S32G Bootloader 定制说明

by John Li (nxa08200)

本文说明在S32G2 RDB2板上如何定制开发Bootloader，本文示例主要实现功能是：

- Bootloader启动一个M核，MCAL驱动测试程序，本文分别测试了MCU，DIO，UART的MCAL驱动示例代码。
- Bootloader同时启动A53 Linux。

历史	说明	作者
V1	● 创建本文	John.Li

目录

1 需要的软件，工具，文档与说明

1.1 软件与工具

软件/工具	名称	说明
开发板	S32G-VNP_RDB2	S32G2 开发板
配置与烧录工具	S32 Design Studio 3.4 with the update 3	(3.4.3_D2112)
EB TresosStudio 27.1	EB TresosStudio 27.1	It is required to modify AUTOSAR configuration of the bootloader.
Bootloader 工程：	S32G2 Platform Software Integration 2022.06	内含 bootloader 的 MCAL 工程
AutoSar MCAL 基础软件	RTD-MCAL 3.0.2: SW32G_RTD_4.4_3.0.2_D2203.exe	Modules configurations were developed and tested using the Tresos Configuration Tool version "EB tresos Studio 27.1.0 b200625-0900"
Linux BSP	BSP 32	

1.2 参考文档

- 《Hands-on S32G2 Multicore application enablement example.pdf》Wang Xuewei。本文主要参考此文。
- 《S32G_RTD_MCAL_Vxxxx.pdf》：JohnLi。https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-MCAL-customization-application-doc/ta-p/1399899，MCAL 定制说明。注意原文针对版本为 2.0.0_hf04，需要更新到 3.0.2。
- 《S32G_Kernel_BSP32_V4-20220513.pdf》：JohnLi。https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-Linux-BSP-customization-doc/ta-p/1399902，Linux 定制说明。

1.3 开发说明

Bootloader 工程的主要难点在于处理 M 核 Bootloader 与 M 核 autosar 系统(本文用 MCAL 驱动示例代码来替代)，以及 M 核与 A 核之间的资源冲突，这些所谓的资源包括：

- SRAM 的空间分配及与 MPU 的配合。
- CLOCK 的初始化冲突。

- MCU 配置相关的冲突。
- PORT 配置的冲突。
- 中断配置的冲突。

等等。

总的解决冲突的原则是：

- Bootloader 与 MCAL 的冲突，则尽量使用 Bootloader 来初始化，去掉 MCAL 的非本驱动相关的初始化与配置。所以 bootloader 负责 clock, mcu 配置与 port 配置的初始化，MCAL 驱动测试示例中就需要去掉，以免冲突。
- M 核与 A 核冲突，使用 M 核来初始化核心与自己所属外设的时钟，PORT 和中断的初始化，涉及到 A 核时钟的尽量与 Linux 中设置一致，这样 A 核就不会再设置。A 核负责 A 核所属的外设时钟，中断和 PORT 的初始化。

声明：Bootloader 本身是非 Autosar 规范要求的，所以没有质量保证声明，本文也仅是提供一些修改指导，不做质量保证，请注意。

2 测试软件安装编译说明

- Cygwin 的安装请参考文档：《S32G_RTD_MCAL_Vxxxx.pdf》：https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-MCAL-customization-application-doc/ta-p/1399899。
- S32DS 3.4.3 的安装，请参考文档 《S32 Design Studio v3.4.1 及 RTD 2.0.0 HF4 安装指南_v20210810.pdf》：https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-Design-studio-amp-RTD-SDK-install-guide/ta-p/1399909。

  注意请安装本文要求的软件版本。

2.1 安装 RTD_MCAL 驱动

双击 SW32G_RTD_4.4_3.0.2_D2203.exe 安装。安装向导中会要求提供 EB Tresos Studio 的路径（C:\EB\tresos\）。

RTD正确安装后，会在安装路径（默认：C:\NXP\ SW32G_RTD_4.4_3.0.2）下看到源码以及相关文档。并且对应的EB Tresos Studio安装路径下的links文件夹中会看到.link的文件，这个文件表明EB Tresos Studio已经和RTD关联起来。之后打开EB Tresos Studio新建或导入RTD配置工程时，它会关联到RTD路径下。如果安装时没有提供EB路径，也可以手动在EB相应的路径下自行新建后缀为.link的文件，写入刚才RTD的安装路径。

C:\EB\tresos\links\SW32G_RTD_4.4_3.0.2.link

```
path=C:/NXP/SW32G_RTD_4.4_3.0.2
```

注意其它的.link 文件要修改为备份名，比如说修改为 SW32G2_RTD_4.4_2.0.0_HF04.bak，以防止冲突。MCAL 的详细情况，可以参考文档《S32G_RTD_MCAL_V*.pdf》。

2.2 编译 MCAL 驱动测试程序(以 MCU 为例)

首先以最简单的MCU MCAL驱动为例。

根据文档《S32G_RTD_MCAL_Vxxxx.pdf》说明，打开MCU MCAL驱动测试程序示例。

在工程Mcu_Example_S32G274A_M7右击，选择generate project，使用EB生成代码后，将C:\EB\tresos\generate*拷贝到C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\generate*

修改编译相关文件：C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\project_parameters.mk：

```
#The path to the GCC installation dir
GCC_DIR = C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi #编译器位置
#The path to the EB Tresos installation dir
TRESOS_DIR = C:/EB/tresos #EB安装位置
#The path to the T32 installation dir
T32_DIR = C:/T32 #Trace32安装位置
```

…

```
COMPILER ?= GCC #编译器选择,这里选择使用S32DS的GCC编译器，
GCC_DIR ?= C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi #编译器位置
PLUGIN_DIR ?= C:/NXP/SW32G_RTD_4.4_3.0.2/eclipse/plugins #MCAL安装位置
LLCE_BIN_DIR ?= ../../firmware/llce_bin/s32g2/bin/ghs/enablement/ #LLCE firmware 位置
TARGET_BOARD ?= S32G_RDB #编译为RDB版本
```

如下修改C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\Makefile：导出*.bin文件。

```
.PHONY: build
build: $(ELFNAME).elf
    $(GCC_DIR)/bin/arm-none-eabi-objcopy.exe -O binary ./out/$(ELFNAME).elf ./$(ODIR)/$(ELFNAME).bin
```

Cygwin中编译MCU MCAL驱动示例程序的命令如下：C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7

```
$ make build
$ make
```

生成镜像在C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\out：

main.elf，main.bin， main.map。

2.3 优化重排 M7 demo 镜像及与 MPU 设置的配合

目前MCU MCAL镜像链接文件Mapping是：

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

```
    int_sram              : ORIGIN = 0x34000000, LENGTH = 0x00400000 /* 4MB */
    int_sram_stack_c0     : ORIGIN = 0x34400000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_stack_c1     : ORIGIN = 0x34402000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_stack_c2     : ORIGIN = 0x34404000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_no_cacheable  : ORIGIN = 0x34500000, LENGTH = 0x00100000 /* 1MB, needs to include int_results */
    ram_rsvd2             : ORIGIN = 0x34600000, LENGTH = 0        /* End of SRAM */
```

而Bootloader的SRAM镜像mapping是：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\linkfiles\autosar_intram.gld：

```
MEMORY
{
  int_sram          (rwx)     : ORIGIN = 0x34700000, LENGTH = 0xDF000   /* ~1MB */
  int_sram_heap     (rwx)     : ORIGIN = 0x347DF000, LENGTH = 0x10000   /* 64k - required for the dynamic memory allocation via stdlib */
  int_sram_stack    (rwx)     : ORIGIN = 0x347EF000, LENGTH = 0x1000    /* 8K - if changed, sync with sys_init */
}
```

Linux Bootloader：fip.s32的是：

```
Image Layout
      DCD:                Offset: 0x200          Size: 0x1c
      IVT:                Offset: 0x1000         Size: 0x100
      AppBootCode Header:     Offset: 0x1200         Size: 0x40
      Application:            Offset: 0x1240         Size: 0x2a800
IVT Location:   SD/eMMC
Load address:   0x343008c0
Entry point:    0x34302000
0x343008c0~0x343008c0+0x1240+0x2a800=0x3432c300
```

所以可以看到M7和fip的SRAM镜像加载运行地址是有冲突的，所以我们需要移动M7镜像。

另外注意，Bootloader工程默认是没有打开MPU配置的，而MCAL驱动示例代码默认是打开的，所以检查其MPU配置如下：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Platform_TS_T40D11M30I2R0\startup\include\core_specific.h:

```
/*
 Region  Description       Start       End         Size[KB]  Type            Inner Cache Policy    Outer Cache Policy    Shareable   Executable   Privileged Access   Unprivileged Access
 -------- ----------------- ---------- ---------- ---------- ---------------- -------------------- -------------------- ----------- ------------ ------------------- ---------------------
      0  Whole memory map  0x0        0xFFFFFFFF    4194304  Strongly Ordered  None                 None                 Yes         No           No Access           No Access
      1  QSPI AHB          0x0        0x1FFFFFFF     524288  Normal            None                 None                 No          Yes          Read/Write          Read/Write
      2  DTCM              0x20000000  0x201FFFFF       2048  Strongly Ordered  None                 None                 Yes         Yes          Read/Write          Read/Write
      3  HSE Shared RAM    0x22C00000  0x22C03FFF         16  Normal            None                 None                 Yes         Yes          Read/Write          Read/Write
      4  Standby RAM       0x24000000  0x24007FFF         32  Normal            Write-Back/Allocate  Write-Back/Allocate  No          Yes          Read/Write          Read/Write
      5  RAM(1st 4MB)      0x34000000  0x343FFFFF       4096  Normal            Write-Back/Allocate
```

```
Write-Back/Allocate   No          Yes          Read/Write          Read/Write
        6  RAM(2MB)          0x34400000  0x345FFFFF       2048  Normal          Write-Back/Allocate
Write-Back/Allocate   No          Yes          Read/Write          Read/Write
        7  Non-Cacheable RAM  0x34500000  0x345FFFFF       1024  Normal          None              None              Yes
Yes          Read/Write          Read/Write
        8  Peripherals       0x40000000  0x5FFFFFFF     524288  Device          None              None              Yes       No
Read/Write          Read/Write
        9  LLCE              0x43800000  0x4383FFFF       256  Device          None              None              Yes       No
Read/Write          Read/Write
        10 PPB               0xE0000000  0xE00FFFFF       1024  Strongly Ordered  None              None              Yes
No          Read/Write          Read/Write
    */

    static const uint32 rbar[CPU_MPU_MEMORY_COUNT] = {0x00000000UL, 0x00000000UL, 0x20000000UL,
0x22C00000UL, 0x24000000UL, 0x34000000UL, 0x34400000UL, 0x34500000UL, 0x40000000UL, 0x43800000UL,
0xE0000000UL};
    static const uint32 rasr[CPU_MPU_MEMORY_COUNT] = {0x1004003FUL, 0x03080039UL, 0x0104001FUL,
0x130C001BUL, 0x030B001DUL, 0x030B002BUL, 0x030B0029UL, 0x130C0027UL, 0x13050039UL,
0x13050023UL, 0x13040027UL};
```

所以将M7镜像移动到4M以上的地址，但是又不能大于6M的地方(7M以上为Bootloader地址空间)，另外要注意如中断处理函数之类的链接地址需要在no_cacheable的地址，所以我们也需要这一段0x34500000的地址，(这样会导致导出的*.bin文件比较大，中间有连续的空洞空间，不过客户最终是使用整个autosar系统，所以本文不再考虑重排以缩小镜像)：

```
    int_sram             : ORIGIN = 0x34400000, LENGTH = 0x00080000 /* 512KB size 4MB offsit */
    int_sram_stack_c0    : ORIGIN = 0x34480000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_stack_c1    : ORIGIN = 0x34482000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_stack_c2    : ORIGIN = 0x34484000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_no_cacheable  : ORIGIN = 0x34500000, LENGTH = 0x00078000 /* 480K, needs to include int_results
*/
    ram_rsvd2            : ORIGIN = 0x34600000, LENGTH = 0        /* End of SRAM */
```

重新编译后，转成*.bin文件，文件大于由原来的5.M多变成了1.25MB。(实际大小 仅有256KB)。得到bin文件大小为：1,310,720 Bytes=0x140000。

然后map文件main.map如下所示：

```
.sram          0x34400000    0x1e21c    //加载地址
               0x34400000                . = ALIGN (0x4)
 *(.core_loop)
 .core_loop     0x34400000       0xc out/startup_cm7.o
               0x34400000                _core_loop
               0x3440000c                . = ALIGN (0x4)
 *(.startup)
 *fill*         0x3440000c       0x4
 .startup       0x34400010     0x1d0 out/startup_cm7.o
               0x34400010                Reset_Handler          //运行地址
0x34400010                _start
```

2.4 去掉 CLOCK INIT

基于以下理由需要去掉clock initial

1. Bootloader 已经配置过 clock 了，所以 MCAL 驱动再次配置可能会有冲突。

2. MCAL 驱动 sample 本身是为了单独运行 lauterbach 脚本才初始化 clock，当集成在 autosar 系统中，推荐是使用 bootloader 来做时钟初始化的。

● Can_llc-pfe-th->ECU(…)->Mcu(…)->Mcu->General->Mcu Init Clock API=unchecked。

之后需要在Bootloader中将时钟的反初始化去掉。

然后在源代码中：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\src\main.c中，将clock初始化删除掉：

```
#if 0
  Mcu_InitClock(McuClockSettingConfig_0);

  while ( MCU_PLL_LOCKED != Mcu_GetPllStatus() )
  {
    /* Busy wait until the System PLL is locked */
  }

  Mcu_DistributePllClock();

 #endif
```

