootloader/ BootSources_Standby Demo.

5 Modify A53 Linux codes

According to the document <<S32G_Linux_STR_V1 -*.pdf>>(Johnli), modify the Linux code, mainly in ATF. Refer to Chapter 5 of the documentation: Custom Modifications.

6 Demo running and testing

6.1 Hardware link

- The 12V power supply is connected to J176, and SW15 is the power switch
- Connect the USB cable from UART0 J2 to the computer, as the programming cable and the debugging serial port of Linux 115200 8n1
- Connect the USB cable from UART1 J1 to the computer as the debugging serial port 115200 8n1 of the M7 core
- SW9=OFFOFF, SW10=OFFOFF is download mode, used for image programming, SW10_1=ON is normal startup mode
- Bootmode all OFF means boot from QSPI NOR

6.2 Image burning

Switch to download mode, power on, and write the image of bootloader with IVT header, fip.bin and mcal sample according to the instructions in <<S32G_Bootloader_V1-*.pdf>>(Johnli).

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

The port names of the COM port is set to the serial port seen in the device manager: COM22 (note that the serial port 0 corresponds to the PC serial port number).

Then click Upload target and algorithm to hardware..., the programming tool will load the algorithm image and configure the programming equipment:

```
Configuring target
Progress: 100
Flash algo is loaded.
Device: Macronix MX25UW51245G
Capacity: 64 MiB (67108864 bytes)
```

Then click Erase memory range... and select 0x0-0x500000

- Use flash tools to burn the bootloader image into QSPI:

Click Upload file to device..., and write "bt_blob.bin" to address 0x0.

- Use flash tools to program A53 fip.bin into QSPI:

Click Upload file to device..., and write "fip.bin" to the address 0x100000, refer to the address before The QSPI source address configured by Bootloader MCAL, when programming, pay attention to programming the fip.bin file, this is without A53 Bootloader fip.s32 for IVT header.

- Use flash tools to program main.bin of M7_0 into QSPI:

Click Upload file to device..., and burn "main.bin" to the address 0x200000. This is the M7 standby Demo test image.

- Burn A53 Linux image to SDcard:

According to the document <<S32G_Kernel_BSP32_V4-20220513.pdf>>, it is stated that using the SDcard reader in Ubuntu Burn the entire image to TFcard in:

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

And update the modified 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 can be directly copied to the FAT partition of TFcard in the windows environment. Then insert the TFcard into the J3 TFcard slot on the RDB2 board, and switch SW3=On to start the TFcard.

6.3 Demo running

Switch to normal mode, and then open two serial ports on the PC: 115200-8n1, power on and start M core serial port 1 and print as follows:

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

Then execute the command on serial port 0 of the Linux terminal:

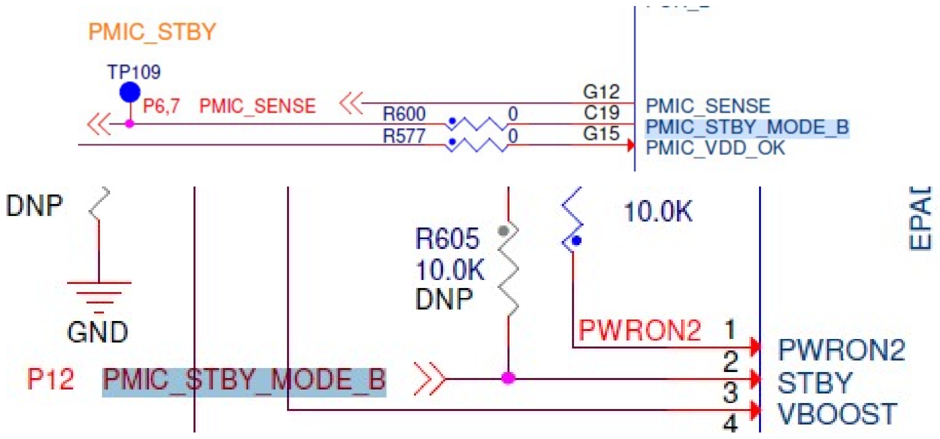
```
echo disabled > /sys/devices/platform/40060000.rtc/power/wakeup
echo mem > /sys/power/state
```

Then enter “y” on the M7 serial port.

Serial port print:

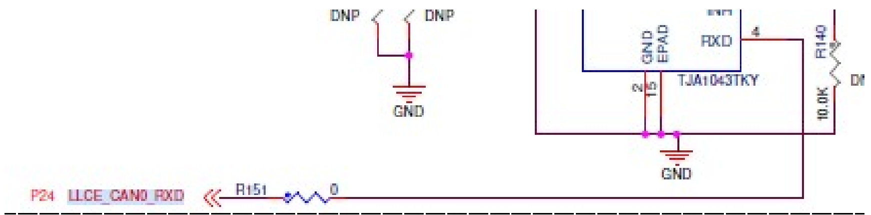
```
Stdby demo: Enter Stdby
```

The test points for whether to enter Standby mode are as follows:



S32G notifies PMIC to enter or exit Standby by pulling PMIC_STBY_MODE_B pin, so you can check whether S32G has entered Standby state by measuring the level change of TP109. This pin level switch is a hardware behavior.