或是在Can_llc-pfe-th->ECU(…)->Mcu(…)->Mcu->McuClockSettingConfig中将McuClockSettingConfig_0删除掉。

2.5 去掉 MCU 相关 INIT

为了避免 Bootloader 和 Mcal 驱动的 MCU 模式设置的冲突，将 MCU 模式设置初始化去掉：（此处会设置 MCU 模式初始化，它会重启 parition，所以需要去掉）：

然后在源代码中：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\src\main.c中，将模式初始化删除掉：

```
  #if 0
  Mcu_SetMode(McuModeSettingConf_0);
 #endif
```

或是在Can_llc-pfe-th->ECU(…)->Mcu(…)->Mcu->McuModeSettingConfig中将McuModeSettingConfig_0删除掉。

另外MCU_init中也会调用RamSectorConf去初始化RAM，而我们之前已经用DCD初始化过了，所以可以将MCU_init也注掉，这样的话MCU的main函数相当与没有代码调用了。

```
#if 0
  /* Initialize the Mcu driver */
  Mcu_Init(NULL_PTR);
    #endif
```

2.6 DIO MCAL 程序去掉 PORT INIT

EB打开工程 C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Dio_ToggleLed_S32G274A_M7\TresosProject。

相关编译，内存移动与MPU设置，CLOCK去掉INIT，MCU去掉INIT，与MCU MCAL驱动相同。

源代码修改如下：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Dio_ToggleLed_S32G274A_M7\src\main.c

```
int main(void)
{…
#if 0
/* 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);
#endif
…
```

MCU MCAL驱动主要包括时钟和MCU模式的初始化，全部去掉后就基本没有代码了。本节以DIO点灯MCAL示例为例说明PORT冲突的解决。基本是两种思路，1：是由Bootloader来负责全部PORT的初始化。2：是Bootloader只做相关驱动的初始化，其它PIN要设置其untouchedPortPin。然后由每个MCAL驱动来负责自己的初始化。实践中以第一种方法居多，所以本文也以第一种办法说明：

Dio_ToggleLed…->someid->Port,右击，disabled。重新生成代码：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Dio_ToggleLed_S32G274A_M7\src\main.c

```
//#include "Port.h" 去掉port.h文件
int main(void)
{…
#if 0
  /* Initialize all pins using the Port driver */
  Port_Init(NULL_PTR);
#endif
```

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Dio_ToggleLed_S32G274A_M7\ project_parameters.mk

```
#MCAL modules used
MCAL_MODULE_LIST := Resource Base Mcu Dem Det EcuC Os Platform Dio Port Rte #去掉Port，不编译。
```

之后Bootloader工程中加入此GPIO的Port配置。

2.7 UART MCAL 程序去掉 PORT INIT

EB打开工程 C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Uart_TS_T40D11M30I2R0\TresosProject。

相关编译，内存移动与MPU设置，CLOCK去掉INIT，MCU模式去掉INIT，与MCU MCAL驱动相同。以及Port在EB中disable的办法与上一节相同。

由于DIO并不需要时钟初始化，所以以UART MCAL工程来说明去掉CLOCK/MCU MODE/PORT的情况。

源代码修改如下：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Uart_TS_T40D11M30I2R0\src\main.c

```
//#include "Port.h"
int main(void)
{…
#if 0
  /* 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);
#endif
```

2.8 UART MCAL 程序修改 CLOCK TREE

参考第3.5节，为了让Bootloader/UART MCAL/Linux的UART CLOCK TREE统一，需要修改UART 的CLOCK TREE：

Uart_example…->someid…->Mcu…->Mcu->McuClockSettingConfig->McuClockSettingConfig_0->McuPeriphPLL：

- RDIV=1；MFD (1 -> 255)=50；MFN (0 -> 32767)=0；
- PHI0 Division value (0 -> 255)* =19；
- PHI1 Division value (0 -> 255)* =24；
- PHI3 Division value (0 -> 255)* =15；PHI3 Divider enable=checked。
- PLL_PHI0 Frequency (Calculated) (dynamic range)*自动计算为1.0E8。
- PLL_PHI1 Frequency (Calculated) (dynamic range) 自动计算为8.0E7。
- PLL_PHI3 Frequency (Calculated) (dynamic range)* 自动计算为1.25E8 。
- PLL_VCO Frequency (Calculated) (dynamic range)* 自动计算为2.0E9。

…McuClockSettingConfig_0-> McuCgm0ClockMux8：

- CGM0 Clock Mux8 Source= PERIPH_PLL_PHI3_CLK。
- Clock Mux8 Frequency (LIN_BAUD_CLK) (dynamic range) 自动计算为1.25E8 。

…McuClockSettingConfig_0->McuClockReferencePoint->UART_CLK

- Mcu Clock Reference Point Frequency (0 -> 5000000000)* 自动计算为1.25E8 。

然后检查：Uart_example…->someid…->Uart…->Uart->UartChannel-> UartChannel_1

- UartClockRef = /Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig_0/UART_CLK=1.25E。

注意，默认波特率为：9600，为了配合调高的根时钟，修改为：

- DesireBaudrate = LINFLEXD_UART_BAUDRATE_115200
- Uart Parity Enable= unchecked。所以其实配置是配置为无奇偶检验的，注意PC端相应配置。

另外注意，UART镜像默认M7的根时钟是48Mhz的FIRC，本文为了配合正式的Bootloader的配置，修改为800Mhz，所以CPU的loop速率理论上快了9倍，所以:

Uart_example…->someid…->Uart…->Uart->General:

- Uart Timeout Duration (0 -> 4294967295)*= 10000 //原来是1000
- Uart_example…->someid…->Mcu…->Mcu->General:

Mcu Loops TimeOut (1 -> 4294967295)*= 100000 //原来是10000，此处可以不修改，因为MCU的init已经注掉了。（以上修改仅在使用MCAL驱动示例这种裸代码时，采用Uart Timeout Method =OSIF_COUNTER_DUMMY。才需要）。

最后：修改C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\src\main.c，去掉多余的测试代码，仅保留一条异步发送即可：

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

  /* Initializes an UART driver*/
  Uart_Init(&Uart_xConfig_VS_0);
  /* Uart_AsyncSend transmit data */
  (void)Uart_AsyncSend(UART_CHANNEL, (const uint8 *)WELCOME_MSG, strlen(WELCOME_MSG));
  /* Wait for Uart successfully send data */
  while(Uart_GetStatus(UART_CHANNEL, &varRemainingBytes, UART_SEND) == UART_STATUS_OPERATION_ONGOING);
#if 0 …#endif
  Uart_Deinit();
  Exit_Example((Uart_Status == UART_STATUS_NO_ERROR) && (Std_Uart_Status == E_OK));
  return (0U);
```

2.9 解决中断冲突

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

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

以上代码会初始化中断，其中 Platform_Init 会注册 A 核 M 核的外设中断，注册的中断处理函数为空，而 Platform_InstallIrqHandler 设置 UART1 的中断处理函数和优化级，实际应用中发现其会影响 A 核的中断。

所以如下代码简化对 UART1 的中断的注册过程：

```
#include "sys_init.h"
…
#if 1
  /* Initialize IRQs with api */
    sys_enableIsrSource(LINFLEXD1_IRQn, (uint8)0x7U); /* Enable interrupt for uart1*/
    sys_registerIsrHandler(LINFLEXD1_IRQn, &LINFLEXD1_UART_IRQHandler);
#else
   /* Initialize IRQs */
   Platform_Init(NULL_PTR);
   Platform_InstallIrqHandler(LINFLEXD1_IRQn, LINFLEXD1_UART_IRQHandler, NULL_PTR);
#endif
```

避免对 A 核驱动的影响。

2.10 准备 A53 Linux 镜像

根据文档《S32G_Kernel_BSP32_V4-20220513.pdf》，准备 A53 SDcard 镜像。

注意点：

- 由于 DMA 搬运要求，如果在不打开 CRC 情况下是 64 Byte 对齐，所以需要修改 ATF 配置

arm-trusted-firmware/plat/nxp/s32/s32_common.mk：`FIP_ALIGN := 16`changed to `FIP_ALIGN := 64` before building.

编译后的打印：

```
Image Layout
      DCD:                Offset: 0x200          Size: 0x1c
      IVT:                Offset: 0x1000         Size: 0x100
      AppBootCode Header:     Offset: 0x1200         Size: 0x40
      Application:          Offset: 0x1240         Size: 0x2a800
IVT Location:   SD/eMMC
Load address:   0x343008c0  //0x40 倍数
Entry point:    0x34302000
```

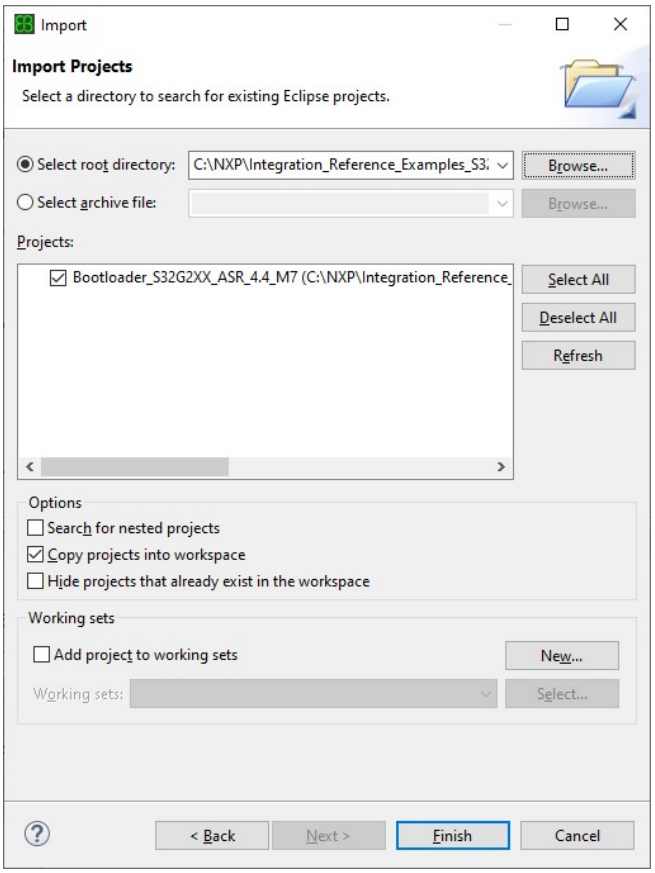
由于 FIP 有修改，所以如果是使用默认的 demo 镜像，需要把 FIP.bin/.S32 更新一下。

- 由于计划测试 MCAL 的 DIO 和 UART1 MCAL 程序，所以 Linux 端的串口 1 与 DIO GPIO_LED 的 IOMUX 相应去掉：目前 BSP32 并没有配置 UART1 的 IOMUX 与初始化。所以不需要修改，而 DIO 的 GPIO 管脚与 SPI1 冲突。需要在 Uboot 和内核的 DTS 中将 SPI1 驱动 disabled。本文不再说明，请参考 Linux 定制文档。

3 Bootloader 工程说明

运行Platform_Software_Integration_S32G2_2022_06.exe安装bootloader工程，然后将C:\NXP\Integration_Reference_Examples_S32G2_2022_06\applications\realtime\Tresos\eclipse\plugins下的所有plugin拷贝到C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins中。

然后打开EB Tresos 27.1.0，File->Import…->General->Existing Projects into Workspace:->Next Select root directory->Browse…->C:\NXP\Integration_Reference_Examples_S32G2_2022_06\applications\realtime\Tresos\workspaces\Bootloader_S32G2XX_ASR_4.4_M7，从而打开工程Bootloader_S32G2XX_ASR_4.4_M7。(勾选 Copy projects into workspace)

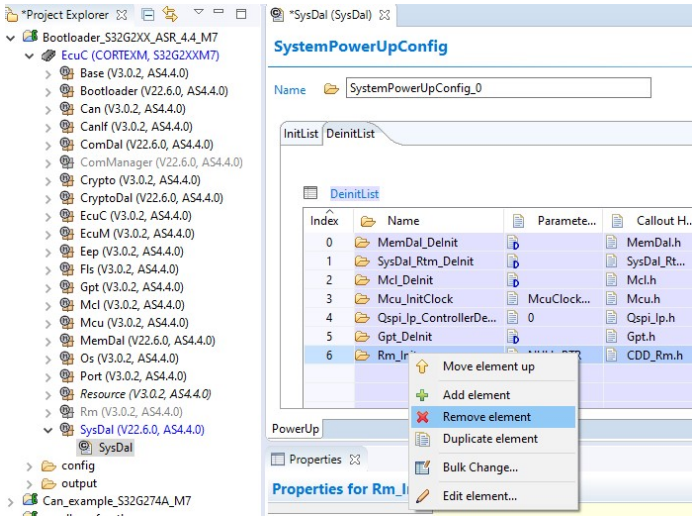


然后双击 Bootloader_S32G2XX_ASR_4.4_M7->EcuC…，则可以打开所有模块(如果有模块加载失败，检查之前是否将 platform 的 plugin 拷贝到了 mcal 中，并且确保 C:\EB\tresos\links 仅有 SW32G_RTD_4.4_3.0.2.link 的连接)。

3.1 关掉 XRDC 支持

为了简单化工程，先去掉 XRDC 支持：

1. 在 Rm(V3.0.2,AS4.4.0)模块上右击，选择 Disable 关掉此模块。
2. 选择 SysDal(V22.6.0,AS4.4.0)->SysDal->PowerUP-> SystemPowerUpConfig_0->DeinitList: 去掉 Rm_Init:



3.2 关掉 eMMC/SD 支持(可选)

由于本 sample 工程中的镜像都是放在 QSPI NOR 中，所以不需要 eMMC 支持，可以如下关闭：

1. Bootloader(…)->EcuC(…)：disabled 掉 Eep 和 MemDal 模块。
2. Bootloader(…)->EcuC(…)->SysDal (…)->SysDal->Powerup-> SystemPowerUpConfig_0:

InitList 中将 MemDal 的 init 删除掉。

DeinitList 中将 MemDal 的 deinit 删除掉。

3. 如后文说明在编译配置中将 SDHC 去掉。

3.3 关掉 secure boot(可选)

本工程不考虑 secure boot，所以可以如下去掉：

1. Bootloader(…)->EcuC(…)：disabled 掉 CryptoDal 和 Crypto 模块。
2. Bootloader(…)->EcuC(…)->Bootloader(…)→ Bootloader->General: Enable Secure Boot= unchecked.
3. Bootloader(…)->EcuC(…)->SysDal (…)->SysDal->Powerup-> SystemPowerUpConfig_0:

InitList 中将 SysDal_Init 的 init 删除掉。

4. 如后文说明在编译配置中将 secure boot 去掉。

3.4 增加 MCAL 驱动所需要的 PORT 的初始化

首先，由于本 Bootloader 工程中去掉了 SDHC 的支持，所以相应的管脚要去掉：

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

将 USDHC 相关管脚全部删除。

然后在 General 里的：PortNumberOfPortPins*自动计算为 40。之后其它管脚的 Port ID 全部要自动计算重排。

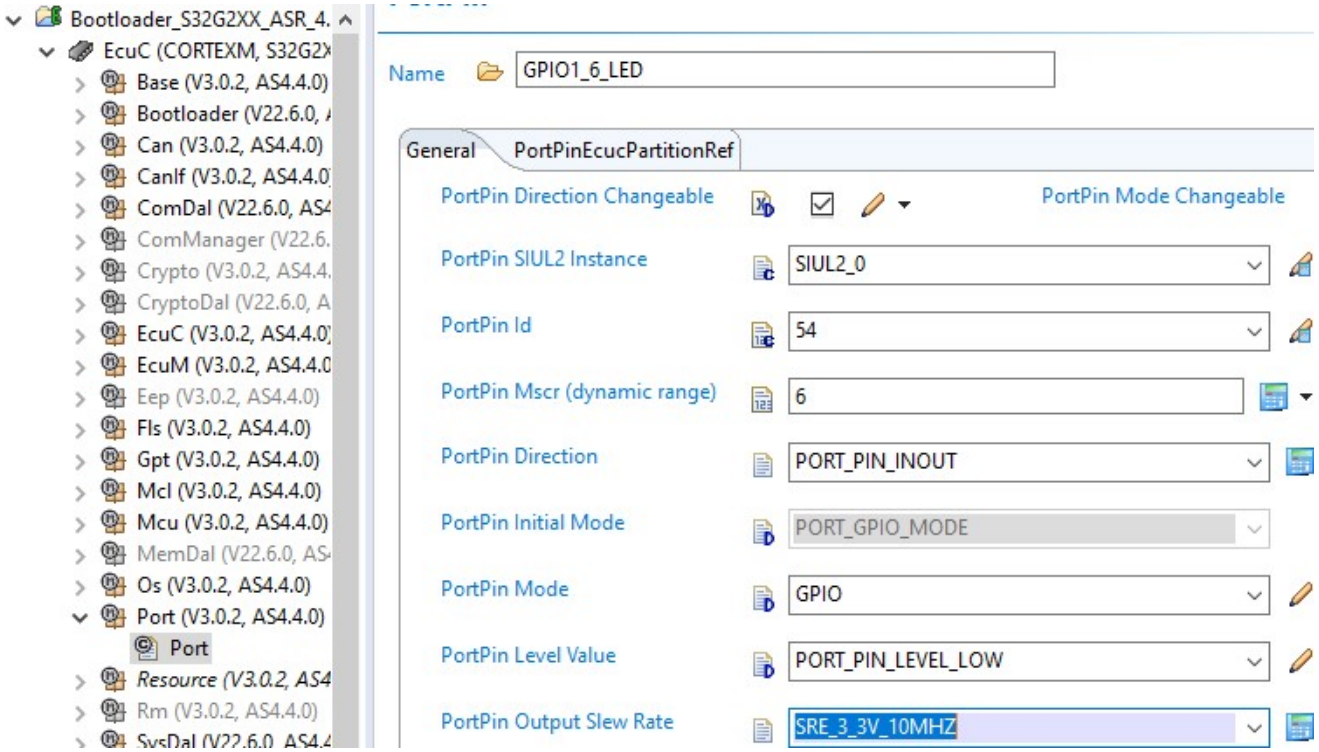
- DIO MCAL 驱动使用的 GPIO：

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

  PortNumberOfPortPins+1=41

  PortPin :点击右上角+增加一个 Port :双击进入：

1. 修改名字为 GPIO1_6_LED
2. PortPin Id=40 自动排序
3. PortPin Mscr (dynamic range)=6
4. PortPin Direction= PORT_PIN_INOUT
5. PortPin Mode= GPIO
6. PortPin Level Value= PORT_PIN_LEVEL_LOW
7. PortPin Output Slew Rate= SRE_3_3V_10MHZ

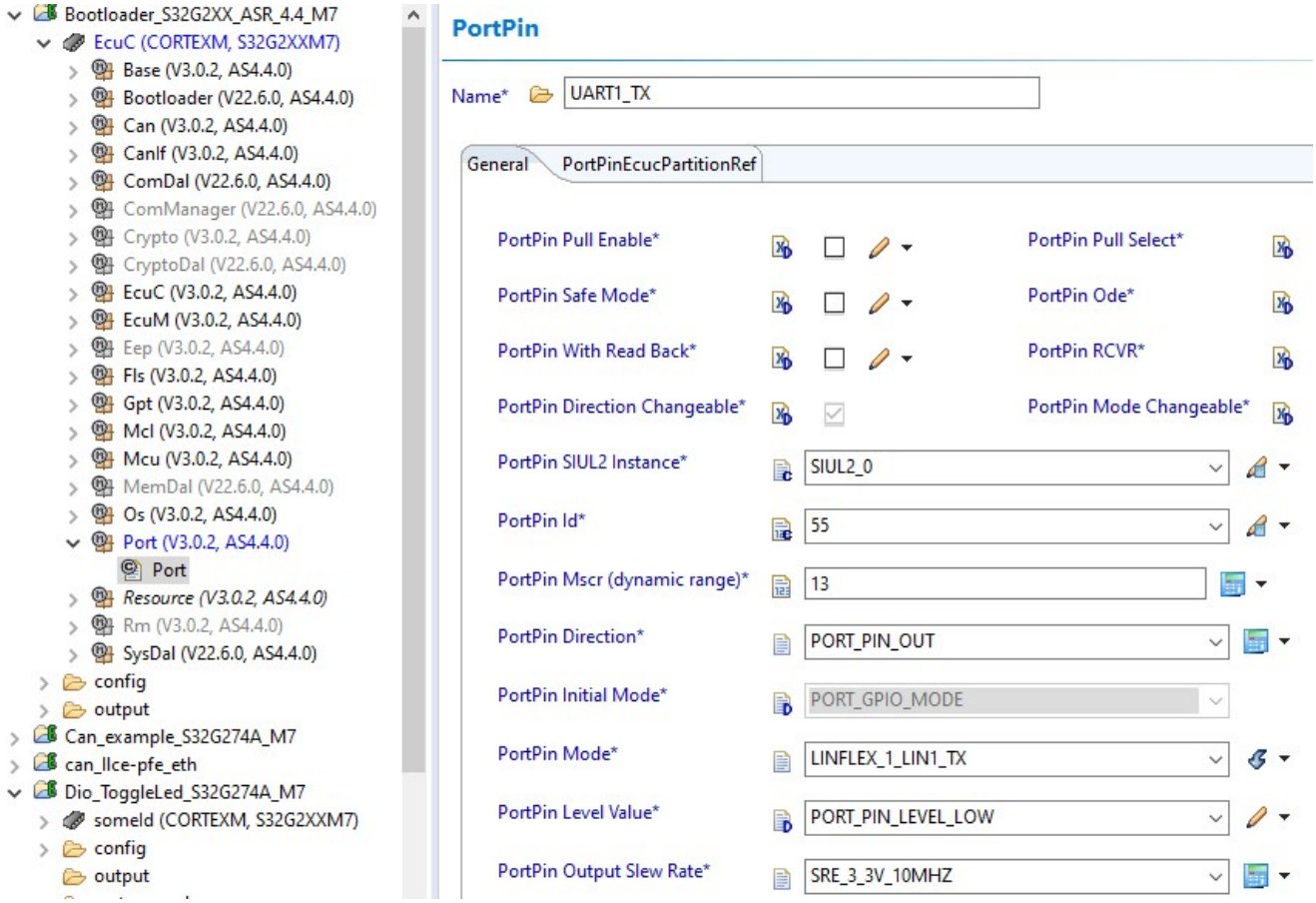


- UART MCAL 驱动使用的 UART1_RX/TX

  Bootloader…->EcuC(…)->Port(…) -> Port->PortContainer-> PortContainer_0->General : PortNumberOfPortPins+2=43

  PortPin :点击右上角+增加一个 Port :双击进入

1. 修改名字为 UART1_TX
2. PortPin Id=41 自动排序
3. PortPin Mscr (dynamic range)=13
4. PortPin Direction= PORT_PIN_OUT
5. PortPin Mode= LINFLEX_1_LIN1_TX
6. PortPin Level Value= PORT_PIN_LEVEL_LOW
7. PortPin Output Slew Rate= SRE_3_3V_10MHZ



RX 的设置与上道理相同：

1. 修改名字为 UART1_RX
2. PortPin Id=42 自动排序
3. PortPin Mscr (dynamic range)=16
4. PortPin Direction= PORT_PIN_IN
5. PortPin Mode= LINFLEX_1_LIN1_RX
6. PortPin Level Value= PORT_PIN_LEVEL_LOW
7. PortPin Output Slew Rate= SRE_3_3V_10MHZ

3.5 解决 Bootloader,MCAL 与 Linux 的 clock 冲突

以 UART MCAL 驱动示例为例：核心原则是：

1. 时钟配置只保留 Bootloader 的初始化处，Bootloader 的反初始化与 Mcal 代码中的初始化代码去掉

2. Bootloader 的初始化要兼顾 M 核时钟与 Mcal 驱动所需要的最终时钟配置，并考虑到 A 核时钟的正确源和值(可以配置为不受 MCU 代码控制)。

所以:首先在 Sysdal->powerup->systempowerupconfig_0->DinitList:

- 将此项 Mcu_InitClock；McuClockSettingsDisablePLL；Mcu.h。删除掉。从而去掉了 Bootloader 中的反初始化。

然后：修改初始化的值，要兼顾 M 核，外设与 A 核时钟：

	Bootloader: clocksettingconfig	UART:MCU clocksettingconfig	Linux	Bootloader: clocksettingconfig
	Config_0(初始化)	Config_0		Config_0(初始化)修改说明：
General	Cgm0cfg:1:48 Cgm1cfg:1:48 Cgm2cfg:1:48 Cgm5cfg:0 Cgm6cfg:0	Cgm0cfg:1:48 Cgm1cfg:1:48 Cgm2cfg:1:48 Cgm5cfg:0 Cgm6cfg:0		保持不变
McuFXOSC	4.0E7	4.0E7	4.0E7	保持不变
McuCgm0Pcs	PCFS_4:8.0E8	PCFS_4:0		保持不变
Cgm0Mux0	Source: CORE_PLL_DFS1_CLK: Name: XBAR_2X_CLK: 8.0E8 LBIST_CLK:5.0E7 DAPB_CLK:1.3E8	Source: FIRC_CLK Name: XBAR_2X_CLK: 4.8E7 LBIST_CLK:2.4E7 DAPB_CLK: 8E6	Source: CORE_PLL_DFS1_CLK: Name: XBAR_2X_CLK: 8.0E8 LBIST_CLK:? DAPB_CLK:?	Source:(此为 M 核的时钟，可以保持不变，M 核 XBAR=400Mhz) CORE_PLL_DFS1_CLK: Name: XBAR_2X_CLK: 8.0E8 LBIST_CLK:5.0E7 DAPB_CLK:1.3E8
Cgm0Mux1	Source: FXOSC_CLK CLKOUT0:0	Source: FXOSC_CLK CLKOUT0:2.0E7	Source: FXOSC_CLK CLKOUT0:0	保持不变=关闭
Cgm0Mux2	Source: FXOSC_CLK CLKOUT1:0	Source: FXOSC_CLK CLKOUT0: 2.0E7	Source: FXOSC_CLK CLKOUT1:0	保持不变=关闭
Cgm0Mux3	Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7	Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7		Source: 保持不变 PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7

Cgm0Mux4	Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 0	Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 5E6	Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 8E7	Source: 保持不变，UART MCAL 驱动没有用到 PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 0
Cgm0Mux5	Source: PERIPH_PLL_PHI1_CLK FTM_1_REF_CLK: 0	Source: PERIPH_PLL_PHI1_CLK FTM_1_REF_CLK: 5E6	Source: PERIPH_PLL_PHI1_CLK FTM_1_REF_CLK: 8E7	Source: 保持不变，UART MCAL 驱动没有用到 PERIPH_PLL_PHI1_CLK FTM_1_REF_CLK: 0
Cgm0Mux6	Source: PERIPH_PLL_PHI1_CLK FLEXRAY_PE_CLK: 0	Source: FIRC_CLK FLEXRAY_PE_CLK: 2.4E7	Source: PERIPH_PLL_PHI1_CLK FLEXRAY_PE_CLK: 0	Source: 保持不变 PERIPH_PLL_PHI1_CLK FLEXRAY_PE_CLK: 0
Cgm0Mux7	Source: PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 4.0E7	Source: FIRC_CLK CAN_PE_CLK: 4.8E7	Source: PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 8.0E7	Source: 保持不变 PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 4.0E7
Cgm0Mux8	**Source: FIRC_CLK** **LIN_BAUD_CLK: 4.8E7**	**Source: FIRC_CLK** **LIN_BAUD_CLK: 4.8E7**	Source: PERIPH_PLL_PHI3_CLK CAN_PE_CLK: 1.25E8	**Source: (修改为 Linxu 使用的时钟源和值，这个是 UART MCAL 驱动关键时钟配置)** **PERIPH_PLL_PHI3_CLK** **CAN_PE_CLK: 1.25E8**
Cgm0Mux9	Source: GMAC_EXT_TS_CLK GMAC_TS_CLK: 1.0E8	Source: FIRC_CLK GMAC_TS_CLK: 2.4E7	Source: PERIPH_PLL_PHI5_CLK CAN_PE_CLK: 1.25E8	Source:修改为不受 MCU 控制 GMAC_EXT_TS_CLK GMAC_TS_CLK: 0
Cgm0Mux10	Source: PERIPH_PLL_PHI5_CLK GMAC_0_TX_CLK: 1.25E8	Source: FIRC_CLK GMAC_0_TX_CLK:2.4E7	Source: PERIPH_PLL_PHI3_CLK CAN_PE_CLK: 1.25E8	Source: 修改为不受 MCU 控制 PERIPH_PLL_PHI5_CLK GMAC_0_TX_CLK: 0
Cgm0Mux11	Source:	Source:	Source:	Source: 修改为不受

	GMAC_0_EXT_RX_CLK GMAC_0_RX_CLK: 1.25E8	FIRC_CLK GMAC_0_RX_CLK: 4.8E7	FIRC_CLK GMAC_0_RX_CLK: 0>	MCU 控制 FIRC_CLK GMAC_0_RX_CLK: 4.8E7
Cgm0Mux12	Source: PERIPH_PLL_DFS1_CLK QSPI_2X_CLK:2.6E8	Source: FIRC_CLK QSPI_2X_CLK:0	Source: PERIPH_PLL_DFS1_CLK QSPI_2X_CLK:8E8	Source: 修改为不受 MCU 控制 PERIPH_PLL_DFS1_CLK QSPI_2X_CLK:0
Cgm0Mux14	Source: PERIPH_PLL_DFS3_CLK SDHC_CLK: 4.0E7	Source: FIRC_CLK SDHC_CLK: 0	Source: PERIPH_PLL_DFS3_CLK SDHC_CLK: 4.0E7	Source: 修改为不受 MCU 控制 PERIPH_PLL_DFS3_CLK SDHC_CLK: 0
Cgm0Mux15	Source: FIRC_CLK GMAC_0_REF_CLK: 4.8E7	Source: FIRC_CLK GMAC_0_REF_CLK: 4.8E7	?	Source: 保持不变 FIRC_CLK GMAC_0_REF_CLK: 4.8E7
Cgm0Mux16	Source: FIRC_CLK SPI_CLK: 4.8E7	Source: FIRC_CLK SPI_CLK: 4.8E7		Source: 保持不变 FIRC_CLK SPI_CLK: 4.8E7
Cgm1Mux0	Source: CORE_PLL_PHI0_CLK A53_CORE_CLK:8.0E8	Source: FIRC_CLK A53_CORE_CLK:4.8E7	Source: CORE_PLL_PHI0_CLK A53_CORE_CLK:1.0E9	Source:此为 A53 clock，修改为与 Linux 相同的源和值 CORE_PLL_PHI0_CLK A53_CORE_CLK:1.0E9
Cgm1Pcs	PCFS_4: 8.0E8	PCFS_4: 0		保持不变=关闭
Cgm2Pcs	PCFS_33: 0	PCFS_33: 0		保持不变=关闭
Cgm2Mux0	Source: FIRC_CLK PFE_PE_CLK, ACCEL_3_CLK:0	Source: FIRC_CLK PFE_PE_CLK, ACCEL_3_CLK:0		Source: 保持不变，不受 MCU 控制 FIRC_CLK PFE_PE_CLK, ACCEL_3_CLK:0
Cgm2Mux1	Source: FIRC_CLK PFE_MAC_0_TX_DIV_CLK, ACCEL_4_CLK:0	Source: FIRC_CLK PFE_MAC_0_TX_DIV_CLK,	Source: FIRC_CLK PFE_MAC_0_TX_DIV_CLK,	Source: 保持不变，不受 MCU 控制 FIRC_CLK

		ACCEL_4_CLK:0	ACCEL_4_CLK:0	PFE_MAC_0_TX_DIV_CLK, ACCEL_4_CLK:0
Mcu GENCTRL1	Source: PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0	Source: PFE_MAC_0_TX_CLK PFE_MAC_0_TX_CLK :0	Source: PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0	Source: 保持不变，不受 MCU 控制 PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0
Cgm2Mux2	Source: FIRC_CLK PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK:0	Source: FIRC_CLK PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK:0	Source: FIRC_CLK PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK:4.8E7	Source: 保持不变，不受 MCU 控制 FIRC_CLK PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK:0
Cgm2Mux3	Source: FIRC_CLK PFE_MAC_2_TX_CLK, GMAC_1_REF_DIV_CL:0	Source: FIRC_CLK PFE_MAC_2_TX_CLK, GMAC_1_REF_DIV_CL:0	Source: FIRC_CLK PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK:4.8E7	Source: 保持不变，不受 MCU 控制 FIRC_CLK PFE_MAC_2_TX_CLK, GMAC_1_REF_DIV_CL:0
Cgm2Mux4	Source: FIRC_CLK PFE_MAC_0_RX_CLK, GMAC_1_RX_CLK：4.8E7	Source: FIRC_CLK PFE_MAC_0_RX_CLK, GMAC_1_RX_CLK：4.8E7		Source: 保持不变，不受 MCU 控制 FIRC_CLK PFE_MAC_0_RX_CLK, GMAC_1_RX_CLK：4.8E7
Cgm2Mux5	Source: FIRC_CLK PFE_MAC_1_RX_CLK, SEQ_CLK：4.8E7	Source: FIRC_CLK PFE_MAC_1_RX_CLK, SEQ_CLK：4.8E7		Source: 保持不变，不受 MCU 控制 FIRC_CLK PFE_MAC_1_RX_CLK, SEQ_CLK：4.8E7
Cgm2Mux6	Source: FIRC_CLK PFE_MAC_2_RX_CLK, APEXD_0_CLK：4.8E7	Source: FIRC_CLK PFE_MAC_2_RX_CLK, APEXD_0_CLK：4.8E7		Source: 保持不变，不受 MCU 控制 FIRC_CLK PFE_MAC_2_RX_CLK, APEXD_0_CLK：4.8E7
Cgm2Mux7	Source: FIRC_CLK PFEMAC0_REF_DIV_CLK：0	Source: FIRC_CLK PFEMAC0_REF_DIV		Source: 保持不变，不受 MCU 控制

		_CLK：0		FIRC_CLK PFEMAC0_REF_DIV_CLK：0
Cgm2Mux8	Source: FIRC_CLK PFEMAC1_REF_DIV_CLK：0	Source: FIRC_CLK PFEMAC1_REF_DIV_CLK：0		Source: 保持不变，不受 MCU 控制 FIRC_CLK PFEMAC1_REF_DIV_CLK：0
Cgm2Mux9	Source: FIRC_CLK PFEMAC2_REF_DIV_CLK：0	Source: FIRC_CLK PFEMAC2_REF_DIV_CLK：0		Source: 保持不变，不受 MCU 控制 FIRC_CLK PFEMAC2_REF_DIV_CLK：0
Cgm5Mux0	Source: FIRC_CLK DDR_CLK: 4.8E7	Source: FIRC_CLK DDR_CLK: 4.8E7	Source: DDRPLL_PHI0 DDR_CLK: 8E8	Source: 保持不变，不受 MCU 控制，linux 自己控制 DDR 的 CLOCK 初始化 FIRC_CLK DDR_CLK: 4.8E7
McuRtc ClockSelect	Source: FIRC_CLK 4.8E7	Source: FIRC_CLK 4.8E7		Source: 保持不变 FIRC_CLK 4.8E7
McuCorePLL	Source: FXOSC_CLK Name: PLL_PHI0: 8.0E8 PLL_PHI1:0 PLL_VCO: 1.6E9	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_PHI1:0 PLL_VCO: 0	Source: FXOSC_CLK Name: PLL_PHI0: 1.0E9 PLL_PHI1: 2.0E9 PLL_VCO: 2.0E9	Source: **修改为 Linux 要求的源和值** FXOSC_CLK Name: PLL_PHI0: **1.0E9** PLL_PHI1: 0 PLL_VCO: 2.0E9
McuCoreDFS	Name: DFS1: 8.0E8 DFS2: 8.0E8 DFS3:0 DFS4: 0 DFS5: 0 DFS6: 0	Name: DFS1: 0 DFS2: 0 DFS3:0 DFS4: 0 DFS5: 0 DFS6: 0	Name: DFS1: 8.0E8 DFS2: 0 DFS3:0 DFS4: 0 DFS5: 0 DFS6: 0	Name: DFS1: **8.0E8 //M 核根时钟，所以保留** DFS2: 0 DFS3:0 DFS4: 0 DFS5: 0 DFS6: 0
McuPeriphPLL	Source: FXOSC_CLK	Source: FXOSC_CLK	Source: FXOSC_CLK	Source: FXOSC_CLK

	Name: PLL_PHI0: 1.0E8 PLL_PHI1: 8.0E7 PLL_PHI2: 4.0E7 PLL_PHI3:0 PLL_PHI4:0 PLL_PHI5:1.25^{E}8 PLL_PHI6:0 PLL_PHI7:0 PLL_PHI8:0 PLL_VCO: 2.0E9	Name: PLL_PHI0: 1.0E8 PLL_PHI1: 8.0E7 PLL_PHI2: 0 PLL_PHI3:0 PLL_PHI4:0 PLL_PHI5: 0 PLL_PHI6:0 PLL_PHI7:0 PLL_PHI8:0 PLL_VCO: 1.6E9	Name: PLL_PHI0: 1.0E8 PLL_PHI1: 8.0E7 PLL_PHI2: 8.0E7 PLL_PHI3: 1.25^{E}8 PLL_PHI4: 2.0E8 PLL_PHI5:1.25^{E}8 PLL_PHI6: 2.0E9 PLL_PHI7: 1.0E8 PLL_PHI8:0 PLL_VCO: 2.0E9	Name: PLL_PHI0: 1.0E8 PLL_PHI1: 8.0E7 PLL_PHI2: 4.0E7 PLL_PHI3: 1.25^{E}8**//UART 的根时钟，所以配置为与 Linux 相同** PLL_PHI4:0 PLL_PHI5:1.25^{E}8 PLL_PHI6:0 PLL_PHI7:0 PLL_PHI8:0 PLL_VCO: 2.0E9
McuPeriphDFS	Name: DFS1: 8.0E8 DFS2: 0 DFS3: 8.0E8 DFS4: 0 DFS5: 0 DFS6: 0	Name: DFS1: 0 DFS2: 0 DFS3: 0 DFS4: 0 DFS5: 0 DFS6: 0	Name: DFS1: 8.0E8 DFS2: 0 DFS3: 8.0E8 DFS4: 0 DFS5: 0 DFS6: 0	Name: DFS1: 8.0E8 DFS2: 0 DFS3: 8.0E8 DFS4: 0 DFS5: 0 DFS6: 0
McuAccelPLL	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_PHI1: 0 PLL_PHI2: 0 PLL_VCO:0	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_PHI1: 0 PLL_PHI2: 0 PLL_VCO:0	Source: FXOSC_CLK Name: PLL_PHI0: 1.8E9 PLL_PHI1: 0 PLL_PHI2: 0 PLL_VCO:0	配置为不受 MCU 控制
McuDDRPLL	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_VCO:0	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_VCO:0	Source: FXOSC_CLK Name: PLL_PHI0: 1.6E9 PLL_VCO:8.0E8	配置为不受 MCU 控制
McuClk Monitor	27 项	27 项		保持
McuClock Reference	Name: A53_CORE_CLK:8.0E8	Name: UART_CLK		Name: A53_CORE_CLK:1.0E

Point	CAN_CLK:4.0E7 CM7_CLK= XBAR_CLK:4.0E8 PIT_CLK= XBAR_DIV3_CLK:1.3E8 uSDHC=SDHC_CLK：4.8E8	=LIN_BAUD_CLK=4.8E7		9 CAN_CLK:4.0E7 CM7_CLK= XBAR_CLK:4.0E8 PIT_CLK= XBAR_DIV3_CLK:1.3E8 UART_CLK=LIN_BAUD_CLK=1.25E8//增加此项

说明：

1. 从 BSP32 开始默认使用了 ATF 的版本的 clk 命令暂时不完整，所以 Linux clock tree，在 BSP30 上使用以下命令导出：

   Uboot:

```
=> clk dump
FIRC                    : 51000000 Hz
SIRC                    : 32000 Hz
FXOSC                   : 40000000 Hz
ARM_PLL_MUX             : 40000000 Hz
ARM_PLL_VCO             : 2000000000 Hz
ARM_PLL_PHI0            : 1000000000 Hz
ARM_PLL_DFS1            : 800000000 Hz
ARM_PLL_DFS2            : 0 Hz
MC_CGM1_MUX0            : 1000000000 Hz
A53_CORE                : 1000000000 Hz
A53_CORE_DIV2           : 500000000 Hz
A53_CORE_DIV10          : 100000000 Hz
MC_CGM0_MUX0            : 800000000 Hz
XBAR_2X                 : 800000000 Hz
XBAR                    : 400000000 Hz
XBAR_DIV2               : 200000000 Hz
XBAR_DIV3               : 133333333 Hz
XBAR_DIV4               : 100000000 Hz
XBAR_DIV6               : 66666666 Hz
PERIPH_PLL_MUX          : 40000000 Hz
```

PERIPH_PLL_VCO : 2000000000 Hz
PERIPH_PLL_PHI0 : 100000000 Hz
PERIPH_PLL_PHI1 : 80000000 Hz
PERIPH_PLL_PHI2 : 80000000 Hz
PERIPH_PLL_PHI3 : 125000000 Hz
PERIPH_PLL_PHI4 : 200000000 Hz
PERIPH_PLL_PHI5 : 125000000 Hz
PERIPH_PLL_PHI7 : 100000000 Hz
PERIPH_PLL_DFS1 : 800000000 Hz
PERIPH_PLL_DFS2 : 0 Hz
PERIPH_PLL_DFS3 : 800000000 Hz
PERIPH_PLL_DFS5 : 0 Hz
SERDES_REF : 100000000 Hz
MC_CGM0_MUX3 : 80000000 Hz
MC_CGM0_MUX4 : 80000000 Hz
FTM0_EXT_REF : 0 Hz
FTM0_REF : 40000000 Hz
MC_CGM0_MUX5 : 80000000 Hz
FTM1_EXT_REF : 0 Hz
FTM1_REF : 40000000 Hz
Mux without a valid source
MC_CGM0_MUX6 : 0 Hz
Mux without a valid source
Failed to get the frequency of CGM MUX
FLEXRAY_PE : 0 Hz
MC_CGM0_MUX7 : 80000000 Hz
MC_CGM0_MUX8 : 125000000 Hz
LIN_BAUD : 125000000 Hz
LINFLEXD : 62500000 Hz
MC_CGM0_MUX10 : 125000000 Hz
GMAC0_TX : 125000000 Hz
GMAC0_EXT_TS : 0 Hz
MC_CGM0_MUX9 : 200000000 Hz
GMAC0_TS : 200000000 Hz