According to the document <<S32G_Standby_Demo_V*-*.pdf>>, touch the CAN0_RX pin R151 to the ground to trigger an interrupt and exit the Standby mode:



M core serial port 1 prints again:

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

A core serial port 0 is printed again:

```
[   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 Project release package

```
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 The generated file of the modified stdy demo
|   |   |   |->workspace
|   |   |   |   |->Bootloader_S32G2XX_ASR_4.4_M7 //bootloader project
|   |   |   |   |->Uart_Example_S32G274A_M7_STDY // Uart sample The project file of the modified
stdy demo
|   |->NXP
|   |   |->Integration_Reference_Examples_S32G2_2022_06. Code. Framework. Realtime. Swc.
Bootloader. Platforms. S32G2XX. Build //bootloader compiling folder
|   |   |   |->bin_bootloader // target image folder
|   |   |   |   |->Bootloader.bin/elf/map //generated bootloader image
|   |   |   |->cmm //debug Lauterbach script
|   |   |   |->launch.bat // compiling configure file
|   |   |->SW32G_RTD_4.4_3.0.2. eclipse. Plugins //stdy demo folder
|   |   |   |->Icu_TS_T40D11M30I2R0 //ICE module Modification
|   |   |   |->Mcu_TS_T40D11M30I2R0 //MCU module Modification
|   |   |   |->Pmic_TS_T40D11M30I2R0 //PMIC module Modification
|   |   |   |->Uart_TS_T40D11M30I2R0 //base on UART sample standby demo main folder
|   |   |   |->Platform_TS_T40D11M30I2R0 //link file
|->test_binary
|   |->bt_blob.bin //bootloader  image, marked with IVT header by IVT tool
|   |->fip.bin/s32 //atf image
|   |->fsl-s32g274a-rdb2.dtb //kernel dtb
|   |->main.bin/elf/map //standby demo image
```

8 Suggestion for the future development

8.1 M/A core sync mechanism

The synchronization mechanism of M/A in Standby needs to be developed by the customer. If it is implemented on the upper layer, synchronization methods such as IPCF driver can be considered. This article uses the method of inputting commands at the M/A serial port in turn when entering STDBY&STR Used for Demo.

If it is to be implemented at the bottom layer, because all drivers will be suspended during Standby, the method that requires driver support cannot be considered, and methods such as setting flags and polling flags in the standby SRAM can be considered.

8.2 Function safety and Information security

Due to the instability introduced by Linux STR itself, it is recommended that the M core consider the possibility that the A core fails to exit the STR, and consider the heartbeat mechanism between the M core and A core (via IPCF or network port) to judge Whether the A core exits STR successfully, if not, reboot.

Since this demo uses the M7 standby demo full boot process, the secure boot can be performed in the full boot, but if you use the standby sram short boot method, you must consider the signature and verification of the image in the standby sram.

9 Remaining issues

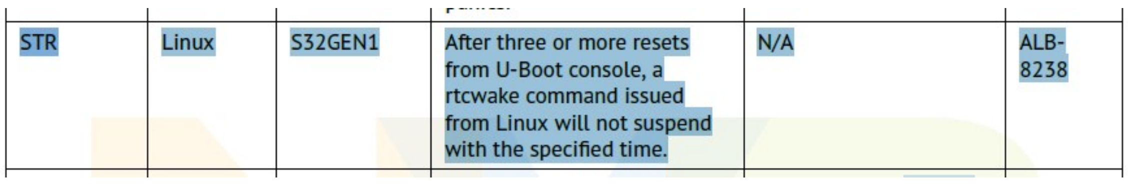
Refer to the release notes of Linux BSP to see the list of drivers that Linux supports STR, as shown in the following example: <<S32G2_LinuxBSP_32.0_Release_Notes.pdf>>:

– 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 support

As of the current Linux BSP release (2023/5/4), there is no claim that the IPCF Linux driver supports STR, but the IPCF driver is relatively simple and has little dependence on hardware, so whether there will be problems with STR needs to be tested, or wait for NXP to confirm IPCF's STR support.

9.2 PFE Slave STR support

As of the current Linux BSP release (2023/5/4), the STR supported by PFE's Linux Slave driver does not meet the needs of the actual situation. It is necessary to modify the two STR dependent functions of suspend/resume, or wait for NXP's Repair confirmed.