```
GMAC0_EXT_TX                : 0 Hz
GMAC0_EXT_REF               : 0 Hz
SERDES0_LANE0_CDR           : 125000000 Hz
SERDES0_LANE0_TX            : 125000000 Hz
GMAC0_EXT_RX                : 25000000 Hz
MC_CGM0_MUX11               : 25000000 Hz
GMAC0_RX                    : 25000000 Hz
Mux without a valid source
MC_CGM0_MUX15               : 0 Hz
Mux without a valid source
GMAC0_REF_DIV               : 0 Hz
Mux without a valid source
GMAC0_REF                   : 0 Hz
MC_CGM0_MUX16               : 100000000 Hz
SPI                         : 100000000 Hz
MC_CGM0_MUX12               : 800000000 Hz
QSPI_2X                     : 400000000 Hz
QSPI                        : 200000000 Hz
MC_CGM0_MUX14               : 800000000 Hz
SDHC                        : 400000000 Hz
DDR_PLL_MUX                 : 40000000 Hz
DDR_PLL_VCO                 : 1600000000 Hz
DDR_PLL_PHI0                : 800000000 Hz
MC_CGM5_MUX0                : 800000000 Hz
DDR                         : 800000000 Hz
ACCEL_PLL_MUX               : 40000000 Hz
ACCEL_PLL_VCO               : 1800000000 Hz
Mux without a valid source
MC_CGM0_MUX1                : 0 Hz
Mux without a valid source
Failed to get the frequency of CGM MUX
CLKOUT0                     : 0 Hz
Mux without a valid source
MC_CGM0_MUX2                : 0 Hz
Mux without a valid source
```

```
Failed to get the frequency of CGM MUX
CLKOUT1                 : 0 Hz
PER                     : 80000000 Hz
CAN_PE                  : 80000000 Hz
ACCEL_PLL_PHI0          : 0 Hz
ACCEL_PLL_PHI1          : 600000000 Hz
SERDES0_LANE1_CDR       : 125000000 Hz
SERDES0_LANE1_TX        : 125000000 Hz
PFE_MAC0_EXT_TX         : 0 Hz
PFE_MAC0_EXT_RX         : 0 Hz
PFE_MAC0_EXT_REF        : 50000000 Hz
PFE_MAC1_EXT_TX         : 0 Hz
PFE_MAC1_EXT_RX         : 0 Hz
PFE_MAC1_EXT_REF        : 50000000 Hz
PFE_MAC2_EXT_TX         : 0 Hz
PFE_MAC2_EXT_RX         : 125000000 Hz
PFE_MAC2_EXT_REF        : 50000000 Hz
SERDES1_LANE0_TX        : 125000000 Hz
SERDES1_LANE0_CDR       : 125000000 Hz
PFE_MAC0_REF_DIV        : 0 Hz
PFE_MAC1_REF_DIV        : 0 Hz
PFE_MAC2_REF_DIV        : 0 Hz
SERDES1_LANE1_TX        : 125000000 Hz
SERDES1_LANE1_CDR       : 125000000 Hz
PFE_MAC0_RX             : 125000000 Hz
PFE_MAC0_TX_DIV         : 125000000 Hz
MC_CGM2_MUX1            : 125000000 Hz
MC_CGM2_MUX4            : 125000000 Hz
MC_CGM2_MUX7            : 50000000 Hz
PFE_MAC1_RX             : 125000000 Hz
PFE_MAC1_TX             : 0 Hz
MC_CGM2_MUX2            : 125000000 Hz
MC_CGM2_MUX5            : 125000000 Hz
MC_CGM2_MUX8            : 50000000 Hz
```

```
PFE_MAC2_RX                 : 125000000 Hz
PFE_MAC2_TX                 : 125000000 Hz
MC_CGM2_MUX3                : 125000000 Hz
MC_CGM2_MUX6                : 125000000 Hz
MC_CGM2_MUX9                : 50000000 Hz
MC_CGM2_MUX0                : 600000000 Hz
PFE_SYS                     : 300000000 Hz
PFE_PE                      : 600000000 Hz
```

Kernel:

```
root@s32g274ardb2:~# cd /sys/kernel/debug/clk/
root@s32g274ardb2:/sys/kernel/debug/clk# cat clk_summary
                                 enable  prepare  protect                                duty
   clock                          count    count    count        rate   accuracy phase  cycle
---------------------------------------------------------------------------------------------
 pcf85063-clkout                     0        0        0       32768          0     0  50000
 serdes_ext                          0        0        0           0          0     0  50000
 fxosc                               3        3        0    40000000          0     0  50000
    accelpll_sel                     1        1        0    40000000          0     0  50000
       accelpll_vco                  1        1        0  1800000000          0     0  50000
          accelpll_phi1              1        1        0   600000000          0     0  50000
             pfe_pe_sel              1        1        0   600000000          0     0  50000
                pfe_sys_part_block   1        1        0   600000000          0     0  50000
                   pfe_pe            2        2        0   600000000          0     0  50000
                      pfe_sys        1        1        0   300000000          0     0  50000
          accelpll_phi0              0        0        0  1800000000          0     0  50000
    ddrpll_sel                       0        0        0    40000000          0     0  50000
       ddr_part_block                0        0        0    40000000          0     0  50000
          ddrpll_vco                 0        0        0  1600000000          0     0  50000
             ddrpll_phi0             0        0        0   800000000          0     0  50000
                ddr                  0        0        0   800000000          0     0  50000
    periphpll_sel                    1        1        0    40000000          0     0  50000
       periphpll_vco                 8        8        0  2000000000          0     0  50000
          periphll_dfs6              0        0        0           0          0     0  50000
          periphll_dfs5              0        0        0           0          0     0  50000
          periphll_dfs4              0        0        0           0          0     0  50000
```

```
periphll_dfs3          1    1    0  800000000    0    0  50000
  sdhc_sel             1    1    0  800000000    0    0  50000
    sdhc_part_block    1    1    0  800000000    0    0  50000
      sdhc             1    1    0  400000000    0    0  50000
periphll_dfs2          0    0    0          0    0    0  50000
periphll_dfs1          1    1    0  800000000    0    0  50000
  qspi_sel             1    1    0  800000000    0    0  50000
    qspi_2x            1    1    0  400000000    0    0  50000
      qspi_1x          2    2    0  200000000    0    0  50000
periphpll_phi7         1    1    0  100000000    0    0  50000
  dspi                 2    2    0  100000000    0    0  50000
periphpll_phi6         0    0    0 2000000000    0    0  50000
periphpll_phi5         2    2    0  125000000    0    0  50000
  pfe_emac_2_tx_sel    1    1    0  125000000    0    0  50000
    pfe2_tx_part_block 1    1    0  125000000    0    0  50000
      pfe_emac_2_tx    1    1    0  125000000    0    0  50000
  gmac_0_tx_sel        1    1    0  125000000    0    0  50000
    gmac_0_tx          1    1    0  125000000    0    0  50000
periphpll_phi4         1    1    0  200000000    0    0  50000
  gmac_0_ts_sel        1    1    0  200000000    0    0  50000
    gmac_0_ts          1    1    0  200000000    0    0  50000
periphpll_phi3         1    1    0  125000000    0    0  50000
  lin_baud             4    4    0  125000000    0    0  50000
    lin                3    3    0   62500000    0    0  50000
periphpll_phi2         0    0    0   80000000    0    0  50000
  can                  0    0    0   80000000    0    0  50000
periphpll_phi1         1    1    0   80000000    0    0  50000
  ftm1_ref_sel         0    0    0   80000000    0    0  50000
    ftm1_ref           0    0    0   40000000    0    0  50000
  ftm0_ref_sel         0    0    0   80000000    0    0  50000
    ftm0_ref           0    0    0   40000000    0    0  50000
  per_sel              1    1    0   80000000    0    0  50000
    per                2    4    0   80000000    0    0  50000
periphpll_phi0         1    1    0  100000000    0    0  50000
```

```
        serdes_int    2     2     0  100000000        0     0  50000
   armpll_sel         1     1     0   40000000        0     0  50000
     armpll_vco       1     1     0 2000000000        0     0  50000
       armpll_dfs6    0     0     0          0        0     0  50000
       armpll_dfs5    0     0     0          0        0     0  50000
       armpll_dfs4    0     0     0          0        0     0  50000
       armpll_dfs3    0     0     0          0        0     0  50000
       armpll_dfs2    0     0     0          0        0     0  50000
       armpll_dfs1    1     1     0  800000000        0     0  50000
         xbar_sel     1     1     0  800000000        0     0  50000
           xbar      11    11     0  400000000        0     0  50000
             xbar_div6  0     0     0  66666666        0     0  50000
             xbar_div4  1     1     0 100000000        0     0  50000
             xbar_div3  3     6     0 133333333        0     0  50000
             xbar_div2  0     0     0 200000000        0     0  50000
       armpll_phi1      0     0     0 2000000000       0     0  50000
       armpll_phi0      0     0     0 1000000000       0     0  50000
         a53_core       0     0     0 1000000000       0     0  50000
           a53_core_div10  0     0     0 100000000        0     0  50000
           a53_core_div2   0     0     0 500000000        0     0  50000
dummy                     0     0     0          0        0     0  50000
sysclk                    0     0     0   10000000        0     0  50000
serdes_100_ext            1     1     0  100000000        0     0  50000
serdes_125_ext            1     1     0  125000000        0     0  50000
sirc                  0     0     0     32000        0     0  50000
firc                  2     2     0  48000000        0     0  50000
  pfe_emac_1_tx_sel       0     0     0  48000000        0     0  50000
    pfe1_tx_part_block    0     0     0  48000000        0     0  50000
      pfe_emac_1_tx       0     0     0  48000000        0     0  50000
pfe_emac_0_tx_sel         1     1     0         0        0     0  50000
  pfe0_tx_part_block      1     1     0         0        0     0  50000
    pfe_emac_0_tx         1     1     0         0        0     0  50000
gmac_0_rx                 1     1     0         0        0     0  50000
root@s32g274ardb2:/sys/kernel/debug/clk#
```

2. 所谓的“保持不变”,即不做修改，而“不受 MCU 控制”是指将此项下的 under MCU control=unchecked。即 M 核代码不去初始化此时钟。所以除开 UART 的时钟，大部分其它外设可以修改为“不受 MCU 控制”。

3. Core CLK tree 如下：

23.1.2.1 Core-related clock overview

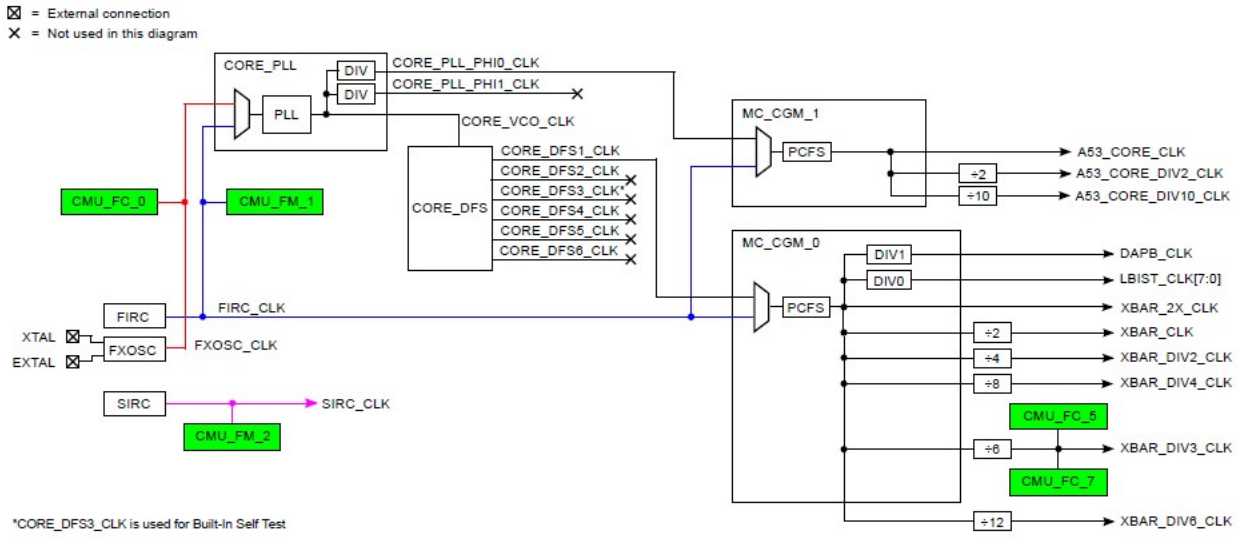


Figure 49. Core-related clock overview

23.7.11.1 Cortex-A53 cluster clocking

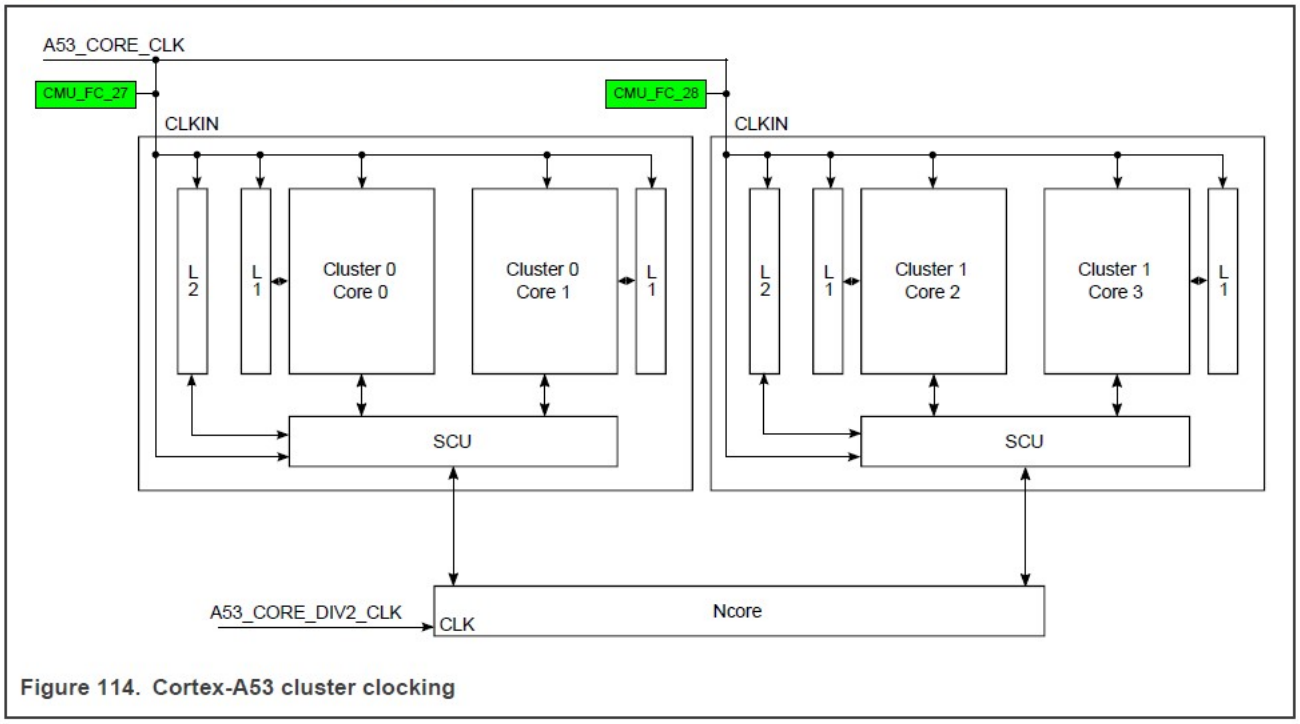


Figure 114. Cortex-A53 cluster clocking

23.7.11.2 Cortex-M7 clocking

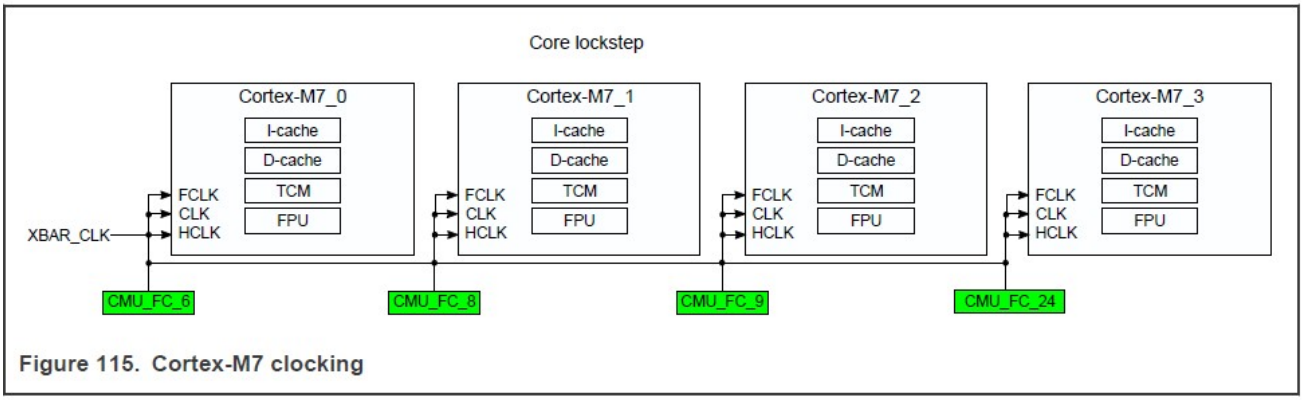


Figure 115. Cortex-M7 clocking

所以：

- …McuClockSettingConfig_0->McuCorePll:

RDIV=2; MFD (1 -> 255)=50; PHI0 Division value (0 -> 255) ,则：

PLL_PHI0 Frequency (Calculated) (dynamic range)= 1.0E9 //A53 的时钟为 1G，修改为与 Linux 相同。

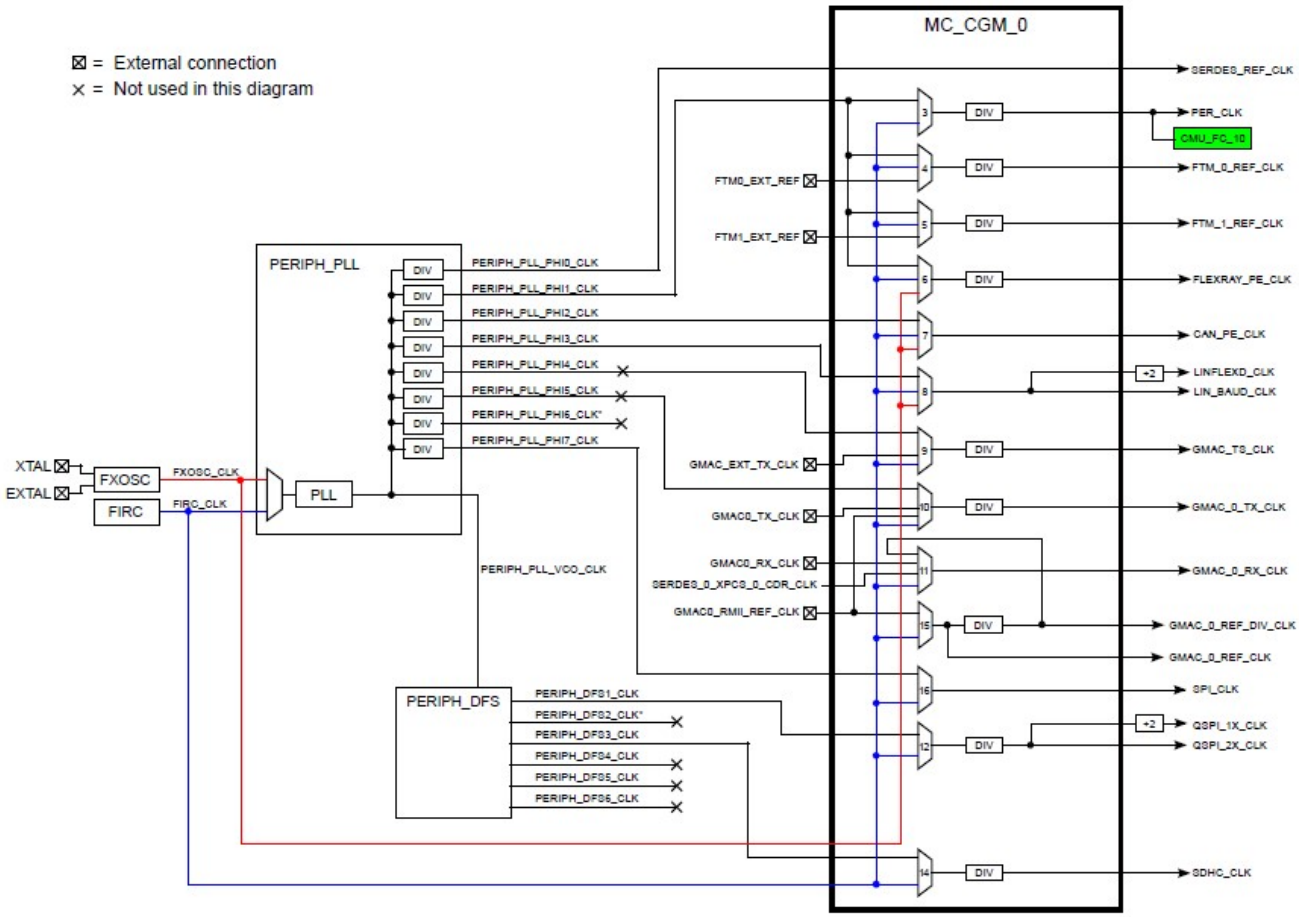
PLL_VCO Frequency (Calculated) (dynamic range)= 2.0E9

- …McuClockSettingConfig_0->McuCoreDFS: McuDfs_1

DFS1 MFI (1 -> 255)=1; DFS1 MFN (0 -> 35) =9;则：

DFS1_CLK Frequency (Calculated) (dynamic range)= 8.0E8 //则 M 核 时钟为 400Mh。

4. 外设 CLK tree 如下：



所以 LIN_BAUD CLOCK 的源是 FXOSC->PERIPH_PLL(2G)->PLL_PHI3_CLK(125M)->LIN_BAUD_CLK(125M)。

- …McuClockSettingConfig_0->McuPeriphPLL:

PHI3 Division value (0 -> 255)*= 15；PHI3 Divider enable=checked。则：

PLL_PHI3 Frequency (Calculated) (dynamic range)= 1.25E8。

- …McuClockSettingConfig_0->McuCgm0ClockMux8:

CGM0 Clock Mux8 Source=PERIPH_PLL_PHI3_CLK；

Clock Mux8 Frequency (LIN_BAUD_CLK) (dynamic range)自动计算为 1.25E8。

- ClockReferencePoint 增加 UART_CLK。

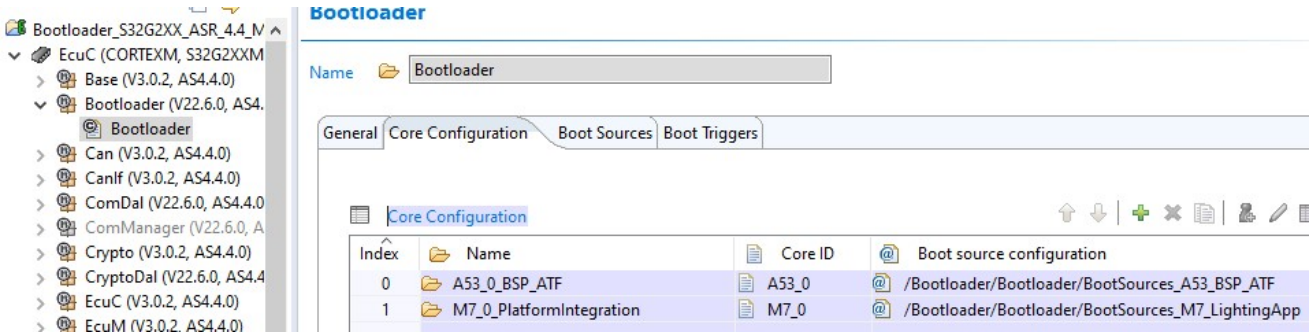
…McuClockSettingConfig_0->McuClockReferencePoint：点击右上角+号，增加一项，点击进入：

1. 修改 Name= UART_CLK
2. Mcu Clock Frequency Select：选择 LIN_BAUD_CLK
3. Mcu Clock Reference Point Frequency (0 -> 5000000000)：点击自动计算得到：1.25E8

其余外设的时钟修改主要就是如上所说，改为不受 MCU 控制。

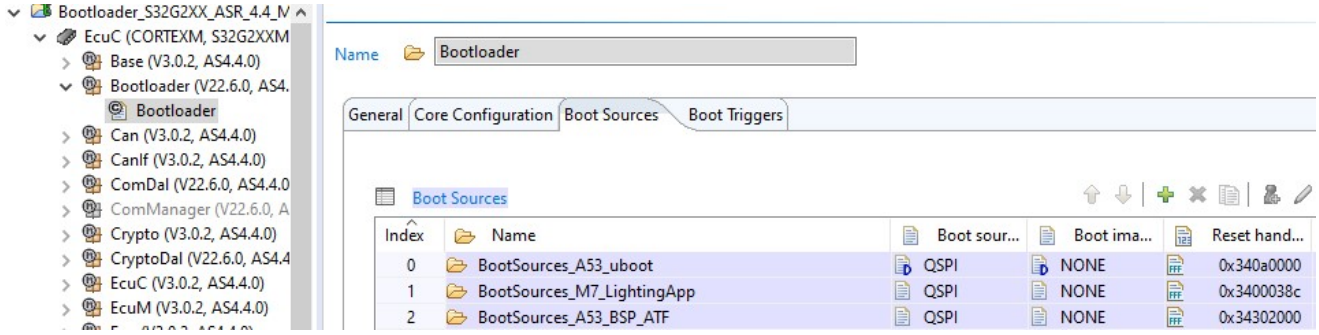
3.6 配置 A53 Boot sources:

1. 打开 EcuC(…)->Bootloader(…)->Bootloader->Core Configuration:默认已经配置为:



所以是启动 A53 的包含 ATF 的 Bootloader，和 M7_0 的一个 APP，注意其实还可以添加启动 M7_1/2，本文不做说明。

2. 打开 Bootloader(…)->Bootloader->Boot Sources: 以下有 reset 地址：



进入->BootSources_A53_BSP_ATF->Boot image fragment->ImageFragments_0:此外需要设置 ATF 的 BL2 的加载地址和大小，可以查看 ATF 的编译 Log 如下：

```
Image Layout
    DCD:                Offset: 0x200       Size: 0x1c
    IVT:                Offset: 0x1000      Size: 0x100
    AppBootCode Header: Offset: 0x1200      Size: 0x40
    Application:        Offset: 0x1240      Size: 0x2a800
IVT Location:   SD/eMMC
```

Load address: 0x343008c0

Entry point: 0x34302000

所以在 BootSources_A53_BSP_ATF->General->Reset handler address 中，保持值 0x34302000 不变。

在 BootSources_A53_BSP_ATF->Boot image fragments->ImageFragments_0 中：

Load image at address (RAM)= 0x343008c0 //load address。

注意注释说明：**“The address to load the image into RAM.**
NOTE !: The start address must be multiple of 8 if you choose CRC32 authentication method, otherwise must be multiple of 64.”

所以 load image 地址在编译 ATF 时要求 64 Byte 对齐。

Image size (bytes)= 262144=256KB= //>0x2a800+0x1240=0x2BA40。

3.7 配置 M7 Boot sources:

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

修改名字：Name= BootSources_MCAL_Test。

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 = 0x00180000 /* 1.5MB size, 4MB offsit */
```

参考编译生成 Mapping 文件：
C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\out\main.map

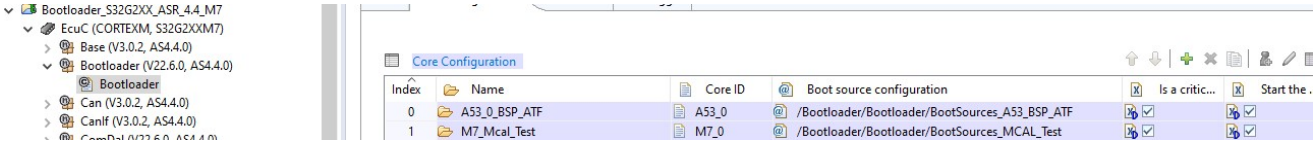
```
.startup      0x34400010     0x1d0 tmp/startup_cm7.o
              0x34400010               Reset_Handler
              0x34400010               _start
```

bin 文件大小为：1.28MBytes=0x140000

综上所述：

1. BootSources_M7_LLCEtoPFE_App->General-> Boot souce=QSPI //保持不变。
2. BootSources_M7_LLCEtoPFE_App->General ->Reset handler address =0x34400010
3. BootSources_M7_LLCEtoPFE_App->Boot image fragments->ImageFragments_0->Load image at address (RAM)= 0x34400000 //保持不变
4. BootSources_M7_LLCEtoPFE_App->Boot image fragments->ImageFragments_0-> Image size (bytes)= 1,314,816=0x141000>0x140000

5. BootSources_M7_LLCEtoPFE_App->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_Mcal_Test, Core ID 修改为：M7_0，Boot source configuraiton 指向/Bootloader/Bootloader/BootSources_MCAL_Test。



3.8 关闭调试软断点：

在 Bootloader(…)->Bootloader->General 中将 Software breakpoint enable uncheck 掉，关掉软件死循环，如下说明：

> This flag indicates whether the bootloader shall execute a wait-for-T32 loop or not. This needs to be configured when the bootloader is running from flash.
> - checked: wait for debugger;
> - unchecked: do not wait for debugger.

源代码说明：

```
/* Compile switch to enable debug breakpoint */
#define BL_USE_BREAKPOINT STD_ON //修改后变成 OFF
//bootloader.c
#if (BL_USE_BREAKPOINT == STD_ON)
/* Debugging macro used for stopping in the main function during debug. */
static volatile uint32 ENABLE_BREAKPOINT_AT_MAIN = 0U;
#endif /* BL_USE_BREAKPOINT == STD_ON */
int main(void)
{
#if (BL_USE_BREAKPOINT == STD_ON)
  while (0U == ENABLE_BREAKPOINT_AT_MAIN) continue;
#endif /* BL_USE_BREAKPOINT == STD_ON */
```

然后生成代码：

在 Bootloader(…)->EcuC(…)上面右击，然后选择 Generate Project 生成代码。生成代码位置在：C:\EB\tresos\workspace\Bootloader_S32G2XX_ASR_4.4_M7\output，注意，每次修改配置后要重新生成时，最好把以前此目录下所有文件删除。

注意 Imagefragments_0 中的 Source address(in QSPI)的配置：A53 是 0x100000，M7_0 是 0x200000，这个地址是接下来使用 flash tools 烧写 A 核 fip.bin 和 M 核*.bin 的地址。

3.9 编译 Bootloader 工程

1. 修改编译配置

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\launch.bat

```
::TRESOS_DIR: Root directory of the Tresos configuration tool, e.g. "C:/EB/tresos"
SET TRESOS_DIR=C:/EB/tresos
::MAKE_DIR: Root directory of the make tool, e.g. "C:/Program Files (x86)/GnuWin32"
SET MAKE_DIR=C:/cygwin64
::GHS_DIR: Root directory of the GreenHills toolchain, e.g. "C:/ghs/comp_201914"
::SET GHS_DIR=PATH/TO/GHS
::GCC_DIR: Root directory of the GCC toolchain, e.g. "C:/NXP/S32DS.3.2/S32DS/build_tools/gcc_v9.2"
SET GCC_DIR=C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2
::TOOLCHAIN
SET TOOLCHAIN=gcc
::CORE
SET CORE=m7
::SRC_PATH_DRIVERS: Path to the drivers plugins folder(default: %TRESOS_DIR%/plugins)
SET SRC_PATH_DRIVERS=C:/NXP/SW32G_RTD_4.4_3.0.2/eclipse/plugins
:: Path to the SDHC stack, e.g. "c:/NXP/S32DesignStudio/S32DS/software/PlatformSDK_S32G_2020_12/stacks/sdhc"
::SET SDHC_STACK_PATH=PATH/TO/SDHC/STACK //不使用 SDHC 驱动


::SRC_PATH_SAF: Path to the SAF plugins folder(default: %TRESOS_DIR%/plugins)
::SET SRC_PATH_SAF=PATH/TO/SAF/PLUGINS
::TRESOS_WORKSPACE_DIR: Tresos workspace folder, e.g. "%TRESOS_DIR%/workspace"
SET TRESOS_WORKSPACE_DIR=C:/EB/tresos/workspace/Bootloader_S32G2XX_ASR_4.4_M7/output
:: HSE_FIRMWARE_DIR Path to the HSE firmware directory. Needs to be set in case CryptoDal services are used
::SET HSE_FIRMWARE_DIR=C:/NXP/HSE_FW_S32G2_0_1_0_5 //不使用 cryptodal 驱动
…
```

在 cygwin 中运行:

```
/cygdrive/c/NXP/Integration_Reference_Examples_S32G2_2022_06/code/framework/realtime/swc/bootloader/platforms/S32G2XX/build
```

./launch.bat

编译。

如果遇到错误：

c:/nxp/s32ds.3.4/s32ds/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi/bin/../lib/gcc/arm-none-eabi/9.2.0/../../../../arm-none-eabi/bin/real

-ld.exe: bin_bootloader/Gpt_PBcfg.o:(.mcal_const_cfg+0x4): undefined reference to `OSIF_Millisecond'

那在 Bootloader(…)->EcuC(…)->Gpt(…)->Gpt->GptChannelConfiguration->GptChannelConfiguration_0:

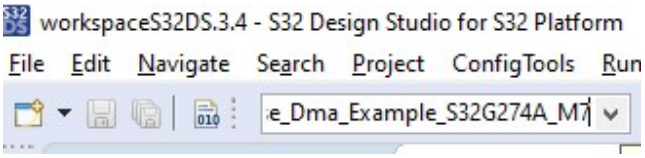
将 GptNotification 关掉。

编译成功后生成镜像：C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\bin_bootloader

Bootloader.bin，Bootloader.elf，Bootloader.map。

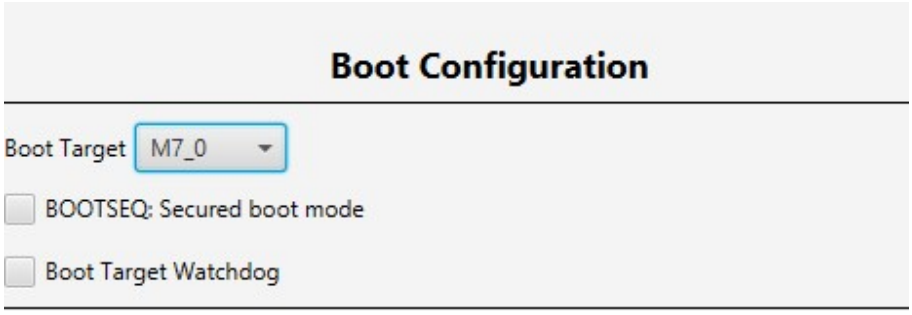
3.10 制造 Bootloader 的带 IVT 的镜像

打开 32 Design Studio for S32 Platform 3.4(3.4.3)，点击 ConfigTools 下拉菜单，选择 IVT。（注意：需要打开某个工程才能选择此菜单，标志为在工程下拉单中有工程名）：



打开 IVTView

- Boot Configuration:在 Boot Target 中选择：M7_0：

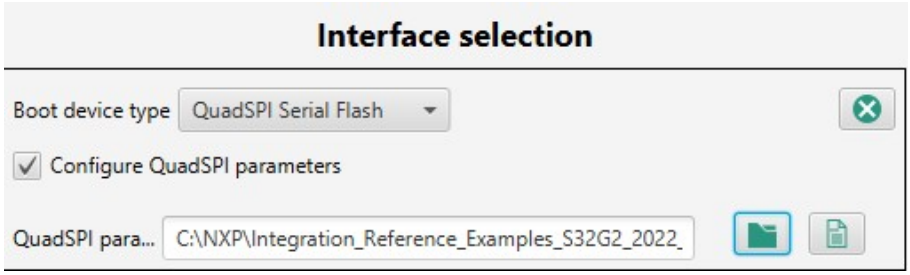


- Interface selection：在 Boot device type 中选择：QuadSPI Serial Flash，然后勾选 Configure QuadSPI parameters，选择 QSPI NOR 的时序头文件：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_QuadSPI_133MHz_DDR_configuration.bin

或者：

C:\NXP\S32DS.3.4\eclipse\mcu_data\processors\S32G274A_Rev2\PlatformSDK_S32XX_2022_03\quadspi\default_boot_images\mx25_sim200ddr.bin



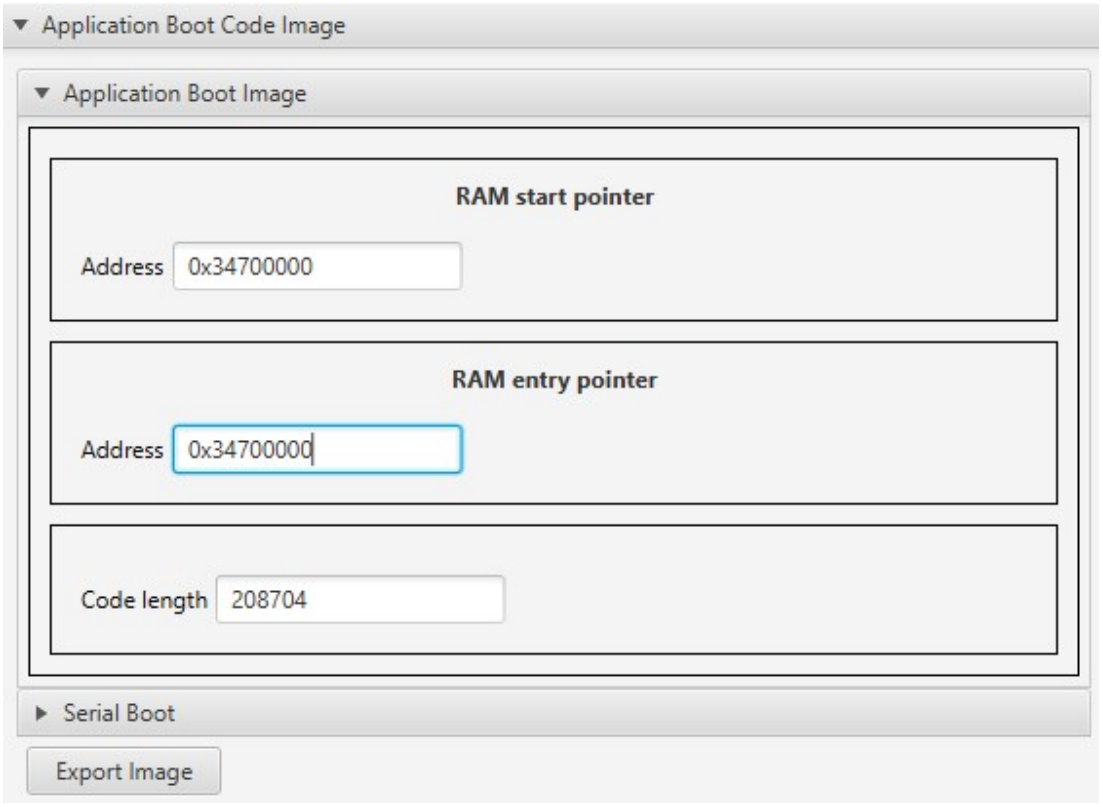
- DCD：DCD 段由内部 ROM 调用 的数据结构，用于初始化内部 SRAM。默认为 On 打开，选择文件为：C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_DCD_InitSRAM.bin

- Application bootloader：选择我们编译出来的 Bootloader：C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\bin_bootloader\Bootloader.bin

RAM start pointer 和 RAM entry pointer 参考 mapping 文件 Bootloader.map：

```
.all            0x34700000    0x29d28
                0x34700000              INT_SRAM_START = .
                0x34700000              . = ALIGN (0x4)
 *(.exception_table)
                0x34700000              . = ALIGN (0x4)
 *(.intc_vector)
 .intc_vector   0x34700000     0x200 bin_bootloader/Vector_core.o
                0x34700000              VTABLE
```

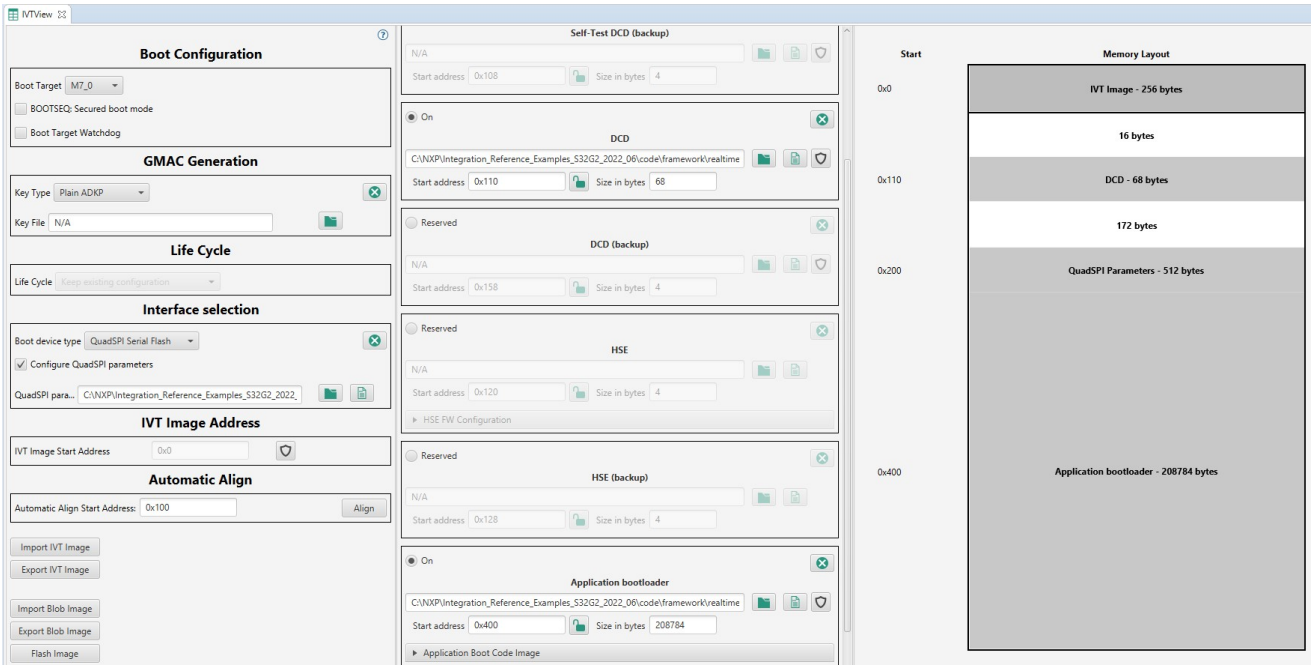
所以都是 0x34700000：



然后在 Automatic Align 中，输入 0x100，然后点击 Align，如果自动 Align 成功，会弹出成功提示框，如果失败，手动调节 Align 值，再试一下。Align 的要求一般是针对 eMMC 这种，QSPI NOR 要求不严格。

最后点击 Application Boot Image 中的 Export Image 按键，取名(bootloader_a_m.bin)保存在本地，然后这个文件会在 Application bootloader 中自动再打开。

- 其它不使用的镜像段全部点击 On 键关闭，变成 Reserved，点击 Export Blob Image 按键，导出最终的 Bootloader 镜像：



保存文件名为：bootloader_a_m_blob.bin。

3.11 烧写镜像

硬件连接：

- 使用 USB 线连接 PC 和 RDB2 板上的 UART0，J2
- RDB2 设置为下载模式：SW10-1=OFF, SW10-2=OFF。上电。

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

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“bootloader_a_m_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，烧写是注意是烧写 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*.bin 到 QSPI 中：

  点击 Upload file to device…，将“int_app.bin”烧写到地址 0x200000 处，这个是 LLCE to PFE 的 MCAL 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，将用 SDcard 读卡器在 Ubuntu 中烧写到 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 Align 后的 fip.s32：

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

  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。

4 测试

4.1 硬件连接

- 设置 RDB2 为正常启动模式，从 QSPI NOR 上启动：SW10-1=ON, SW10-2=OFF,SW4 all=OFF。
- 使用 USB 线连接 PC 和 RDB2 板上的 UART0，J2，打开 PC 端串口终端，设置为 115200, 8-n-1。
- DIO MCAL 测试需要 SW11=on，这样管脚连接到 LED GPIO 上，而不是 SPI。
- UART MCAL 测试需要再 PC 与 RDB2 板上的 UART1，，打开 PC 端串口终端，设置为 115200, 8-n-1。

4.2 MCU MCAL+Linux 测试过程

直接上电，在正常启动模式下启动，如果 A 核 Linux 终端正常启动，则说明 Bootloader Boot A 核成功，M 核 MCAL 镜像没有影响到 A 核启动。

4.3 DIO MCAL+Linux 测试过程

直接上电，在正常启动模式下启动，如果 A 核 Linux 终端正常启动，则说明 Bootloader Boot A 核成功。U128 GPIO LED 灯闪烁一段时间(A 核驱动加载 SPI 后停止)，说明 DIO MCAL M7_0 示例代码正常启动。

4.4 UART MCAL+Linux 测试过程

直接上电，在正常启动模式下启动，使用两根 USB 线分别连接 UART0/UART1。A 核 Linux 终端正常启动的打印在 UART0 终端中，则说明 Bootloader Boot A 核成功。UART1 连接的串口(115200,8-n-1)终端打印：

"This example is an simple echo using UART The board will greet you if you send 'Hello Board'
Now you can begin typing:"

5 Bootloader 源代码说明

Startup.s:
\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\src\m7

|-> SystemInit // Initialize the system (MPU, interrupts)

|->main:
\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\bootloader.c

| |->SysDal_Init

| | |->SysDal_StartUpInit

| | | |->SysDal_McuPlatformInitSeq

| | | | |->Mcu_Init，Mcu_SetMode，Mcu_InitClock(McuClockSettingConfig_0);，Mcu_DistributePllClock();// clock initial

| | | |->SysDal_WakeUpInit

| | | | |->InitBlockOneCallout

| | | | | |->//外设驱动初始化

Port_Init(&Port_Config);

Mcl_Init(&Mcl_Config);

Fls_Init(NULL_PTR);

SysDal_Rtm_Init();

CryptoDal_Init(&CryptoDal_Config[0]);

```
   Gpt_Init(&Gpt_Config);
```
| |->Bl_Run

| | |->Bl_BootApplications

| | | |->Bl_LoadApplication

| | | | |->Bl_FetchApplication

```
   switch (bootApplications[u8ApplicationId].bootSource)
   {
   case BS_QSPI:
   {
      Status = Bl_LoadAndAuthFromQspi(u8ApplicationId, u8FragmentIdx);
      break;
   }
#if (STD_ON == BL_SDHC_ENABLED)
   case BS_SDMMC:
   {
      Status = Bl_LoadFromSdhc(u8ApplicationId, u8FragmentIdx);
      break;
   }
#endif /* STD_ON == BL_SDHC_ENABLED */
```
| | | |->Bl_StartApplication //此为启动每个核的主程序：

输入参数为：

```
coreStartAddress = (applicationConfig->u32ResetHandler) & 0xFFFFFFFC; //reset 地址
   coreId = Bl_GetCoreInternalId(applicationConfig->core); //比如说 coreID=0，是第一个 A53 核
   partition = BL_GetPartition(applicationConfig->core);//比如说 A53 在 partition 1 里。
```

详细说明请参考芯片手册 Mode Entry Module MC_ME 一章中的 partition mapping。

以下为启动一个核的流程：

```
/* Set the core Vector Table address before enabling the partition. */
/* Enable clock partition. */
/* Trigger hardware process for enabling clocks. */
  /* Write the valid key sequence. */
/* Wait for partition clock status bit. */
/* Unlock software reset domain control register. */
/* Enable the interconnect interface of software reset domain */
/* Wait for software reset domain status register */
```

/* to acknowledge interconnect interface not disabled. */

/* Cluster reset. */

/* Write the valid key sequence. */

/* Wait until cluster is not in reset. */

/* Lock the reset domain controller. */

/* Enable core clock. */

/* Partition peripherals are always enabled in partition 0. */

/* Trigger hardware process for clock update. */

/* Write key sequence. */

/* Wait for clock to be enabled. */

/* Pull the core out of reset and wait for it. */

| | | |->SysDal_DeInit：/* De-init the peripherals and disable all interrupts, */ /* after all the applications have been loaded. */

| | | | |->SysDal_DeinitBlockOne //反初始化

SysDal_Rtm_DeInit();

Mcl_DeInit();

Mcu_InitClock(McuClockSettingsDisablePLL); //此代码应该消失掉了

Qspi_Ip_ControllerDeinit(0);

Gpt_DeInit();

6 Bootloader 定制说明

6.1 QSPI NOR 驱动说明

Bootloader 工程集成了 QSPI NOR 驱动，注意本驱动仅针对 RDB2 板上设计的 QSPI NOR flash，如果修改了此 QSPI NOR flash，请参考文档：

1. AN13563.pdf：《S32G QuadSPI Deep Dive》：本文主要说明 QSPI NOR MCAL 驱动定制。

https://www.nxp.com/products/processors-and-microcontrollers/s32-automotive-platform/s32g-vehicle-network-processors/s32g2-processors-for-vehicle-networking:S32G2

2. 《S32G_QSPINOR_定制_xxxxxxxx_Vx.pdf》：本文主要说明 QSPI NOR IVT 头 timming 配置头定制，以及 Linux 及 flash tools 工具定制。

https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-QSPI-Nor-customization-doc/ta-p/1399906

6.2 eMMC/SDcard 启动支持

Bootloader 工程默认是支持 SDHC 的 M7 核驱动的，请参考文档 Hands-on S32G2 Multicore application enablement example.pdf》Wang Xuewei 说明。

但是因为 eMMC/SDcard 启动比较慢，所以将 BL2 ATF 放在 QSPI-NOR 是比较通行的做法。

6.3 DDR 初始化

目前的 Linux BSP，从 BSP32 开始，均是默认支持 ATF 的版本，所以 Bootloader 会先加载 ATF 到内部 SRAM 中，再由 ATF 初始化外部 DDR，然后 ATF 负责从 eMMC/SDcard 中加载 Uboot 代码到外部 DDR 中。

而之前的 BSP 版本默认是不支持 ATF 的，所以需要把整个 Uboot 镜像加载到内部 SRAM 中，这样做占据的空间太大，所以一般的做法是在 Bootloader 中先初始化外部 DDR，然后直接用 Bootloader 从 eMMC 把 Uboot 读取到外部 DDR 中。所以需要将 Uboot 中的 LPDDR4 初始化的代码移植到 Bootloader 中。

随着 BSP 的升级，目前已经不需要。

6.4 Secure Boot 支持

请参考文档《Hands-on S32G2 Multicore application enablement example.pdf》Wang Xuewei 说明。

7 调试说明

7.1 Bootloader 的调试

Lauterbach 脚本位与：C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm

所以如 3.8 节，不关闭软件中断，则 Bootloader 代码会停留在最开始的地方，这个时候运行脚本连接调试器，即可以调试。

7.2 MCAL 驱动的调试

修改 Lauterbach 脚本：copy C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm to connect_s32gxx_m7_uart.cmm。修改如下：

Data.Load.Elf C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\out\main.elf /GLOBTYPES /NoCode

然后在 UART MCAL 的源代码 main 函数中增加：C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\src\main.c

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

则代码就会停在 while 将，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。